Programación en Lua

Luis Eduardo Muñoz

ISBN: 978-958-53925-9-5

Editado en Colombia - Abril 2023

Primera edición

Editorial CIMTED

Universidad Tecnológica de Pereira

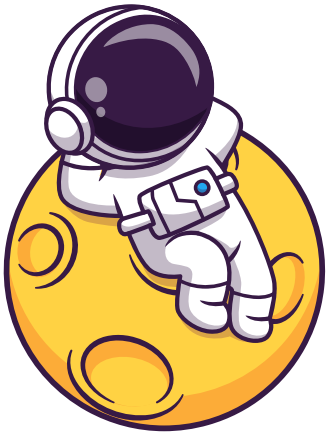
Programación en Lua

Autor:

Phd. Luis Eduardo Muñoz

Universidad Tecnológica de Pereira

Página legal



Título de la obra: Programación en Lua

ISBN: 978-958-53925-9-5

Autor: Luis Eduardo Muñoz Guerrero

Sello editorial: Corporación Centro Internacional de Marketing Territorial para la Educación y el Desarrollo

Editor: Corporación Centro Internacional de Marketing Territorial para la Educación y el Desarrollo.

Corporación CIMTED

NIT: 811043395-0

Materia: Programación

Tipo de Contenido: Computación y sistemas

Clasificación THEMA: - Lenguajes de programación y extensión/scripting: general

Público objetivo: Enseñanza universitaria o superior

Ciudad de Edición: Medellín

Idioma: Español

1ª Edición, Editado en Medellín - Colombia. Abril 2023

Fecha de aparición: 2023-04-14

Edición: Primera

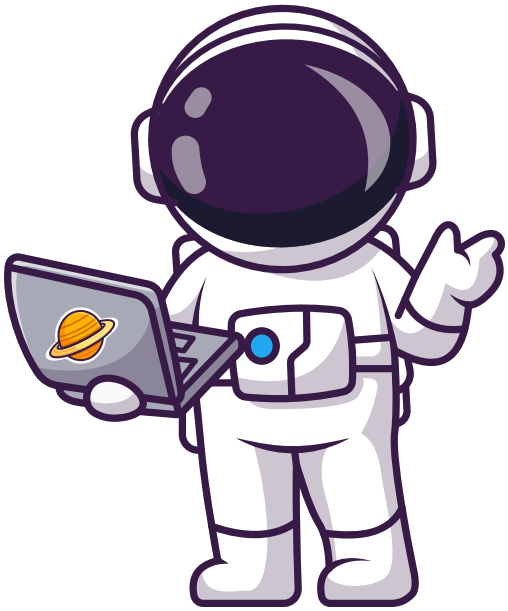
Tipo de soporte: Libro digital descargable

Formato: Pdf (.pdf)

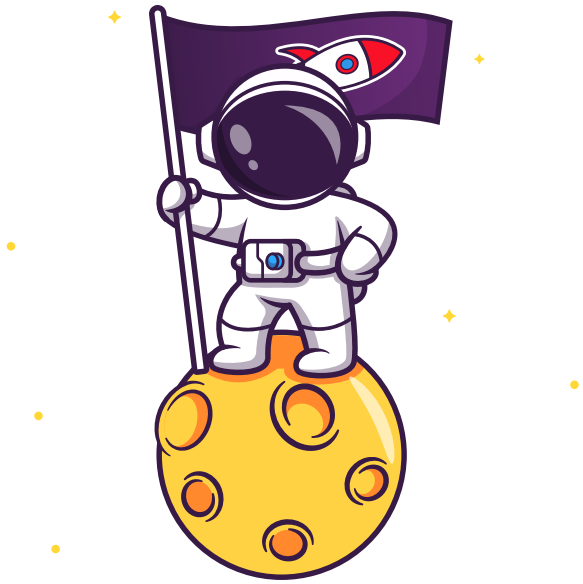
Tipo de contenido: Texto (legible a simple vista)

www.editorialcimted.com

www.memoriascimted.com



Editorial CIMTED



Cuidado de la edición:

Juliana Escobar Gómez

Calle 41 no 80b 120 int 301

Código postal: 050031

Medellín - Colombia

Abril - 2023

Programación en Lua/Editor: Corporación Centro Internacional de Marketing Territorial para la Educación y el Desarrollo CIMTED. 1ª edición, Medellín - Colombia

Corporación CIMTED sello editorial Centro Internacional de Marketing Territorial para la educación y el desarrollo. 2023

Páginas: 188

Incluye bibliografía

ISBN: 978-958-53925-9-5

Formato electrónico. Distribución gratuita. Puede descargarse desde: www.editorialcimted.com
www.memoriascimted.com

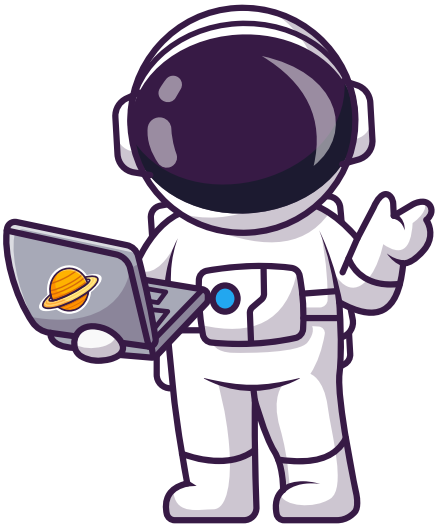
Sobre el autor:

Luis Eduardo Muñoz Guerrero

Universidad Tecnológica de Pereira Colombia

Ingeniero de sistemas, Magister en Ingeniera de Sistemas por la Universidad Nacional de Colombia, PhD en ciencias de la educación RudeColombia Cade UTP. Su experiencia de trabajo ha girado, principalmente, alrededor del campo educativo, sus proyectos están asociados con áreas de evaluación educativa, educación basada en competencias, software educativo, enseñanza de la programación. Ha publicado artículos en revistas nacionales e internaciones. Autor de los libros; Programación Moderna con aplicaciones y Programación Funcional con Racket. Actualmente es profesor titular de tiempo completo programa de Ingeniería de Sistemas y Computación de la Universidad Tecnológica de Pereira y Pertenece al grupo de investigación informática.

Correspondencia: lemunozg@utp.edu.co



Dedico este libro a mi madre por mostrarme la fortaleza, dedicación, intuición y confianza que me permiten continuar creciendo como padre de Alejandro, Luis y Camilo

Tabla de contenido

Tabla de ilustraciones

Tabla de fotografías

Introducción

Bienvenidos al libro "Programación en Lua". Este libro está diseñado para enseñar los fundamentos de la programación utilizando el lenguaje de programación Lua. Lua es un lenguaje de programación multiparadigma que tiene una sintaxis muy sencilla y fácil de aprender. Esto lo hace ideal para cualquier persona que quiera iniciarse en el mundo de la programación.

En este libro, aprenderás los fundamentos de la programación en Lua, así como la lógica necesaria para entender otros lenguajes de programación. No se requiere de una gran experiencia previa en programación, ya que cada sección ha sido diseñada para explicar los conceptos de manera sencilla y con ejemplos prácticos. ¡Incluso los principiantes podrán entenderlo!

Lua es un lenguaje de programación muy popular que se utiliza en muchas aplicaciones diferentes. Por ejemplo, muchos usuarios de la plataforma de videojuegos Roblox lo utilizan para crear sus propias experiencias de juego. Roblox permite personalizar muchos aspectos de la experiencia y el comportamiento mediante bloques de código escritos en Lua. ¡Esto hace que aprender Lua sea muy divertido!

Este libro ha sido diseñado para que cualquier persona, sin importar su edad o su nivel de experiencia, pueda aprender a programar en Lua. Incluso los niños pueden aprender a programar en Lua de una manera divertida e intuitiva. ¡Esperamos que disfrutes aprendiendo a programar en Lua y que te diviertas creando tus propios programas!

Fotografía 1 - La popular plataforma de videojuegos Roblox ofrece posibilidades de aprendizaje para niños, incluyendo formación en Lua (Baszucki, 2021).

¿Qué se aprenderá en este libro?

Al terminar la lectura de este libro, esperamos que el lector adquiera una sólida base de conocimientos en diferentes aspectos de programación en el lenguaje de programación en Lua, al practicar los ejemplos y realizar los ejercicios propuestos.

Para ello, se abordarán temas teóricos y prácticos centrados en Lua, tales como: qué es la programación y su importancia, cómo instalar y utilizar Lua en editores de código y terminales de ejecución, el uso de funciones básicas, buenas prácticas en la escritura de código, variables y tipos de datos, operaciones aritméticas, interacción con el usuario, operaciones de cadenas de texto, lógica proposicional, operadores lógicos, condicionales, bucles, iteraciones y algoritmos en Lua, solución de problemas complejos, anidación de estructuras condicionales y cíclicas, uso de variables locales y globales, tablas y listas, programación funcional, librerías y módulos en Lua, y manejo de errores.

Este libro cubre una amplia gama de temas esenciales para aquellos que deseen automatizar tareas o desarrollar sus propios proyectos en Lua, proporcionando una base sólida para el aprendizaje y la práctica.

Capítulo 1

¿Qué es la programación?

En esta sección, vamos a tener una breve introducción que te ayudará a entender de una forma sencilla y práctica los roles importantes que las computadoras y la programación han tenido en nuestra vida. Es emocionante saber que la programación es una habilidad cada vez más popular debido a la influencia tan importante que las computadoras han tenido en nuestra vida cotidiana.

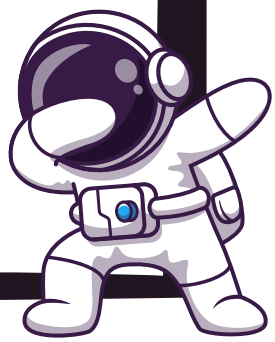
En este libro, profundizaremos en los temas que están directamente relacionados con la programación, explicando con más detalle lo que significa y los impactos relevantes que ha tenido en nuestras vidas. En términos simples, la programación trata de la habilidad de darle instrucciones a nuestra computadora. La habilidad de programar independientemente del lenguaje de programación (concepto que abordaremos en secciones posteriores) es fundamental en el mundo de la informática y la tecnología.

Es innegable que las computadoras son una presencia constante en nuestra sociedad actual. Durante décadas, han logrado grandes hazañas, como el alunizaje, la salvación de vidas y los avances tecnológicos que antes eran inimaginables. Indudablemente, las computadoras son una de las herramientas más útiles para el progreso humano, y lo seguirán siendo durante mucho tiempo.

A pesar de que es evidente la importancia que tienen las computadoras en nuestra vida diaria, no está de más saber cuál es su origen y cómo llegaron a ser tan relevantes en nuestro mundo moderno. Para entender su papel actual, es fundamental conocer su historia, los logros y los eventos más destacados que han estado ligados a ellas a lo largo del tiempo. De esta forma, podemos comprender mejor el papel que han desempeñado en la construcción de nuestra sociedad actual y cómo se han convertido en una herramienta indispensable para nuestra vida cotidiana.

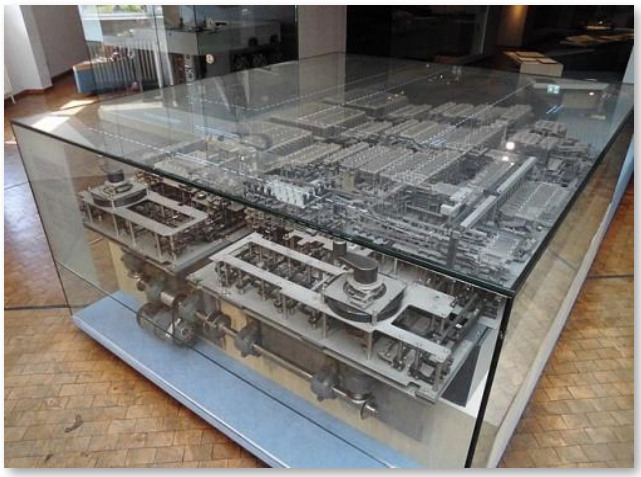
En conclusión

Presentamos en este capítulo introductorio el propósito y enfoque de "Programación en Lua", destacando la simplicidad y adaptabilidad de Lua como un lenguaje de programación perfecto tanto para principiantes como expertos. Asimismo, hemos introducido el contexto y relevancia de la programación en la sociedad actual, junto con el impacto que las computadoras han tenido en nuestras vidas. A lo largo de este libro, abordaremos diversos aspectos de la programación en Lua, desde conceptos básicos hasta temas más avanzados, con el objetivo de brindar a los lectores una base sólida y las habilidades requeridas para desarrollar sus propios proyectos y aplicaciones. Esta travesía de aprendizaje ha sido diseñada para ser accesible y entretenida, sin importar la edad o el nivel de experiencia del lector. Estamos entusiasmados por acompañarte en este emocionante camino hacia el dominio de Lua y la programación en general. ¡Adelante!



La historia de las computadoras y la programación

Una de las primeras computadoras en la historia fue la impresionante Z1, una computadora mecánica que se destacó en 1936 gracias a su capacidad para realizar cálculos en números binarios. Esta fue la primera computadora programable en el mundo que tenía la capacidad de usar lógica booleana, lo que marcó el inicio de una nueva era de desarrollo y avance tecnológico para la humanidad.



Fotografía 2 - Museo Alemán de Tecnología, Réplica de Z1 (Mike Peel)

Sin embargo, programar en la Z1 no era una tarea sencilla. Se requería el uso de largas y rígidas cintas perforadas con pequeños agujeros circulares que contenían información e instrucciones para la computadora. Estas cintas debían ser leídas por un lector de cintas dentro del mecanismo de la computadora. A pesar de las dificultades, la Z1 fue una pionera en su época y allanó el camino para las computadoras programables modernas que conocemos hoy en día.

En 1946, se dio un gran paso en la historia de la computación con la creación de ENIAC, la primera computadora eléctrica digital de propósito general. Este logro marcó un hito en la solución de problemas numéricos y permitió un avance tecnológico sin precedentes.

Una de las curiosidades más fascinantes de ENIAC es que su equipo de programación estaba conformado por seis mujeres, lo que representaba un gran cambio en la época. Además, esta computadora ofrecía la posibilidad de ser reprogramada para ejecutar diferentes tareas, lo que la hacía única en su tipo.

Sin duda, un gran logro para la humanidad que ha cambiado la forma en que interactuamos con el mundo actualmente.

Fotografía 3 - "Ejército de EUA" Patsy Simmers, Gail Taylor, Milly Beck y Norma Stec sosteniendo componentes de ENIAC, EDVAC, ORDVAC y BRLESC-I (Fotógrafo desconocido)

En 1953, se fabricó la computadora IBM 650, una de las primeras computadoras que se produjeron a gran escala para su venta. Esta máquina era programable en lenguaje ensamblador, que es un lenguaje de programación de bajo nivel. (En secciones posteriores del libro, se explicará qué son los lenguajes de programación de bajo nivel y cómo funcionan). El hecho de que la IBM 650 utilizara lenguaje ensamblador como "estándar" de programación para esa época, marcó un hito importante en la historia de la programación y la informática en general.

En 1975, se fundó una compañía que cambiaría la forma en que usamos las computadoras: Microsoft. Esta empresa informática es conocida por su línea de sistemas operativos, incluyendo el popular Microsoft Windows, el cual se ha convertido en el sistema operativo más utilizado en las computadoras de todo el mundo. Sin duda alguna, Microsoft ha dejado una huella en la historia de la informática y ha sido una pieza clave en la evolución de la tecnología de las computadoras.

Fotografía 4 - "En 1975, se fundó una compañía que cambiaría la forma en que usamos las computadoras: Microsoft" (Sipa Press/Rex Features)

En 1976, se fundó Apple, una compañía informática que revolucionó la industria con la creación del sistema operativo Mac OS, el segundo más utilizado en las computadoras actuales.

Con su enfoque en la innovación y el diseño, Apple rápidamente se convirtió en el competidor más grande de Microsoft, liderando el camino hacia una nueva era de tecnología y mejorando la experiencia de usuario en todo el mundo.

Fotografía 5 - "En 1976, se fundó Apple, una compañía informática que revolucionó la industria con la creación del sistema operativo Mac OS" (SAL VEDER/ ASSOCIATED PRESS)

En 1991, apareció un lenguaje de programación que hoy en día es uno de los más populares y usados: ¡Python! Es un lenguaje de programación de alto nivel con una gran comunidad de usuarios y desarrolladores que no deja de crecer. Actualmente, Python se utiliza en una amplia variedad de aplicaciones y mercados, lo que lo convierte en una herramienta muy versátil y útil.

¡Genial! Ahora podemos comprender aún mejor por qué las computadoras son tan importantes para nuestra sociedad. Pero ¿alguna vez te has preguntado cómo es que las computadoras funcionan en su interior? ¿Cómo es que siguen un conjunto de instrucciones y realizan diferentes tareas? ¡Es hora de descubrirlo!

Afortunadamente, los lenguajes de programación son la clave para entender cómo las computadoras realizan todas estas operaciones. Son los lenguajes de programación los que permiten que los programadores escriban un conjunto de instrucciones que las computadoras pueden seguir. Pero ¿cómo es que los lenguajes de programación logran esto? ¿Cómo es que las computadoras saben en qué orden deben realizar estas instrucciones y en qué condiciones? ¡Es hora de adentrarse en el mundo de la programación para descubrirlo!

La sociedad actual está en constante evolución, impulsada en gran medida por el desarrollo y uso de herramientas digitales. Estas herramientas, desde las más sencillas hasta las más complejas, son programadas para realizar tareas específicas. Gracias a la programación, las computadoras se han convertido en una de las herramientas más valiosas de la humanidad, permitiéndonos realizar tareas que antes parecían imposibles.

Ilustración 1 – "Estas herramientas, desde las más sencillas hasta las más complejas, son programadas para realizar tareas específicas"

La programación es un campo en constante crecimiento y su demanda laboral sigue aumentando en todo el mundo. Cada vez se necesitan más programadores capaces de desarrollar programas que se ajusten a las necesidades particulares de las empresas y ciudadanos. Esto se debe a que el avance y desarrollo continuo de nuevas tecnologías es cada vez mayor conforme pasan los años.

Al programar una computadora, se le da una serie de instrucciones detalladas que deben ser ejecutadas para lograr completar alguna tarea. Las computadoras son máquinas capaces de realizar operaciones matemáticas a una velocidad impresionante. Gracias a la programación y los avances en las ciencias de la computación, podemos utilizar esta capacidad a nuestro favor para realizar tareas complejas a una velocidad y eficiencia sin precedentes.

Por ejemplo, podemos darle a una computadora órdenes como: "Calcular la suma de los números 5 y 2", "Mostrar en pantalla el texto 'Hola mundo'", o "Pedir al usuario que escriba algún número y usarlo para multiplicarlo por 10". Estas órdenes pueden parecer triviales, pero son la base de la programación. Y a partir de ellas, podemos resolver problemas mucho más complejos, desde calcular velocidades hasta realizar operaciones con enormes conjuntos de datos.

La programación es fascinante porque nos permite resolver problemas complejos a partir de operaciones internas muy sencillas. Con el avance constante de la tecnología, la programación seguirá siendo una herramienta esencial para el progreso de la humanidad en el futuro.

La programación es una forma de dar instrucciones a una computadora para que realice tareas específicas. Es como enseñarle a un niño cómo hacer algo, pero en este caso, el "niño" es la computadora.

Ilustración 2 – "Se puede decir que la programación trata de indicar un conjunto de instrucciones dirigidas a una computadora"

Si estás interesado en aprender a programar, ¡no te preocupes! No es necesario ser un genio en matemáticas ni tener habilidades especiales. ¡Cualquier persona puede hacerlo! Además, las posibilidades que ofrece la programación son infinitas. Puedes crear aplicaciones, juegos, automatizar tareas y mucho más. Aprender a programar es una experiencia única y enriquecedora que puede abrir muchas puertas en el mundo de la tecnología.

Lua es un lenguaje de programación que ha demostrado ser muy útil a lo largo de la historia, ofreciendo un rendimiento notable en diversas tareas. Al ser un lenguaje de programación compilado, su compilador es realmente rápido, llegando a tener una velocidad similar al lenguaje C, desarrollado por Dennis Ritchie.

Este lenguaje no solo se aplica en el mundo de los videojuegos, como se menciona y se abordará en varias secciones de este libro. De hecho, Lua ofrece un amplio espectro de posibilidades y aplicaciones en distintos ámbitos. Es cierto que este libro se centra en enseñar los fundamentos básicos para que cualquier persona pueda aprender a programar desde cero con Lua. Por lo tanto, es probable que no se aborden todas las aplicaciones técnicas en detalle, ya que el objetivo principal es enseñar la programación desde un punto de vista básico y cómo se puede aplicar para resolver problemas sencillos.

Al igual que cualquier otro lenguaje de programación, no se puede decir que Lua sea superior a otro, ya que todos cumplen la misma función. La diversidad de aplicaciones de los lenguajes de programación en gran escala depende en gran medida de sus bibliotecas y de los aportes de la comunidad. En este aspecto, Lua no se queda atrás. A lo largo del libro, especialmente en las secciones introductorias, se mostrarán las diferentes aplicaciones y relevancias que Lua ha tenido en la industria, principalmente, pero no exclusivamente, en el desarrollo de videojuegos.

En Lua, se pueden aplicar diversos conceptos que pueden parecer complejos para el usuario promedio. Algunos de estos conceptos no se tratarán a fondo en este libro para mantener su naturaleza fácil de entender y centrarse en los fundamentos. Lua permite la implementación de interfaces gráficas, software 3D, control de sonido de alta fidelidad, inteligencia artificial, estructuras de datos personalizadas, administración de bases de datos, abstracción de procesos industriales y matemáticos complejos, y uso en sistemas embebidos, entre otros.

Una aplicación reciente e importante de Lua es el caso de MAD-NG, una herramienta versátil para el diseño, modelado y optimización de aceleradores de partículas, así como para estudios de dinámica de haz y óptica. Esta herramienta utiliza Lua como lenguaje de programación de propósito general debido a su simplicidad y potencia para manejar una herramienta de tal envergadura.

Otro ejemplo destacado de Lua es su uso como intermediario en la compilación de documentos en LaTeX, como en el caso de LuaTeX. La importancia de LuaTeX radica en su flexibilidad, la tipografía avanzada que ofrece y la posibilidad de programar en Lua dentro de los documentos de LaTeX, especialmente para automatizar procesos.

Ahora bien, aprender Lua también puede ser una excelente introducción al mundo de la programación para aquellos que deseen explorar otros lenguajes más adelante. Gracias a su simplicidad y enfoque en la legibilidad del código, los principiantes pueden adquirir habilidades esenciales de programación y luego aplicarlas en otros lenguajes más complejos, si así lo desean. Además, Lua se utiliza a menudo como lenguaje de scripting en otros lenguajes de programación, lo que amplía aún más su versatilidad y utilidad en diferentes campos.

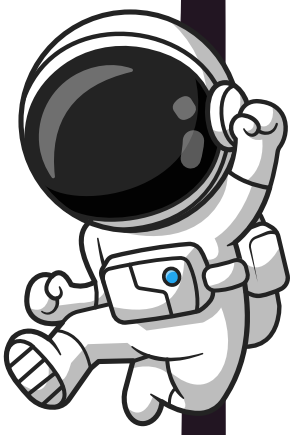
En este libro, los lectores también aprenderán sobre el enfoque de Lua en la portabilidad y su capacidad para funcionar en prácticamente cualquier sistema operativo. Esta característica hace que Lua sea una opción atractiva para desarrolladores que buscan crear soluciones multiplataforma. Además, el lenguaje es altamente extensible, lo que significa que los programadores pueden añadir funcionalidades adicionales según sea necesario, facilitando así la adaptación a las necesidades específicas de cada proyecto.

Uno de los aspectos más destacados de Lua es su eficiente recolector de basura. En programación, la gestión de la memoria es un componente crítico para garantizar el rendimiento y la estabilidad de las aplicaciones. El recolector de basura de Lua se encarga de liberar automáticamente la memoria que ya no se necesita, permitiendo a los programadores centrarse en el desarrollo de sus aplicaciones sin preocuparse tanto por la administración de la memoria. Esta característica es especialmente útil para aquellos que se inician en la programación, ya que minimiza los errores relacionados con la gestión de la memoria y facilita el aprendizaje.

Por último, en este libro se guiará a los lectores a través de ejemplos prácticos y ejercicios que les permitirán aplicar los conceptos aprendidos de manera efectiva. Estos ejemplos cubrirán una amplia gama de temas. Al completar estos ejercicios, los lectores no solo adquirirán una sólida comprensión de los fundamentos de Lua, sino que también desarrollarán habilidades prácticas que les permitirán abordar sus propios proyectos con confianza.

En resumen

Exploramos la historia de las computadoras y la programación, destacando la evolución de las máquinas desde la Z1 hasta las computadoras modernas. Hemos visto cómo se han creado empresas tecnológicas influyentes como Microsoft y Apple, y cómo han surgido lenguajes de programación populares como Python. La programación es una herramienta esencial para el progreso humano y su demanda laboral sigue en aumento. Lua es un lenguaje de programación versátil y útil que ofrece una introducción accesible al mundo de la programación. Este libro se centra en enseñar los fundamentos de Lua y, a través de ejemplos prácticos y ejercicios, los lectores adquirirán habilidades esenciales de programación que les permitirán enfrentar sus propios proyectos con confianza.



Los lenguajes de programación.

A lo largo de la historia, los seres humanos han encontrado diferentes formas de comunicarse con aquellos que les rodean. A medida que evolucionamos, hemos desarrollado una variedad de idiomas que nos permiten expresar ideas y comunicar información de manera efectiva. El español, inglés, japonés, catalán, alemán y muchos otros, son algunos ejemplos de estos idiomas.

De manera similar, existen diferentes lenguajes de programación, cada uno con sus propias reglas sintácticas y lógicas. No hay un lenguaje de programación universal y cada uno se enfoca en una aplicación específica. Algunos lenguajes están diseñados para ser amigables y comprensibles para niños, mientras que otros están enfocados en resolver problemas matemáticos y físicos a gran escala. Hay una gran variedad de lenguajes de programación disponibles para cubrir una amplia gama de necesidades.

En resumen, así como la variedad de idiomas permite una comunicación efectiva entre las personas, la variedad de lenguajes de programación permite que los programadores elijan la herramienta adecuada para su proyecto específico.

Un lenguaje de programación es como un puente entre la mente del programador y la computadora. Es una forma de comunicarse con la máquina y darle una serie de instrucciones para que realice una tarea determinada.

Hay muchos lenguajes de programación disponibles en la actualidad, como Lua, Python, Rust, C++ y GoLang, por nombrar algunos. En este libro, nos centraremos en el lenguaje de programación Lua, que es un lenguaje de programación de alto nivel que ofrece una gran cantidad de facilidades y produce resultados rápidos y efectivos.

Ilustración 3 – Lenguajes de programación (C++, Lua y Python en la imagen)

Es común tener preguntas acerca de los lenguajes de programación, como qué son exactamente y cuál es el mejor para comenzar a aprender. Sin embargo, antes de responder estas preguntas, es importante tener algunos conceptos claros sobre el tema.

Las computadoras de hoy en día utilizan un sistema numérico binario para funcionar, lo que significa que todo lo que realizan se basa en únicamente dos números: el número uno (1) y el número cero (0). Aunque pueda sonar un poco extraño para aquellos que no están familiarizados con el tema, es gracias a este sistema binario que las computadoras pueden hacer cosas increíbles, incluyendo la creación de interfaces gráficas y la interacción con el usuario.

Ilustración 4– Esquematización de la función del lenguaje de programación.

Aunque pueda parecer que las computadoras hacen todo por sí solas, en realidad detrás de su gran poder de procesamiento matemático se encuentran operaciones numéricas realizadas en el sistema binario. Por supuesto, no es necesario entender completamente cómo funciona todo esto para poder usar una computadora, ya que los lenguajes de programación facilitan la comunicación con ellas y permiten que las personas creen programas de manera eficiente y efectiva. En otras palabras, no es necesario ser un experto en matemáticas o informática para sacar provecho de las computadoras y sus capacidades.

Las computadoras son una parte esencial de nuestra vida cotidiana. Han permitido grandes avances tecnológicos, como llevar al hombre a la luna o salvar incontables vidas. Es indudable que su papel en la sociedad actual es fundamental para el progreso continuo de la humanidad.

Para entender el papel que tienen las computadoras en nuestra vida diaria, es importante conocer su origen. Así podremos entender su evolución y cómo se han convertido en herramientas vitales para nuestra sociedad.

Con el surgimiento de nuevos lenguajes de programación más rápidos y poderosos, el desarrollo de aplicaciones en informática se ha facilitado. Estos lenguajes hacen que el manejo de código e interpretación de este sea más sencillo, utilizando declaraciones relacionadas con palabras que podemos comprender fácilmente.

Las computadoras funcionan con números binarios, lo que significa que trabajan con electricidad. Los transistores son los encargados de conducir esa electricidad en su interior. A simple vista, leer código binario es una tarea difícil para los seres humanos. Sin embargo, somos capaces de entender escritos más complejos, ya que contienen una mayor cantidad de posibles caracteres, como la oración "Los planetas orbitan alrededor de sus estrellas".

La diferencia entre los bloques de información radica en que el código binario está diseñado para ser interpretado por una computadora, mientras que la oración sobre los planetas está pensada para ser comprendida por los seres humanos. Es por eso por lo que el código binario se conoce como "lenguaje de máquina", mientras que los lenguajes de programación actúan como puentes entre el lenguaje humano y el de máquina, siendo diseñados para ser fáciles de leer por seres humanos y ser traducidos a lenguaje de máquina para su ejecución en la computadora.

En la actualidad, existen dos tipos de lenguajes de programación: los lenguajes de programación de alto nivel y los lenguajes de programación de bajo nivel. Los lenguajes de programación de alto nivel tienen una estructura similar al lenguaje humano, lo que significa que gran parte de su código es fácil de entender si se conocen las palabras en su idioma, generalmente en inglés. En contraste, los lenguajes de programación de bajo nivel tienen una estructura más compleja y difícil de leer a simple vista.

Si estás empezando en el mundo de la programación, te alegrará saber que Lua, el lenguaje de programación que se trata en este libro es un lenguaje de programación de alto nivel. Esto lo hace fácil de aprender, ya que no se necesita un gran conocimiento de informática para poder entender y escribir código en este lenguaje. De hecho, te mostraremos un ejemplo de código escrito en Lua:

```
1    print(“Hola mundo”)
```

¡Genial! Has encontrado el primer bloque de código escrito en Lua de este libro. Pero no te preocupes si parece complicado al principio, ya que lo único que hace este código es mostrar el texto "Hola mundo" en la pantalla.

Por otro lado, los lenguajes de programación de bajo nivel se asemejan más al lenguaje de la máquina que al lenguaje humano, lo que significa que tienen una escritura de código mucho más compleja y difícil de leer. Aunque no es necesario tener conocimientos previos en lenguajes de programación de bajo nivel, es importante comprender la diferencia entre los dos tipos de lenguajes de programación. Al ser Lua un lenguaje de programación de alto nivel, la comprensión del código y la detección de errores son mucho más sencillas y amigables para el usuario.

Cuando se desea aprender a programar es muy común toparse con la duda: ¿Qué lenguaje de programación se debe aprender primero? Y la respuesta es mucho más sencilla de lo que parece. Como se mencionó en varias ocasiones a lo largo de este libro, un lenguaje de programación es solamente un intermediario entre los seres humanos y el lenguaje de máquina.

Existen una gran variedad de lenguajes de programación, tanto de alto como de bajo nivel, lo cual ofrece a quienes desean aprender a programar una amplia gama de opciones para facilitar su proceso de aprendizaje.

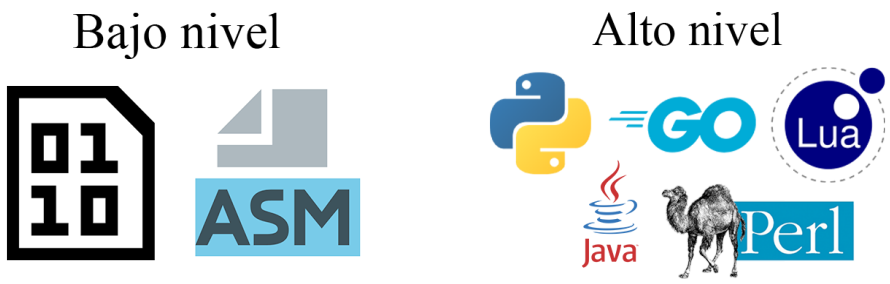
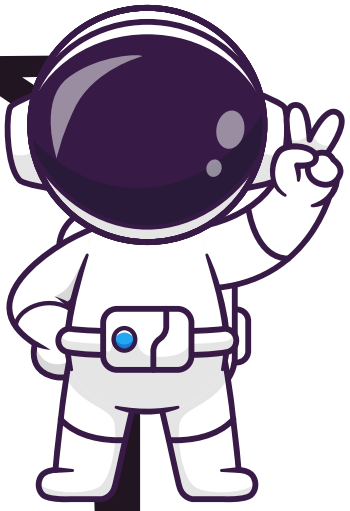


Ilustración 5 – Ejemplos de diferentes lenguajes de programación de bajo y alto nivel

Al igual que los humanos pueden expresar el mismo mensaje en diferentes idiomas, los principios de programación son universales y no cambian, independientemente del lenguaje de programación utilizado.

El lenguaje de programación Lua ha sido siempre fiel a este concepto, lo que significa que aprender Lua no solo permite entender los principios fundamentales de la programación, sino que también es una excelente base para aprender otros lenguajes de programación.

En conclusión



En conclusión, el mundo de la programación es amplio y diverso, con una gran variedad de lenguajes de programación disponibles para satisfacer diferentes necesidades y niveles de experiencia. Lua es un lenguaje de programación de alto nivel que facilita el aprendizaje de los conceptos básicos de programación y proporciona una base sólida para explorar otros lenguajes en el futuro. Al igual que los idiomas humanos, cada lenguaje de programación tiene su propósito y aplicación, y aprender a programar es una habilidad valiosa que puede abrir muchas puertas en el mundo tecnológico actual

Capítulo 2

Primeros pasos en Lua: ¿Qué es Lua?

Antes de sumergirnos en el mundo de Lua, es importante entender de qué se trata. El nombre de este lenguaje de programación significa "Luna" en portugués, haciendo referencia al satélite natural de nuestro planeta. La imagen que evoca la órbita lunar alrededor de la Tierra se refleja en la facilidad y la elegancia que caracterizan a este lenguaje.

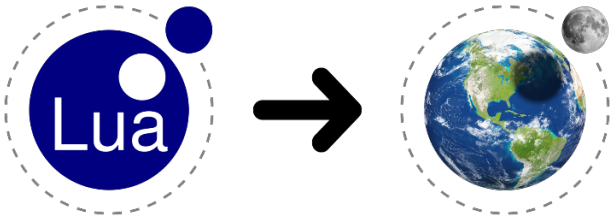


Ilustración 6 - El nombre de este lenguaje de programación significa "Luna" en portugués, haciendo referencia al satélite natural de nuestro planeta

Lua es un lenguaje de programación de alto nivel que resulta muy accesible para quienes quieren iniciarse en la programación sin experiencia previa. Sin embargo, su facilidad no significa que sea un lenguaje limitado, al contrario, posee una gran potencia que permite aprovechar al máximo las posibilidades de otros lenguajes más complejos.

Una de las ventajas más destacadas de Lua es que es un lenguaje liviano y fácil de utilizar, lo cual no afecta su capacidad para desarrollar aplicaciones complejas y sofisticadas. Además, es un lenguaje multiparadigma, lo que significa que se puede utilizar en diferentes tipos de programación: orientada a objetos, funcional o imperativa. En un capítulo posterior del libro profundizaremos en estos conceptos y en cómo aplicarlos a Lua.

Lua fue creado en 1993 por el Grupo de Tecnología en Computación Gráfica de la Pontificia Universidad Católica de Río de Janeiro, Brasil. Desde entonces, ha evolucionado para convertirse en uno de los lenguajes más versátiles y populares del mundo del desarrollo de videojuegos, aunque también ha sido utilizado en otros campos laborales y de estudio, como servidores y aplicaciones de diferentes tipos.

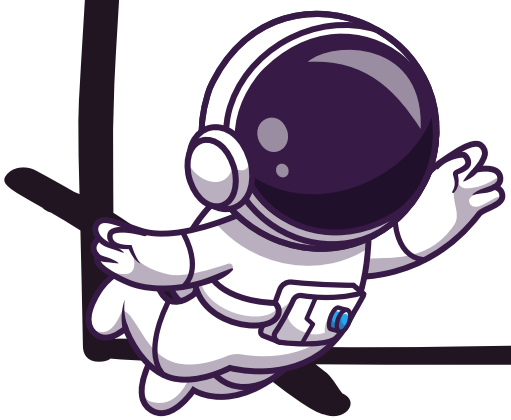
Dentro del mundo de los videojuegos, Lua se ha utilizado para desarrollar algunos de los juegos más famosos, como World of Warcraft, Stepmania y Roblox, entre otros. Su portabilidad y velocidad lo hacen ideal para ser incorporado en aplicaciones, lo que lo convierte en una herramienta muy poderosa y versátil.

Fotografía 6 - World of Warcraft y Roblox, dos ejemplos de juegos que usan Lua.

Aquí te ofreceremos la oportunidad de explorar en profundidad el fascinante mundo de Lua. Aprenderemos juntos cómo crear diversos tipos de programas y resolver problemas a través del uso de este lenguaje. Pero eso no es todo, porque programar va más allá de estas tareas básicas: ¡te abrirá las puertas a un sinfín de posibilidades emocionantes!

En conclusión

En este primer capítulo introducimos el lenguaje de programación Lua, destacando su origen, facilidad de uso y versatilidad en distintos ámbitos, especialmente en el desarrollo de videojuegos. Mencionamos también su capacidad para abordar diferentes paradigmas de programación, lo cual exploraremos en capítulos posteriores. Al aprender Lua, no solo adquirirás habilidades en programación, sino que también te adentrarás en un mundo lleno de posibilidades emocionantes e ilimitadas.



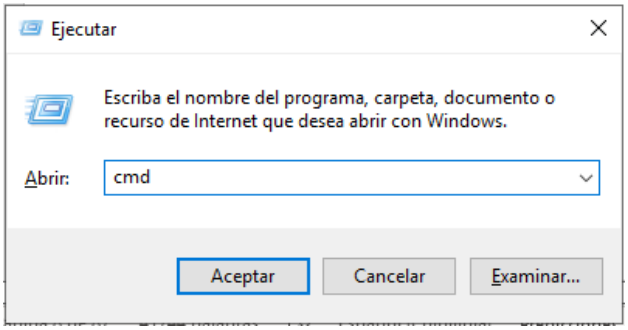
Instalación de Lua en Windows: ¿Cómo saber si nuestra computadora es de 32 o 64 bits?

Cuando queremos instalar software en nuestra computadora Windows, es importante tener en cuenta algunos conceptos clave relacionados con el procesador y la compatibilidad del software que deseamos instalar. En general, hay dos tipos de procesadores utilizados en las computadoras modernas: los procesadores de 64 bits y los procesadores de 32 bits.

Aunque es probable que estos conceptos te resulten familiares, es importante saber que un procesador de 64 bits tiene teóricamente un mejor rendimiento y capacidad para procesar datos que uno de 32 bits. Aunque cada vez son menos comunes, todavía podemos encontrar computadoras con procesadores de 32 bits, por lo que muchos desarrolladores de software consideran a estos usuarios al ofrecer soporte para sistemas operativos de 32 bits. En el caso de Lua, sus desarrolladores han tenido en cuenta a los usuarios con computadoras de 32 bits para asegurarse de que puedan ejecutar este lenguaje de programación sin problemas de compatibilidad.

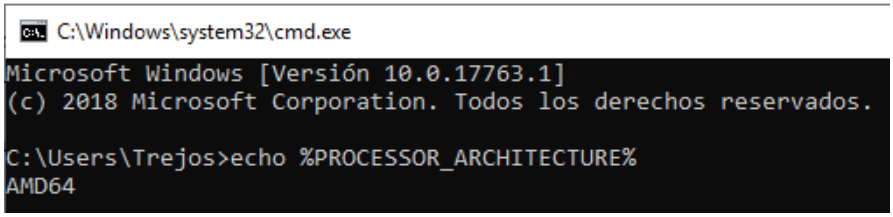
En esta guía paso a paso, te ayudaremos a identificar el sistema operativo que utiliza tu computadora, ya sea de 32 o 64 bits, para que puedas instalar la versión correspondiente de Lua. Si ya sabes qué tipo de procesador tiene tu sistema operativo, puedes saltarte este paso, ya que solo necesitamos esta información para instalar el software de Lua. No te preocupes, los pasos son sencillos y te guiaremos en todo momento para que puedas instalar Lua sin problemas.

1. Para empezar, abramos una nueva terminal de comandos en nuestro sistema. Para hacerlo, podemos presionar [Windows + R] en nuestro teclado y escribir "cmd" en la casilla que aparece. ¡Listo!



2. Una vez que tengamos la nueva terminal abierta, escribimos el siguiente comando:

echo %PROCESSOR_ARCHITECTURE%. Este comando nos mostrará la arquitectura de nuestro procesador y sistema operativo, que puede ser de 64 bits (AMD64) o de 32 bits.



Si el resultado del comando no es "AMD64", entonces nuestro procesador y sistema operativo son de 32 bits. Es importante saber esto para poder instalar el software correcto en base a la arquitectura de nuestro sistema.

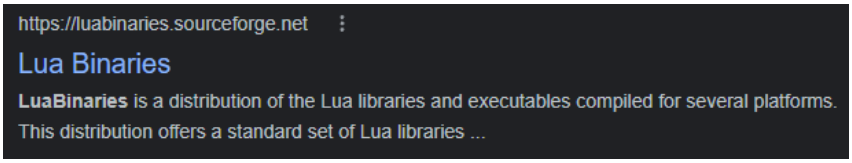
Instalación de Lua en Windows: ¿Cómo descargar e instalar Lua dentro del sistema operativo Windows?

Ahora, en esta sección vamos a aprender cómo podemos instalar el lenguaje de programación Lua dentro de nuestro sistema operativo Windows.

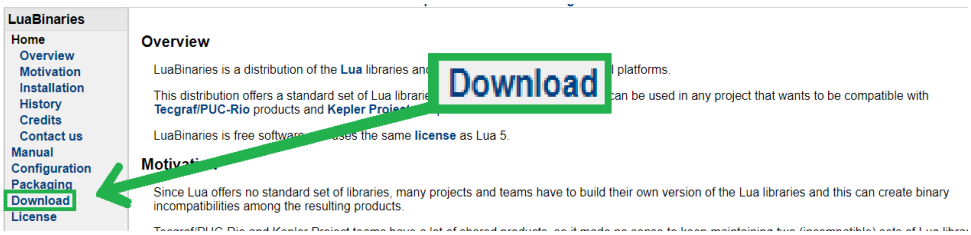
Aunque no es necesario tener instalado el lenguaje de programación para escribir código en Lua, es importante tener un intérprete que pueda ejecutar los archivos y comandos relacionados con Lua en nuestro sistema. Desafortunadamente, Windows no viene con un intérprete de Lua incorporado, así que necesitamos instalar uno nosotros mismos. ¡Pero no te preocupes! Esta sección te guiará a través de los pasos necesarios para instalar Lua en tu sistema sin problemas.

Los pasos son fáciles de seguir, pero debemos hacerlo con cuidado para evitar errores durante la instalación. Si nos encontramos con algún problema, siempre podemos buscar guías o tutoriales en internet para ayudarnos a solucionarlo.

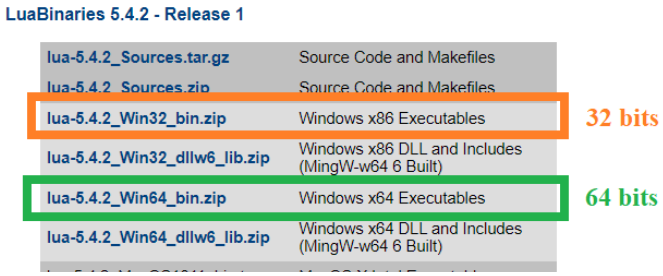
1. Lo primero que necesitamos hacer es buscar "lua binaries" en Google. Una buena opción suele ser el sitio web de SourceForge, que suele aparecer en los primeros resultados de búsqueda.



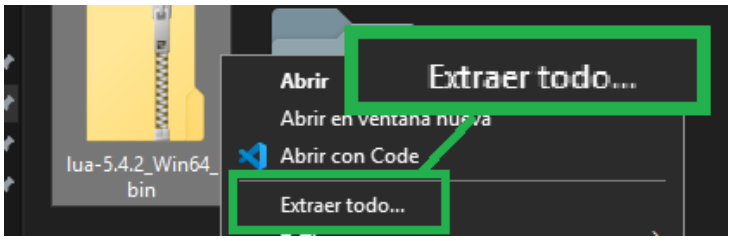
2. Una vez en el sitio, buscaremos la sección "Download" en el menú de navegación izquierdo y haremos clic en ella.



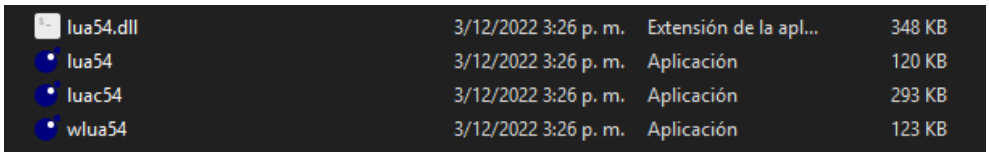
3. En la sección de descarga, encontraremos una lista de todas las versiones de Lua disponibles. Siempre es recomendable instalar la versión más reciente para disfrutar de todas las novedades. También debemos asegurarnos de descargar la versión correspondiente a nuestra arquitectura (32 o 64 bits) y la versión de Windows.

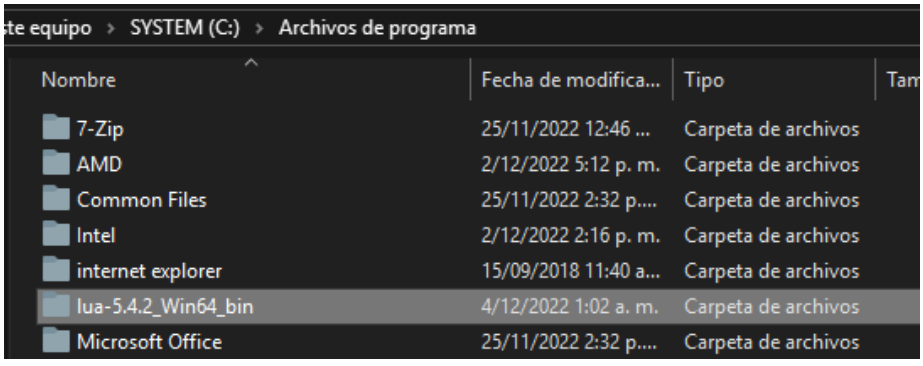


4. Después de descargar el archivo, lo extraemos con la herramienta de extracción de archivos .zip que viene preinstalada en Windows. Podemos hacer esto haciendo clic derecho en el archivo y seleccionando "Extraer todo".

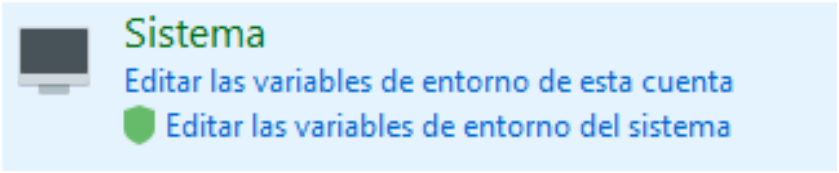


5. Veremos que se nos ha generado una carpeta con los ejecutables de Lua. Esta carpeta será la que necesitamos indicarle a nuestro programa para ejecutar nuestro código. Copiamos la carpeta a una ubicación segura en nuestro disco duro, por ejemplo, dentro de los Archivos de programa.

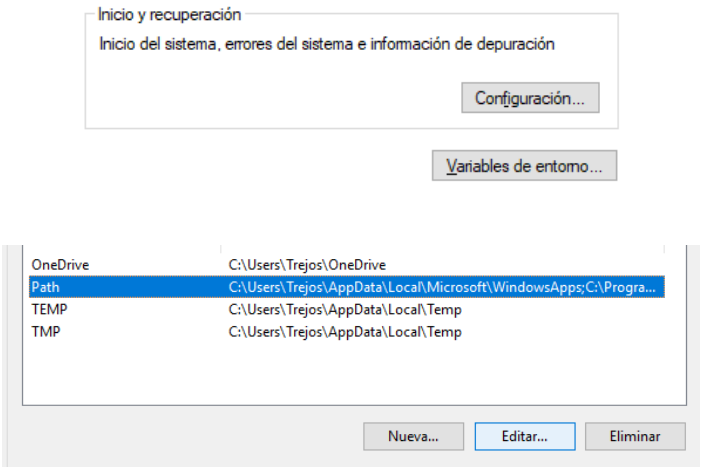




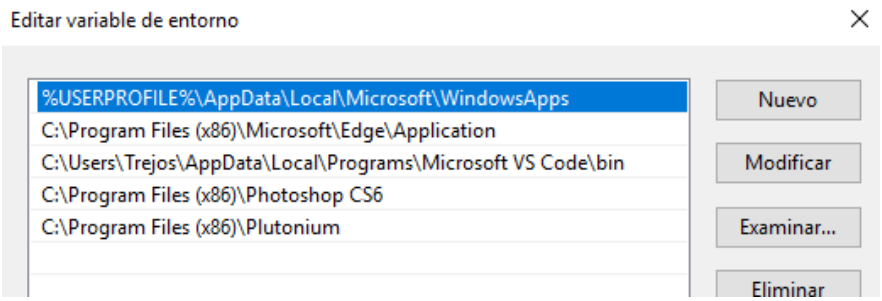
6. Ahora, abrimos el panel de control de Windows y buscamos "variables" en el cuadro de búsqueda. Seleccionamos la opción de "Editar las variables de entorno del sistema".



7. En la sección de "Opciones avanzadas", seleccionamos la opción de "Variables de entorno...". Aquí, encontramos la variable PATH y la editamos.



8. Creamos una nueva variable dentro de PATH y pegamos la ruta que copiamos antes.

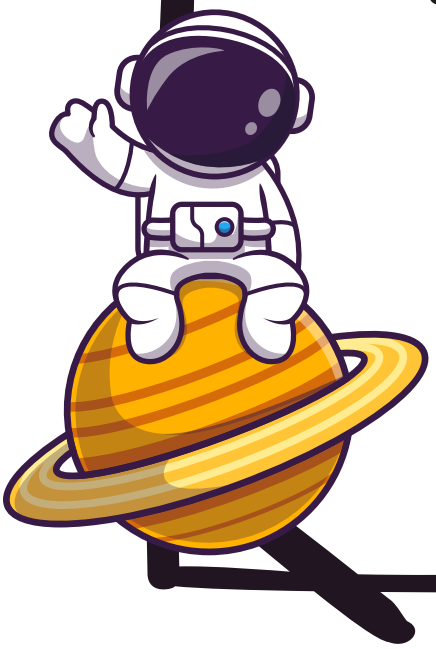


¡Listo! Si todo ha ido bien, deberíamos poder ejecutar Lua desde la terminal de comandos y ver la versión instalada. ¡Ya tienes Lua instalado en tu sistema Windows!

```
C:\Windows\system32\cmd.exe
Microsoft Windows [Versión 10.0.17763.1]
(c) 2018 Microsoft Corporation. Todos los derechos reservados.

C:\Users\Trejos>lua54 -v
Lua 5.4.2  Copyright (C) 1994-2020 Lua.org, PUC-Rio
```

En conclusión



En este capítulo aprendimos cómo determinar si nuestra computadora es de 32 o 64 bits, lo que nos permitió instalar la versión adecuada de Lua para nuestra arquitectura. Además, hemos seguido los pasos para descargar e instalar Lua en un sistema operativo Windows, configurando correctamente las variables de entorno para que podamos ejecutar nuestros programas en Lua desde la terminal de comandos. Con estos conocimientos, ahora estás preparado para comenzar a explorar el lenguaje de programación Lua y desarrollar tus propios proyectos

Instalación de Visual Studio Code y uso básico de su interfaz gráfica

¡Genial! Ahora que ya hemos instalado nuestro lenguaje de programación, es hora de que abordemos otra de las partes más importantes en el trabajo de un programador: su entorno de trabajo, también conocido como IDE (del inglés Integrated Development Environment) o espacio de desarrollo integrado. Este aspecto es fundamental, ya que, como bien sabemos, estamos aprendiendo un nuevo lenguaje de programación y necesitamos una herramienta que nos permita escribir, editar y visualizar nuestro código de forma clara y agradable.

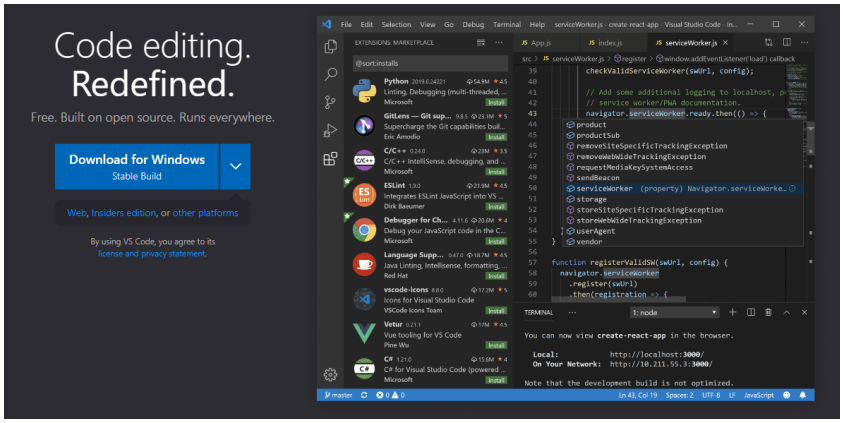
En nuestro caso, vamos a utilizar un editor de código llamado Visual Studio Code, que es ampliamente utilizado por la comunidad de programadores y desarrolladores en todo el mundo. Este editor de código ofrece una interfaz muy cómoda y agradable a la vista, así como una extensa lista de soporte de lenguajes, complementos y extensiones desarrollados principalmente por la comunidad.

En teoría, podríamos utilizar cualquier otro editor de código y obtener los mismos resultados, pero elegimos Visual Studio Code para este libro debido a su practicidad, popularidad, comodidad y, sobre todo, su soporte multiplataforma.

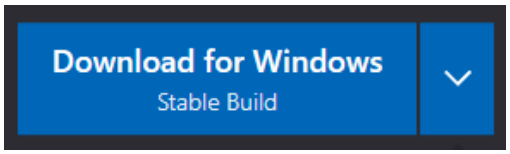
Una vez que conocemos el editor de código que utilizaremos para nuestros programas, es hora de descargarlo en nuestro sistema operativo. El sitio web oficial de Visual Studio Code nos proporciona enlaces de descarga para cada sistema operativo, ya sea Windows, Linux o MacOS. Por lo tanto, esta guía será explicada de manera general, independientemente del sistema operativo que tengamos instalado en nuestra computadora.

Para instalar este editor de código en nuestro sistema operativo, debemos seguir los siguientes pasos:

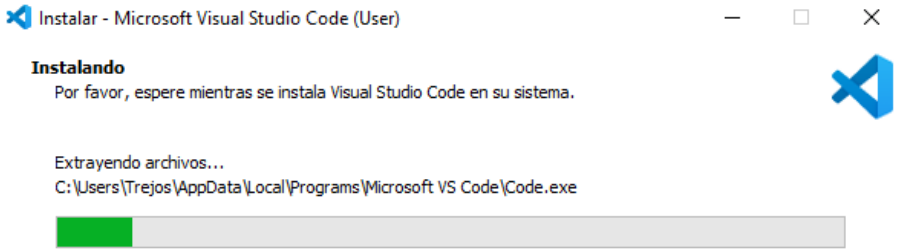
1. Dirigirnos al sitio web oficial de Visual Studio Code (https://code.visualstudio.com). También podemos buscar en nuestro motor de búsqueda los términos "Visual Studio Code" y entrar al sitio web oficial de Visual Studio Code, que suele aparecer entre los primeros resultados relevantes de nuestra búsqueda.



2. Descargar el instalador correspondiente a nuestro sistema operativo. Generalmente, el sitio web detecta automáticamente nuestro sistema operativo al ingresar a la página principal y nos muestra directamente el botón correspondiente para descargar el archivo para nuestro sistema. En este caso, como estamos realizando el ejemplo en una computadora con Windows, aparece el botón "Descargar para Windows" directamente en la página principal de Visual Studio Code.



3. Una vez que hemos descargado el archivo, solo tenemos que abrirlo y seguir el proceso de instalación según el sistema operativo en el que se está instalando. Al abrir el archivo ejecutable y seguir el proceso de instalación de Visual Studio Code, el asistente de actualización se encargará de instalar Visual Studio Code en nuestra computadora.



¡Y ya está! Ahora tenemos Visual Studio Code instalado en nuestro sistema operativo y podemos utilizarlo para escribir todo nuestro código de Lua de manera rápida, sencilla e intuitiva, con la ayuda de herramientas como un asistente para autocompletar el código a medida que escribimos.

¡Hola mundo! En Lua

¡Bienvenido al mundo de la programación! En este capítulo vamos a aprender a escribir nuestro primer código en Lua. Para empezar, haremos un "Hola mundo", que es una práctica muy común para los principiantes. No te preocupes si nunca has programado antes o si te sientes un poco intimidado por trabajar con código. Con esta práctica, verás que no es tan difícil como parece y que puedes obtener resultados incluso sin tener mucho conocimiento del lenguaje.

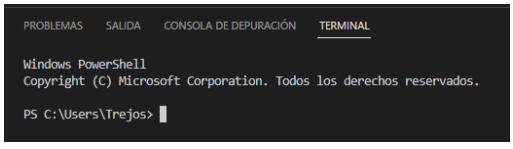
Es importante recordar que todos los programadores han empezado desde cero en algún momento, y este es el inicio que han tomado incluso los más grandes y exitosos desarrolladores. Para hacer nuestro "Hola mundo", vamos a utilizar la función print() de Lua. Esta función nos permite mostrar en pantalla el texto que escribamos dentro de sus paréntesis (es decir, sus parámetros).

Es fundamental escribir el texto entre comillas simples o dobles para que Lua sepa que se trata de un valor de tipo string. Pero no te preocupes si no sabes lo que eso significa todavía, ya lo explicaremos más adelante. Para empezar, crearemos un nuevo archivo en Visual Studio Code y lo guardaremos como "HolaMundo.lua" para indicar que estamos trabajando con Lua. Luego, escribiremos la siguiente línea de código:

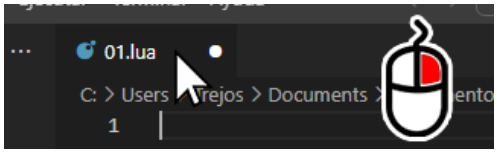
```
1    print(“Hola mundo”)
```

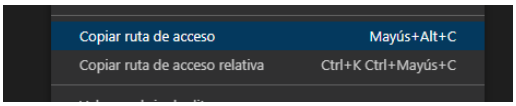
¡Genial! Acabas de escribir tu primer programa en Lua. Ahora, vamos a ejecutarlo siguiendo estos pasos:

1. Haz clic en la barra de herramientas superior de Visual Studio Code llamada "Terminal" y selecciona "Nueva terminal". Esto abrirá una ventana de terminal donde podremos ejecutar líneas de comando.

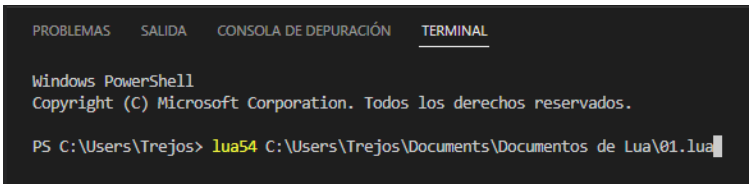


2. Copia la ruta de acceso de nuestro documento presionando las teclas [Shift + Alt + C] o haciendo clic derecho en la pestaña del documento y seleccionando "Copiar ruta de acceso".





3. Escribe en la terminal la palabra clave de Lua que tengas instalada en tu sistema (en este ejemplo, "lua54" para la versión 5.4 de Lua) seguida de un espacio y pega la ubicación de tu archivo. Finalmente, presiona <Enter> para ejecutar el programa.



Verás que en la ventana de la terminal aparecerá el texto “Hola mundo". ¡Lo lograste!

Recuerda que hacer un "Hola mundo" es solo el comienzo. A partir de ahora, podrás explorar todas las posibilidades que te ofrece la programación. Por ejemplo, puedes mostrar en pantalla cualquier texto que quieras, como hicimos con estos ejemplos:

```
1  print(“Esta es la primera línea”)
2  print(“Esta es la segunda línea de texto”)
3  print(“Tercera”)
```

¡Muy bien! Ahora que ya sabes cómo hacer un "Hola mundo" y mostrar texto en pantalla, es hora de seguir practicando y experimentando por tu cuenta. Recuerda que la programación es una habilidad que requiere práctica constante para mejorar, así que no tengas miedo de probar cosas nuevas y desarrollar tus propios programas basados en lo que aprendiste. ¡Sigue adelante!

Una alternativa a usar print() en “Hola mundo”

En ocasiones, puede resultar tedioso utilizar la función print() para mostrar información en pantalla. Pero no te preocupes, ¡hay una solución! Podemos utilizar la librería de entrada y salida de datos (Librería I/O) para conseguir el mismo resultado.

Al principio, es normal que estos conceptos te resulten un poco desconocidos. Sin embargo, no te preocupes, no son tan complicados como parecen. Además, es importante tener en cuenta que existen diferentes funciones y librerías que pueden variar en su rendimiento y compatibilidad con distintas versiones de un mismo lenguaje de programación.

Veamos un ejemplo para entenderlo mejor. En lugar de usar la función print(), podemos utilizar la función io.write() para mostrar "Hola mundo" en pantalla.

```
1  io.write(“Hola mundo”)
```

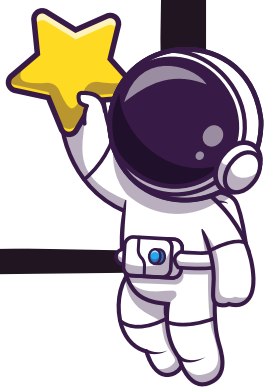
Luego, ejecutamos el código en la terminal y obtenemos la misma salida que con la función print().

Salida	Hola mundo

La idea principal es que existen diversas formas de lograr el mismo resultado dentro de la programación. Con el tiempo, profundizaremos en estos conceptos para que puedas desarrollar proyectos más complejos en Lua. No te preocupes si todavía no lo tienes claro, lo importante es seguir practicando.

En conclusión

En este capítulo dimos nuestros primeros pasos en la programación en Lua, aprendiendo a escribir y ejecutar un programa "Hola mundo". Utilizamos la función print() y explorado una alternativa con la librería I/O y la función io.write(). A medida que avancemos en el libro, iremos adquiriendo más conocimientos y habilidades para crear programas más complejos y eficientes. Recuerda que la práctica constante es clave para mejorar tus habilidades en programación, así que no dudes en experimentar y probar diferentes enfoques.



Capítulo 3

Algunas ideas y convenciones generales dentro de Lua

Antes de adentrarnos en ejemplos complejos en Lua, es importante tener en cuenta algunas de las características que hacen especial a este lenguaje de programación y que pueden representar buenas prácticas al programar. A menudo, estas características no se utilizan tanto como deberían, pero es útil entenderlas y estudiarlas para familiarizarnos con nuestro lenguaje y comprender lo que lo distingue de otros.

Es hora de conocer algunas de las principales características de Lua. La mayoría de estos conceptos son fáciles y amigables de entender, curiosidades o "trucos" que te ayudarán a programar mejor. Sin embargo, es importante destacar que no todo lo que es posible en programación es necesariamente una buena práctica. A medida que avanzamos en nuestro conocimiento, aprenderemos más sobre las mejores prácticas que nos ayudarán a escribir programas limpios y estables.

El primer concepto que vamos a explorar son las fracciones de código, que se refieren a un conjunto de instrucciones dentro de nuestro sistema que indican varias líneas de código en una sola. Esto se puede utilizar en diferentes situaciones y aplicaciones. Por ejemplo, podemos tener un bloque de código donde cada instrucción se escribe en una línea de código separada:

```
1    variable = 17
z    for i = 1, 10, 1 do
3        print(17+i)
4    end
```

Pero también podemos escribir el mismo programa en una sola línea de código, utilizando las fracciones de código:

```
1    variable = 17; for i = 1, 10, 1 do;  (17+i); end
```

Si bien este ejemplo ocupa menos espacio, hace que el código sea más difícil de leer y entender con facilidad. A pesar de que las fracciones de código son ejecutables dentro de una sola línea, no significa que sea una buena práctica escribir todo el código en una sola línea. En lugar de utilizar los ";", podemos utilizar espacios en blanco para lograr el mismo resultado y hacer el código más fácil de leer:

```
1    variable = 17 for i = 1, 10, 1 do print(17+i) end
```

Ambos programas pueden ejecutarse correctamente, pero es importante recordar que es mejor seguir las buenas prácticas para mantener un código claro y fácil de entender. Si bien es cierto que las fracciones de código son útiles en ciertas situaciones, no es recomendable abusar de ellas.

Comentarios en Lua

Antes de comenzar a programar en Lua, es importante entender uno de los temas básicos de programación: los comentarios. Los comentarios son líneas de texto que colocamos dentro de nuestro código, pero que el intérprete del lenguaje no ejecutará como instrucciones para el programa.

Cada lenguaje de programación tiene su propia forma de hacer comentarios. Algunos permiten comentarios de una sola línea, mientras que otros admiten comentarios de varias líneas. En algunos casos, los comentarios de varias líneas requieren una estructura específica que debe repetirse en cada línea.

El propósito de los comentarios en la programación es brindar información textual al programador para guiarlo de manera sencilla dentro del código. Además, también pueden incluir anotaciones útiles para el programador cuando vuelve a trabajar en el programa.

En Lua, los comentarios se crean usando el símbolo de doble guion "--". Esto indica que lo que sigue es un comentario de una sola línea. Dentro de esta línea de código, podemos escribir cualquier cosa que deseemos sin preocuparnos de que el programa lo ejecute. Por ejemplo:

```
1  -- Esto es un comentario y no será ejecutado
2  print(“Hola mundo”)
3  -- print(“Adiós”)
```

La salida que se muestra al ejecutar el código es: "Hola mundo". Como puedes ver, hemos agregado dos líneas de texto como comentario y el programa no las ejecuta. La última línea "--print("Adiós")" es un comentario y no se muestra en pantalla.

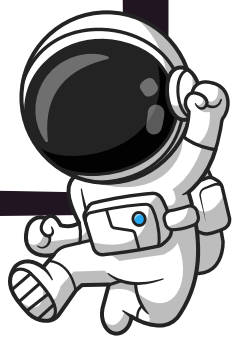
Además de los comentarios de una sola línea, también podemos usar comentarios de varias líneas. Si necesitamos escribir un comentario que abarque varias líneas, podemos usar este formato:

```
1  --[[
2       Esto es un comentario muy largo
3       que puede ocupar múltiples líneas de código.
4       print(“Hola ”)
5  --]]
6  print(“mundo”)
```

La salida que se muestra al ejecutar el código es: "mundo". En este ejemplo, usamos "--[[--]]" para crear el comentario de varias líneas. Todo lo que esté dentro de los corchetes no se ejecutará como una orden directa del programa.

En conclusión

Exploramos algunas de las características y convenciones generales en Lua, como las fracciones de código y los comentarios, que pueden ser útiles para escribir programas eficientes y legibles. Aprendimos sobre la importancia de las buenas prácticas al programar y cómo mantener un código claro y fácil de entender. Recordemos que no todas las posibilidades en programación son buenas prácticas, por lo que es esencial continuar aprendiendo y aplicando las mejores prácticas en nuestro trabajo. Al familiarizarnos con estas características y convenciones, podremos aprovechar al máximo el lenguaje Lua y crear programas más efectivos y estables.



Programación interactiva en Lua

Lua nos ofrece la posibilidad de interactuar directamente con el lenguaje sin tener que escribir código en un archivo. Esto se llama programación interactiva y es una función que también está disponible en otros lenguajes como Python.

La idea de la programación interactiva es que podemos probar el código y hacer pruebas directamente con el intérprete sin tener que crear un archivo para guardar las instrucciones. Esto tiene sus ventajas y desventajas. Por un lado, es más fácil interactuar con el lenguaje sin tener que preocuparse por la ubicación del archivo. Por otro lado, el código que escribamos en la programación interactiva no se guardará y tendremos que escribirlo de nuevo si lo necesitamos más tarde. Además, un entorno de desarrollo ofrece herramientas útiles como el resaltado de errores y el autocompletado del código.

Para interactuar con Lua de forma interactiva, abrimos una terminal y escribimos "lua54" para iniciar la versión 5.4 de Lua. Esto abrirá el intérprete de Lua y podremos escribir comandos directamente.

Podemos escribir tanto una línea de texto como varias líneas de código, incluso con indexaciones. Aprenderemos más sobre las indexaciones en secciones posteriores. Por ejemplo, si escribimos el siguiente código de varias líneas, podemos ejecutarlo en la ventana interactiva de Lua:

```
1  Numero1 = 20
2  Numero2 = 80
3  print(Numero1 + Numero2)
```

Después de escribir este código en la ventana interactiva, obtendremos la siguiente salida:

```
C:\Users\Trejos>lua54
Lua 5.4.2  Copyright (C) 1994-2020 Lua.org, PUC-Rio
> Numero1 = 20
> Numero2 = 80
> print(Numero1 + Numero2)
100
```

Como puedes ver, cada línea de código que escribamos en la ventana interactiva se almacenará por el resto de la sesión.

Capítulo 4

Variables y tipos de datos en Lua

Las variables son conceptos muy importantes en la programación. Nos permiten guardar información en nuestro código. Piensa en una variable como una caja: puedes guardar muchas cosas diferentes dentro de ella, incluso puede estar vacía.

Ilustración 7- Podemos imaginar que una variable es un tipo de caja; en una caja podemos guardar muchas cosas.

Una variable es un espacio en la memoria de la computadora donde podemos guardar información, como números, texto o incluso nada. La ventaja de usar variables es que podemos almacenar información que luego podemos usar en el futuro en nuestro programa. Sin embargo, no es necesario guardar toda la información en una variable, ya que tener muchas variables puede ocupar mucho espacio en la memoria de la computadora. Además, las variables en Lua tienen una estructura para definir su nombre y el tipo de información que almacenarán.

Hay muchos tipos de datos que podemos guardar en una variable, incluso una variable vacía.

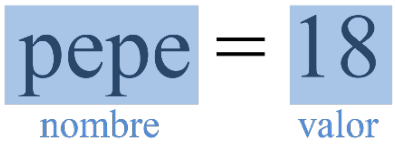
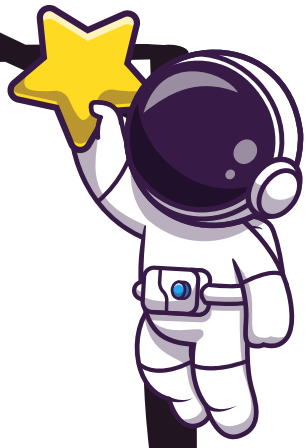


Ilustración 8 – Estructura de asignación de una variable

También es posible eliminar variables cuando ya no son necesarias. Las variables consumen espacio en la memoria de la computadora, por lo que, si ya no las necesitas, es mejor eliminarlas para liberar espacio. Aprenderás más sobre cómo trabajar con variables a lo largo de este libro. ¡No te pierdas nada! Hay muchos conceptos interesantes que debes conocer para convertirte en un programador experto en Lua.



En conclusión

la programación interactiva en Lua nos brinda la facilidad de probar y experimentar con el código directamente en el intérprete sin la necesidad de crear un archivo para guardar las instrucciones. Aunque tiene sus ventajas y desventajas, es una herramienta útil para aprender y comprender el lenguaje. Además, las variables y los tipos de datos en Lua son fundamentales para almacenar información en nuestros programas, permitiéndonos manejar datos de diversas formas.

Tipos de datos numéricos

El tipo de dato "number" en Lua se refiere a números que pueden ser utilizados y manipulados con operaciones matemáticas. Estos números son parte del conjunto de números racionales.

Es importante destacar que el tipo de dato "number" es uno de los más importantes en la programación y se utiliza en muchos lenguajes. Por ejemplo, en Lua, podemos utilizar números naturales y enteros como -17, 6, 0, -999, 1650, 15, entre otros. También podemos usar valores racionales como 16.7, -18.005, 1.11, 9.67, -15.87, y exponentes decimales como 5e+12, -7e-78, 17.5e+5, -6.007e-28.

En resumen, el tipo de dato "number" en Lua es un tipo de dato fundamental para la solución de problemas en la programación.

Estructura de los valores numéricos en Lua

Antes de continuar, es importante conocer la estructura de los valores numéricos en Lua. Como hemos visto, podemos tener tres tipos diferentes de valores numéricos en este lenguaje de programación, pero esto no significa que tengan diferentes estructuras. De hecho, todos estos tres tipos de números comparten la misma estructura, aunque puedan parecer un poco diferentes.

Ilustración 9 - Estructura y elementos de valores numéricos.

Veamos la ilustración 13, que muestra la estructura y elementos de los valores numéricos. La parte esencial de todo valor numérico en Lua es la base numérica. Sin ella, no podemos considerar al valor como un número.

Las otras tres partes de un valor numérico en Lua son la orientación, ya sea positiva o negativa; el complemento decimal y la extensión exponencial decimal. La orientación indica si el valor es positivo o negativo, mientras que el complemento decimal se refiere a los valores decimales que conforman el número. La extensión exponencial decimal amplía el alcance de los valores decimales.

Extensión exponencial decimal en valores numéricos

Es importante saber cómo funciona exactamente la notación científica en Lua. Puedes reconocer un número con una notación científica cuando tiene una estructura como la que se muestra en la siguiente ilustración.

Ilustración 10- Indicador de una extensión exponencial de un valor numérico.

La notación científica también se conoce como extensión exponencial decimal. Puedes ver algunos ejemplos de cómo se ven estos números en la notación científica en la tabla siguiente.

Número en Lua	Notación científica
5e+12	$5x10^{12}$
-17e-5	$-17x10^{-5}$
5687e+16	$5687x10^{16}$

En Lua, los números son un tipo de dato que puede ser utilizado para realizar operaciones matemáticas complejas gracias a la biblioteca matemática integrada en el lenguaje. Por ejemplo, puedes calcular fórmulas físicas, ángulos en radianes, operaciones con números muy grandes, entre otras cosas.

Además, es importante tener en cuenta que no solo los números pueden ser operados en Lua, sino que también otros tipos de datos pueden ser manipulados dependiendo del tipo de dato con el que estemos trabajando.

Por ejemplo, puedes realizar operaciones aritméticas con las variables numéricas. Aquí tienes un ejemplo:

```
1  Numero1 = 20
2  print(Numero1 * 10)
```

Al ejecutar este código, obtendrás la salida "200".

Notación de los diferentes operadores numéricos en Lua

En Lua, podemos realizar operaciones aritméticas entre valores numéricos. Aquí veremos algunas de las operaciones más comunes que podemos hacer usando valores numéricos. Esta es solo una lista de las operaciones más comunes, pero la lista es extensa y las posibilidades son amplias.

Operación matemática	Operador en Lua	Ejemplos
Suma	+	17 + 2 || 6 + (-5)
Resta	-	10 – 6 || -5 – (-6)
Producto	*	2 * 5 || 5 * -3
División	/	10 / 5 || 5 / 6
Negación unitaria	-	- 5 || - (-7)

Además de los operadores aritméticos, Lua también nos permite realizar otras operaciones numéricas, como se muestra en la siguiente tabla:

Operación matemática	Operador en Lua	Ejemplos
Valor absoluto	math.abs(x)	math.abs(-67) || math.abs(-17e+10)
Logaritmo natural	math.log(x)	math.log(57) || math.log(1)
Raíz cuadrada	math.sqrt(x)	math.sqrt(16) || math.sqrt(100)
Elevación cuadrada	math.pow(x, 2)	math.pow(5, 2) || math.pow(-17, 2)

En Lua, tenemos una gran cantidad de herramientas matemáticas a nuestra disposición. Con ellas, podemos realizar todo tipo de operaciones numéricas. Más adelante en el libro, profundizaremos en estas librerías para aprender todo lo que podemos hacer con ellas.

Adicionalmente, no solo podemos trabajar con números en sí, sino que también podemos guardarlos en variables y usarlos en operaciones matemáticas. Aquí podemos ver algunos ejemplos de código en Lua que ilustra esto:

Operaciones aritméticas

Es hora de profundizar en el primer ejemplo de operaciones aritméticas en Lua. Ya vimos un ejemplo sencillo antes, pero veamos más para entender completamente las operaciones que podemos realizar con datos numéricos en Lua.

```
1   print(10+5.5)
```

Al ejecutar este código, obtenemos la siguiente salida:

```
Salida   15.5
```

Es importante entender lo que sucede en el código. Es probable que esta explicación sea muy básica, pero lo haremos para que te familiarices con los cuadros explicativos que utilizaremos a lo largo de este libro.

Explicación del código	
1	Escribimos la función “print()”, que se encarga de mostrar todos los elementos dentro de ella en la terminal. Luego, le decimos a la función que muestre en pantalla el resultado de la operación aritmética de la suma de los valores numéricos 10 y 5.5.

2	Finalmente, podemos ver cómo nuestro programa muestra en la pantalla el resultado de la suma de estos dos valores numéricos. El resultado de la operación 10 + 5.5 es 15.5, y ese es el resultado que se muestra en la pantalla.

Veamos otro ejemplo donde realizamos más operaciones aritméticas usando los operadores que vimos anteriormente.

```
1  print(8 * 8)
2  print(16 - 5)
3  print(16 / 4)
4  print(- ( -6 ))
```

Al ejecutar este código en el intérprete de Lua, obtenemos la siguiente salida:

```
Salida  64
        11
        4
        6
```

Veamos la explicación del código:

Explicación del código	
1	Este ejemplo es similar al anterior. El código consta de 4 líneas, y cada línea hace uso de la función print(), que solo muestra en pantalla lo que está dentro de la función.
2	Mostramos en pantalla 4 casos diferentes, cada uno con diferentes operaciones aritméticas dentro; incluimos operaciones de multiplicación, resta, división y negación unitaria. Finalmente, podemos ver cómo la salida genera el resultado esperado de cada operación numérica.

Así, podemos concluir esta sección sobre operaciones y el uso de operadores aritméticos en Lua.

Las operaciones aritméticas en Lua siguen un orden de ejecución en caso de ambigüedad en el momento de ejecutar el código. La siguiente sección nos enseñará en profundidad el orden en que Lua realiza las operaciones aritméticas y cómo este orden está relacionado con las reglas comunes de jerarquía de ejecución de cálculos matemáticos.

Orden de ejecución de operaciones aritméticas

Antes de continuar en el mundo de la programación, es importante conocer la teoría aritmética para entender cómo funciona Lua con sus datos y operaciones. Hablaremos de operadores simples, pero también es importante saber en qué orden ejecuta Lua operaciones similares. En el ejemplo anterior, vimos una simple suma, pero ¿qué sucedería si el programa fuera más complicado?

```
1  print(8+12*7+5)
```

La jerarquía de operaciones aritméticas es un estándar establecido por la matemática que indica el orden en que deben ser ejecutadas antes de que el resto lo hagan. Incluso se extiende a operaciones como la potenciación, pero por ahora nos enfocaremos en las 4 operaciones básicas: suma, resta, multiplicación y división. Según la teoría, el orden de ejecución de operaciones aritméticas es el siguiente: primero, resolvemos las multiplicaciones o divisiones, y luego las sumas y restas, siempre y cuando no haya paréntesis que indiquen una excepción a este estándar.

Entonces, siguiendo este principio de jerarquía de operadores aritméticos, podemos entender fácilmente el resultado de nuestro problema. En este caso, tenemos dos operadores aritméticos: multiplicación y suma. Si seguimos el estándar de jerarquía de operadores aritméticos, primero resolvemos la multiplicación y luego las sumas correspondientes al resultado de la multiplicación.

```
1    print(8+12*7+5)
```

Al ejecutar este código, obtenemos la siguiente salida:

```
Salida    97
```

Como podemos ver, Lua también sigue los estándares de jerarquía de operadores aritméticos de la matemática para dar resultados. La siguiente ilustración muestra de una forma más fácil de entender cómo funciona este orden jerárquico de operaciones aritméticas en Lua y en la matemática.

Ilustración 11 - Orden de ejecución de operaciones aritméticas en base a su posición jerárquica

Ahora tenemos un conocimiento más amplio sobre las operaciones aritméticas que resuelve Lua y el orden que sigue. Si no se especifica una prioridad con paréntesis, Lua tomará en cuenta este orden jerárquico para ejecutar las operaciones aritméticas. En la siguiente sección, exploraremos más en detalle lo que significa indicar una prioridad al realizar operaciones.

Paréntesis y su papel en la jerarquía aritmética

Antes estudiamos la jerarquía de operadores aritméticos para operaciones simples en matemáticas. Ahora, vamos a ver cómo los paréntesis nos permiten hacer excepciones a esta jerarquía. Los paréntesis son dos símbolos: ( ) y sirven para definir un orden específico en el cálculo de operaciones aritméticas mixtas. Los paréntesis son la máxima prioridad en la jerarquía aritmética. Veamos algunos ejemplos para entender mejor cómo funcionan.

Paréntesis dentro de la jerarquía aritmética

```
1    print((2+3)*5+7)
2    print(((((25*6)+16)*7)+6)*6)
```

Al ejecutar el código en un intérprete de Lua, obtenemos la siguiente salida:

```
Salida    32
          7008
```

La función "print" muestra en la pantalla el resultado de las operaciones dentro de ella. Las operaciones prioritarias están marcadas con paréntesis y se realizan primero. El resultado final es el resultado de la operación aritmética con prioridades de paréntesis y la jerarquía de las operaciones aritméticas.

En resumen, los paréntesis nos permiten controlar el orden en el cálculo de operaciones aritméticas mixtas y son la máxima prioridad en la jerarquía aritmética. Ahora ya entendemos mejor su papel en las matemáticas y en la informática.

Variables numéricas

Antes mencionamos que las variables son muy útiles en la programación y es un recurso esencial para guardar información en un programa. En este capítulo, aprenderás cómo crear variables en Lua. Hay dos tipos principales de variables: las variables locales y las variables globales, pero no te preocupes, eso lo estudiaremos más adelante. Por ahora, te enseñaremos cómo crear una variable y asignarle un valor, es muy fácil.

Una variable está compuesta por tres partes:

3. **Nombre:** es el identificador único que usarás posteriormente en el programa para acceder a los datos almacenados en ella.

4. **Valor:** es la información o datos que guardarás en la variable.

5. **Tipo de datos:** es el tipo de información que estás guardando, por ejemplo, números, texto, etc.

El nombre de la variable debe cumplir ciertas reglas, como no ser una palabra reservada. Las palabras reservadas son términos que ya están asignados por el lenguaje de programación, como "function", "if", "while", "do", "end", etc. Es importante evitar usar estas palabras para nombrar tus variables.

Lua es un lenguaje sensible a las mayúsculas, por lo que las palabras "function" y "FunctioN" son diferentes. Además, Lua tiene muy pocas palabras reservadas, solo 21, lo que lo hace más fácil de aprender en comparación con otros lenguajes de programación como Python, que tiene 33 palabras reservadas.

Puedes usar cualquier nombre que no sea una palabra reservada de Lua para nombrar tus variables y evitar errores al ejecutar tu programa. Además, puedes usar variantes de las palabras reservadas, como "FunCtiOn", "wHILE", "End", etc.

Veamos algunos ejemplos de variables numéricas y cómo asignarles valores:

```
1  num1 = 7
2  num2 = 13
3  num3 = num1 + num2
4  print( num3 * 10 )
```

Al ejecutar este código, se obtiene la siguiente salida:

```
Salida  200
```

Explicación del código	
1	En este ejemplo, creamos tres variables numéricas. La primera y segunda variables son asignadas directamente con valores numéricos, mientras que la tercera variable es asignada indirectamente con el resultado de la suma de las otras dos variables. Luego, usamos la función print() para mostrar el resultado de multiplicar el valor de la tercera variable por 10.

Veamos otro ejemplo similar al anterior:

```
1  NumbeR = 20
2  DecimaL = -15.7432
3  FunCtiOn = NumbeR * DecimaL
4  print(“El resultado es: ” .. FunCtiOn)
```

Al ejecutar este bloque de código, obtenemos la siguiente salida:

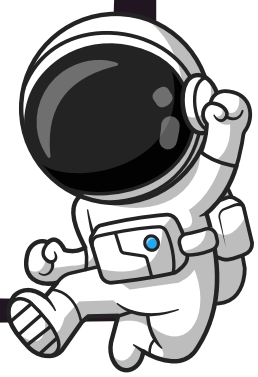
```
Salida  El resultado es: -314.864
```

En este ejemplo, creamos dos variables numéricas. La primera es un número entero y la segunda es un número decimal negativo. Después, creamos una tercera variable que es el resultado de multiplicar las dos variables anteriores.

Finalmente, usamos la función print para mostrar en pantalla la frase "El resultado es: " seguida del valor de la tercera variable. Con esto, veremos en la terminal el resultado de la multiplicación.

En resumen

El tipo de dato "number" en Lua es un tipo de dato esencial en la programación, que permite a los programadores realizar cálculos y manipulaciones matemáticas. Las variables numéricas son fundamentales para almacenar y operar con datos numéricos, y las operaciones aritméticas en Lua siguen un orden jerárquico similar al de las matemáticas. Los paréntesis pueden utilizarse para modificar el orden de las operaciones aritméticas y especificar prioridades, y Lua es sensible a las mayúsculas, lo que significa que los nombres de variables y palabras clave se tratan como diferentes si difieren en el uso de mayúsculas y minúsculas.



Almacenar datos escritos por el usuario en variables

En lugar de almacenar datos en variables, ¿qué tal si les damos vida a tus programas permitiéndoles que el usuario ingrese los datos que necesitan a través del teclado? Esto es uno de los conceptos más importantes y útiles que usaremos en nuestros programas.

Aunque hay una gran variedad de tipos de datos en programación, la lista de datos que podemos hacer que el usuario ingrese es relativamente limitada. Por ejemplo, los datos de entrada solo pueden ser valores numéricos o de texto.

La capacidad de que el usuario ingrese datos les da vida a tus programas, haciéndolos más personalizados y específicos para cada usuario. De esta forma, el programa puede ser reutilizado y ser útil en diferentes situaciones, sin tener que ser reprogramado desde cero.

Veamos un ejemplo de cómo podemos hacer esto:

```
1  io.write("Escribe tu edad: ")
2  edad=io.read()
3  io.write("Tu edad es ", edad, " y pronto tendrás ", edad+1)
```

Si el usuario ingresa 18, la salida será:

```
Salida  Escribe tu edad: 18
        Tu edad es 18 y pronto tendrás 19
```

En este ejemplo, usamos la librería de entrada y salida de datos de Lua para pedirle al usuario que escriba su edad. Luego, almacenamos esta información en una variable y la usamos para mostrar un mensaje en la pantalla.

También debes saber que la forma de hacer que el programa tome datos escritos desde el teclado es muy amplia, pero con la práctica y el tiempo, podrás identificar las formas más comunes y eficientes. Mientras aprendes a programar en Lua, usaremos la función io.read(). Además, también puedes usar variables de texto y aprender más sobre ellas en secciones posteriores de este libro.

Proyecto de interacción con el usuario

En este proyecto, vamos a crear un programa complejo en Lua para mejorar nuestras habilidades de programación. Utilizaremos funciones y librerías para interactuar con variables numéricas y recibir datos del usuario a través del teclado. También manejaremos valores de cadena de texto, aunque aún no hemos estudiado profundamente este tipo de datos.

El objetivo de este proyecto es practicar de forma divertida los conceptos que hemos aprendido hasta ahora en Lua. La mayoría de las funciones y programas que usaremos aquí son cosas que ya vimos en ejemplos anteriores. Así que no te preocupes si aún no estás seguro de cómo funcionan los tipos de datos de cadena de texto, porque aquí seguiremos trabajando con lo que ya conocemos.

Ilustración 12 - "¿Cuál es el IVA de este producto?"

Nuestra tarea como programadores es crear un programa fácil de usar para el operador, donde solo tenga que ingresar el precio del artículo que está comprando en ese momento. Después, el programa

calculará y mostrará en la pantalla el monto a pagar en impuestos (IVA) y el costo total, incluyendo el precio del producto y el IVA. Así, el operador tendrá una idea clara del monto total que debe pagar.

Análisis y solución al problema:

Necesitamos hacer que nuestro programa sea amigable para el usuario, por lo que debemos crear una interfaz de usuario sencilla para interactuar con el programa. Podemos hacer esto con una bienvenida y algunas instrucciones en la pantalla.

El siguiente código es un ejemplo:

```
1  print("¡Bienvenido a nuestro programa!")
2  print("Este programa te ayudará a calcular fácilmente el IVA de un producto.")
3  print("Ingresa el valor del producto (sin incluir el IVA), y déjanos el resto a nosotros.")
4  io.write("Precio del producto: ")
```

Si ejecutamos este código, obtendremos la siguiente salida:

```
Salida  ¡Bienvenido a nuestro programa!
        Este programa te ayudará a calcular fácilmente el IVA de un producto.
        Ingresa el valor del producto (sin incluir el IVA), y déjanos el resto a nosotros.
        Precio del producto:
```

Ahora tenemos una interfaz de usuario sencilla que hace que el usuario se sienta más cómodo al usar el programa.

Después, el usuario ingresará el valor del producto para calcular el IVA. Primero, debemos considerar que el IVA actual es del 19%, así que lo almacenaremos en una variable para su uso posterior.

Podemos hacer esto y almacenar los datos ingresados por teclado de la siguiente manera:

```
5  Porcentaje = 19
6  Valor = io.read()
```

Supongamos que el usuario ingresa el precio del producto. Este valor numérico se almacenará en la variable "Valor". El objetivo principal de este programa es calcular el valor del IVA, así que también almacenaremos este valor en otras variables en el programa.

```
7  IVA= (0.1 * Porcentaje) * Valor
8  Final = Valor + IVA
```

Ahora tenemos almacenado tanto el valor del IVA del producto como el total a pagar, incluyendo el precio original del producto y el valor del IVA.

¡Felicidades! Terminamos gran parte del trabajo. Ahora solo necesitamos mostrar esta información al usuario para que pueda entender fácilmente el resultado.

Podemos hacer esto con pocas líneas de código utilizando la función print():

```
9   print("El valor del IVA es de $" .. IVA)
10  print("Por lo tanto, el total a pagar es de $" .. Final)
```

¡Listo! Ya tenemos el código de nuestro programa completo. Todo debería funcionar bien, pero siempre es una buena idea hacer algunas pruebas en un caso de simulación que podría ser real en la vida cotidiana.

Aquí está el código completo:

```
1   print("¡Bienvenido a nuestro programa!")
2   print("Este programa te ayudará a calcular fácilmente el IVA de un producto.")
3   print("Ingresa el valor del producto (sin incluir el IVA), y déjanos el resto a nosotros.")
4   io.write("Precio del producto: ")
5   Porcentaje = 19
6   Valor = io.read()
7   IVA = (0.1 * Porcentaje) * Valor
8   Final = Valor + IVA
9   print("El valor del IVA es de $" .. IVA)
10  print("Por lo tanto, el total a pagar es de $" .. Final)
```

Supongamos que se ha entregado nuestro producto como programador a nuestro cliente, este cliente se encarga de personalmente darle una breve introducción a aquellos operarios que usarán el programa. Bajo este caso hipotético, un cliente de un supermercado se acerca a la caja registradora preguntando cuánto debe pagar en total por un producto, ya que en su etiqueta marca el texto “IVA no incluido”. Este mismo cliente quiere saber cuál es el precio que le cobrarán en la caja al momento de considerar el precio del IVA en su compra.

Ilustración 13 - "El cliente se pregunta por qué no está escrito el IVA en el precio y le interesaría saber cuánto debe pagar".

Por lo tanto, el operador de nuestro programa tendrá que usar el programa que hemos desarrollado para calcular el valor del impuesto que el cliente debe pagar.

Supongamos que este producto tiene un precio total de $10,000. Este será el precio que el operador de nuestro programa debe introducir. Al introducir adecuadamente este valor, obtenemos la siguiente salida:

Salida	Este programa te ayudará a calcular fácilmente el IVA de un producto. Ingresa el valor del producto (sin incluir el IVA), y déjanos el resto a nosotros. Precio del producto: 10000 El valor del IVA es de $1600 Por lo tanto, el total a pagar es de $11,600

¡Y listo! Ahora tenemos la información suficiente para responder la pregunta del cliente, y también conocemos el resultado de calcular el valor de impuesto IVA que se debe pagar por su producto en específico.

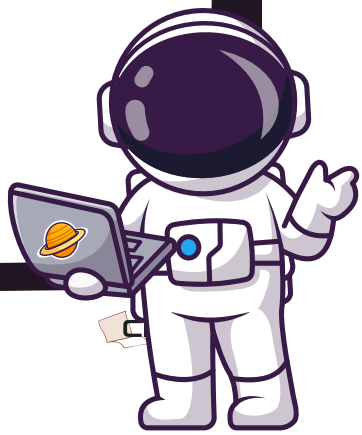
Ahora es momento de analizar un poco más a fondo todo lo que ocurrió en el código:

Explicación del código	
1	Al igual que explicamos en el proceso de creación del código, hacemos uso de la función "print()" para mostrar en pantalla lo que está dentro de la función. Durante varias líneas de código, usamos esta función para brindar una agradable introducción al operador de nuestro programa.
2	Después de mostrarlo en pantalla, llega el momento de usar la librería de entrada y salida de datos en Lua. Usamos dos funciones de esta librería: primero, usamos "io.write()" para escribir en pantalla lo que está dentro de la función, evitando así un salto de línea. La segunda función que usamos es "io.read()", que permite pausar la ejecución del programa y abrir un espacio para que el usuario pueda escribir algo en la terminal. Cuando se detecta que se presionó la tecla Enter, se usa la información escrita por el usuario para almacenarla en una variable. En este caso, supongamos que el operador escribió el valor numérico 10000 en su teclado, y este valor se guardó en la variable "Valor".
3	Luego creamos dos variables finales que usaremos. Ambas son variables numéricas. En la primera, calculamos el valor del IVA que se debe pagar, y en la segunda, guardamos la suma total del valor del producto más el valor del impuesto IVA. Finalmente, mostramos en pantalla los resultados de ambos valores para tener el precio adecuado.

¡Y eso es todo! Ahora tenemos una explicación más detallada de todo lo que ocurre en el código, lo que nos ayuda a entender mejor cada parte de este.

En conclusión

En este capítulo exploramos cómo interactuar con el usuario y almacenar datos proporcionados por ellos en variables. Aprendimos a utilizar la librería de entrada y salida de datos de Lua y a aplicar funciones como io.write() e io.read() para lograr esta interacción. Además, creamos un proyecto práctico para calcular el IVA de un producto, utilizando las habilidades adquiridas en la lectura y almacenamiento de datos ingresados por el usuario. Este proyecto nos ha permitido consolidar nuestro conocimiento sobre variables numéricas, operaciones matemáticas y cómo presentar la información al usuario de una manera clara y amigable.



Cambiar datos dentro de una variable numérica

Ahora, vamos a ver un concepto simple en programación.

Cuando creamos una variable en nuestro programa, estamos simplemente asignando un espacio temporal para almacenar un dato. Pero ese dato no tiene por qué quedarse estático durante toda la ejecución del programa. También podemos cambiar ese dato en cualquier momento que queramos.

Ejemplo:

```
1  num = 10
2  print(num)
3  num = num*2
4  print(num)
```

La salida será:

```
Salida  10
        20
```

Este programa es fácil de entender. Aquí hay una breve explicación:

Explicación del código	
1	En la línea 1, creamos una variable numérica llamada num y le asignamos el valor numérico 10. Luego, mostramos en la pantalla el valor almacenado en esta variable.
2	En la línea 3, cambiamos el valor de la variable num multiplicándolo por 2. Luego, mostramos en la pantalla el nuevo valor de num.

Como puedes ver, no es necesario crear nuevas variables cada vez que necesitemos usar un nuevo dato. De hecho, podemos reutilizar una variable que ya esté ocupando espacio y probablemente no vayamos a usar más en el resto del programa.

Este concepto también se aplica a diferentes tipos de datos, no solo a valores numéricos. Por ejemplo, un dato almacenado en una variable puede ser numérico en un momento y después booleano. Aprenderemos más sobre los diferentes tipos de datos en diferentes secciones del libro.

Tipos de datos string o de cadena de texto

Los tipos de datos String, también conocidos como "cadenas de texto" en la programación, se refieren a datos que están en forma de texto. Por ejemplo, "Hola mundo", "Estoy aprendiendo a programar" y "Astronomía" son ejemplos de tipos de datos String. En el código, debemos encerrar el texto entre comillas, pero estas comillas no serán visibles en la salida del programa.

¿Por qué llamamos "cadenas de texto" a los tipos de datos de texto?

Puede ser confuso para aquellos que recién están empezando a programar entender la razón detrás del término "cadenas de texto". Para entenderlo, es necesario conocer algunos conceptos básicos pero importantes de la informática, como qué es un carácter en la programación.

Al inicio de este libro, hablamos sobre el funcionamiento básico de una computadora. Ahí entendimos un poco sobre los bits y su papel importante en la programación. Si quisiéramos crear un nuevo cifrado de texto y un patrón de combinación de bits para almacenar una línea de texto en específico dentro de la computadora, no sería muy eficiente.

Es aquí donde entra en juego el concepto de "cadena de caracteres". El objetivo es optimizar el uso de la memoria y mejorar el rendimiento y eficiencia de la computadora al trabajar con texto. Una cadena de caracteres es simplemente un conjunto de caracteres unidos para formar oraciones, pero, aunque reciba el nombre de "cadena de caracteres", está hecha de varios otros tipos de datos, en este caso, caracteres.

La siguiente imagen te ayudará a entender mejor lo que estamos hablando.

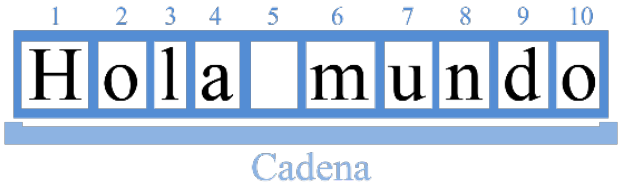


Ilustración 14 - Cadena de texto y sus componentes.

Como puedes ver en la imagen, una cadena de texto no es un objeto único, sino un tipo de dato que contiene varios otros tipos de datos, caracteres. Ahora, veamos algunos ejemplos de cómo aplicar este nuevo concepto de "cadenas de caracteres de texto" en nuestros programas.

Cambio de tipos de datos en las variables de Lua

Manejar correctamente los tipos de datos en Lua es fundamental para que nuestro código funcione correctamente. No podemos usar operaciones destinadas a tipos de datos numéricos con tipos de datos como texto o listas. Cada tipo de dato tiene sus propias operaciones que pueden ser utilizadas dentro de su mismo grupo de tipo de datos en el lenguaje de programación. Por lo tanto, es importante tener en cuenta el tipo de datos que tenemos almacenados y las operaciones que deseamos realizar con ellos.

```
1  variable = 17
2  print(type(variable))
3  variable = “17”
4  print(type(variable))
```

Al ejecutar este código, obtendremos la siguiente salida:

Salida	Number String

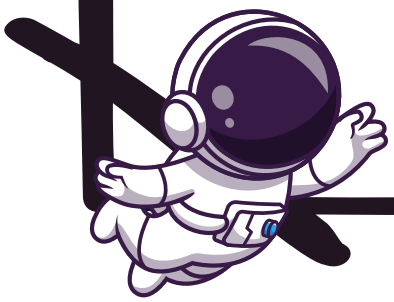
Este programa es sencillo, pero vamos a explicar lo que sucede en él.

Explicación del código	
1	Primero, creamos una variable y la inicializamos con el valor numérico 17.
2	Luego, usamos la función type() integrada en Lua para mostrar en pantalla el tipo de dato almacenado en la variable. La función type() devuelve el tipo de dato en forma de texto. En este caso, obtenemos number como respuesta.
3	Después de mostrar el tipo de dato, cambiamos el valor numérico de la variable por una cadena de caracteres que representa el número 17. Mostramos de nuevo el tipo de dato y vemos que ahora es una cadena de caracteres.

Al analizar la salida, podemos ver que, aunque ambas variables tienen el mismo número escrito (17), no son del mismo tipo de dato. La primera es un valor numérico y se pueden realizar operaciones aritméticas con él. La segunda es una cadena de texto y no se pueden realizar operaciones aritméticas con ella, solamente se pueden realizar operaciones destinadas a tipos de texto.

En conclusión

Exploramos cómo cambiar datos dentro de una variable numérica, así como el concepto de cadenas de texto y la importancia de manejar correctamente los tipos de datos en Lua. Aprendimos que podemos modificar los valores almacenados en las variables y que las variables pueden cambiar de tipo de dato según sea necesario. Además, vimos cómo identificar el tipo de dato almacenado en una variable utilizando la función type(). Al comprender y aplicar estos conceptos correctamente, podemos escribir código más eficiente y optimizado, permitiéndonos sacar el máximo provecho de las capacidades de Lua.



Concatenación de dos strings usando print()

Aprenderemos cómo hacer una de las operaciones más simples pero útiles con los tipos de datos de cadena en nuestros programas: la concatenación de dos cadenas. La concatenación es simplemente unir dos cadenas de caracteres en una sola, es decir, juntar dos listas de caracteres para formar una.

En el siguiente código, creamos dos variables de texto y las concatenamos:

```
1  texto1=”Hola “
2  texto2=”mundo”
3  print(texto1 .. texto2)
```

Al ejecutar este código, obtendremos la siguiente salida:

```
Salida  Hola mundo
```

Aquí hay una explicación detallada de lo que sucede en cada paso:

Explicación del código	
1	Creamos dos variables de texto que usaremos para mostrar cómo se vería el resultado de concatenarlas.
2	Luego, usamos la función print() para mostrar el resultado de la concatenación de ambas variables de texto, usando el operador de concatenación de texto “+”.
3	Finalmente, podemos ver que la salida del programa es el resultado de la concatenación de ambas variables de texto.

Y ¡eso es todo! Ahora sabes cómo concatenar valores de tipo de cadena de texto en tus programas. Pero recuerda que es importante practicar mucho para asegurarte de comprender completamente los conceptos y poder aplicarlos en futuros programas y temas que veas en este libro.

Concatenación de dos strings usando io.write()

Es hora de aprender dos formas diferentes de concatenar diferentes valores de tipos de cadena de texto dentro de la función io.write() de la biblioteca de entrada y salida de datos en Lua. Al igual que con la función print(), podemos usar el operador de concatenación ".." dentro de io.write(). Además, también podemos concatenar valores de tipo de cadena de texto como si fueran múltiples parámetros de la función io.write().

Antes de continuar: ¿Qué es un parámetro en una función?

En pocas palabras, los parámetros son las entradas que se le pasan a una función. Son los datos que usaremos dentro de la función y que deben ser definidos cuando escribimos el código. No te preocupes si todavía no entiendes del todo las funciones, en adelante profundizaremos más en este tema a medida que avancemos en el libro.

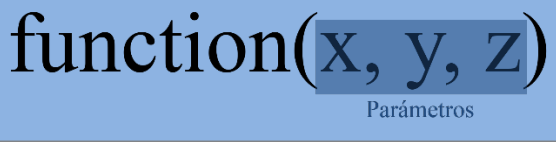


Ilustración 15 - Parámetros de una lista

Veamos un ejemplo en código para entender estos conceptos:

```
1  texto0=”Los aviones“
2  texto1=” vuelan”
3  io.write(texto0 .. texto1)
```

Ejecutando este código, obtendríamos la siguiente salida:

```
Salida  Los aviones vuelan
```

Explicación del código	
1	En este programa creamos dos variables de tipo de cadena de texto. La función principal de estas dos variables es ser usadas para mostrar cómo se vería el programa cuando ambas cadenas de texto se concatenen. Luego, usamos la función io.write() para mostrar en pantalla la concatenación de las cadenas de texto en ambas variables, usando el operador de concatenación "..".

En este ejemplo, veremos cómo concatenar cosas de una forma diferente a la que acabamos de ver. En lugar de usar un operador de concatenación, usaremos la función io.write() y usaremos cada valor de cadena de texto como un elemento más en los parámetros de la función.

Aquí tenemos un ejemplo de código que nos ayudará a entender mejor el concepto:

```
1  texto0=”Los aviones “
2  texto2=” aterrizan”
3  io.write(texto0, texto2)
```

Cuando ejecutamos este bloque de código, obtenemos la siguiente salida:

```
Salida  Los aviones aterrizan
```

En este ejemplo, estamos haciendo la misma concatenación que en el ejemplo anterior, pero sin usar un operador de concatenación. En cambio, usamos ambas variables como parámetros de la función io.write(). Obtenemos el mismo resultado que al usar el operador de concatenación.

Es importante destacar dos cosas: estamos haciendo dos casos de concatenación, uno usando el operador "," y el otro usando la función io.write(). También es importante mencionar el string "\n", que indica al programa que hay que hacer un salto de línea. Finalmente, se muestran ambos resultados en pantalla.

En conclusión, en este capítulo aprendimos a concatenar cadenas de texto en Lua utilizando tanto la función print() como la función io.write(). Hemos explorado dos enfoques diferentes para lograr la concatenación: mediante el uso del operador ".." y utilizando múltiples parámetros en la función io.write(). Ambos métodos son efectivos y proporcionan el mismo resultado. Ahora que conocemos estas técnicas, podemos aplicarlas en nuestros programas para manipular y unir cadenas de texto según sea necesario.

Concatenación entre números y strings

Antes mencionamos que la concatenación se trata de una operación que se aplica principalmente a los datos de tipo de cadena de texto, pero no significa que esté limitada a ellos. De hecho, podemos concatenar otros tipos de datos usando el operador de concatenación. Es importante destacar que, aunque podamos hacer esto, el resultado de nuestro programa seguirá siendo un valor de tipo de cadena de texto. El siguiente ejemplo ilustra esto:

```
1  io.write("Digite el primer valor: ")
2  valor1=io.read()
3  io.write("Digite el segundo valor: ")
4  valor2=io.read()
5  print("Has escrito los valores " .. tostring(valor1) .. " y " .. tostring(valor2))
```

Antes de continuar, es importante mencionar la función tostring(). Se trata de una función de conversión de tipo de dato que transforma casi cualquier tipo de dato en una cadena de texto. Por ejemplo, en el ejemplo anterior, el valor numérico almacenado en la variable que se utiliza como parámetro de la función tostring() se convierte en su equivalente en cadena de texto.

Si ejecutamos el código anterior y suponemos que el usuario ingresa 10 como el primer valor y 20 como el segundo, obtendríamos la siguiente salida:

```
Salida  Digite el primer valor: 10
        Digite el segundo valor: 20
        Has escrito los valores 10 y 20
```

Este código hace uso de la librería de entrada y salida de datos de Lua para mostrar una orden al usuario para que ingrese dos números y almacenarlos en dos variables. Luego, la función print() muestra en pantalla lo que se escribió en ambas variables. La función tostring() juega un papel importante en la concatenación, ya que permite concatenar un valor que no es de tipo de cadena, previamente convirtiéndolo.

Concatenación entre datos de cadena de texto y otros tipos de datos no numéricos

Aprendimos en los ejemplos anteriores cómo concatenar valores numéricos con valores de texto en Lua. Ahora, aunque los siguientes ejemplos abarcan los mismos temas, nos permitirán entender que la concatenación de valores de texto puede ser combinada con una amplia gama de otros tipos de datos, al igual que lo hicimos con los tipos numéricos.

Veamos algunos ejemplos de otros tipos de datos que también pueden ser convertidos a texto y luego concatenados con valores de texto en nuestros programas.

Hay algunos tipos de datos que aún no hemos visto en los ejemplos anteriores, pero no te preocupes por entender completamente lo que sucede en el código. Lo importante es que comprendas que puedes concatenar valores de texto con otros tipos de datos, siempre y cuando los conviertas a texto.

Todos los tipos de datos que veamos en el futuro serán estudiados con más profundidad en secciones posteriores de este libro, por lo que por ahora no es necesario preocuparse por entender completamente el código y su teoría detrás.

El siguiente ejemplo es similar a los anteriores y muestra cómo también podemos concatenar valores booleanos, después de convertirlos a texto.

```
1  string1 = "Escrito "
2  bool = true
3  string2 = " como valor booleano"
4  stringTotal = string1 .. tostring(bool) .. string2
5  io.write(stringTotal)
```

Al ejecutar este bloque de código, obtendremos la siguiente salida:

```
Salida  Escrito true como valor booleano
```

Explicación del código	
1	En este programa, usamos un total de 4 variables, 3 de ellas son de texto y una es un valor booleano. También usamos una variable para concatenar las otras dos variables de texto y la conversión correspondiente del valor booleano.
2	Finalmente, mostramos en pantalla el valor de la variable que contiene la concatenación de las otras variables en forma de texto.

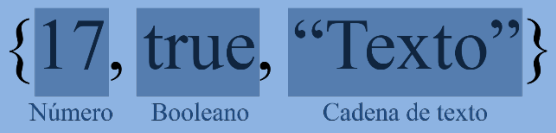
Lo mismo puede hacerse con muchos otros tipos de datos.

Veamos qué sucede cuando intentamos concatenar una lista completa en nuestro programa. ¿Qué pasa cuando usamos un valor con múltiples datos y sin unidad? La respuesta es interesante, ya que es un tipo de dato que no contiene un solo valor, por lo que su conversión a texto no es convencional.

En resumen, la concatenación de valores de texto con otros tipos de datos es posible siempre y cuando se conviertan a texto antes de la concatenación. Veamos más ejemplos a medida que avanzamos en este libro para entender mejor cómo funciona esta técnica.

Antes de continuar: ¿Qué es una lista o tabla en Lua?

Una lista o tabla en Lua es simplemente un tipo de datos que puede contener varios otros tipos de datos dentro de sí mismo. No es necesario que comprendamos todo lo que sucede en el código, sólo debemos saber que se trata de un tipo de dato que almacena otros tipos de datos. Más adelante en el libro, exploraremos este concepto en profundidad y entenderemos mejor qué es una lista.



Como se puede ver en la Ilustración 16, las tablas en Lua son una forma de almacenar múltiples tipos de datos juntos en un solo lugar. Veremos más detalles sobre esto a medida que avancemos en el libro.

Ilustración 16 - Tablas en Lua

El siguiente bloque de código muestra un ejemplo de concatenación de diferentes tipos de datos:

```
1  table = {17, 16, "hola"}
2  print("La lista es: " .. tostring(table))
```

Al ejecutar este código, obtenemos la siguiente salida:

```
Salida  La lista es: table: 00000000001690b0
```

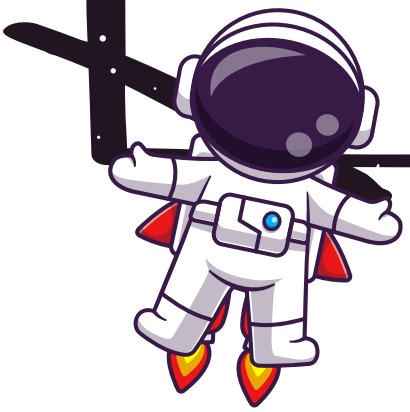
Como puedes ver, la salida no es exactamente lo que está dentro de la lista. Permíteme explicarte por qué sucede esto.

Inicialmente, creamos una variable que contiene una lista con valores numéricos y una cadena de texto. Después, pedimos a nuestro programa que muestre en pantalla la conversión de la lista a su versión en cadena de texto. Sin embargo, la salida no es lo que esperábamos. En lugar de ver los valores dentro de la lista, vemos un extraño código que dice "table: 00000000001690b0".

En realidad, este código hexadecimal muestra la dirección de memoria ocupada por la lista dentro del programa. La explicación detrás de esta salida puede ser un poco complicada, pero lo importante es entender que la conversión textual de algunos tipos de datos puede no tener sentido y generar una salida extraña.

En conclusión

Exploramos la concatenación de diferentes tipos de datos con cadenas de texto en Lua. Aprendimos que podemos concatenar números, booleanos y otros tipos de datos con cadenas de texto siempre y cuando los convirtamos a texto antes de realizar la concatenación. Sin embargo, también vimos que algunos tipos de datos, como las tablas, pueden generar resultados inesperados al ser convertidos a texto, ya que su representación en cadena muestra la dirección de memoria en lugar de los valores almacenados en sí.



Manejando y operando con cadenas de texto

Para hacer más cosas con nuestras cadenas de texto, podemos utilizar algo llamado librerías. Estudiarás más sobre librerías en detalle en otras partes de este libro.

Llamemos a esto "manejo de cadenas de texto", donde usaremos funciones especiales del lenguaje de programación para trabajar y hacer operaciones más útiles y avanzadas con nuestras cadenas de texto. Por ejemplo, podremos convertir todas las letras de una oración en mayúsculas o minúsculas, revertir el texto, entre otras cosas.

Aquí hay algunos ejemplos escritos en Lua de estas funcionalidades adicionales que podemos hacer con las cadenas de texto. ¡Es hora de ponerse en marcha y empezar a programar en este maravilloso lenguaje de programación!

Conversión de una cadena de texto a mayúsculas

En este ejemplo, veremos cómo realizar una operación sencilla en los tipos de datos de cadena de texto utilizando la librería "string", la cual es la librería para manejar cadenas de texto en Lua.

Aquí tienes el siguiente bloque de código:

```
1    cadena = "mayúsculas"
```

```
2  print("Antes de la conversión: " .. cadena)
3  cadena = string.upper(cadena)
4  print("Después de la conversión: " .. cadena)
```

Al ejecutar este bloque de código, obtendrás la siguiente salida:

```
Salida  Antes de la conversión: mayúsculas
        Después de la conversión: MAYUSCULAS
```

Como puedes ver, lo que está sucediendo en este código es bastante simple: estamos generando una salida que toma cada carácter de una cadena de texto y lo convierte a su equivalente en mayúsculas.

Veamos una explicación breve del ejemplo anterior, aunque sea algo muy básico para entender conceptualmente lo que sucede en el código.

Explicación del código	
1	En primer lugar, usamos una variable llamada "cadena" y le asignamos un valor de tipo cadena de texto que contiene la palabra "mayúsculas".
2	Luego, mostramos en la pantalla el valor almacenado en esa variable. Después de eso, entra en acción la librería "string".
3	Utilizamos la función "string.upper()" de la librería "string", que convierte el texto dentro de ella a su equivalente en mayúsculas. Después, volvemos a mostrar en la pantalla el nuevo valor dentro de la variable y confirmamos que se ha convertido correctamente a mayúsculas.

Y así, hemos convertido correctamente los caracteres en minúsculas de una cadena de texto a su equivalente en mayúsculas.

string.upper(x)

La función "string.upper(x)" es una herramienta muy útil de la librería de manejo de cadenas de texto en Lua. Simplemente necesitas pasarle una cadena de texto como parámetro, y automáticamente te devolverá una nueva cadena con todas las letras en mayúsculas.

Por ejemplo, si tienes la cadena "hola mundo", puedes usar "string.upper(x)" para convertirla en "HOLA MUNDO". Es una forma fácil y rápida de darle un formato uniforme a tus cadenas de texto.

Ilustración 17 - Función string.upper(x)

Convirtiendo una cadena de texto a minúsculas

En este ejemplo, aprenderás cómo convertir todos los caracteres de una cadena de texto en su equivalente en minúsculas.

```
1  cadena = “MINUSCULAS”
2  print(“Antes de hacer la conversión: “ .. cadena)
```

```
3   cadena = string.lower(cadena)
4   print("Después de hacer la conversión: " .. cadena)
```

Al ejecutar este código, obtendrás la siguiente salida:

```
Salida   Antes de hacer la conversión: MINUSCULAS
         Después de hacer la conversión: minúsculas
```

En este ejemplo, usamos una variable llamada "cadena" y le asignamos un valor de tipo texto "MINUSCULAS". Luego, mostramos en pantalla el valor de la variable antes de la conversión. A continuación, usamos la función string.lower() de la librería de manejo de texto de Lua para convertir el texto en su equivalente en minúsculas. Finalmente, mostramos en pantalla el nuevo valor de la variable después de la conversión.

¡Y eso es todo! Ahora ya sabes cómo convertir una cadena de texto a minúsculas usando la librería de manejo de texto de Lua. ¡Practica y agrega esta habilidad a tu lista de funciones en Lua!

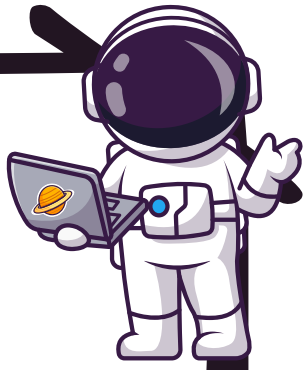
string.lower(x)

La función string.lower(x) es la encargada de convertir todos los caracteres de una cadena de texto a minúsculas. Esta función es el proceso inverso de string.upper(), que convierte los caracteres a mayúsculas.

Al proporcionar una cadena de texto como entrada, la función string.lower() generará una nueva cadena con todos los caracteres en minúsculas. Por ejemplo, si proporcionamos la cadena 'HOLA MUNDO' como entrada, la función generará la cadena 'hola mundo'.

Ilustración 18 - Función string.lower()

Ahora, podemos decir que tenemos una idea más clara sobre las operaciones que podemos realizar con valores de tipo de texto en Lua. En las siguientes secciones de este libro, veremos diferentes ejemplos y prácticas que nos ayudarán a poner en práctica estos dos operadores. De esta forma, no olvidaremos su función con el tiempo y mientras aumenta nuestro conocimiento sobre las diferentes funciones en Lua. Además, comprenderemos cómo estas funciones pueden ser aplicadas de manera práctica y real en la creación de programas. A continuación, revisaremos un breve caso de estudio donde utilizaremos estas dos funciones que acabamos de aprender en un solo programa.



En conclusión

Exploramos el manejo de cadenas de texto en Lua y cómo realizar operaciones útiles y avanzadas con ellas utilizando la librería "string". Aprendimos a convertir cadenas de texto a mayúsculas y minúsculas utilizando las funciones "string.upper()" y "string.lower()" respectivamente. Estas habilidades nos permiten manipular y dar formato a nuestras cadenas de texto de manera efectiva.

Caso de estudio: Generación de correos electrónicos y contraseñas para trabajadores

En este caso de estudio, supongamos que estamos trabajando en una empresa que nos ha pedido crear un programa que, a partir del nombre y apellido de un trabajador, genere un usuario y una contraseña de correo electrónico que cumplan con ciertas condiciones específicas. Aquí explicaremos estas condiciones en detalle:

El programa requerirá dos entradas de texto. La primera entrada será el nombre del trabajador y la segunda entrada será el primer apellido. Luego, el programa deberá devolver dos salidas. La primera salida será el correo electrónico generado, el cual deberá estar en minúsculas y compuesto por el nombre y apellido del trabajador, separados por un punto, seguido del identificador de la empresa (@empresa.com). La segunda salida será la contraseña, la cual estará compuesta por el apellido del trabajador en mayúsculas, seguido de un guion bajo y un número aleatorio de 4 dígitos.

Ilustración 19 - "El programa debe de generar el correo electrónico y contraseña del trabajador"

Nuestro objetivo como programadores es crear un programa que realice esta tarea para facilitar el proceso de creación de correos electrónicos para los trabajadores de la empresa.

Ejemplos:

A continuación, se muestra una tabla con algunos ejemplos de entradas y salidas esperadas del programa:

Nombre: Juan Apellido: Pérez Correo electrónico: juan.perez@empresa.com Contraseña: PEREZ_2765	Nombre: Martha Apellido: Mina Correo electrónico: martha.mina@empresa.com Contraseña: MINA_6543
Nombre: Luis Apellido: Pino Correo electrónico: luis.pino@empresa.com Contraseña: PINO_2070	Nombre: Viviana Apellido: Rojas Correo electrónico: viviana.rojas@empresa.com Contraseña: ROJAS_7490

Ahora que tenemos la información necesaria para solucionar nuestro programa, podemos comenzar a trabajar en la solución. Empezaremos con una breve bienvenida cada vez que se ejecute nuestro programa. Esta bienvenida será una pequeña muestra de amabilidad hacia el operador de nuestro programa. El siguiente código muestra cómo podemos hacer que nuestro programa enseñe esta bienvenida:

```
1  print("Bienvenido")
2  print("La función de este programa es generar correos electrónicos con sus contraseñas correspondientes, basados en el primer nombre y apellido que proporcionemos dentro del programa.")
3  print("Asegúrate de proporcionar y escribir correctamente la información solicitada dentro del programa.")
```

Cuando ejecutes este código, se generará la siguiente bienvenida en la pantalla de la terminal de ejecución:

```
Salida  Bienvenido
        La función de este programa es generar correos electrónicos con sus contraseñas correspondientes, basados en el primer nombre y apellido que proporcionemos dentro del programa.
        Asegúrate de proporcionar y escribir correctamente la información solicitada dentro del programa.
```

Ahora que mostramos la bienvenida e introducción al operador de nuestro programa, es hora de pedir algunos datos para poder usarlos más adelante en el programa. Podemos hacer esto con el siguiente código:

```
4  print("Por favor, escriba el primer nombre del trabajador:")
5  nombreTrabajador = io.read()
6  print("Por favor, escriba el primer apellido del trabajador:")
7  apellidoTrabajador = io.read()
```

De esta manera, el usuario y operador del programa serán los responsables de escribir y proporcionar el nombre y apellido del trabajador para su uso dentro del programa.

Finalmente, ahora que tenemos todos los datos necesarios proporcionados por el usuario dentro de nuestro programa, es hora de generar otras variables con estos datos. Podemos hacer esto con el siguiente código, comenzando por crear el correo electrónico del trabajador:

```
8  correoElectronico = string.lower(nombreTrabajador .. "." .. apellidoTrabajador) .. "@empresa.com"
9  print("El correo electrónico asignado a este trabajador es: " .. correoElectronico)
```

De esta manera, podemos generar el correo electrónico del trabajador y mostrarlo en pantalla para su revisión.

Es hora de generar una contraseña para el correo electrónico. Es importante recordar que la contraseña debe seguir un formato específico, que consiste en el apellido del trabajador en mayúsculas, seguido de un guion bajo "_" y, finalmente, cuatro dígitos aleatorios.

Podemos generar la contraseña del correo electrónico de la siguiente manera:

```
10 claveCorreo = string.upper(apellidoTrabajador) .. "_" .. tostring(math.random(0, 9)) .. tostring(math.random(0, 9)) .. tostring(math.random(0, 9)) .. tostring(math.random(0, 9))
11 print("La clave asignada a este correo electrónico es: " .. claveCorreo)
```

Es posible que algunas partes del código parezcan confusas, pero no te preocupes, estudiaremos mucho más a fondo el funcionamiento de estas librerías y sus funciones en secciones posteriores del libro.

Antes de continuar: ¿Qué hace math.random(0, 9)?

Se trata de una función que forma parte de la librería matemática de Lua. Estudiaremos mucho más a fondo el funcionamiento y la utilidad general de esta función en secciones posteriores de este libro. Sin embargo, podemos entender de manera sencilla que esta función genera un número entero aleatorio en el rango [0, 9].

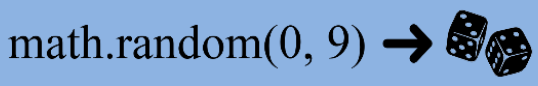


Ilustración 20- Función math.random()

¡Y eso es todo! Finalmente podemos decir que hemos cumplido con todas las condiciones necesarias para nuestro programa. Ahora podemos entender completamente lo que ocurre en nuestro código.

Si escribimos y analizamos el código que acabamos de crear, podemos ver que el bloque de código completo de nuestro programa es el siguiente:

```
1  print("Bienvenido")
2  print("La función de este programa es generar correos electrónicos con sus contraseñas correspondientes, basados en el primer nombre y apellido que proporcionemos dentro del programa.")
3  print("Asegúrate de proporcionar y escribir correctamente la información solicitada dentro del programa.")
4  print("Por favor, escriba el primer nombre del trabajador:")
5  nombreTrabajador = io.read()
6  print("Por favor, escriba el primer apellido del trabajador:")
7  apellidoTrabajador = io.read()
8  correoElectronico = string.lower(nombreTrabajador .. "." .. apellidoTrabajador) .. "@empresa.com"
```

```
9  print("El correo electrónico asignado a este trabajador es: " .. correoElectronico)
10 claveCorreo = string.upper(apellidoTrabajador) .. "_" .. tostring(math.random(0, 9)) .. tostring(math.random(0, 9)) .. tostring(math.random(0, 9)) .. tostring(math.random(0, 9))
11 print("La clave asignada a este correo electrónico es: " .. claveCorreo)
```

Si suponemos que el operador escribe el nombre "Daniel" y el apellido "Duran", entonces el programa generará la siguiente salida en la terminal de ejecución:

```
Salida
Bienvenido
La función de este programa es generar correos electrónicos con sus contraseñas correspondientes, basados en el primer nombre y apellido que proporcionemos dentro del programa.
Asegúrate de proporcionar y escribir correctamente la información solicitada dentro del programa.
Por favor, escriba el primer nombre del trabajador: Daniel
Por favor, escriba el primer apellido del trabajador: Duran
El correo electrónico asignado a este trabajador es: daniel.duran@empresa.com
La clave asignada a este correo electrónico es: DURAN_9218
```

Veamos una breve explicación de todo lo que ocurre dentro de nuestro código:

Explicación del código	
1	Al inicio, pedimos al usuario que ingrese el nombre y apellido del trabajador para crear su correo electrónico. Les damos una cálida bienvenida y les explicamos brevemente de qué trata nuestro programa.
2	Luego, utilizamos las funciones string.lower() y string.upper() para crear el correo electrónico y la contraseña correspondiente, respectivamente.
3	Por último, mostramos en pantalla la información obtenida para verificar que el correo electrónico y la contraseña se han creado correctamente.

¡Eso es todo! Con este ejemplo, hemos aplicado las funciones string.lower() y string.upper() en un caso práctico relacionado con el mundo laboral. Hemos demostrado cómo podemos aplicar nuestros conocimientos adquiridos sobre este tema en un desarrollo de programa real.

En conclusión

Creamos un programa que cumple con las condiciones específicas de la empresa para la generación de correos electrónicos y contraseñas de sus trabajadores. Para lograr esto, utilizamos las funciones string.lower() y string.upper() para crear el correo electrónico y la contraseña, respectivamente. Además, Aprendimos a utilizar la función math.random() para generar números aleatorios y cómo podemos aplicar nuestros conocimientos adquiridos sobre programación en un caso práctico relacionado con el mundo laboral.

Reemplazar caracteres en una cadena de texto con Lua

Antes aprendimos algunas operaciones básicas con cadenas de texto en Lua, pero hay muchas más funciones disponibles. Ya entendimos que una cadena de texto es simplemente un conjunto de caracteres unidos para formar oraciones, y que cada carácter en la cadena es independiente de los demás.

Con esto en mente, podemos explorar operaciones como reemplazar caracteres específicos en una cadena con otros caracteres definidos por nosotros. Es decir, podemos hacer que nuestro programa busque caracteres específicos en una cadena de texto y los reemplace con otros que definamos en el código.

Ilustración 21 - Cambio de bloque de caracteres

También podemos experimentar con reemplazar bloques de caracteres en lugar de solo un carácter.

Ilustración 22 - Cambio de caracteres individuales

Veamos un ejemplo de código para ver cómo aplicar esto en la práctica:

```
1  saludo = “Hola, Adam. Me alegra bastante volver a verte Adam”
2  print(“Primer saludo:”)
3  print(saludo)
4  saludo = string.gsub(saludo, “Adam”, “Mike”)
5  print(“Segundo saludo:”)
6  print(saludo)
```

Al ejecutar este bloque de código, obtendríamos la siguiente salida en la terminal:

```
Salida
Primer saludo:
Hola, Adam. Me alegra bastante volver a verte Adam
Segundo saludo:
Hola, Mike. Me alegra bastante volver a verte Mike
```

Ahora, es cierto que este programa es bastante fácil de entender, pero vamos a ver una breve explicación del código para asegurarnos:

Explicación del código	
1	Creamos una variable llamada "saludo" que contiene una cadena de texto "Hola, Adam. Me alegra bastante volver a verte Adam". Luego mostramos en pantalla el valor de la variable.
2	Usamos la función string.gsub() para reemplazar "Adam" por "Mike". Esto hace que el valor de la variable cambie a "Hola, Mike. Me alegra bastante volver a verte Mike".
3	Finalmente, mostramos en pantalla el valor final de la variable para verificar que el reemplazo se haya realizado correctamente.

Ahora que estamos estudiando una nueva función, es importante entender su estructura y cómo funciona para poder aplicarla correctamente en nuestros programas. La estructura de la función string.gsub() es la siguiente:

string.gsub(Cadena base, Caracteres de búsqueda, Caracteres de reemplazo)	
Cadena base	Es el valor o variable donde se aplicará la función. Buscaremos caracteres dentro de esta cadena.
Caracteres de búsqueda	Son los caracteres que buscaremos dentro de la cadena base.
Caracteres de reemplazo	Son los caracteres que reemplazaremos una vez encontrados los caracteres de búsqueda.

El siguiente ejemplo muestra cómo podemos usar la función string.gsub() para reemplazar un solo carácter por un valor vacío, lo que se puede ver como "eliminar caracteres" dentro de una cadena de texto.

```
1  cadena = "Autzonzomzíza"
2  print("Antes: " .. cadena)
3  cadena = string.gsub(cadena, "z", "")
4  print("Después: " .. cadena)
```

Al ejecutar este código, obtendríamos la siguiente salida en la terminal:

```
Salida  Antes: Autzonzomzíza
        Después: Autonomía
```

Finalmente, vamos a ver una explicación detallada del código para entender mejor su funcionamiento:

Explicación del código	
1	Primero, creamos una variable llamada "cadena" y le asignamos un valor de tipo cadena de texto, específicamente "Autzonzomzíza". Después de definir la variable, mostramos su valor en la terminal de ejecución.
2	Luego, hacemos uso de la función string.gsub() para reemplazar los caracteres "z" por un valor vacío.
3	Finalmente, mostramos en la terminal de ejecución el valor final de la cadena después de haber aplicado la función string.gsub() y verificamos que cada ocurrencia del carácter "z" ha sido eliminada correctamente.

```
string.gsub(x, y, z)
Esta es una función que permite reemplazar caracteres específicos en un valor de tipo de cadena de texto por otros
caracteres también definidos dentro de la misma función.
Esta función recibe tres parámetros:
x: El valor de la cadena de texto base.
y: Los caracteres a buscar y reemplazar.
z: Los caracteres de reemplazo.
La función genera una cadena de texto con los caracteres reemplazados.
```

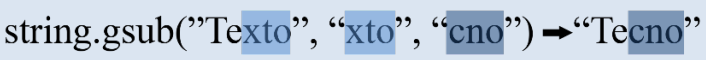


Ilustración 23 - Función string.gsub()

Ahora que tenemos una comprensión clara y detallada de cómo reemplazar caracteres dentro de valores de tipo de cadena de texto, podemos seguir explorando y descubriendo más funcionalidades y posibilidades dentro de Lua. Este es solo el comienzo de lo que podemos hacer con este poderoso lenguaje de programación.

Invertir cadenas de texto en Lua

Es hora de ver una operación más con las cadenas de texto en Lua. Esta vez, nos enfocaremos en cómo invertir el orden de los caracteres en una cadena de texto. Es un concepto fácil de entender y te será útil en tus programas.

Echa un vistazo a la siguiente ilustración para entenderlo mejor:

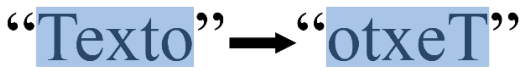


Ilustración 24 - Inversión de una cadena de texto

Para lograr esto, necesitamos una función que pueda escanear carácter por carácter la cadena de texto y cambiar el orden de los caracteres. Para hacerlo, usaremos la librería de manejo de cadenas de texto en Lua. Te presentaremos algunos ejemplos simples para que puedas lograr esto fácilmente en tus programas.

Ejemplo de inversión de una cadena de texto en Lua

Este ejemplo es sencillo de entender. Aquí tienes el código:

```
1  var = “Hola, mundo”
2  print("Antes de revertir: " .. var)
3  var = string.reverse(var)
4  print("Después de revertir: " .. var)
```

Al ejecutar este código, obtendrás la siguiente salida:

Salida	Antes de revertir: Hola, mundo Después de revertir: odnum ,aloH

Explicación del código	
1	Al principio, creamos una variable llamada "var" y le asignamos el valor de cadena de texto "Hola mundo". Luego, usamos la función print() para mostrar el valor que se encuentra en la variable.
2	A continuación, usamos la función string.reverse() de la librería de manejo de texto para invertir el orden de los caracteres en la cadena de texto. Esta función asigna el resultado invertido a la variable "var" y finalmente, usamos print() para mostrar el resultado invertido en la ventana de ejecución.
3	Por último, podemos ver que los caracteres de la cadena de texto se han invertido con éxito.

¡Y eso es todo! Ahora sabes cómo invertir el orden de los caracteres en una cadena de texto en Lua. Esta es una operación sencilla, pero es útil para recordar y practicar para futuros usos.

string.reverse(x)

La función string.reverse(x) tiene como objetivo cambiar el orden de los caracteres en una cadena de texto. Invierte la cadena de texto que se utiliza como parámetro y la escribe en sentido contrario.

Parámetro: Cadena de texto

Resultado: Cadena de texto escrita al revés

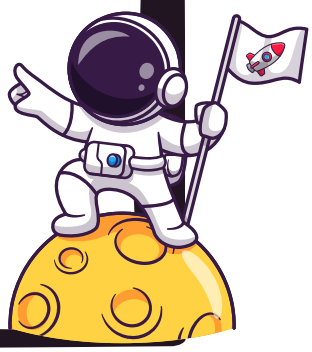
string.reverse(”Espacio”) ➜ “oicapsE”

Ilustración 25- Función string.reverse()

Ahora tienes una comprensión clara de cómo invertir el orden de los caracteres en una cadena de texto en Lua. Aunque es una operación sencilla, es importante practicar y recordar esta función para futuros usos en diferentes tipos de casos. ¡Buen trabajo!

En conclusión

Aprendimos a reemplazar caracteres y bloques de caracteres en una cadena de texto usando la función string.gsub() en Lua. Además, también aprendimos a invertir el orden de los caracteres en una cadena de texto utilizando la función string.reverse(). Es importante tener en cuenta que estas funciones nos brindan muchas posibilidades y es útil comprender su estructura y funcionamiento para aplicarlas correctamente en nuestros programas.



Repetir una cantidad definida de veces una cadena de texto

Ahora es momento de aprender otra habilidad con los tipos de datos de cadena de texto en este lenguaje de programación. Se trata de una tarea sencilla y, en cierto modo, está relacionada con los bucles y las operaciones repetitivas que veremos más adelante en este libro. La habilidad de repetir una cadena de texto una cantidad definida de veces. No te preocupes por entender los bucles ahora. Lo estudiaremos en profundidad más adelante. Lo que veremos ahora es cómo repetir una cadena de texto una cantidad determinada de veces. Veamos un ejemplo sencillo para entender cómo hacer esto en nuestros programas.

```
1  text = “Repito”
2  print(string.rep(text, 5))
```

Al ejecutar este código, obtendremos lo siguiente en la terminal:

```
Salida  RepitoRepitoRepitoRepitoRepito
```

Podemos ver que la función trabaja de manera cíclica concatenando una cadena de texto una cantidad determinada de veces. Entenderemos mejor cómo se genera esta salida y cómo se relaciona con los bucles en Lua más adelante.

Antes de continuar: ¿Qué es un bucle?

En términos simples, un bucle en programación se refiere a un bloque de código que se ejecuta repetidamente una cantidad determinada de veces, es decir, un código que se repite una y otra vez hasta que se cumpla alguna condición.

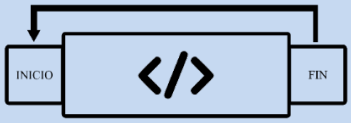


Ilustración 26 – Estructura general de un bucle

Ahora entendemos cómo utilizar la función de la librería de manejo de texto en Lua para repetir una cadena de texto una cantidad definida de veces en la terminal de nuestro programa.

string.rep(x, y)

Esta función es muy fácil de entender tanto en su uso práctico como en su concepto. Su función principal es repetir una cadena de texto una cantidad determinada de veces, según los parámetros definidos en la función.
Los parámetros son:
x: Cadena de texto que será repetida
y: Cantidad de veces que se repetirá la cadena de texto
La función genera una cadena de texto repetida secuencialmente la cantidad de veces definida.

string.rep(”Hola”, 3)➡“Hola Hola Hola”

Ilustración 27 - Función string.rep()

Es normal sentirse un poco confundido acerca del uso de los bucles en programación, pero no te preocupes, estudiaremos mucho más sobre ellos en el futuro. De hecho, la mayoría de las veces, las salidas repetitivas se logran a través de bucles. Por ahora, solo debemos concentrarnos en aprender la función de repetición de cadenas de texto en Lua. ¡Buen trabajo!

Medir la longitud de una cadena en Lua

En esta sección, aprenderemos cómo medir la longitud de una cadena de texto en Lua. Aunque puede ser confuso para aquellos que no tienen experiencia previa en programación, en realidad, medir la longitud de una cadena de texto es un concepto sencillo. Una cadena de texto es simplemente una secuencia de caracteres que forman un texto. Por lo tanto, medir la longitud de una cadena de texto significa simplemente contar cuántos caracteres la conforman.

Ilustración 28 - Longitud de una cadena de texto

Podemos hacer esto utilizando la librería de manejo de texto de Lua, la cual hemos estado usando para realizar diferentes operaciones con las cadenas de texto. Medir la longitud de un texto en un programa puede tener muchas aplicaciones, pero no es necesario preocuparse por ellas ahora. Más adelante en el libro, estudiaremos algunos ejemplos de cómo aplicar esta técnica en diferentes situaciones. El siguiente bloque de código ilustra lo que hemos estado hablando:

```
1  string1 = "Longitud de texto"
2  print("Primera cadena de texto: " .. string.len(string1))
3  string2 = "Manzana"
4  print("Segunda cadena de texto: " .. string.len(string2))
```

Cuando ejecutamos este código, veremos la siguiente salida en la terminal:

```
Salida  Primera cadena de texto: 17
        Segunda cadena de texto: 7
```

En conclusión

En este capítulo aprendimos dos habilidades importantes para trabajar con cadenas de texto en Lua. En primer lugar, vimos cómo repetir una cadena de texto una cantidad definida de veces utilizando la función string.rep(). Esta función es útil para generar salidas repetitivas en nuestros programas. En segundo lugar, aprendimos cómo medir la longitud de una cadena de texto utilizando la función string.len(). Esta habilidad es importante para realizar diferentes operaciones con cadenas de texto en nuestro programa. Aunque medir la longitud de una cadena de texto puede parecer una tarea sencilla, es fundamental para trabajar con ellas en Lua

Ejercicios de práctica	
1	Crea un programa que sume los valores numéricos 10 y 20 y muestre el resultado en pantalla. Para resolver este problema, debes crear dos variables para almacenar ambos números, luego sumarlas y usar la función print() para mostrar el resultado en la pantalla.
2	Completa el siguiente bloque de código: 1 variable1 = 2 = 37 3 print(* variable2) La salida esperada es: Salida 370
3	Crea un programa que pida dos valores de tipo de cadena de texto al usuario y los concatene, mostrando el resultado en pantalla. Debes usar la función io.read() para recibir los valores de teclado y la función print() para mostrar el resultado en pantalla. Recuerda que el operador de concatenación para cadenas de texto es "..".
4	Crea un programa que reciba un valor numérico del usuario y muestre en pantalla la mitad de ese valor. Debes usar la función io.read() para recibir el valor numérico y la función print() para mostrar el resultado en pantalla. También puedes usar la función tostring() si es necesario.
5	Crea un programa que muestre en pantalla la siguiente salida: Salida Hola mundo! Debes usar solo una línea de código y la función io.write(). Recuerda que los caracteres "\n" son interpretados como un salto de línea en la terminal de ejecución.
6	Crea un programa que pida un valor numérico y una cadena de texto al usuario, y los concatene, mostrando el resultado en pantalla. Debes usar tres variables, dos para almacenar los valores recibidos de teclado y una tercera para almacenar el resultado de la concatenación. Usa la función io.read() para recibir los valores de teclado y la función print() para mostrar el resultado en pantalla. También puedes usar la función tostring() si es necesario.

7	Crea un programa que realice una secuencia de operaciones numéricas según el siguiente orden: 1 Pide dos valores numéricos al usuario. 2 Divide el primer valor entre el segundo y almacena el resultado en una tercera variable. 3 Multiplica el valor de la tercera variable por cuatro (4). 4 Muestra el valor final de la tercera variable en pantalla. Debes usar las variables y la cantidad de operaciones indicadas en la descripción.

Tipos de datos booleanos, lógica booleana y operadores lógicos

¡Vamos a explorar uno de los temas más importantes en el mundo de la programación: la lógica booleana y los operadores lógicos! La lógica booleana, también conocida como álgebra de Boole, es una estructura matemática que nos permite representar valores de verdad con un simple "Verdadero" o "Falso".

En Lua, los tipos de datos booleanos son justamente esos dos valores: "Verdadero" y "Falso", que se representan como "true" y "false". Al principio, entender cómo funciona la lógica booleana y sus operadores puede ser un poco confuso, pero no te preocupes, vamos a profundizar en estos conceptos con ejemplos para que puedas entenderlos de manera clara y sencilla.

Operador relacional de igualdad

El operador relacional de igualdad es uno de los operadores lógicos que debemos conocer en la lógica booleana. Es importante estar familiarizados con su funcionamiento y notación antes de usarlo en nuestro código, ya que puede confundirse con el operador de asignación de valor de Lua.

Para comprender bien el funcionamiento del operador de igualdad, es necesario hacer una serie de preguntas como: ¿Son iguales ambos elementos?, ¿son del mismo tipo de dato ambos elementos? Si la respuesta a ambas preguntas es sí, entonces el resultado será un valor booleano "true", de lo contrario, el resultado será "false".

Este operador nos permite comparar dos elementos y determinar si son iguales o no. La notación correcta es importante para evitar errores en nuestro código. En resumen, el operador relacional de igualdad es una herramienta esencial en la lógica booleana y debemos conocerlo bien antes de usarlo en nuestros programas.

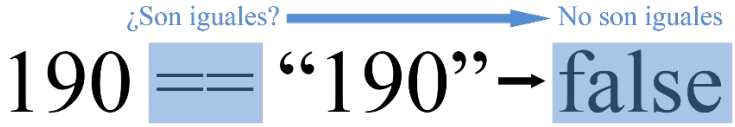


Ilustración 29 - Operador lógico de igualdad

La estructura correcta para el operador lógico de igualdad es la siguiente:

Estructura [Elemento A] == [Elemento B]	
[Elemento A]	Es el primer elemento que se usará para comparar valores. Se considera su tipo de dato y su valor.
==	Es el operador de igualdad, se escribe como dos operadores de asignación consecutivos (==). Este operador indica al programa que se realizará una operación lógica.
[Elemento B]	Es el segundo elemento que se usará para comparar valores. Se considera su tipo de dato y su valor.
Salida	
[Resultado booleano]	Si [Elemento A] y [Elemento B] son iguales en tipo de dato y valor, entonces la salida será "true". De lo contrario, la salida será "false". La salida es un resultado booleano que puede ser "true" o "false", dependiendo de si [Elemento A] y [Elemento B] son iguales o no.

Como puedes ver, el concepto detrás del operador lógico de igualdad es fácil de comprender. Veamos un ejemplo para ver cómo funciona en práctica:

En el siguiente bloque de código, usamos el operador lógico de igualdad:

```
1  num1=7
2  num2=3
3  print((num1+num2)==10)
```

Al ejecutar este código, obtenemos la siguiente salida:

```
Salida  true
```

En este ejemplo, estamos comparando si la suma de num1 y num2 es igual a 10. Como 7 + 3 = 10, la salida es "true". Esto demuestra cómo funciona el operador lógico de igualdad en la práctica.

Veamos otro ejemplo de código para aplicar el operador lógico de igualdad:

```
1  Variable = “17”
2  Variable2 = 17
3  Relación = Variable == Variable2
4  print(“¿Son las dos variables iguales?: “ .. tostring(Relación))
```

Al ejecutar este código, obtenemos la siguiente salida:

```
Salida  ¿Son las dos variables iguales?: false
```

Explicación del código	
1	Primero, creamos dos variables: "Variable" y "Variable2". La primera recibe un valor de tipo de cadena de texto "17", mientras que la segunda recibe un valor numérico 17. Luego, creamos una tercera variable "Relación" que contiene el resultado booleano de la comparación entre "Variable" y "Variable2" usando el operador lógico de igualdad.
2	Finalmente, imprimimos en pantalla si ambas variables son iguales o no. Como "17" es una cadena de texto y 17 es un número, la salida es "false".

Este ejemplo nos muestra cómo funciona el operador lógico de igualdad en la práctica. Con estos ejemplos, deberías tener una buena comprensión de cómo funciona este operador en los programas.

Operador relacional de desigualdad

Vamos a echar un vistazo al operador relacional de desigualdad. Este operador es similar al operador de igualdad que acabamos de ver.

El operador de igualdad nos devuelve un valor de verdad si ambos operadores son iguales. Por otro lado, el operador de desigualdad verifica que ambos valores sean diferentes entre sí, incluso si son del mismo tipo de dato. Si los dos valores son diferentes, ya sea en su tipo de dato o en su contenido, entonces el operador de desigualdad funcionará correctamente.

Al principio, puede resultar un poco confuso de entender, pero en realidad, este operador devuelve un valor de verdad cuando los valores comparados son diferentes entre sí. Por otro lado, devuelve un valor booleano falso cuando los valores comparados son exactamente iguales, tanto en su contenido como en su tipo de dato.

Vamos a ver algunos ejemplos prácticos con código para que podamos entender mejor los conceptos que hemos estado discutiendo. Echemos un vistazo a este bloque de código:

```
1  Nombre = ”Juan”
2  print(nombre ~= ”Ernesto”)
```

Cuando ejecutamos este código, obtenemos la siguiente salida en la pantalla de terminal de nuestro programa:

```
Salida  true
```

Veamos una breve explicación de lo que sucede en este código:

Explicación del código	
1	En este ejemplo, creamos una variable llamada "nombre" y le asignamos el valor de tipo de cadena de texto "Juan".
2	Luego, mostramos algo en la terminal de ejecución de nuestro programa. En este caso, estamos pidiendo al programa que nos muestre el valor de verdad resultante del uso del operador de desigualdad entre la variable "nombre" y el valor de tipo de cadena de texto "Ernesto". Como es obvio, "Juan" no es igual a "Ernesto", por lo que la desigualdad se cumple y, por supuesto, la salida generada por nuestro programa es el valor booleano "true".

Veamos un segundo ejemplo para asegurarnos de entender completamente cómo funciona el operador de desigualdad en Lua:

```
1  Numero1 = 20
2  Numero2 = Numero1 / 2
3  Numero3 = 10
4  print(“¿Son los dos números desiguales?: ” .. tostring(Numero2 ~= Numero3))
```

Al ejecutar este código, obtenemos la siguiente salida en la terminal de nuestro programa:

```
Salida  ¿Son los dos números desiguales?: false
```

Explicación del código	
1	En este ejemplo, creamos una variable llamada "Numero1" y le asignamos el valor numérico 20. Luego, creamos otras dos variables llamadas "Numero2" y "Numero3", a las cuales les asignamos los valores de la división de "Numero1" entre 2 y el número 10, respectivamente.
2	Finalmente, le pedimos al programa que muestre en la pantalla el valor de verdad resultante de la comparación de la desigualdad entre "Numero2" y "Numero3". Debido a que "Numero2" contiene el resultado de la división de "Numero1" (cuyo valor es 20) entre 2 y "Numero3" contiene el valor 10, ambos son iguales. Por lo tanto, la desigualdad no se cumple y la salida de nuestro programa es el valor booleano "false".

Operadores lógicos “mayor qué” y “menor qué”

Vamos a ver los operadores lógicos "mayor que" y "menor que". Son muy útiles en ejemplos numéricos que realizaremos en nuestros programas y prácticas. La importancia de estos dos operadores radica en su capacidad para comparar el tamaño y magnitud de valores numéricos y controlar acciones en bloques de código que se manejan por medio de condicionales o casos de estudio.

Estos dos operadores lógicos comparan si un valor numérico es mayor o menor que otro valor numérico, y devuelven un valor booleano verdadero o falso dependiendo de la comparación. En los próximos ejemplos, profundizaremos en cómo aplicar estos operadores en nuestros programas. Por ahora, echemos un vistazo a una explicación gráfica para comprender mejor este tema.

El siguiente gráfico muestra cómo funcionan y qué valores de verdad generan estos dos operadores lógicos "mayor que" y "menor que". En la primera línea, estamos usando el operador "mayor que" para comparar si 17 es mayor que 19, que es falso. En la segunda línea, usamos el operador "menor que" y preguntamos si 17 es menor que 19, lo que es verdadero.

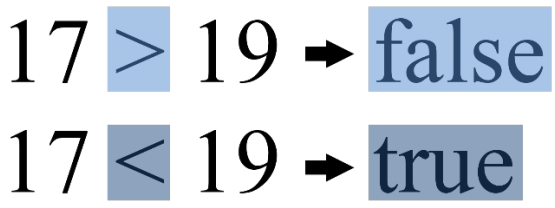


Ilustración 30 - Operador de comparación de valor numérico mayor qué y menor qué

¡Y eso es todo! Ahora que entendemos cómo funcionan estos operadores, podemos ver ejemplos más prácticos que usen bloques de código para comprender mejor estos nuevos temas.

Estos siguientes ejemplos de código te muestran diferentes situaciones en las que puedes ver y entender cómo usamos estos dos operadores lógicos:

Primer ejemplo:

```
1  num1 = 10
2  num2 = 7
```

```
3   print(num1 > num2)
```

```
Salida   true
```

Segundo ejemplo:

```
1   num = 30
2   print(30 < (10 * 10))
```

```
Salida   true
```

Vamos a analizar lo que sucede en este código:

Explicación del código	
1	Estos ejemplos son fáciles de entender. En el primer ejemplo, le pedimos a nuestro programa que nos muestre si 10 es mayor que 7, que es verdadero. En el segundo ejemplo, comparamos si 30 es menor que 100, también verdadero. Por lo tanto, nuestro programa devuelve como resultado true en ambos casos.
2	En resumen, en ambos ejemplos hicimos uso de los operadores lógicos "mayor que" y "menor que" para comparar valores numéricos y obtener un resultado booleano verdadero o falso."

En conclusión

En este capítulo exploramos la lógica booleana y los operadores lógicos en Lua, específicamente el operador relacional de igualdad y de desigualdad, así como los operadores lógicos "mayor que" y "menor que". Estos operadores son fundamentales para la programación en Lua, ya que nos permiten comparar y evaluar los valores de verdad y realizar acciones basadas en estos resultados.

Valores de verdad por ausencia de *nil*

Es hora de conocer uno de los conceptos más novedosos y particulares de Lua como lenguaje de programación: los valores de verdad basados en la ausencia de nil.

¿Suena confuso? No te preocupes, no necesitas entenderlo completamente en este momento. Simplemente debes saber que podemos determinar si un valor existe o no en nuestro programa usando valores de verdad. Esto se utiliza principalmente en la ejecución de condicionales en diferentes

escenarios que implican la ejecución de código bajo ciertas condiciones. Aunque todavía no hemos abordado este tema en profundidad, lo veremos más adelante mientras avanzamos en el libro. Por ahora, es importante que tengas en cuenta que esto será una pre-introducción al estudio de los condicionales en Lua, que es un tema divertido e interesante de explorar.

Antes de que surja algún otro concepto nuevo, vamos a ver una explicación breve de qué son las condicionales en Lua.

Antes de continuar: ¿Qué es un condicional?

De manera sencilla, un condicional es un bloque de código que se ejecutará solo cuando se cumpla una determinada condición. Profundizaremos más en este tema en las secciones posteriores del libro.

Ilustración 31- Estructura básica de un condicional en Lua

Ahora que tenemos una idea superficial de lo que es un condicional, podemos ver cómo se puede aplicar el concepto de valor de verdad basado en la ausencia de nil. Como recordarás, cuando estudiábamos los tipos de datos, dijimos que el programa devolverá un valor de verdad positivo (true) cuando detecte que una variable o una cantidad de datos no es nil, es decir, no está vacía. Podemos verlo como si el programa se hiciera la pregunta: ¿Hay algún tipo de datos en este espacio de memoria? Veamos un ejemplo escrito como bloque de código para entender esto mejor:

```
1  x = 17
2  if x then
3       print(“La variable no es nula”)
4  end
```

Al ejecutar este bloque de código, obtendríamos la siguiente salida en la terminal de ejecución:

```
Salida  La variable no es nula
```

Veamos otro ejemplo para comprender mejor el concepto. En este caso, usaremos el mismo código, solo cambiaremos el valor dentro de la variable x.

```
1  x = nil
2  if x then
3       print(“La variable no es nula”)
4  End
```

Al ejecutar este bloque de código, obtendríamos la siguiente salida en la terminal de ejecución:

```
Salida
```

Como puedes ver, la salida es completamente vacía. Esto se debe a que el valor es nil, por lo que el programa no encontró ningún tipo de dato relevante dentro de nuestro programa para ejecutar el bloque de código dentro del condicional.

Veamos una explicación detallada de lo que ocurre en nuestro programa para comprenderlo completamente. Primero, creamos una variable llamada x, que será comparada y analizada más tarde para ver si hay algún tipo de datos dentro de ella.

Como probablemente hayas notado, nuestros dos ejemplos manejan casi el mismo código, pero para entender y estudiar más a fondo este concepto, debemos prestar atención a una pequeña diferencia en nuestro código, que naturalmente está en la variable y el tipo de datos que se almacenan en ella.

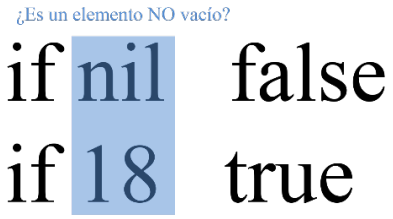


Ilustración 32 - Valor de verdad por ausencia de nil en condicionales

Analicemos el primer caso. En esta ocasión, la variable tiene un valor diferente de nil, es decir, podemos decir que la variable existe y no está vacía. Sin embargo, en el segundo caso que también estamos estudiando, no ocurre lo mismo. Esto se debe a que asignamos a la variable nil, que es un tipo de dato vacío que no ocupa espacio en memoria. Por lo tanto, el programa no considera que la variable exista.

Es importante destacar que la simple existencia del valor en el primer ejemplo no significa necesariamente un valor booleano positivo true por sí solo. Esto solo se aplica dentro de los condicionales.

No podemos mostrar el valor de verdad que indica que este valor existe y es relevante. Para confirmar esto, podemos ejecutar el siguiente bloque de código:

```
1  x = 17
2  print(x == true)
```

Al ejecutar este bloque de código, obtendremos la siguiente salida:

```
Salida  false
```

Entonces, ¿por qué, al ejecutarlo dentro de un condicional, toma un valor de verdad y se ejecuta su bloque interno de código de condicional? Como mencionamos anteriormente, esto se debe a la particularidad de manejo de los condicionales en Lua. Es una clase de excepción en la ejecución de nuestro código, ya que de ninguna otra forma diferente a la evaluación para ejecutar un condicional podemos ver que se trate de un tipo de valor booleano.

Por lo tanto, podemos concluir que la estructura condicional juega un papel importante al momento de verificar si hay algún tipo de dato relevante dentro de una variable para poder ejecutar su bloque de código interno. En el primer ejemplo, el bloque de código se ejecuta debido a que la variable contiene

un tipo de dato diferente a un valor nulo. Sin embargo, en el segundo ejemplo, como le habíamos asignado un valor nulo, la estructura condicional no se aplica y no se ejecuta su bloque de código interno.

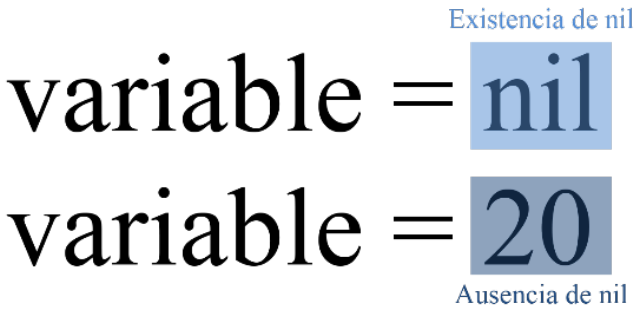


Ilustración 33- Identificar la ausencia de un valor nil

En resumen, podemos decir que el valor de verdad por ausencia de nil es un concepto importante en Lua y se utiliza principalmente en la ejecución de condicionales. La simple existencia de un valor no significa necesariamente un valor booleano positivo true, pero dentro de una estructura condicional, la existencia de un valor diferente a nil determina si se ejecuta o no su bloque de código interno.

Caso de estudio: Uso de los condicionales lógicos

En este caso de estudio, queremos fortalecer nuestras habilidades y conocimientos sobre el uso de los operadores lógicos que hemos aprendido hasta ahora. La idea es tener una buena base antes de enfrentarnos a resolver ejercicios más complejos. No te preocupes, los ejemplos que vamos a ver aquí no son muy difíciles. Con el tiempo, la complejidad de los programas irá aumentando.

Nuestra tarea como programadores es sencilla: debemos desarrollar un programa que compare información personal de dos usuarios y nos diga si tienen el mismo nombre, si uno es mayor que el otro, si nacieron en el mismo año y si tienen el mismo color favorito. El usuario deberá proporcionar la información de los dos usuarios a través del teclado y almacenarla en variables. Luego, podremos realizar las operaciones necesarias en el bloque de código de nuestro programa.

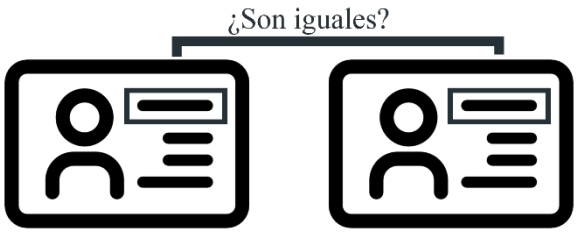


Ilustración 34 - '"Comparar diferentes usuarios frente a su información personal"

Una vez entendido esto, podemos empezar a desarrollar nuestro programa. En primer lugar, es importante crear una bienvenida amigable para el usuario de nuestro programa. Podemos hacer esto

utilizando la función print() o io.write() si es necesario. El siguiente bloque de código muestra un ejemplo:

```
1  print("Bienvenido!")
2  print("La función principal de este programa es comparar información brindada por dos personas.")
3  print("Asegúrate de escribir la información correctamente.")
```

Al ejecutar este bloque de código, obtendrás la siguiente bienvenida para el programa, que ayudará a entender la función principal del programa y a no sentirse confundido:

```
Salida  Bienvenido!
        La función principal de este programa es comparar información brindada por dos personas.
        Asegúrate de escribir la información correctamente.
```

Ahora que hemos dado la bienvenida al usuario, es momento de pedir los datos que necesitamos. El programa requiere que pidamos los siguientes datos:

Información requerida	
primerNombre	El primer nombre de la persona
primerApellido	El primer apellido de la persona
nacimiento	El año de nacimiento de la persona
edadActual	La edad actual de la persona
color	El color favorito de la persona

Es importante tener en cuenta que estamos trabajando con información de dos personas, por lo que debemos identificarlas de alguna manera, por ejemplo, primerNombre1 y primerNombre2 o nacimiento1 y nacimiento2. Ahora es momento de pedir los datos a través del teclado. Podemos hacer esto agregando el siguiente bloque de código a nuestro programa:

```
4  io.write("[Primera persona] Escribe el primer nombre: ")
5  primerNombre1 = io.read()
6  io.write("[Primera persona] Escribe el primer apellido: ")
7  primerApellido1 = io.read()
8  io.write("[Primera persona] Escribe el año de nacimiento: ")
```

Y repetimos el mismo proceso para la segunda persona:

```
9  io.write("[Segunda persona] Escribe el primer nombre: ")
10 primerNombre2 = io.read()
11 io.write("[Segunda persona] Escribe el primer apellido: ")
12 primerApellido2 = io.read()
13 io.write("[Segunda persona] Escribe el año de nacimiento: ")
```

Supongamos que estamos comparando a dos personas con la siguiente información:

	Primer sujeto	Segundo sujeto
Primer Nombre	Julián	Andrés
Primer Apellido	Castro	Castro
Año de nacimiento	2003	2004
Edad Actual	19	19
Color favorito	Naranja	Rojo

Entonces, al escribir esta información en nuestro programa, obtendríamos la siguiente salida en la terminal:

```
Salida  [Primera persona] Escribe el primer nombre: Julián
        [Primera persona] Escribe el primer apellido: Castro
        [Primera persona] Escribe el año de nacimiento: 2003
        [Primera persona] Escribe la edad actual: 19
        [Primera persona] Escribe el color favorito: Naranja
        [Segunda persona] Escribe el primer nombre: Andrés
        [Segunda persona] Escribe el primer apellido: Castro
        [Segunda persona] Escribe el año de nacimiento: 2004
        [Segunda persona] Escribe la edad actual: 19
        [Segunda persona] Escribe el color favorito: Rojo
```

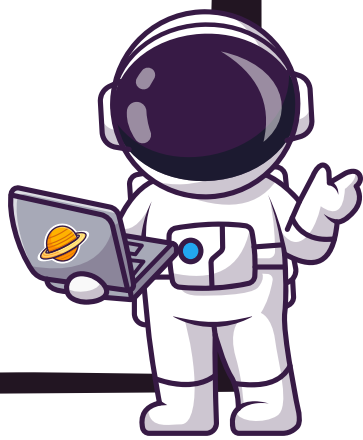
¡Y eso es todo! Ahora el operador puede descansar y dejar que el programa ejecute el resto de las operaciones necesarias para obtener el resultado deseado. Ahora debemos mostrar en la pantalla si alguna de las informaciones incluidas coincide. Podemos hacerlo con el siguiente bloque de código:

```
14  print("¿Ambos sujetos comparten el mismo nombre?: " .. tostring(primerNombre1 == primerNombre2))
15  print("¿Ambos sujetos comparten el mismo apellido?: " .. tostring(primerApellido1 == primerApellido2))
16  print("¿Ambos sujetos comparten el mismo año de nacimiento?: " .. tostring(nacimiento1 == nacimiento2))
17  print("¿Ambos sujetos comparten la misma edad?: " .. tostring(edadActual1 == edadActual2))
18  print("¿Es el primer sujeto mayor al segundo sujeto?: " .. tostring(edadActual1 > edadActual2))
19  print("¿Es el segundo sujeto mayor al primer sujeto?: " .. tostring(edadActual1 < edadActual2))
20  print("¿Ambos sujetos comparten el mismo color favorito?: " .. tostring(color1 == color2))
```

Con este ejemplo podrás tener una idea más clara de cómo puedes hacer uso de lenguajes de programación como Lua para realizar operaciones de comparación y visualización de resultados en la pantalla. Recuerda que este es solo un ejemplo básico y que existen muchas formas de mejorarlo y ajustarlo a tus necesidades específicas, por lo que es importante seguir aprendiendo.

En conclusión

Los valores de verdad basados en la ausencia de nil son un concepto importante en Lua que se utiliza principalmente en la ejecución de condicionales. Es importante recordar que la existencia de un valor no implica necesariamente un valor booleano positivo true, sino que solo se aplica dentro de las estructuras condicionales. En el caso de estudio presentado, hemos visto cómo podemos utilizar los operadores lógicos para comparar información personal de dos usuarios y determinar si comparten el mismo nombre, año de nacimiento, edad actual y color favorito. Este es solo un ejemplo básico que puede ser mejorado y ajustado a las necesidades específicas de cada programador.



Ejercicios de práctica	
1	Desarrollar un programa fácil de usar que te muestre si dos números son iguales o no. Para hacer esto, siga estos sencillos pasos: 1 Pídele al usuario que introduzca dos números a través del teclado. Asegúrate de hacerlo de manera clara y amigable, por ejemplo: "Por favor, ingresa el primer número". 2 Una vez que tengas ambos números, muestra en pantalla un mensaje que diga: "¿Son ambos números iguales?:", seguido del resultado de la comparación. 3 Por último, muestra un mensaje de despedida amigable, como: "¡Eso es todo! Gracias por usar este programa".
2	En este ejercicio, nuestro objetivo es crear un programa que muestre en pantalla alternadamente los valores "true" y "false", siguiendo ciertas condiciones específicas. La salida que esperamos obtener es la siguiente: Salida false true false true Para lograrlo, debemos cumplir con las siguientes restricciones: 1 Solo podemos utilizar dos variables numéricas. 2 No podemos imprimir directamente los valores "true" o "false". Por ejemplo, no debemos hacer algo como: "print(true); print(false)". 3 Solo podemos utilizar un único operador lógico para generar los valores "true" y "false". 4 Está permitido cambiar los valores dentro de las variables, pero no su tipo de dato. 5 El programa debe contener un máximo de 9 líneas de código.
3	Nuestro objetivo es crear un programa que verifique si una cadena de texto introducida por el usuario es igual a "Manzana". Para ello, debemos asegurarnos de que la presentación en pantalla sea clara y concisa. Un ejemplo de la salida en la terminal podría ser: Salida ¿La cadena de texto es “Manzana”?: true Además, para obtener la cadena de texto que se va a comparar, debemos hacer que el usuario la introduzca a través del teclado, usando la función io.read().

4	Completar las partes faltantes del código para que este genere la salida esperada del programa. La salida esperada es la siguiente: Salida El valor de verdad es true Y el código incompleto es el siguiente: 1 num = 17 2 print(“El de verdad es ” .. to (num >))

Capítulo 5

Operadores lógicos, algoritmos y condicionales en Lua

Es hora de profundizar en los operadores lógicos exclusivos para los valores de tipo booleano en Lua. Ya hemos visto que cada tipo de datos tiene una serie de operadores correspondientes que se pueden aplicar a esos mismos tipos de datos. Con eso en mente, es hora de ver también los operadores de tipo booleano, así como hemos visto diferentes tipos de operadores y operaciones que podemos realizar con operadores numéricos o de cadenas de texto.

Estos temas relacionados con las operaciones booleanas incluyen algunos conceptos matemáticos que pueden ser algo complicados para aquellos que no tienen una amplia comprensión de la lógica antes de leer este libro. Sin embargo, una de las metas de este libro es que sea accesible para cualquier persona, independientemente de su formación o experiencia previa. Aprender a programar en Lua puede ser fácil de entender, incluso si no tienes una formación teórica en los temas que veremos.

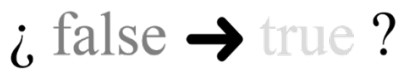


Ilustración 35 - "Alguna operación que pueda generar diferentes valores de verdad"

En resumen, los operadores lógicos booleanos son una serie de operadores que se pueden aplicar a valores booleanos para calcular su valor de verdad basado en otros valores booleanos.

A continuación, se presenta una lista de los posibles tipos de operadores y operaciones que podemos realizar con valores de tipo booleano en Lua:

Operación	Operador	Descripción
Conjunción	and	Verifica si todos los elementos de la comparación de la operación tienen un valor lógico positivo.
Disyunción	or	Verifica si al menos un elemento de la comparación de la operación tiene un valor positivo de verdad.
Negación	not	Invierte el valor de verdad del elemento que forma parte de la operación lógica.

Estos son los 3 operadores lógicos principales en Lua, pero es posible realizar cualquier otra operación en Lua mediante operaciones equivalentes a sus respectivas operaciones en la lógica booleana. Estos operadores lógicos son útiles para evaluar diferentes valores de verdad y definir la validez de las sentencias en una expresión booleana.

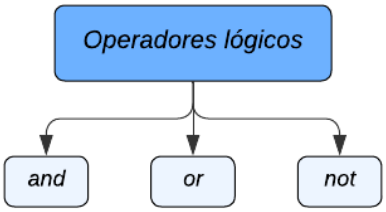


Ilustración 36 - Operadores lógicos en Lua

Uso del operador lógico and en Lua.

Vamos a estudiar el uso de uno de los operadores más comunes en la programación: el operador de conjunción "and". Este operador se utiliza para verificar si ambas partes de una operación lógica son verdaderas y se representa con el símbolo "and" en muchos lenguajes de programación.

El siguiente gráfico te puede ayudar a entender el concepto de operador de conjunción:

Ilustración 37- Uso del operador and

Antes de ver algunos ejemplos de cómo usar este operador en un programa, vamos a explicar las partes que conforman una operación de conjunción:

Ilustración 38 - Partes de una operación and

Además de entender las partes que conforman una operación de conjunción, es importante conocer los diferentes resultados que puede generar. La siguiente tabla muestra los posibles resultados:

Primer valor	Operador	Segundo valor	Salida
true	and	true	true
true	and	false	false
false	and	true	false
false	and	false	false

Como puedes ver, la operación de conjunción sólo generará una salida verdadera (true) si ambos valores son verdaderos (true). Si alguno de los dos valores es falso (false), entonces la salida será false.

Veamos un ejemplo para asegurarnos de entender cómo funciona el operador de conjunción "and" en código.

```
1  bool1=true
2  bool2=false
3  print(bool1 and bool2)
```

Al ejecutar este bloque de código, obtendríamos la siguiente salida en la terminal:

```
Salida  false
```

Como puedes ver, al hacer la operación de conjunción entre bool1 y bool2, donde bool1 es verdadero (true) y bool2 es falso (false), la salida es falso (false).

Veamos otro ejemplo de código para entender mejor cómo funciona el operador de conjunción "and".

En este caso, supongamos que tenemos un programa que verifica si dos personas tienen la misma edad y el mismo nombre. Si ambas condiciones se cumplen, entonces el programa mostrará en pantalla un mensaje afirmativo. Aquí está el código:

```
1  edadPersona1 = 18
2  edadPersona2 = 18
3  nombrePersona1 = "Daniel"
4  nombrePersona2 = "Alex"
5  mismaEdad = edadPersona1 == edadPersona2
6  mismoNombre = nombrePersona1 == nombrePersona2
7  print("¿Tienen las dos personas tanto el mismo nombre como la misma edad?: " .. tostring(mismoNombre and
   mismaEdad))
```

Al ejecutar este bloque de código, obtendríamos la siguiente salida en la terminal:

```
Salida  ¿Tienen las dos personas tanto el mismo nombre como la misma edad?: false
```

El programa es conceptualmente fácil de entender. Sin embargo, veamos una breve explicación de lo que sucede dentro de nuestro código para asegurarnos de comprenderlo completamente:

```
Explicación del código
```

1	En primer lugar, el programa crea cuatro variables. Dos de ellas son números y las otras dos son cadenas de texto. Cada variable almacena información personal de dos personas, incluyendo su nombre y edad.
2	En segundo lugar, el programa usa dos variables booleanas que son el resultado de comparar las edades y nombres de ambas personas. Estas variables contienen el valor verdadero si tanto la edad como el nombre coinciden.
3	En tercer lugar, el programa muestra el resultado de usar el operador lógico "and" en las dos variables booleanas mencionadas anteriormente. Este resultado indica si tanto la edad como el nombre de las dos personas coinciden.
4	Finalmente, vemos que la salida del programa es "falso". Esto es porque, aunque ambas personas tienen la misma edad, no tienen el mismo nombre. Como resultado, ninguna de las dos variables booleanas contiene un valor verdadero, lo que da como resultado una salida falsa.

¡Y eso es todo! Ahora podemos decir que hemos comprendido el uso del operador lógico de conjunción en Lua, el operador "and". Cuando estudiemos temas más avanzados, como condicionales y bucles, veremos que este operador lógico es muy útil y necesario para el correcto funcionamiento de diferentes programas.

Ahora que hemos comprendido este operador lógico, es hora de explorar otros operadores lógicos y aprender a usarlos correctamente en nuestros programas.

Uso del operador lógico or en Lua

Es hora de aprender sobre nuestro segundo operador lógico: el operador "or". Al igual que el operador anterior, este también está compuesto por dos partes.

También es posible usar más de dos partes en una operación, pero esto requeriría de más de un operador para crear una cadena entre cada uno de los valores almacenados en nuestros elementos del operador lógico.

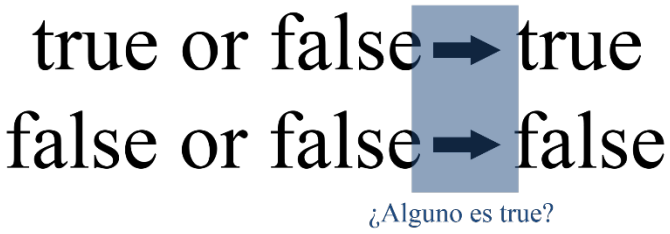


Ilustración 39- Uso del operador or

Veamos el uso del operador "or". Este operador es diferente al operador de conjunción que estudiamos antes. Con el operador "or", no es necesario que ambos elementos de la operación lógica tengan un valor de verdad positivo para obtener una salida positiva. Con solo uno de los elementos siendo verdadero, se generará una salida booleana verdadera. Solo si ambos elementos son falsos, la salida será booleana negativa. En cualquier otro caso, la salida será booleana verdadera.

Podemos entender mejor esta idea con la siguiente tabla de verdad que muestra las diferentes salidas posibles y los elementos que las causan.

Primer valor	Operador	Segundo valor	Salida
true	or	true	true
true	or	false	true
false	or	true	true
false	or	false	false

Como mencionamos antes, la salida será un valor booleano verdadero cada vez que se encuentre al menos un elemento verdadero, incluso si el otro elemento es falso.

¿ [ELEMENTO] ó [ELEMENTO] ?

Ilustración 40- Interpretación del operador or

Veamos un pequeño ejemplo para confirmar cómo funciona el operador "or" en un bloque de código.

Este es un ejemplo de código que muestra cómo usar el operador "or" en un programa:

```
1  bool1= false
2  bool2= true
3  print(bool1 or bool1); print(bool1 or bool2)
```

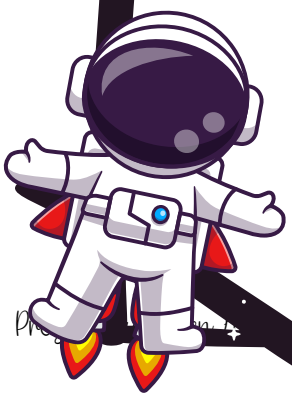
Al ejecutar este código, obtendríamos la siguiente salida:

```
Salida  false
        true
```

Como mencionamos antes, este código es bastante simple y se puede entender fácilmente a partir de la tabla de verdad que hicimos anteriormente.

En conclusión

Los operadores lógicos en Lua son una herramienta poderosa para evaluar diferentes valores de verdad y definir la validez de las sentencias en una expresión booleana. Aunque los conceptos matemáticos pueden ser complicados, este libro ha sido diseñado para ser accesible para cualquier persona, independientemente de su formación o experiencia previa. El operador de conjunción "and" y el operador de disyunción "or" son los dos principales operadores lógicos que se pueden aplicar a valores booleanos en Lua. Es importante conocer las diferentes salidas posibles de cada operación y cómo aplicarlas en diferentes programas para obtener resultados precisos y eficientes.



Algoritmos y condicionales en Lua

Es hora de que aprendamos sobre los condicionales y cómo desarrollar diferentes algoritmos para que nuestro programa funcione correctamente, dependiendo de nuestras necesidades.

En esta sección, empezaremos a profundizar en el desarrollo de programas más complejos que requieren más atención a su ejecución. Los algoritmos también nos ayudarán a crear diferentes situaciones y comportamientos para que nuestro programa tome acción ante condiciones específicas que plasmemos en nuestro código.

Paso 1 ➜ Paso 2 ➜ FINAL

En resumen, un algoritmo es una serie de pasos que nuestro programa debe seguir para lograr un objetivo. También podemos ver un algoritmo como una secuencia de pasos lineales que nuestro programa debe ejecutar para realizar una tarea. Puede parecer que estos conceptos suenen complicados, pero no hay que preocuparse. Nos aseguraremos de estudiar y explicar cada uno de estos temas de una manera clara y fácil de entender, para que eventualmente podamos desarrollar nuestros propios algoritmos y entender otros algoritmos desarrollados por otras personas sin confusión.

Para empezar a comprender estos nuevos conceptos, estudiaremos algo conocido como estructura en el mundo de la programación. En pocas palabras, una estructura en el mundo de la programación es simplemente una forma que tenemos dentro de nuestro bloque de código que contiene un bloque de código interno. Este bloque de código generalmente se ejecuta cuando se cumplen ciertas especificaciones o condiciones, de lo contrario, es probable que no se ejecute.

En el mundo de la programación, y en el lenguaje de programación Lua en particular, hay muchas estructuras, pero hay algunas que son universales o muy comunes entre la gran variedad de otros lenguajes de programación. Estas estructuras populares o comunes comparten una misma idea y configuración entre los diferentes lenguajes de programación, pero esto también dependerá de la sintaxis del lenguaje en particular.

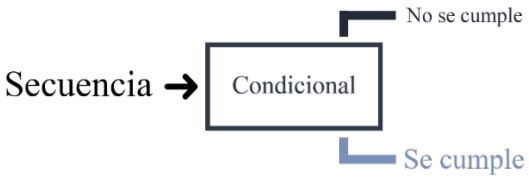


Ilustración 41 - Estructura general de un condicional

A continuación, en la siguiente tabla podemos ver algunas de las diferentes estructuras que puedes encontrar en Lua y en muchos otros lenguajes de programación debido a su popularidad y utilidad.

Estructura	Sintaxis	Descripción
Condicional simple	if [ ] then	Este bloque de código se ejecutará solo cuando se cumpla la condición indicada en [ ].
Condicional elseif	elseif [ ] then	Esta estructura es similar al condicional simple, pero es un equivalente a usar "else if [ ] then".

Condicional final else	else	Este bloque se ejecutará cuando ninguna otra condición se haya cumplido. Es la contra ejecución de un condicional simple.

Estas estructuras son comunes en muchos lenguajes de programación y serán las primeras que estudiaremos en las secciones siguientes de este libro. Debemos prepararnos porque es el momento de aprender una de las partes más interesantes y de nivel intermedio de los conceptos teóricos que podemos aprender fácilmente usando este libro en este lenguaje de programación.

El condicional if en Lua

Es hora de echar un vistazo a la primera estructura condicional que veremos como ejemplo para poder ejecutar diferentes algoritmos dentro de nuestro código. Esta estructura es la más simple de todas: el condicional simple if.

En pocas palabras, la función de esta estructura es ejecutar un bloque de código interno después de que se haya cumplido una condición específica definida dentro de la estructura de nuestro programa. Esto nos permite crear diferentes escenarios y secuencias de pasos que deben ser realizados bajo ciertas condiciones y especificaciones de nuestro programa.

A continuación, podemos ver una ilustración de cómo se ve exactamente este tipo de estructura:

Ilustración 42 - Estructura de condicional if

Así que podemos identificar fácilmente cómo encontrar un condicional simple en Lua dentro de diferentes bloques de código. Ahora veamos la estructura que se utiliza para una ejecución correcta:

Estructura if [CONDICIONAL] then [BLOQUE] end	
if	Es el primer elemento que inicializa la estructura. Le indicamos a la computadora que se trata del inicio de un condicional simple.
[CONDICIONAL]	Es donde va el condicional del programa. Es un valor de tipo booleano, puede ser positivo o negativo, dependiendo del caso. Normalmente se usa una operación que devuelva un valor de verdad, pero también es posible escribir el valor de verdad directamente.
then	Es una palabra reservada que indica el inicio del bloque de código interno de la estructura. Indica que a partir de ese punto, se empezará a escribir el código que se ejecutará en el caso de que la condición sea cumplida.
[BLOQUE]	Es el bloque de código que se ejecutará una vez que la condición haya sido cumplida satisfactoriamente. Aquí es donde escribiremos todo lo que queremos que ejecute el programa.
end	es una palabra reservada que indica el final del bloque de código interno de la estructura del condicional simple. Una vez que el programa encuentre esta palabra reservada, detendrá la ejecución del bloque interno de la estructura.

Utilizar condicionales en nuestros programas tiene muchas aplicaciones, y las posibilidades de lo que podemos desarrollar con ellos son infinitas. Ahora que entiendes todo esto, es hora de empezar a programar algunos bloques de código que te sirvan como ejemplo de lo que puedes hacer con condicionales simples en tus programas.

Este es un primer ejemplo de código que te ayudará a entender cómo funcionan los condicionales en Lua. Aquí te muestro el código:

```
1  nombre=”Ernesto”
2  if nombre == ”Ernesto” then
3      print(“Hola, Ernesto!”)
4  end
```

Al ejecutar este código, obtendrás la siguiente salida:

```
Salida  Hola, Ernesto!
```

Estructura condicional elseif en Lua

La vida está llena de decisiones y actuamos en base a las condiciones de nuestro entorno. En la programación, esto también es cierto, y hemos visto cómo funciona el condicional en Lua, pero ¿qué pasa si queremos hacer escenarios más complejos? Ahí es donde entra en juego la estructura elseif de Lua. Nos permite crear escenarios de condiciones más complejas. Por ejemplo, "Si llueve, llevaré el paraguas, de lo contrario lo dejaré en casa". La estructura elseif es como un plan B en la ejecución de nuestro código y nos será muy útil para crear programas más complejos usando Lua.

Entender el concepto de elseif no es difícil. Podemos pensar en él como una alternativa de ejecución en caso de que alguna condición no se cumpla positivamente. El siguiente gráfico puede ayudarnos a comprenderlo mejor:

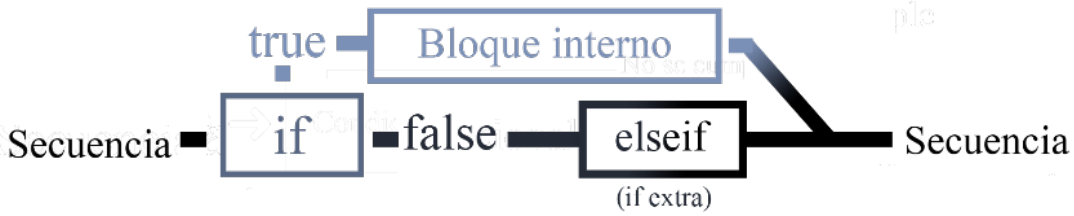


Ilustración 43 - Estructura elseif

Ahora, podemos profundizar en el funcionamiento de la estructura elseif analizando sus diferentes partes:

Estructura [CONDICIONAL SIMPLE] else [BLOQUE] end	
[CONDICIONAL SIMPLE]	Es una estructura de condicional simple, también conocida como estructura if. Una estructura else no sería posible sin una estructura if antes de ella.
else	Es la palabra reservada que marca el inicio de la estructura condicional else en nuestro programa. A diferencia de la estructura if, no contiene la palabra reservada "then", por lo que la palabra "else" cumple la función de marcar el inicio del bloque de ejecución.
[BLOQUE]	Es el bloque de código interno de la estructura, es decir, las órdenes que debe ejecutar nuestro programa cuando se inicia la ejecución de esta estructura.
end	Marca el final de la ejecución de nuestro bloque de código.

Con esto, podemos concluir la explicación de las partes que conforman la estructura elseif en Lua. Veamos un ejemplo práctico de cómo usar la estructura elseif en Lua:

```
1  bool=false
2  if (bool) then
3       print(“Bool es true”)
4  Else
5       print(“Bool es false”)
6  End
```

Al ejecutar este bloque de código, obtendremos la siguiente salida:

```
Salida  Bool es false!
```

Explicación del código	
1	Declaramos una variable de tipo booleano llamada "bool" y le asignamos el valor false.
2	A continuación, usamos una estructura condicional simple, donde la condición es el valor de la variable bool.
3	Dentro de la estructura condicional, si la condición se cumple (bool es true), mostramos en pantalla la cadena de texto "Bool es true”.
4	Si la condición no se cumple, entra en acción la estructura else, que marca el escenario en el que la condición no se cumple.
5	Dentro de la estructura else, mostramos en pantalla la cadena de texto "Bool es false".
6	Finalmente, podemos ver que la salida de nuestro programa es "Bool es false", ya que el valor de bool es false.

En resumen, en este ejemplo vemos cómo se utiliza la estructura elseif para marcar un escenario alternativo cuando la condición no se cumple en la estructura if.

Es importante tener en cuenta que muchos de los ejemplos a partir de ahora incluirán diferentes tipos de condicionales. Por lo tanto, es crucial tener una buena comprensión de este tema para evitar

dificultades futuras al desarrollar y comprender ejemplos, actividades o estudios de caso. Por eso, desarrollaremos más ejemplos para fortalecer aún más nuestras habilidades y conocimiento sobre estos temas.

Veamos un ejemplo más con la estructura elseif en Lua:

```
1  edadJuan = 18
2  edadPedro = io.read()
3  if edadJuan < edadPedro then
4       print(“¡Juan es menor que Pedro!”)
5  else
6       print(“¡Juan es mayor o igual a Pedro!”)
7  end
```

Si suponemos que el usuario escribe que la edad de Pedro es 20, la salida sería:

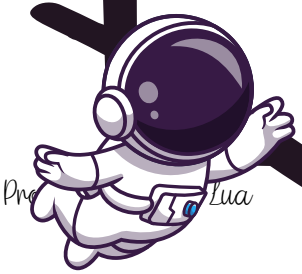
```
Salida  ¡Juan es menor que Pedro!
```

Explicación del código	
1	Declaramos dos variables numéricas, una con la edad de Juan (18) y otra con la edad de Pedro (que será ingresada posteriormente por el usuario).
2	Usamos la estructura condicional if para definir qué acción se ejecutará dependiendo de la comparación entre las edades de Juan y Pedro. Si la edad de Juan es menor que la de Pedro, se imprimirá en pantalla "¡Juan es menor que Pedro!". De lo contrario, se imprimirá en pantalla "¡Juan es mayor o igual a Pedro!".
3	La salida final muestra que Juan es menor que Pedro, cumpliendo con lo que se esperaba en base a los valores asignados a cada una de las variables.

Con estos ejemplos, podemos concluir nuestro estudio sobre el uso de condicionales en Lua.

En conclusión

En este capítulo aprendimos sobre la importancia de los condicionales y los algoritmos en la programación en Lua. Vimos cómo desarrollar diferentes algoritmos para que nuestro programa funcione correctamente, dependiendo de nuestras necesidades. Además, estudiamos las diferentes estructuras de condicionales en Lua, como el condicional simple "if" y la estructura condicional "elseif". Analizando detalladamente sus diferentes partes y cómo se utilizan en la creación de programas más complejos.



Condicionales extensos usando la estructura elseif

Antes de profundizar en el uso de las diferentes estructuras de condicionales en Lua, hicimos una tabla resumen de los tipos de condicionales que podemos encontrar. Aquí está la tabla de nuevo para recordar:

Estructura	Sintaxis	Descripción
Condicional simple	if [ ] then	Esta estructura ejecuta el bloque de código dentro de ella solo cuando se cumple la condición indicada en [ ].
Condicional elseif	elseif [ ] then	Esta estructura es similar al condicional simple, pero es equivalente a usar "else if [ ] then".
Condicional final else	else	Esta estructura se usa para definir un caso hipotético que se ejecutará si ninguna otra condición se cumple. Es la opción contraria a un condicional simple.

Es importante prestar atención al segundo elemento de la tabla: la estructura de extensión condicional elseif. Ya mencionamos que es equivalente a usar "else if [ ] then". Analicemos el siguiente gráfico:

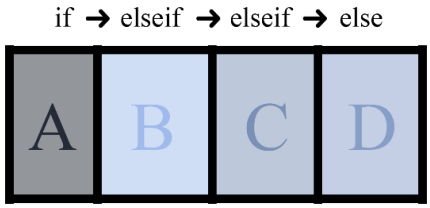


Ilustración 44- Diferentes posibles casos en estructuras elseif

Como podemos ver, el uso de esta extensión de condición es esencial cuando queremos plantear diferentes casos de ejecución o situaciones en relación con una condición. También podemos verlo como una manera de tener varias soluciones posibles en nuestro programa en función de los resultados esperados y su acción correspondiente.

En otras palabras, en lugar de tener solo un plan B usando la estructura "else", ahora podemos tener muchos más planes (B, C, D, ...) dependiendo de nuestras necesidades. Veamos el siguiente bloque de código para entender mejor lo que estamos tratando:

```
1   numero = io.read()
2   if numero == 1 then
3       print(“¡Primer caso!”)
4   elseif numero == 2 then
5       print(“¡Segundo caso!”)
6   elseif numero == 3 then
7       print(“¡Tercer caso!”)
8   else
9       print(“ERROR”)
10  end
```

Supongamos que el operador escribió el número 20, en ese caso, obtendríamos la siguiente salida:

```
Salida    ERROR
```

En este ejemplo, se activó el caso en el que el valor escrito no corresponde a ninguno de los escenarios planteados en el código.

Otro ejemplo de uso de la estructura elseif:

```
1   nombreCliente = “Erick”
2   bebida = io.read()
3   if bebida == “Agua” then
4       print(“¡Hola ” .. nombreCliente .. “, pronto serviremos tú “ .. bebida .. “!”)
5   elseif bebida == “Limonada” then
6       print(“¡Hola ” .. nombreCliente .. “, pronto serviremos tú “ .. bebida .. “!”)
7   elseif bebida == “Jugo” then
8       print(“¡Hola ” .. nombreCliente .. “, pronto serviremos tú “ .. bebida .. “!”)
9   else
10      print(“Uh oh, parece ser que no tenemos esa bebida”)
11  end
```

En este ejemplo, podemos ver diferentes casos y escenarios posibles en función de la entrada del operador del programa. Por ejemplo, si el operador escribió "Jugo", obtendríamos la siguiente salida:

```
Salida    ¡Hola Erick, pronto serviremos tú Jugo!
```

Si el operador escribió "Agua", obtendríamos:

```
Salida    ¡Hola Erick, pronto serviremos tú Agua!
```

Pero si escribió "Malteada", obtendríamos:

```
Salida    Uh oh, parece ser que no tenemos esa bebida
```

Podemos decir que usar múltiples escenarios definidos con la estructura elseif es muy útil para marcar diferentes opciones en nuestros programas en función de los datos con los que trabajemos y la acción necesaria para cada uno de ellos.

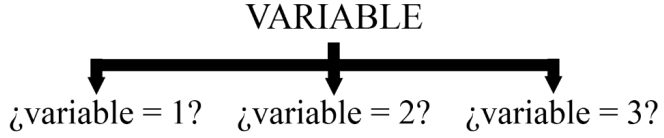


Ilustración 45 - Uso de diferentes escenarios en base al valor de una variable

Veamos un pequeño caso de estudio para entender mejor el tema. Este conocimiento nos abre muchas posibilidades para crear programas más complejos que los que veníamos haciendo antes.

Caso de estudio del uso de condicionales en Lua

Estudiemos un caso práctico sobre el uso de condicionales en Lua. En este ejemplo, veremos cómo podemos aplicar estos conceptos para crear programas más efectivos y laborales. El objetivo es mejorar nuestra habilidad y conocimiento como programadores de Lua.

Imaginemos que somos programadores para un asadero de pollo. Nuestra tarea es desarrollar un programa que muestre la factura de compra de un usuario en la salida de la terminal. La factura debe incluir la información del usuario y la compra realizada, y debe presentarse de manera ordenada y agradable en la pantalla.

Empecemos con una introducción amigable para el usuario de nuestro programa. Esto permitirá al usuario entender mejor lo que está haciendo y qué tipo de información debe ingresar en el programa. Aquí está el bloque de código para la introducción y bienvenida al programa:

```
1  print("¡Hola! Bienvenido a nuestro programa.")
2  print("Este programa generará facturas imprimibles basadas en los productos solicitados por el usuario.")
3  print("Por favor, ingrese la información personal del usuario que ha realizado el pedido.")
4  print("Después, escriba la información sobre el tipo de producto y la cantidad solicitados.")
```

Y aquí está la salida esperada en la terminal de nuestro programa:

```
Salida  ¡Hola! Bienvenido a nuestro programa.
        Este programa generará facturas imprimibles basadas en los productos solicitados por el usuario.
        Por favor, ingrese la información personal del usuario que ha realizado el pedido.
        Después, escriba la información sobre el tipo de producto y la cantidad solicitados.
```

Es hora de pedir al usuario que ingrese los datos necesarios para el correcto funcionamiento del programa. Aquí está el bloque de código para pedir la información del usuario:

```
5  print("Por favor, escriba el nombre del comprador:")
6  nombreComprador = io.read()
7  print("Escriba el nombre del producto solicitado:")
8  nombreProducto = io.read()
9  print("Por último, escriba la cantidad de productos solicitados:")
10 cantidadProducto = io.read()
```

De esta manera, podemos pedir al usuario que ingrese la información de la compra de una manera clara y concisa. Supongamos que tenemos la siguiente información de un escenario hipotético:

Sujeto	
Nombre completo	Ernesto Hernández

Orden pedida	Muslo de pollo
Cantidad	13

Si ingresamos estos datos en el programa, podemos esperar la siguiente salida en la terminal de ejecución:

Salida	Por favor, escriba el nombre del comprador: Ernesto Hernández Escriba el nombre del producto solicitado: Muslo de pollo Por último, escriba la cantidad de productos solicitados: 13

Ahora que tenemos los datos del usuario almacenados en diferentes variables, es hora de empezar a realizar los procesos necesarios para generar la salida esperada de nuestro programa.

Primero, consideremos el menú de productos que ofrece el asadero. Aquí está el menú:

Menú del asadero	
Elemento	Precio por unidad
Almuerzo personal	$14,000
Combo muslos de pollo	$36,000
Pechuga de pollo	$8,000
Muslo de pollo	$6,000
Alas de pollo	$5,000
Bebidas	$3,000

Para definir los precios de cada producto en nuestras variables, podemos usar el siguiente bloque de código:

```
11  if nombreProducto == “Almuerzo personal” then precioProducto = 14000
12  elseif nombreProducto == “Combo muslos de pollo” then precioProducto = 36000
13  elseif nombreProducto == “Pechuga de pollo” then precioProducto = 8000
14  elseif nombreProducto == “Muslo de pollo” then precioProducto = 6000
15  elseif nombreProducto == “Alas de pollo” then precioProducto = 5000
16  elseif nombreProducto == “Bebidas” then precioProducto = 3000
17  else precioProducto = 0 end
```

De esta manera, almacenamos en una variable el precio correspondiente al producto solicitado por el comprador, lo que facilitará los cálculos posteriores en el programa.

Excelente, es hora de calcular el valor total que debe pagar el usuario. Podemos hacerlo con la siguiente línea de código:

```
18  total = precioProducto * cantidadProducto
```

Y con esto, podemos decir que hemos terminado con la parte de cálculo de nuestro programa. Por último, solo nos queda mostrar en pantalla la factura generada.

Aquí está el bloque de código para hacer lo que mencionamos anteriormente:

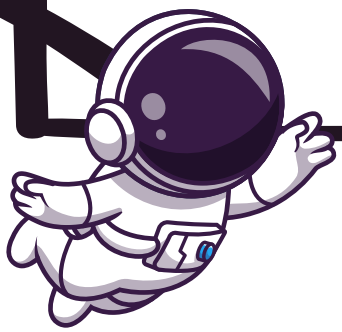
```
19  print("A continuación, se muestra la factura generada.")
20  print("Señor " .. nombreComprador .. ", a continuación, se muestra el total que debe de pagar:")
21  print(" 1) " .. tostring(cantidadProducto) .. " unidades de " .. nombreProducto)
22  print("TOTAL A PAGAR: $" .. tostring(total))
23  print("¡Muchas gracias por su compra!")
```

Con esto, concluimos nuestro programa, cuya función es mostrar una factura en base a un pedido realizado por un usuario en el asadero. La salida generada por nuestro programa sería:

```
Salida  ¡Hola! Bienvenido a nuestro programa.
        Este programa generará facturas imprimibles basadas en los productos solicitados por el usuario.
        Por favor, ingrese la información personal del usuario que ha realizado el pedido.
        Después, escriba la información sobre el tipo de producto y la cantidad solicitados.
        Por favor, escriba el nombre del comprador:
        Ernesto Hernández
        Escriba el nombre del producto solicitado:
        Muslo de pollo
        Por último, escriba la cantidad de productos solicitados:
        13
        A continuación, se muestra la factura generada.
        Señor Ernesto Hernández, a continuación, se muestra el total que debe de pagar:
         1) 13 unidades de Muslo de pollo
        TOTAL A PAGAR: $78000
        ¡Muchas gracias por su compra!
```

En conclusión

En este capítulo aprendimos sobre el uso de la estructura elseif en Lua, que nos permite definir múltiples casos o escenarios en función de una condición. A través de varios ejemplos, pudimos ver cómo esta estructura es esencial para plantear diferentes soluciones en nuestro programa en función de los resultados esperados y su acción correspondiente. Además, exploramos un caso de estudio que nos permitió aplicar estos conceptos para crear un programa que genera una factura de compra en un asadero de pollo



Algoritmos y solución de problemas

Llegamos a una parte importante de la programación: Los algoritmos. En esta sección, aprenderemos cómo solucionar problemas a través de ellos. Además de ver código escrito en Lua, también exploraremos herramientas gráficas que nos ayudarán a comprender, leer e incluso crear soluciones a diferentes problemas.

Los algoritmos son la clave para solucionar problemas más complejos. A veces, cuando estamos programando para resolver problemas difíciles, es fácil perderse en la secuencia de ejecución de nuestro programa y no entender qué otros pasos debemos realizar. Por eso, es útil desarrollar algoritmos para solucionar este tipo de situaciones.

Hay muchas formas de definir, explicar y visualizar algoritmos de solución de problemas, pero una herramienta gráfica que es muy útil es el diagrama de flujo. Los diagramas de flujo tienen muchas aplicaciones en diferentes campos, no solo en la programación y la informática en general. Veamos un ejemplo de un diagrama de flujo para un programa sencillo:

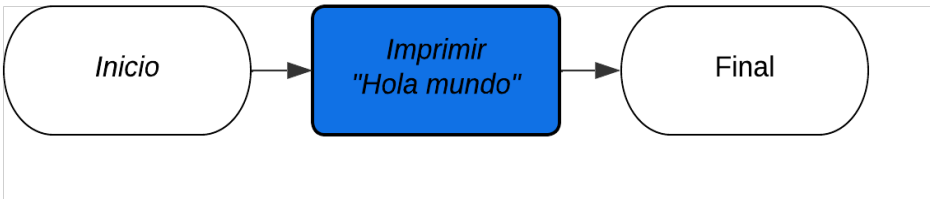


Ilustración 46 - Ejemplo de diagrama de flujo

Este tipo de representación de algoritmos se conoce como diagrama de flujo. Es una representación gráfica de un proceso o algoritmo. En la parte superior izquierda del diagrama, encontramos el inicio del algoritmo, y luego hay flechas que nos muestran el flujo de las decisiones que podrían ocurrir en el programa. Podemos ver cómo se desarrolla el programa paso a paso cuando se encuentra con ciertos datos.

Esta es solo una de las muchas formas en las que podemos representar un algoritmo. Nos facilita tener una idea clara y visual de lo que debemos hacer en lugar de detenernos a leer el código.

Hay muchas formas de presentar algoritmos para solucionar un problema, pero en esta sección nos centraremos en el uso de diagramas de flujo y pseudocódigo. Los diagramas de flujo tienen múltiples usos fuera del mundo de la programación. Por ejemplo, algunas personas los usan para describir el progreso paso a paso de proyectos, escribir ideas o planificar actividades. En la programación, los diagramas de flujo son utilizados para explicar y crear algoritmos para solucionar problemas.

Aunque no existe una regla para realizar diagramas de flujo para explicar algoritmos de solución de problemas en la programación, es importante seguir ciertos estándares. Cada lenguaje de programación puede variar en su estructura y funcionamiento, pero los conceptos básicos de la programación se aplican en todos ellos. Podemos llamar a estos procedimientos y clasificarlos en base a características comunes que comparten algunas declaraciones o bloques de código e información.

Pseudocódigos como representación de algoritmos de programación

Ahora vamos a ver otro método para presentar y explicar algoritmos de programación. Se llama pseudocódigo.

Como recordarás, al principio del libro hablamos sobre la diferencia entre los lenguajes de programación de alto y bajo nivel. Este concepto es crucial para entender qué es el pseudocódigo. Podemos definirlo como una representación de muy alto nivel que no se ajusta a ninguna estructura de lenguaje de programación en particular.

Al programar, debemos indicar a la computadora lo que queremos que haga a través de órdenes escritas en un lenguaje de programación determinado. El pseudocódigo nos permite liberarnos de esta limitación y explicar o esquematizar con mayor facilidad lo que debemos hacer. Además, el pseudocódigo suele ser fácil de entender para el lector y está diseñado para ser transcrito posteriormente a un lenguaje de programación. Algunos componentes comunes de los códigos en pseudocódigo incluyen "crear variable", "Inicio de función", "Fin de función", "Mostrar [elemento] en pantalla", entre otros.

El pseudocódigo no sigue un estándar y está diseñado para ser escrito, leído y entendido con facilidad, independientemente del lenguaje de programación o los principios léxicos que se deban seguir al

escribir código. Una de las principales ventajas del pseudocódigo es su facilidad para explicar algoritmos y solucionar problemas. Veamos algunos ejemplos de códigos escritos en pseudocódigo para comprender mejor este tema nuevo.

“Hola mundo” en pseudocódigo

Veamos cómo sería "Hola mundo" en pseudocódigo:

```
1   Iniciar programa
2   Mostrar en pantalla “Hola mundo”
3   Finalizar programa
```

Si ejecutamos este bloque de código en pseudocódigo, obtendremos la siguiente salida:

```
Salida   Hola mundo
```

Variables y otras operaciones en pseudocódigo

Veamos cómo serían las variables y otras operaciones en pseudocódigo:

```
1   Iniciar programa
2   Crear variables num1 = 14, num2 = 13, num3 = 5
3   Mostrar en pantalla num1 + (num2 / num3)
4   Finalizar programa
```

Si ejecutamos este bloque de código en pseudocódigo, obtendríamos la siguiente salida:

```
Salida   20
```

Condicionales en pseudocódigo

```
1   Iniciar programa
2   Crear variable texto = “Hola”
3   Si texto == “Hola” entonces
4       Imprimir “Todo está bien”
5   De lo contrario entonces
6       Imprimir “Algo ha salido mal”
7   Fin condicional
8   Finalizar programa
```

Salida	Todo está bien

Variantes de pseudocódigo

Como mencionamos, el pseudocódigo no sigue un estándar específico, por lo que también podemos escribir el ejemplo anterior de la siguiente manera:

```
1  INICIO
2  texto = “Hola”
3  Si texto == “Hola” entonces:
4       Mostrar en pantalla “Todo está bien”
5  De lo contrario:
6       Mostrar en pantalla “Algo ha salido mal”
7  FIN
```

Esto generaría la misma salida:

Salida	Todo está bien

En conclusión

En este capítulo aprendimos sobre la importancia de los algoritmos en la programación y cómo utilizar herramientas gráficas como los diagramas de flujo para representarlos. Exploramos el uso del pseudocódigo como una forma de explicar los algoritmos de programación en un lenguaje de alto nivel que es fácil de entender y de transcribir posteriormente a un lenguaje de programación.

Capítulo 6

Ciclos, bucles e iteraciones en Lua

Los ciclos, bucles e iteraciones en Lua se refieren a bloques de código que se ejecutan una y otra vez hasta que cierta condición deje de cumplirse. Si esta condición deja de cumplirse, entonces el bloque de código dejará de ejecutarse. Si la condición sigue cumpliéndose, el bloque de código seguirá ejecutándose. Estos elementos son muy útiles en programación, ya que nos permiten ejecutar un bloque de código repetidas veces sin tener que volver a escribirlo. La práctica "Nunca te repitas a ti mismo" es algo importante que los desarrolladores deben considerar al escribir código, es decir, tratar de escribir código sin tener que volver a escribir el mismo bloque más de una vez, a menos que sea absolutamente necesario.

Como mencionamos antes, la estructura de estos elementos se ejecutará una cantidad indefinida de veces hasta que la condición deje de cumplirse. Esto podría dar la idea de que los programas que usan estas estructuras podrían ejecutar un bloque de código una cantidad infinita de veces. Es importante tener en cuenta que cambiar la condición dentro de un bucle puede ser un error fatal para el funcionamiento del programa. Pero no te preocupes, en este libro aprenderás cómo solucionar este tipo de errores y hacer que tu código vuelva a funcionar correctamente.

Ciclos while en Lua

Vamos a aprender sobre nuestra primera estructura de control de flujo, el ciclo while. Aquí hay un diagrama de flujo que nos ayudará a comprender cómo funciona:

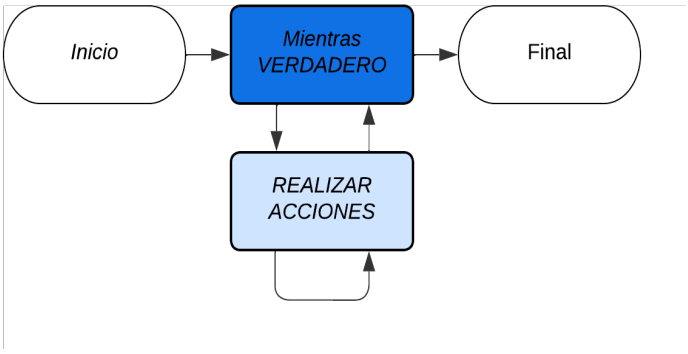


Ilustración 47 - Diagrama de flujo ciclo while

Como puedes ver en el diagrama, el ciclo while ejecuta un bloque de código mientras se cumpla una condición. Cada vez que el bloque de código se termina, la condición se vuelve a evaluar. Si sigue siendo verdadera, el bloque de código se ejecutará de nuevo. Esto continuará indefinidamente hasta que la condición ya no sea verdadera.

Ahora es el momento de ver cómo aplicamos esta estructura en Lua y escribir algunos códigos. ¡Prepárate para escribir algunas líneas de código emocionantes!

El concepto de break en bucles

Ya hemos adquirido una base teórica sobre cómo funcionan los condicionales en Lua y también vimos algunos ejemplos de cómo utilizarlos para realizar tareas más complejas.

Hay varias formas de controlar el flujo de ejecución en un programa y, en este caso específico, dentro de un condicional. Sabemos que estos condicionales se ejecutan solo si su valor de evaluación es verdadero, pero en muchas ocasiones queremos controlar el flujo de ejecución de tal manera que toda la estructura condicional se cumpla solo si se cumple cierta condición.

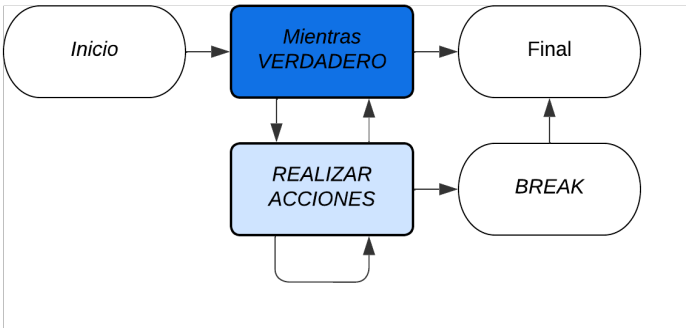


Ilustración 48 - Diagrama de flujo break en ciclos

Esto se puede ver como una condición de detención interna, que puede interrumpir la ejecución del programa sin tener que ejecutar el resto del código dentro de la estructura. Este concepto se conoce como "break" en programación.

Podemos traducir la palabra "break" como "romper". Entonces, el concepto de "break" se refiere a interrumpir la secuencia de ejecución del programa. Lograr esto es más sencillo de lo que parece, solo debemos utilizar la palabra "break", que ya está establecida en el programa para esta función. Veamos un ejemplo para entender mejor:

```
1  i = 1
2  while i do
3  io.write(i .. " ")
4       if i == 5 then
5            break
6       end
7  i = i + 1
8  end
```

Al ejecutar este código, obtendremos la siguiente salida:

```
Salida  1 2 3 4 5
```

Como podemos ver, esta vez la condición de detención está dentro del mismo bucle de ejecución. Si el valor llega a 5, el código dentro del bucle se detiene y la ejecución continúa fuera del bucle.

Uso de ciclos while y contadores en Lua

En el mundo de la programación, es muy útil entender cómo usar contadores dentro de tus programas. Es un concepto similar a las iteraciones matemáticas, pero no es necesario entenderlo completamente para poder utilizarlo. Un contador es simplemente una variable temporal que aumenta su valor a medida que el ciclo se repite.

En Lua, una de las características de los bucles y ciclos es que pueden repetirse y ejecutarse varias veces. Un contador es una variable que aumenta su valor en cada vuelta del ciclo. A menudo, la variable del contador es también la condición de parada del ciclo, pero en algunos casos puede no ser así.

Veamos un ejemplo de código para ver cómo funcionan los contadores en un ciclo "while":

```
1  contador=0
2  while(contador<7) do
3       print(contador)
4       contador = contador + 1
5  end
```

Al ejecutar este bloque de código, obtendrás la siguiente salida:

```
Salida  0
        1
        2
        3
        4
        5
        6
```

Vamos a explicar el código para que lo puedas entender mejor:

Explicación del código	
1	En la primera línea, creamos una variable numérica llamada "contador" y le asignamos el valor 0.
2	Luego, iniciamos un ciclo "while" y le decimos a la computadora que ejecute el código dentro del ciclo siempre y cuando el valor de "contador" sea menor que 7.
3	Dentro del ciclo, pedimos a la computadora que muestre en pantalla el valor de "contador" y, después, aumente el valor de "contador" en 1.
4	El ciclo se repite hasta que el valor de "contador" sea igual o mayor que 7, momento en el que el ciclo termina y el programa se detiene.

Este ejemplo te muestra cómo puedes usar un contador dentro de un ciclo "while" para hacer que el ciclo se repita varias veces y hacer que la variable "contador" aumente su valor en cada iteración.

Uso de ciclos while y contadores en reversa en Lua

Veamos cómo funcionan los contadores inversos en comparación con los contadores regulares. La diferencia principal es que los contadores inversos disminuyen su valor en cada iteración del ciclo, mientras que los contadores regulares aumentan su valor.

Resumiendo, los contadores regulares aumentan su valor en una unidad en cada iteración, mientras que los contadores inversos disminuyen su valor en una unidad en cada iteración.

Veamos un ejemplo de código que muestra una de las muchas formas en que podemos usar contadores inversos.

```
1  contador=5
2  while(contador>=0) do
3       print(contador * 2)
4       contador=contador - 1
5  end
```

Este código nos dará la siguiente salida:

```
Salida  10
        8
        6
        4
        2
        0
```

Explicación del código	
1	Primero, creamos una variable llamada "contador" que almacena el valor 5.
2	Luego, usamos un ciclo "while" que verifica si el valor en "contador" es mayor o igual a 0.
3	Dentro del ciclo, imprimimos en pantalla el valor de "contador" multiplicado por 2.
4	Finalmente, restamos 1 al valor en "contador".

¡Y eso es todo! Con este ejemplo, hemos visto cómo usar contadores inversos dentro de nuestros programas en Lua.

Uso de ciclos while anidados en Lua

Es hora de ver otro concepto sencillo en el estudio de los ciclos while en Lua. Hablamos de los ciclos anidados, una estructura cíclica compuesta por muchos otros ciclos dentro de ella. Un ciclo anidado es simplemente un ciclo que contiene otros ciclos dentro de su estructura. Por ejemplo, podemos entender esto mejor con unos cuantos ejemplos prácticos. Mira este ejemplo de código que demuestra lo que acabamos de describir:

```
1   contador1=0
2   contador2=0
3   while(contador2<5) do
4       contador1=0
5       while(contador1<3) do
6           io.write(“a”)
7           contador1=contador1+1
8       end
9       io.write(“\n”)
10      contador2=contador2+1
11  end
```

Ejecutando este bloque de código, obtendrás la siguiente salida:

```
Salida  aaa
        aaa
        aaa
        aaa
        aaa
```

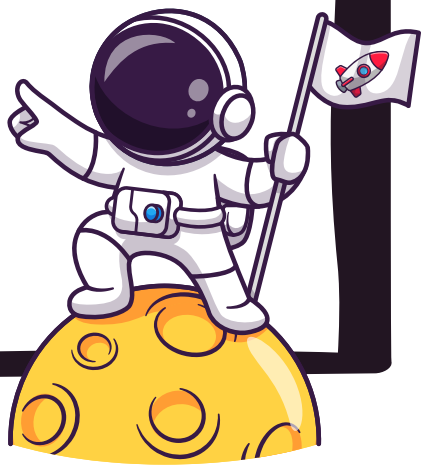
Explicación del código	
1	En este programa, vemos cómo se utilizan ciclos while anidados, es decir, una estructura while dentro de otra estructura while. Primero, inicializamos las variables contador1 y contador2 con el valor 0.
2	Luego, nos encontramos con nuestro primer ciclo while. Este ciclo se ejecutará mientras la condición se evalúe como verdadera y contador2 sea menor que 5. Dentro de este ciclo, encontramos otro ciclo while anidado, después de reiniciar el valor de contador1 a 0. Este segundo ciclo se encarga de mostrar en pantalla la letra "a" tres veces, terminar el ciclo y ejecutar el resto del código fuera de este bloque.
3	Después de mostrar las letras "a", agregamos un salto de línea y repetimos el proceso un total de 5 veces.

Ejercicios de práctica

1	Desarrolla un programa que muestre los 20 primeros números naturales en pantalla. Asegúrate de usar solo una variable para realizar esta tarea. La variable debería ser modificable dentro del programa. Utiliza la función io.write() para mostrar la salida en una sola línea de la terminal. La salida esperada es la siguiente: Salida 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
2	Crea un programa que muestre los primeros elementos de la sucesión de Fibonacci en pantalla. Por ejemplo, si se escribe el número 10, la salida esperada es: Salida 1 1 2 3 4 7 11 18 29 47 Asegúrate de que el programa cumpla con las siguientes condiciones: 1 El programa solo debe usar un máximo de 4 variables numéricas, incluyendo el contador, y una variable de texto. En total, debes usar 5 variables. 2 Debe usar una sola estructura cíclica. 3 Debe escribir el resultado usando una sola línea de código con la función print().
3	Crea un programa que muestre cada uno de los números en el rango [1, n], donde n es un valor numérico escrito por el operador del programa. Por ejemplo, si el valor escrito es 4, la salida esperada es: Salida 1, 2, 3, 4 Asegúrate de que cada número esté separado por una coma, excepto el último.
4	Crea un repetidor de pantalla, un programa que repita una cadena de texto específica un número determinado de veces. El programa debe tomar como entrada una cadena de texto y un valor numérico. Por ejemplo, si la cadena de texto es "Cadena" y se desea repetir 7 veces, la salida esperada es: Salida Cadena Cadena Cadena Cadena Cadena Cadena Cadena
5	Completa el siguiente bloque de código para que la ejecución corresponda a la salida esperada del programa. Salida Ingrese un número: 10 Ingrese un número: -7 Ingrese un número: 56 El menor número es -7 Y aquí está el bloque de código que debes completar: 1 contador = 1; menor = 999 2 while contador < 3 do 3 io.write(“Ingrese un número: “); actual = io.read() 4 if actual < menor then menor = actual end 5 contador = contador + 1 6 end 7 print(“El menor número es ” .. menor)

En conclusión

Los ciclos, bucles e iteraciones son una herramienta fundamental en la programación en Lua. Nos permiten ejecutar un bloque de código repetidas veces sin tener que volver a escribirlo, siguiendo el principio de "Nunca te repitas a ti mismo". Es importante recordar que la estructura de estos elementos se ejecutará una cantidad indefinida de veces hasta que la condición deje de cumplirse, por lo que debemos tener cuidado al cambiar la condición dentro de un bucle para evitar errores en el programa. Además, aprendimos cómo usar el ciclo while en Lua y cómo controlar el flujo de ejecución mediante el uso del concepto de "break". También vimos cómo usar contadores en ciclos while y ciclos anidados para realizar tareas más complejas en nuestros programas. Ahora que hemos adquirido una base teórica sólida, es hora de poner en práctica lo aprendido con algunos ejercicios.



Los bucles repeat en Lua

Es hora de conocer nuestra segunda estructura de repetición en Lua, se trata de la estructura "repeat".

Esta estructura se puede ver como el inverso del ciclo "while". Mientras que antes, un ciclo "while" se ejecutaba mientras se cumplía una condición, ahora una estructura "repeat" se ejecutará indefinidamente hasta que se cumpla una condición. Una vez que se cumpla la condición, el programa dejará de ejecutar el bloque de código dentro de la estructura. Es similar a la estructura "while", ya que ambas se utilizan para repetir un bloque de código bajo ciertas condiciones, pero con una ligera diferencia en su funcionamiento. Veamos el diagrama de flujo de la estructura "repeat":

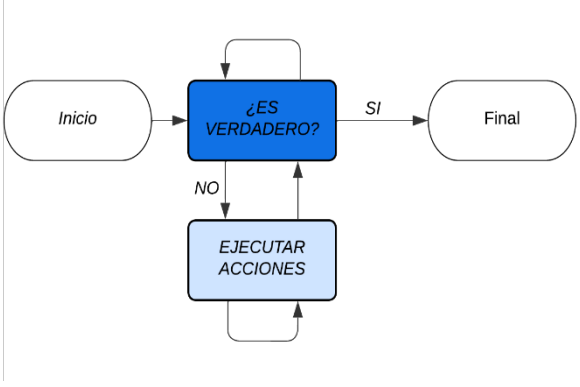


Ilustración 49- Diagrama de flujo de estructura repeat

Este diagrama muestra una visualización de la estructura general de un bucle "repeat". Como se mencionó anteriormente, el bucle se ejecutará indefinidamente hasta que se cumpla la condición.

Los bucles "repeat" pueden ser útiles en muchos aspectos de nuestros programas, pero debido a su similitud con la estructura "while", sus usos principales son similares. Podemos usar esta estructura cada vez que necesitemos repetir un bloque de código bajo ciertas condiciones. A continuación, veamos algunos ejemplos de cómo escribir estos bucles en Lua.

Uso simple de la estructura *repeat*

En este ejemplo, vamos a ver cómo usar de manera sencilla la estructura repeat. Como mencionamos antes, esta estructura es similar a la estructura de ciclos while que vimos previamente.

```
1  numero=10
2  repeat
3  io.write(numero, " ")
4  numero = numero - 1
5  until (numero==0)
```

Al ejecutar este código, obtenemos la siguiente salida en la terminal:

```
Salida  10 9 8 7 6 5 4 3 2 1
```

Veamos una explicación más detallada de lo que sucede en el código:

Explicación del código	
1	Primero, declaramos la variable "numero" con el valor 10.
2	Luego, iniciamos la estructura repeat.
3	Dentro de la estructura repeat, mostramos en pantalla el valor de "numero" con un espacio.
4	Después, restamos 1 a "numero".
5	Finalmente, continuamos con la estructura repeat hasta que "numero" sea igual a 0.

Ejemplo de uso de la estructura repeat

Vamos a ver otro ejemplo práctico de cómo podemos utilizar esta estructura cíclica en nuestros programas usando Lua:

```
1  numero=2
2  potencia=1
3  suma=0
4  repeat
5      suma = suma + (numero ^ potencia)
```

```
6       potencia = potencia + 1
7   until(potencia==5)
8   print(“El resultado de la suma es igual a: “ .. suma)
```

Al ejecutar este código, obtendremos la siguiente salida en la terminal de nuestro programa:

```
Salida   El resultado de la suma es igual a 30.0
```

Veamos una breve explicación de lo que sucede dentro de nuestro código:

Explicación del código	
1	En este programa estamos usando 3 variables: "numero", "potencia" y "suma". Todas ellas tienen valores numéricos: 2, 1 y 0 respectivamente.
2	Luego, usamos la estructura "repeat". Por cada iteración, aumentamos el valor de "potencia" y realizamos el cálculo. El programa efectúa la operación hasta que se cumpla la condición de igualdad, que detiene el bucle. El resultado final es el número 30 o 30.0.

Con esto, tenemos una comprensión más sólida del funcionamiento de la estructura "repeat" en Lua como lenguaje de programación.

Este ejemplo nos muestra algunos de los conceptos y funcionalidades fundamentales de los ciclos "repeat". Sin embargo, hay muchos otros conceptos que se pueden aplicar a esta estructura. En futuras oportunidades, podremos explorarlos más profundamente y aprender a usarlos en nuestros programas y proyectos en Lua.

Estructuras repeat anidadas

En este ejemplo, vamos a explorar cómo anidar estructuras de tipo 'repeat' en nuestro código. Esto significa que vamos a ver cómo una estructura 'repeat' puede estar dentro de otra estructura 'repeat'. Echemos un vistazo al siguiente bloque de código para entender mejor:

```
1   índice=1
2   indiceInverso=5
3   repeat
4       indiceInverso=5
5       repeat
6           io.write(“_”)
7           indiceInverso=indiceInverso-1
8       until (indiceInverso==0)
9       io.write(“ “)
10      índice=indice+1
11  until (índice==5)
```

Al ejecutar este código, veremos la siguiente salida en la pantalla:

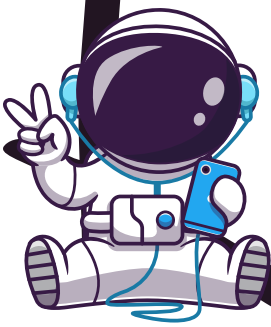
Salida	77777 77777 77777 77777

Como puedes ver, este código muestra cómo se pueden anidar estructuras 'repeat' para lograr un resultado deseado. Vamos a explicar el código que acabamos de ver para que podamos comprenderlo completamente.

Explicación del código	
1	Usamos dos variables contadoras llamadas índice e indiceInverso. Ya estamos familiarizados con los ciclos y los bucles anidados, aquí podemos ver que este concepto de anidación también se aplica a esta estructura específica.
2	Usamos un nuevo ciclo 'repeat' dentro de nuestro ciclo 'repeat' inicial. La estructura anidada es responsable de mostrar en la pantalla la cantidad de números deseados, mientras que la estructura más externa es responsable de mostrar los espacios en blanco la cantidad de veces especificada en el código."

En conclusión

La estructura de repetición "repeat" en Lua es una herramienta poderosa y útil para repetir un bloque de código bajo ciertas condiciones. Aunque es similar a la estructura "while", su funcionamiento es ligeramente diferente, ya que se ejecuta indefinidamente hasta que se cumpla una condición. En este capítulo, aprendimos cómo escribir bucles "repeat" simples y complejos con ejemplos prácticos y hemos explorado cómo se pueden anidar las estructuras "repeat" para lograr un resultado deseado. Con esta comprensión más sólida, podemos utilizar eficazmente la estructura "repeat" en nuestros programas y proyectos en Lua.



Iteraciones for en Lua

En esta sección, exploraremos la última estructura de ciclos y bucles en Lua: las iteraciones for. Antes de continuar con la explicación, es importante comprender lo que son las iteraciones y su importancia en la solución de problemas.

El concepto de iteración es más sencillo de lo que parece. De hecho, ya hemos utilizado este concepto en varias ocasiones con estructuras cíclicas y de bucle. Entonces, ¿por qué apenas estamos empezando a estudiarlo? Aunque hemos usado parcialmente el concepto de iteración en los ejemplos anteriores, esto no significa que los ciclos repeat o while estén enfocados en realizar iteraciones. Sin embargo, la estructura for sí tiene un enfoque fuerte en la aplicación directa de las iteraciones dentro de la misma estructura.

Matemáticamente, una iteración es la repetición cíclica de una función, donde la salida se convierte en la entrada de la siguiente función. Esta definición es parcialmente aplicable en programación. No es

correcto considerar las iteraciones en programación como el uso de una función que devuelve una salida para retomarla como entrada. Es mejor entender las iteraciones en programación como la repetición cíclica de un conjunto de instrucciones, donde parte de la entrada de los datos utilizados en este conjunto varía en su valor en cada iteración hasta que se cumpla una condición de parada.

Al igual que con otras estructuras que hemos visto en este libro, también usaremos la estructura for en muchos casos de estudio donde necesitemos una estructura más avanzada que pueda generar una iteración dentro de un bloque de código sin la necesidad de crear una variable adicional que solo consuma recursos durante la ejecución de nuestro programa.

Hay muchas características y partes importantes de una estructura de ciclo for en nuestros programas, pero no es necesario entender todas ellas completamente. Las verás en las secciones posteriores de este libro.

Partes de un ciclo for en Lua

Ahora tenemos una idea general de cómo funciona un ciclo for en la mayoría de los lenguajes de programación. La sintaxis de estos ciclos puede variar según los estándares del lenguaje de programación específico. Lua tiene ciertas condiciones específicas al escribir ciclos for, por lo que es importante comprender las partes que componen un ciclo for en este lenguaje de programación.

La primera parte importante es la variable índice. Esta variable es evaluada en comparación con la condición planteada dentro del programa como la condición de parada de la estructura. La variable índice puede ser declarada tanto dentro de la estructura del ciclo como fuera de ella. La segunda parte es la condición de parada. Esta es la información que detendrá el programa una vez que la variable índice sea igual a ella.

La tercera parte es el paso o avance del índice. Normalmente, se trata de un valor numérico que se agrega a la variable índice en cada iteración. Por ejemplo, si el paso es 2, entonces la variable índice aumentará en 2 unidades en cada iteración.

Finalmente, la última parte es el bloque de código, donde se escribirá el código que se ejecutará en cada iteración de la estructura.

Es fundamental comprender correctamente el funcionamiento y los roles de cada parte de un ciclo for, ya que este tipo de estructuras se usan comúnmente para solucionar problemas complejos que requieren la repetición de un bloque de código en un programa.

Como se mencionó anteriormente, un ciclo for está enfocado en aplicar el concepto de iteraciones dentro de su propia estructura. Aunque puede ser un poco complicado para los que recién comienzan en el mundo de la programación, en realidad no es muy complejo.

Veamos un ejemplo de cómo se desarrolla un ciclo for en el siguiente diagrama de flujo:

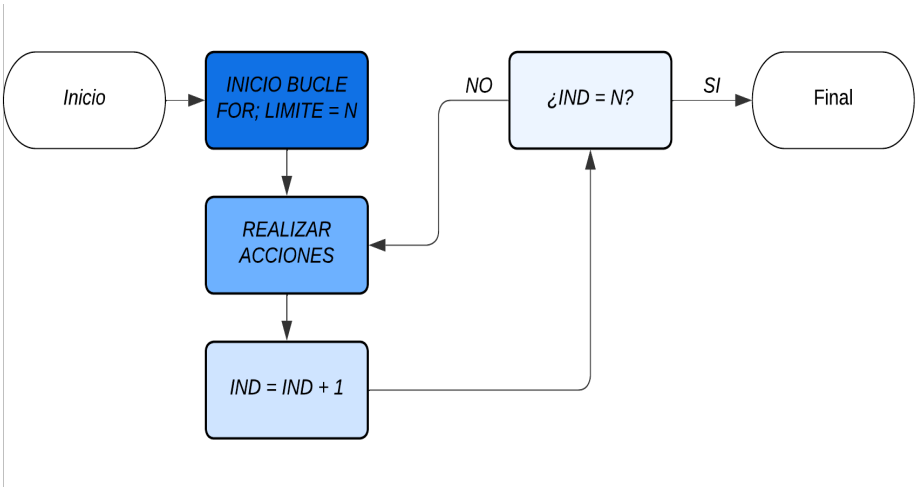


Ilustración 50 - Diagrama de flujo de iteraciones for

¿Lo ves más claro ahora? Eso demuestra lo importante que es visualizar y comprender correctamente la solución de problemas mediante diagramas de flujo. Ahora es el momento de aplicar lo que aprendimos y traducirlo a código Lua, ya que tenemos un conocimiento más amplio de qué son y cómo funcionan las iteraciones en programación.

Uso básico de las iteraciones for en Lua

Es hora de empezar a ver algunos ejemplos prácticos de cómo podemos utilizar las estructuras de iteración for en nuestro código. En este primer ejemplo, aprenderemos cómo hacer uso de esta estructura de una manera sencilla y básica.

Veamos el siguiente bloque de código:

```
1  i for index=0, 10, 1 do
2       io.write(index, “ “)
3  end
```

Al ejecutar este código, obtendremos la siguiente salida en la terminal:

```
Salida  0 1 2 3 4 5 6 7 8 9 10
```

Explicación del código	
1	En este ejemplo, no hemos declarado ninguna variable externa a la estructura for. Solo hemos utilizado tres líneas de código para obtener el resultado deseado.
2	La variable "index" es la que estamos utilizando en la estructura for. Le indicamos el valor de parada y el paso de aumento que debe tomar.
3	Dentro del bloque de código, mostramos en pantalla el valor actual de "index", y luego agregamos un espacio en blanco para separar los números.

¡Eso es todo! Con este ejemplo sencillo, aprendimos cómo utilizar la estructura de iteración for en nuestro código.

Variables externas en iteraciones for

Veamos ahora cómo funcionan las variables que aumentan o disminuyen su valor en cada repetición, pero esta vez usando variables externas en lugar de la variable que se utiliza como índice dentro del ciclo "for".

Aunque puede parecer algo sin importancia, ya hemos mencionado antes la utilidad de no tener que usar una variable externa que consuma espacio en la memoria y que probablemente no se vuelva a utilizar en otro lugar del programa. Sin embargo, si es necesario, también es posible hacerlo.

El siguiente código ejemplifica lo que mencionamos:

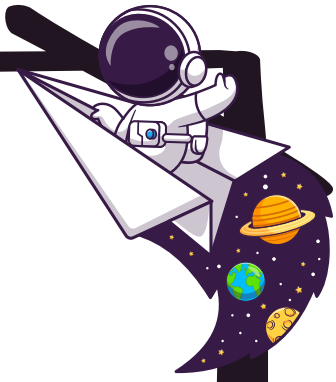
```
1  numero=10
2  for numero=numero, 0, -1 do
3       io.write(numero, “ ”)
4  end
5  print(“- ” .. numero)
```

Al ejecutar este código, obtendrás la siguiente salida en la terminal de tu programa:

```
Salida  10 9 8 7 6 5 4 3 2 1 0 – 10
```

Explicación del código	
1	En este ejemplo, inicialmente establecemos una variable llamada "numero" y le asignamos el valor numérico 10.
2	Luego, utilizamos el valor de la variable "numero" dentro del programa. Escribimos "numero=numero" para darle a la variable local dentro del ciclo "for" el valor global de la variable "numero" fuera de la estructura.
3	El valor de la condición de parada del ciclo "for" es 0, y su paso es la disminución en una unidad de la variable "numero" en cada iteración.
4	Dentro del bloque de código del ciclo "for", mostramos en pantalla el valor de la variable "numero".
5	Después de que se ejecute el ciclo "for", el programa muestra en pantalla el valor final de la variable "numero", que es 10.

Es importante destacar que, aunque estamos usando una variable externa como índice dentro del programa, lo que realmente estamos haciendo es asignar el mismo valor de la variable externa a la variable local dentro del ciclo "for". Al final del programa, no alteramos el valor de la variable externa.



En conclusión

En este capítulo aprendimos acerca de las iteraciones for en Lua y cómo se diferencian de otras estructuras de ciclos y bucles. También hemos explorado las partes que componen un ciclo for en Lua, incluyendo la variable índice, la condición de parada, el paso o avance del índice y el bloque de código. Además, hemos visto cómo utilizar las iteraciones for en Lua de manera básica y cómo trabajar con variables externas en lugar de la variable de índice.

Iteraciones for por cada elemento de una lista

Exploraremos cómo usar ciclos for para iterar a través de cada elemento de una lista. Esta técnica es útil para problemas de iteración relacionados con la lista. No necesitamos trabajar directamente con el índice de la variable. Con el siguiente diagrama de flujo, entenderemos cómo iterar con ciclos for en nuestros programas.

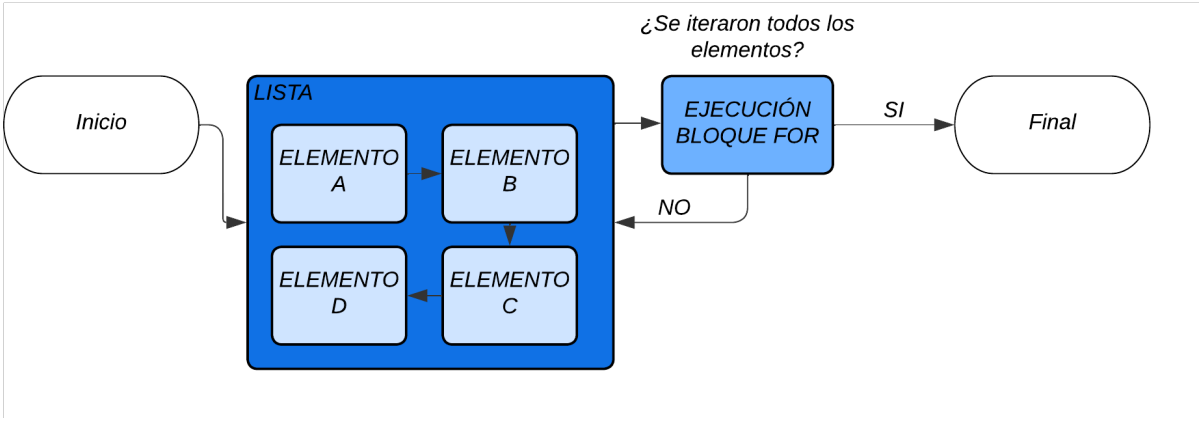


Ilustración 51- Iteración entre elementos de una lista

Como puedes ver, simplemente estamos yendo a través de cada uno de los elementos de la lista, asegurándonos de que la ejecución del programa ocurra en cada una de sus partes. Si la lista en la que estamos tratando de iterar contiene miles de elementos, el programa solo se detendrá después de haber ejecutado cada uno de ellos. Si la lista está vacía, el programa no se ejecutará. Veamos algunos ejemplos para entender mejor este concepto. Hay varias formas de lograr esto, como usar la función ipairs() (que se explicará y usará en ejemplos prácticos en secciones posteriores) o buscar la longitud de la lista e iterar dentro de ella. Ahora, es el momento de ver algunos ejemplos escritos directamente en el

lenguaje de programación. Con estos ejemplos podrás tener una base de cómo usar estos conceptos de manera práctica en tus programas.

Iterar por cada elemento de una lista usando for

Ahora vamos a ver ejemplos prácticos de cómo aplicar los conceptos que hemos aprendido. Como mencionamos antes, hay varias formas de lograr esto, pero para fines educativos, usaremos la función ipairs(). Es posible que esta función sea un poco confusa al principio, pero no debemos preocuparnos. En resumen, la función ipairs() es una función que se encarga de hacer una iteración a través de cada elemento de la lista que le proporcionemos como parámetro. A continuación, tenemos un ejemplo de código que ilustra cómo iterar por cada elemento de una lista con for:

```
1  lista = {0, 1, 2, 3, 4, 5}
2  for indice, element in ipairs(lista) do
3       print(element)
4  end
```

Así, al ejecutar este bloque de código obtenemos la siguiente salida:

```
Salida  0
        1
        2
        3
        4
        5
```

Explicación del código	
1	En primer lugar, creamos una lista que contiene seis elementos numéricos: 0, 1, 2, 3, 4 y 5.
2	Luego, usamos una estructura for para iterar a través de cada elemento de la lista. La sintaxis puede parecer un poco complicada, pero es fácil de entender.
3	La estructura for comienza con for indice, elemento in ipairs(lista) do. Este for es el encargado de realizar la iteración por cada elemento en la lista.
4	En este caso, indice hace referencia al número del índice del elemento en la lista que estamos iterando; elemento hace referencia al elemento mismo en la lista y será usado dentro de la iteración; e in ipairs(lista) es la función que hace que cada iteración sea hecha en base a los elementos de la lista que le hayamos asignado.

Ahora que entendemos cómo funciona la iteración en Lua, veamos otros ejemplos más simples. Además, veamos cómo podemos hacer esto mismo sin tener que usar la función ipairs().

Iterar por cada elemento de una lista usando for sin la función ipairs()

Como mencionamos anteriormente, vamos a ver cómo podemos hacer una iteración por cada elemento de una lista sin usar la función ipairs(). La forma de lograr esto es buscando la longitud de la lista y luego realizar la iteración hasta el último elemento. Veamos el siguiente diagrama de flujo para entender mejor cómo funciona esta idea en nuestro programa:

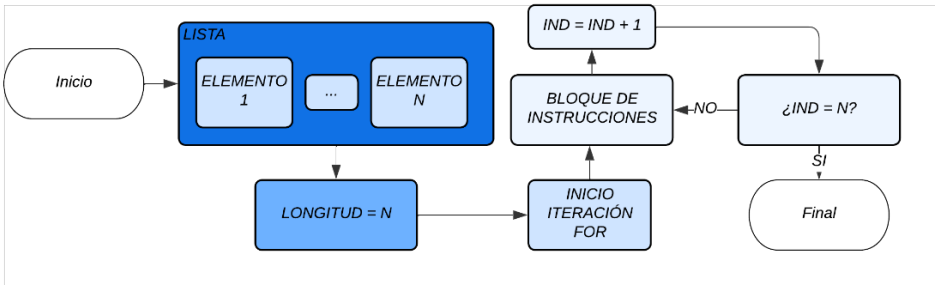


Ilustración 52 - Iterar en una lista sin ipairs()

Como se puede ver en el diagrama, primero estamos buscando la longitud de la lista, es decir, estamos encontrando un valor numérico que represente la cantidad de elementos en la lista. Luego, realizamos la iteración hasta el último elemento de la lista. Es posible que aún no entendamos completamente qué es una lista y su importancia en nuestros programas, pero esto será cubierto en detalle en secciones posteriores.

Recapitulando: ¿Qué es una lista?

En pocas palabras, una lista es un tipo de dato que almacena varios otros tipos de datos dentro de sí. Es como una colección de elementos organizados juntos.

Aquí está un ejemplo más práctico para entender mejor el concepto. Este código muestra cómo podemos iterar a través de cada elemento en una lista:

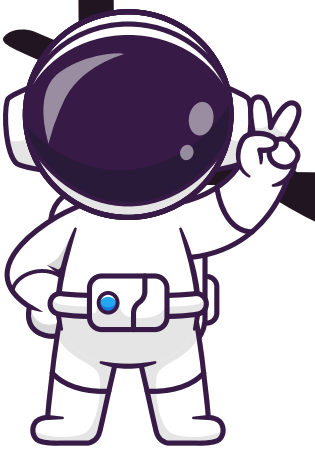
```
1  lista = {0, 1, 2, 3, 4, 5}
2  longitud = #lista
3  for i=1, longitud do
4      print(lista[i])
5  end
```

La salida en la terminal de nuestro programa sería la siguiente:

Salida	0 1 2 3 4 5

En conclusión

Aprendimos cómo usar ciclos for para iterar a través de cada elemento de una lista en Lua. Esto es especialmente útil para problemas relacionados con la iteración de una lista. No necesitamos trabajar con el índice de la variable y podemos iterar directamente sobre cada uno de los elementos en la lista. En este capítulo, hemos visto diferentes maneras de lograr esto, incluyendo el uso de la función ipairs() y la búsqueda de la longitud de la lista para realizar la iteración. Esperamos que estos ejemplos hayan sido útiles para comprender cómo aplicar estos conceptos en la práctica. Además, también discutimos qué es una lista y cómo funciona como un tipo de dato en Lua.



Capítulo 7

Variables locales y globales en Lua

En la sección anterior, aprendimos sobre el funcionamiento de las estructuras de tipo "for" en Lua. En uno de los ejemplos, también mencionamos brevemente el uso de variables locales y globales en Lua, pero aún teníamos dudas sobre lo que exactamente eran estas variables y cómo funcionaban.

En el mundo de la programación, existen muchos conceptos comunes que se aplican independientemente del lenguaje de programación que estemos utilizando. Uno de estos conceptos es el de "scopes", que se refiere a la forma en que la información y el código se desarrollan durante la programación. Este concepto de variables locales y globales en Lua no es único a este lenguaje, sino que también se aplica a una amplia gama de otros lenguajes de programación.

Pero ¿por qué es importante entender el uso de variables locales y globales en la programación y cómo afectan a nuestro código? Podemos volver a revisar el ejemplo anterior de la estructura "for" en Lua para entender mejor este concepto:

```
1  numero=10
2  for numero=numero, 0, -1 do
3       io.write(numero, “ ”)
4  end
5  print(“- ” .. numero)
```

Al ejecutar este bloque de código, obtenemos la siguiente salida:

```
Salida  10 9 8 7 6 5 4 3 2 1 0 - 10
```

Anteriormente, se explicó en general lo que hace este código, pero vamos a repasarlo de nuevo para asegurarnos de entender adecuadamente este importante tema. Hasta ahora, hemos trabajado con las llamadas variables globales. Una variable global es aquella que puede ser llamada desde cualquier parte del programa y, por lo tanto, su valor y espacio en memoria persistirán desde su inicialización hasta que el programa finalice.

Claro está, siempre y cuando la variable no sea eliminada posteriormente a su inicialización en algún otro lugar del código (veremos cómo eliminar variables y optimizar correctamente el uso de la memoria más adelante en este escrito). Si hacemos un cambio en algún lugar del código, la variable global (ya que es universal) se verá afectada en su valor. Esto también ocurrirá en el caso hipotético de que esta variable sea mencionada en algún punto más interno de una estructura cíclica aparte del bloque de código principal de nuestro programa.

Las variables globales suelen ser inicializadas dentro del bloque de código más externo, es decir, el bloque global. No importa en qué bloque de código se utilice esta variable, siempre estará disponible y podremos acceder a la información que se almacena en ella.

Ahora hablemos de las variables locales en Lua. Se pueden definir usando el código "local" de la siguiente manera:

```
1  do
2       local numero = 10
3       io.write(numero, " - ")
4  end
5  print(numero)
```

Al ejecutar este código, obtenemos la siguiente salida:

```
Salida  10 - nil
```

Como podemos ver, creamos un bloque de código vacío con "do" y dentro de él declaramos una variable local llamada "numero" y le asignamos el valor 10. Dentro del bloque, podemos ver el valor almacenado en la variable "numero". Pero cuando el programa sale del bloque e intentamos mostrar de nuevo el valor de "numero", obtenemos "nil".

Esto se debe a que las variables locales solo están disponibles en el bloque donde fueron declaradas y su valor desaparece una vez que se sale de ese bloque. Por lo tanto, las variables locales no pueden ser accedidas por bloques de código externos. En resumen, las variables locales son útiles para limitar el alcance de una variable y evitar conflictos con otras variables con el mismo nombre. Es importante tener en cuenta estas limitaciones al trabajar con variables locales en Lua. Veamos lo que sucede si declaramos la variable en el ejemplo anterior como una variable global:

```
1  do
2       numero = 10
3       io.write(numero, " - ")
4  end
5  print(numero)
```

Al ejecutar este código, obtenemos la siguiente salida:

```
Salida  10 - 10
```

Ahora tenemos una comprensión más profunda de las variables locales y globales en Lua, pero ¿para qué nos sirve esto en la práctica? A primera vista, puede parecer que no hay mucha diferencia entre ambas.

Sin embargo, es importante tener en cuenta que las variables locales solo están disponibles dentro del bloque de código donde fueron declaradas y su valor desaparece una vez que se sale de ese bloque. Esto significa que, al usar variables locales, estamos optimizando el uso de la memoria ya que la variable y su asignación en la memoria solo existirán durante la ejecución de ese bloque de código. Por lo tanto, es conveniente usar variables locales en la mayoría de los casos, a menos que sea necesario utilizar una variable global.

Para declarar una variable local, simplemente debemos usar la palabra clave "local" antes del nombre de la variable. Si no usamos esta palabra clave, el programa considerará que la variable es global. Ahora, veamos algunos ejemplos prácticos de cómo podemos aplicar el uso de variables locales en Lua para mejorar nuestro código.

Uso simple de variables locales en Lua

En este primer ejemplo, vamos a practicar el uso de variables locales en nuestros programas. Ya mencionamos que las variables locales suelen limitarse a un solo bloque de código y esto puede ser muy útil cuando queremos utilizar variables con una duración temporal. ¡Vamos a poner en práctica lo que hemos aprendido!

El siguiente bloque de código ilustra lo mencionado:

```
1  numero = 20
2  if true then
3       local numero = 10
4  end
5  print(“Número registrado: “ .. numero)
```

Al ejecutar este código, obtenemos la siguiente salida:

```
Salida  Número registrado: 20
```

Como puedes ver, el número registrado es 20, lo cual demuestra que la variable "numero" dentro del bloque "if" es una variable local y no afecta al valor de la variable "numero" en el ámbito global. Finalmente, es hora de explicar el código que hemos visto:

Explicación del código	
1	En primer lugar, creamos una variable llamada "número" y le asignamos el valor numérico de 20. Cualquier variable que no sea declarada como "local" se considera una variable global.
2	Después, tenemos un bloque de código dentro de un condicional con el valor de "true", lo que significa que este bloque será ejecutado.
3	Dentro de este bloque, creamos una variable local llamada "número" y le asignamos el valor numérico de 10.
4	Una vez que se ha ejecutado el bloque de código, mostramos en pantalla el valor de la variable "número". La salida es 20.
5	Aquí es donde entra en juego la diferencia entre variables locales y globales. La variable local "número" y la variable global con el mismo nombre son dos variables diferentes, cada una con su propio espacio de memoria.
6	La variable local "número" solo está disponible dentro de su bloque de código, por lo que cuando ejecutamos la línea de código que muestra el valor de la variable en un bloque de código externo, se muestra el valor de la variable global, que es 20.

En conclusión

Las variables locales y globales en Lua son conceptos importantes que deben entenderse para optimizar el uso de memoria en el código. Mientras que las variables globales están disponibles en todo el programa y su valor persiste hasta el final de la ejecución del programa, las variables locales solo están disponibles dentro del bloque de código donde fueron declaradas y su valor desaparece una vez que se sale de ese bloque. Es importante tener en cuenta estas diferencias al trabajar con variables en Lua y utilizar variables locales siempre que sea posible para evitar conflictos y optimizar el uso de memoria en nuestro código.

Uso de las variables locales en el bloque global

Es común pensar que las variables locales solo se utilizan en bloques internos, pero en realidad es una buena práctica crear variables locales en cualquier bloque, incluido el bloque global. Esto ayuda a evitar situaciones donde las variables deban ser globales en el proceso de ejecución del programa.

Aunque crear una variable en el bloque principal de nuestro programa no afecta directamente el concepto de una variable local, ya que eventualmente se comportará como una variable global durante toda la ejecución, también es útil crear variables locales dentro del bloque global para evitar problemas relacionados con la ejecución de diferentes archivos y la importación de estos. Al tener un ecosistema múltiple de diferentes bloques de ejecución con sus respectivas variables, podemos evitar confusiones y asegurarnos de tener un control claro de las variables incluso en el bloque global de nuestro programa.

El siguiente bloque de código es un ejemplo de cómo utilizar variables locales en el bloque global:

```
1  local booleano = true
2  if booleano then
3       print(“Ok”)
4  end
```

Al ejecutar este código, verás lo siguiente en la terminal de tu programa:

```
Salida  Ok
```

Explicación del código	
1	Este programa es muy sencillo y su única finalidad es demostrar que es posible utilizar variables locales en el bloque global.
2	Primero creamos una variable booleana llamada "booleano". Luego, ejecutamos un bloque de código que verifica el valor de esta variable y muestra "Ok" en pantalla, lo que demuestra que la variable puede ser llamada desde cualquier otro bloque de código.

Veremos más adelante las diferentes aplicaciones que tiene esto y por qué se considera una buena práctica.

Esto es importante porque a menudo trabajamos con diferentes programas y podemos importar su contenido para utilizarlo en nuestro programa. Al crear variables locales en el bloque global, aseguramos que estas variables solo sean utilizadas en el contexto de nuestro archivo y no causen conflictos con otros programas importados.

Podemos ver una representación gráfica de los diferentes bloques de ejecución en nuestros programas a través del siguiente diagrama:

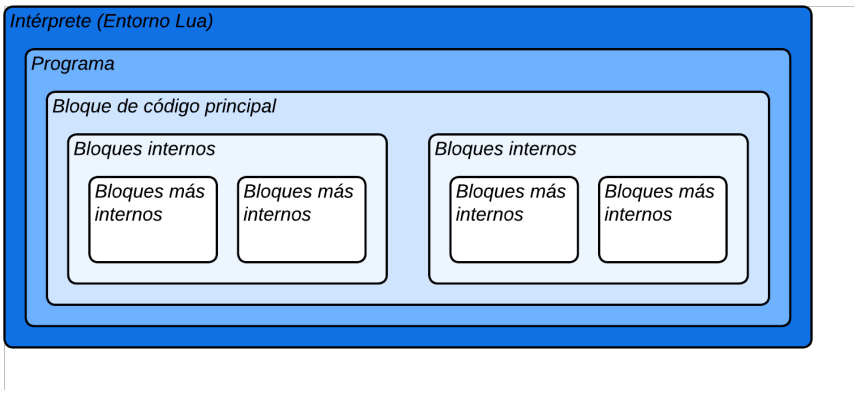


Ilustración 53 - Diferentes bloques de ejecución

Como se puede ver, tenemos un ecosistema dividido en diferentes partes que conforman los diferentes entornos que puede tener nuestro programa para ejecutarse correctamente. Podemos unir estos entornos con otros mediante la creación de variables locales dentro del bloque de ejecución principal, lo que nos permite tener un sistema cerrado en la ejecución de nuestro programa y evitar posibles conflictos al importar desde otros bloques de código.

Aunque no es necesario hacer esto en cada uno de los ejemplos que realizamos, seguir esta práctica es considerada un buen estándar léxico dentro de Lua, especialmente cuando se trata de importar diferentes sistemas a nuestro bloque de código.

Uso de variables locales en múltiples bloques de ejecución

Vamos a explorar un poco más sobre el uso de variables locales en Lua. En este caso, veremos cómo podemos utilizar variables locales en múltiples bloques de ejecución.

No te preocupes si te suena confuso. Simplemente estamos hablando de la posibilidad de usar una variable local en varios bloques de código, aunque la idea principal de una variable local es que solo se ejecute dentro de un bloque de código interno, pero, aun así, la declaración de esta variable también será aplicada a bloques de ejecución más internos al bloque donde se haya creado.

Veamos el uso de las variables locales con la siguiente ilustración:

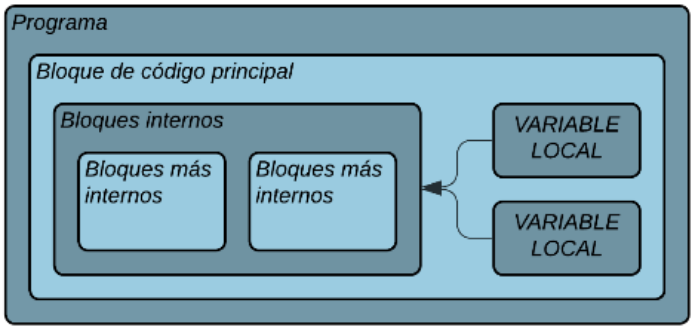


Ilustración 54- Variables locales y disponibilidad en bloques más internos

Como puedes ver en la ilustración, estamos declarando una variable local dentro de un bloque un poco más interno en comparación con el bloque principal de ejecución de nuestro programa. Este bloque también contiene otros bloques de ejecución, y todos ellos tienen acceso a la información almacenada en la variable local que hemos creado dentro del bloque interno, a pesar de que se trate de una variable local. Este código muestra un ejemplo de lo que acabamos de mencionar sobre el uso de variables locales en múltiples bloques de ejecución.

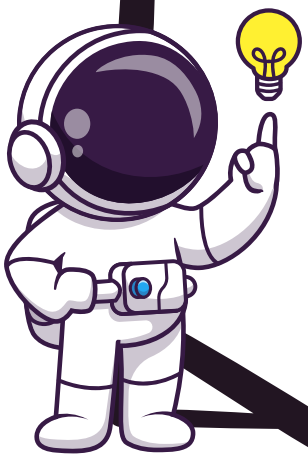
```
1  local numero = 10
2  if true then
3      local segundo = 5
4      local suma = numero + segundo
5      if true then
6          print(“La suma es igual a ” .. numero+segundo+suma)
7      end
8  end
```

Al ejecutar este código, se obtiene la siguiente salida en la terminal de nuestro programa:

```
Salida  La suma es igual a 30
```

Explicación del código	
1	En primer lugar, creamos una variable local llamada "numero" y le asignamos el valor 10. Luego, creamos un bloque de código interno usando una condición "if" con un valor booleano verdadero.
2	Dentro de este bloque, creamos otra variable local llamada "segundo" y le asignamos el valor 5. También creamos una tercera variable local llamada "suma", que es el resultado de la suma de "numero" y "segundo".
3	Finalmente, creamos un tercer bloque de código interno usando otra condición "if" y, dentro de este bloque, mostramos en la pantalla el resultado de la suma de estas tres variables.
4	También debemos tener en cuenta que podemos realizar este tipo de operaciones porque las variables locales son definidas dentro de un bloque de código que se encuentra en una posición más alta en la jerarquía, o bien han sido asignadas previamente dentro del mismo bloque de código.

En conclusión



Utilizar variables locales en cualquier bloque, incluido el bloque global, es una buena práctica que nos ayuda a evitar posibles conflictos al importar diferentes sistemas a nuestro bloque de código. Además, el uso de variables locales en múltiples bloques de ejecución nos permite tener un sistema cerrado en la ejecución de nuestro programa y asegurarnos de tener un control claro de las variables

Capítulo 8:

Tablas y listas en Lua

Las tablas en Lua son similares a las listas en otros lenguajes de programación. Entonces, ¿para qué son útiles las tablas en Lua? Ya vimos diferentes tipos de datos que podemos usar en Lua, como números, booleanos, cadenas, y tipos de datos nulos. Las tablas son un tipo de dato híbrido que nos permite almacenar múltiples tipos de datos en una misma variable. Almacenar diferentes tipos de datos en una misma variable puede ser un poco confuso, pero es aquí donde entra en juego la indexación. La indexación nos permite asignar valores numéricos secuenciales a cada elemento dentro de una tabla, lo que nos permite identificar y acceder a cada dato de manera fácil.

Las tablas en Lua también nos permiten aumentar o disminuir el número de elementos almacenados en ellas. Además, es importante tener en cuenta que la indexación en Lua no comienza de forma tradicional. En resumen, las tablas en Lua son una estructura muy útil que nos permite almacenar diferentes tipos de datos en una misma variable y acceder a ellos de manera fácil a través de la indexación. ¡Vamos a aprender cómo usarlas en práctica en el lenguaje de programación Lua!

Consideremos una lista ejemplar en algún otro lenguaje de programación:

```
1    lista = [60, 40, 99]
```

En este ejemplo, los valores dentro de la lista tienen un índice numérico que los identifica. Generalmente, este índice es un número que va desde 0 hasta n-1, donde n es la cantidad de elementos en la lista. Por lo tanto, si queremos acceder al primer valor en la lista, entenderíamos que su índice es 0. Por ejemplo:

```
2    Imprimir(lista[0])
```

La salida sería 60. La lista tiene un total de 3 elementos, por lo que si queremos acceder al último valor, usaríamos el índice de n-1, es decir, 2:

```
3    Imprimir(lista[2])
```

La salida sería 99.

¿Y qué pasa con Lua? A diferencia de otros lenguajes de programación, Lua no comienza la indexación en 0, sino en 1. Por lo tanto, si queremos acceder al primer valor en la lista en Lua, su índice sería 1 en lugar de 0. Es hora de ponernos manos a la obra y aprender a usar las tablas en Lua para resolver algunos problemas. ¡Vamos!

Uso básico de las tablas en Lua

En este apartado se explicará de manera más práctica cómo utilizar las tablas en Lua. Como se sabe, las tablas en Lua son similares a las listas y se indexan a partir del número 1.

A continuación, se presenta un ejemplo para ilustrar su uso:

```
1    tabla = {12, 27, 34, 45, 100}
```

```
2    print(“1. “ .. tabla[1] .. “ / 4. “ .. tabla[4])
```

Al ejecutar este código, se obtiene la siguiente salida en la terminal del programa:

```
Salida    1. 12 / 4. 45
```

Este código es bastante sencillo. En primer lugar, se crea una tabla en Lua que contiene cinco elementos: los números 12, 27, 34, 45 y 100. Es importante tener en cuenta que cada elemento tiene un índice que lo identifica. Los valores almacenados en la primera y cuarta posición de la lista corresponden a los números 12 y 45. Esto se logra de acuerdo con nuestra solicitud de mostrar el primer y cuarto valor almacenado en la lista.

Operaciones aritméticas y diferentes tipos de datos en una tabla

Las operaciones aritméticas y los diferentes tipos de datos pueden ser incluidos en una tabla en un programa. Las listas en el programa no solo tienen que estar limitadas a un solo tipo de dato. De hecho, una lista puede contener diferentes tipos de datos, como números, booleanos, texto o incluso otras listas.

No solo podemos almacenar un solo tipo de dato en una lista, sino que también podemos almacenar varios tipos de datos en una misma lista. Cada espacio en la lista puede tener un tipo de dato diferente, y eso no afecta el almacenamiento de los otros tipos de datos en los demás espacios de la lista.

El siguiente código es un ejemplo de cómo podemos incluir diferentes tipos de datos en una lista:

```
1    miLista = {“Fresas”, true, 17, 28*5, 0, “Mangos”}
2    miLista[1] = miLista[1] .. “ rojas”
3    miLista[3] = 10
4    for i = 1, 6, 1 do
5        print(miLista[i])
6    end
```

Si ejecutamos este bloque de código, obtendremos la siguiente salida en la terminal de nuestro programa:

```
Salida    Fresas rojas
          true
          10
          140
          0
          Mangos
```

Explicación del código	
1	Al inicio del programa, creamos una lista con seis elementos. De estos seis elementos, dos son cadenas de texto, tres son números y uno es un valor booleano.

2	En el código, modificamos el valor del primer elemento en la lista. Lo hacemos concatenando la cadena original con la cadena "rojas", dando como resultado "Fresas rojas". Este nuevo valor se guarda en la lista. También cambiamos el valor del tercer elemento en la lista y, finalmente, imprimimos los elementos de la lista en la pantalla.

Agregar elementos dentro de una lista

Las listas nos permiten almacenar diferentes valores, y podemos añadir elementos a ellas aunque ya estén definidas. A continuación, se muestra un ejemplo de cómo hacerlo.

```
1  lista = {11, 22, 33}
2  table.insert(lista, 4, 44)
3  for I = 1, 4, 1 do
4       print( lista[i] )
5  end
```

Al ejecutar este código, se obtiene la siguiente salida en la terminal:

```
Salida  11
        22
        33
        44
```

Explicación del código	
1	En primer lugar, creamos una lista llamada "lista" que contiene números.
2	Luego, utilizamos la librería "table" que es esencial para agregar elementos a una lista en Lua.
3	La función "table.insert()" tiene la estructura "table.insert(lista, índice, valor)", donde "lista" se refiere a la variable que se asigna a la lista a la que queremos agregar un nuevo elemento, "índice" se refiere a la posición en la que queremos agregar el nuevo valor y "valor" es el valor que queremos guardar en la lista.
4	Finalmente, usamos un ciclo for para mostrar todos los valores en la lista en la pantalla.

Eliminar elementos dentro de una lista

Después de ver cómo agregar elementos a una lista, veamos cómo podemos eliminar elementos que ya se encuentran dentro de nuestra lista. Aquí te muestro un ejemplo de código para ver cómo eliminar elementos de una lista que ya ha sido definida:

```
1  frutas  = {“Fresa”, “Manzana”, “Verde”, “Pera”, “Naranja”}
2  table.remove(frutas, 3)
3  for i = 1, 4, 1 do
4       io.write(frutas[i] .. “  “)
5  end
```

Al ejecutar este código, se obtiene la siguiente salida en la terminal:

Salida	Fresa Manzana Pera Naranja

Explicación del código	
1	Este código es similar al ejemplo anterior, pero en lugar de agregar elementos, eliminamos elementos de una lista. Primero, creamos una lista que contiene valores de cadena de caracteres: "Fresa", "Manzana", "Verde", "Pera", "Naranja".
2	Luego, usamos la librería "table" para eliminar elementos de la lista, en lugar de usar "table.insert", usamos "table.remove". La estructura de la función es "table.remove(lista, índice)", donde "lista" es la variable que se asigna a la lista y "índice" es la posición del objeto que deseamos eliminar. En este ejemplo, la lista se llama "frutas" y la mayoría de los elementos son frutas, excepto "verde" que no es una fruta. Este elemento está en la posición 3, por lo que lo eliminamos con "table.remove()". Finalmente, mostramos los nuevos elementos en la lista en la pantalla.

Cambiar elementos dentro de una lista

Supongamos que tenemos un elemento en una lista, por ejemplo, un elemento numérico. Durante la ejecución de nuestro programa, es posible que queramos hacer más que solo acceder al valor que está almacenado en la lista. Es posible que queramos realizar operaciones más avanzadas y adaptadas a nuestras necesidades, como, por ejemplo, cambiar el valor en la lista. Afortunadamente, también tenemos la posibilidad de modificar los elementos en la lista de manera bastante sencilla. Aquí se muestra un ejemplo de código para ver cómo modificar elementos en una lista:

```
1  animales = {“Pollo”, “Vaca”, “Abeja”, “Cuchillo”}
2  animales[4] = “Jirafa”
3  for I = 1, 4, 1 do
4       print(animales[i])
5  end
```

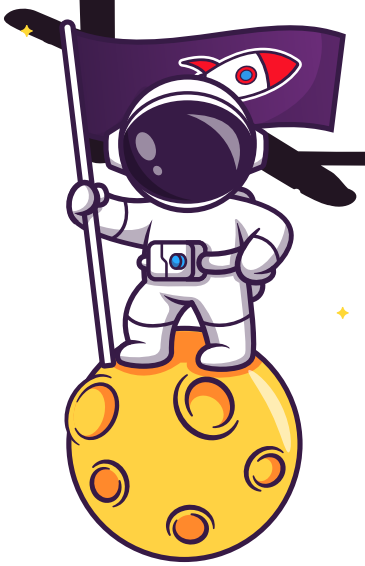
Al ejecutar este código, se obtiene la siguiente salida en la terminal:

Salida	Pollo Vaca Abeja Jirafa

Explicación del código	
1	En este código, creamos una lista llamada "animales" que contiene 4 elementos de tipo de cadena de texto: "Pollo", "Vaca", "Abeja", "Cuchillo".
2	Como se puede ver, el último elemento no es un animal, por lo que más tarde en el código cambiamos su valor. Finalmente, hacemos que el programa muestre en pantalla cada elemento en la lista.

En conclusión

Vimos que las tablas en Lua son una estructura muy útil que nos permite almacenar múltiples tipos de datos en una misma variable y acceder a ellos de manera sencilla a través de la indexación. Además, aprendimos cómo utilizar las tablas en Lua de manera práctica para resolver diferentes problemas, como el uso de operaciones aritméticas y diferentes tipos de datos en una tabla, agregar, eliminar y cambiar elementos dentro de una lista.



Ejercicios de práctica	
1	Crear un programa que agregue 1000 números aleatorios a una lista. Para que podamos desarrollar correctamente este programa, debemos tomar en consideración los siguientes pasos: 1 Creamos una variable que almacene una lista vacía. 2 Luego, utilizamos un ciclo para repetir 1000 veces el proceso de agregar nuevos números aleatorios a la lista. 3 Dentro del ciclo, utilizamos la función de generación aleatoria de números de la librería matemática de Lua para agregar un nuevo número a la lista.

2	Completar el siguiente código para que su ejecución muestre la salida esperada: La salida esperada es: 1 – 2 – 3 – 4 – 5 – 6 – 7 – El código incompleto es: 1 lista = {1, 2, 3, 4 2 for i = 1, 7, 1 do 3 io.write(lista[i] 4 e Es importante probar nuestra solución en Visual Studio Code para asegurarnos de que funciona correctamente.
3	Crear un programa que realice una serie de instrucciones específicas. Para hacerlo, debemos seguir estos pasos: 1 Primero, creamos una variable que almacene una lista vacía. 2 Luego, usamos un ciclo para agregar 10 elementos de tipo texto a la lista, cada uno de los cuales debe ser el valor de texto "A". 3 Finalmente, modificamos cada elemento de la lista mediante la concatenación para obtener los siguientes valores en el orden correcto: Elemento número 1 Adarce Elemento número 2 Aladar Elemento número 3 Arroz Elemento número 4 Albanado Elemento número 5 Atención Elemento número 6 Amonarse 4 Finalmente, mostramos cada elemento de la lista en pantalla mediante la función ipairs().

Capítulo 9

Paradigmas de la programación

Antes de profundizar en el uso de Lua, es fundamental comprender lo que es un paradigma de programación y cuál es su papel en el momento de elegir un estilo o metodología de programación.

Como puedes imaginar, la programación se enfoca en resolver problemas a través de la computadora y los programas que queremos que ejecute. Al igual que en la vida real, en el mundo de la programación existen diferentes tipos de problemas y, por lo tanto, diferentes soluciones.

Es aquí donde entran los paradigmas de programación. Aunque a menudo se refiere a un estilo de programación, la definición más precisa de los paradigmas de programación se refiere a la estructura general que puede tener un lenguaje de programación para ejecutar correctamente la estructura principal del programa.

Lua es principalmente un lenguaje de programación imperativo, lo que significa que está enfocado en resolver problemas de manera lineal y secuencial. Aunque es un lenguaje imperativo, Lua es consciente de las muchas ventajas que ofrece la programación funcional, lo que lo convierte en un lenguaje de programación multiparadigma.

Esto significa que Lua es capaz de manejar más de un paradigma en una misma estructura, ofreciendo una experiencia híbrida y flexible. Esto es muy útil a la hora de programar, ya que no nos limita a usar un solo tipo de paradigma para resolver un problema.

Antes de continuar, veamos brevemente los tres paradigmas de programación más populares y conocidos: imperativo, funcional y orientado a objetos. Lua es capaz de manejar todos estos tres paradigmas en un mismo programa, si es necesario.

Sin embargo, es importante comprender en profundidad cada uno de ellos antes de empezar a escribir código.

Paradigma de programación imperativo

Mencionamos previamente la importancia de estudiar los distintos paradigmas de programación, y ahora es momento de concentrarnos en el primer paradigma que vamos a explorar: la programación imperativa.

Este paradigma es uno de los más fáciles de entender para aquellos que están aprendiendo a programar, ya que muchos de los lenguajes de programación más populares y estandarizados en el mercado, como Python y Lua, utilizan este estilo. Además, es muy útil para resolver problemas mediante un enfoque lineal y algorítmico en lugar de tener que recurrir a conceptos más complejos de flujo de trabajo de código.

Para entender mejor cómo funciona este paradigma, podemos hacer una analogía con alguna actividad cotidiana que sigamos un paso a paso para solucionar un problema. De esta manera, podemos comparar la programación imperativa con un proceso lineal que nos ayuda a resolver un problema de manera eficiente.

Si queremos ir al supermercado a hacer compras, es más fácil si creamos una lista con todo lo que necesitamos comprar. De esta forma, podemos asegurarnos de no olvidar nada y de hacer las compras de forma eficiente.

Esta forma de hacer las compras se puede relacionar con el paradigma de programación imperativa. Ambos tienen como objetivo resolver un problema o lograr un resultado siguiendo una secuencia de pasos.

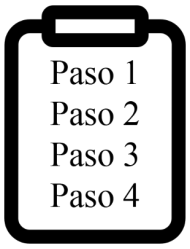


Ilustración 55 - Programación imperativa, puede verse como una serie de pasos

Otra característica importante de este paradigma es que podemos ver cómo se desarrollan los pasos en el programa gracias a su algoritmo y diagrama de flujo. Podemos seguir el flujo de ejecución del programa sin necesidad de saltar a ninguna parte diferente de la línea principal.

Paradigma de programación funcional

Es hora de explorar el segundo de los paradigmas de programación más populares y comúnmente utilizados en el mundo: el paradigma de programación funcional.

Como su nombre sugiere, en este tipo de programación, las funciones son la pieza clave. Las funciones son bloques de código que se pueden guardar y ejecutar simplemente llamándolas.

Es probable que ya estés familiarizado con el concepto de función, ya que también se usa en otras áreas, como la matemática. En matemáticas, una función puede verse como una máquina que toma una entrada y produce una salida. De manera similar, en la programación, una función normalmente recibe una entrada, realiza algún procesamiento y finalmente produce una salida basada en la entrada o en el procesamiento realizado.

La programación funcional es muy útil cuando necesitamos escribir código que debemos ejecutar varias veces sin tener que escribir las mismas líneas una y otra vez. El siguiente diagrama de flujo nos brinda una comprensión visual más clara de este concepto y cómo utilizarlo.

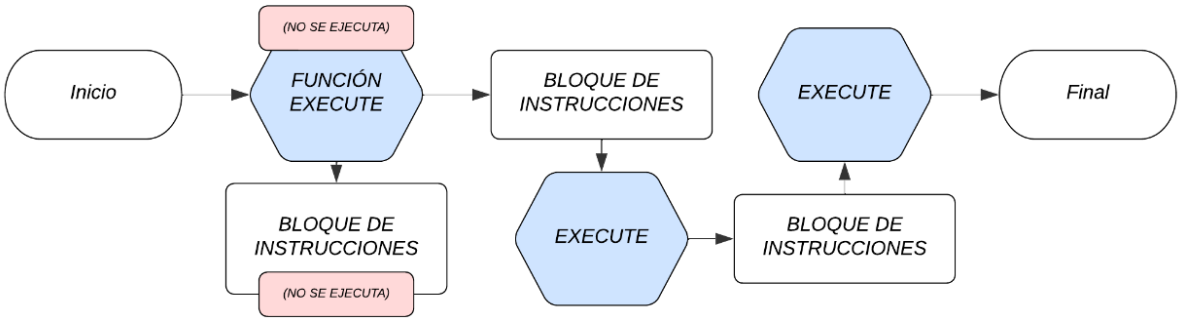


Ilustración 56 - Declaración de funciones y su ejecución

En este diagrama podemos ver cómo el paradigma de programación funcional utiliza funciones para su funcionamiento general. Podemos ver que dentro del bloque principal de ejecución del código se crean las funciones, pero no se ejecutan hasta que se llamen posteriormente en el mismo bloque de código.

Como se mencionó anteriormente, es importante declarar que una función es realmente una función antes de ejecutarla, de lo contrario, podríamos encontrar que el programa ejecuta el código del bloque sin querer, lo cual va en contra del concepto de la programación funcional. En la programación, una función es simplemente un programa que se ejecutará eventualmente. Podemos decir que en la programación y en las matemáticas, las funciones tienen la naturaleza de tener una entrada y una salida de datos.

Este concepto de entrada y salida de datos tiene un nombre en el mundo de la programación: input y output de datos. Este concepto se aplica específicamente a la entrada de valores y la salida generada por cada valor de entrada. En la programación, una función debe contener un bloque de código que permita realizar algún cambio o acción en base a la entrada de datos. Ahora que hemos explicado en detalle qué es una función en la programación, es hora de comparar y diferenciar este paradigma de programación con el paradigma imperativo de programación. Pero antes, es necesario aprender a usar estos dos paradigmas en Lua. Es hora de ponerse nuestros sombreros de programadores y empezar a aprender código en Lua.

Estructura simple del paradigma imperativo

Es hora de conocer los diferentes tipos de paradigmas de programación que podemos utilizar en Lua, comenzando con el paradigma con el que ya estamos familiarizados, ya que es el paradigma que hemos usado en la mayoría de los ejemplos y casos de estudio que hemos realizado en este libro y sus respectivos bloques de ejecución.

Como mencionamos anteriormente, ya explicamos los conceptos principales que rigen el paradigma de programación imperativo, por lo que es fácil entender su funcionamiento general debido a nuestra familiaridad con él. Aquí está un ejemplo de código que ilustra la estructura del paradigma imperativo:

```
1  numero1 = 2
2  numero2 = 5
3  print((numero1+numero2)+(numero1*numero2))
```

Al ejecutar este código, obtendremos la siguiente salida en la terminal de nuestro programa:

Salida	17

Explicación del código	
1	Este programa es similar a otros programas que hemos visto antes en este libro, ya que los ejemplos anteriores están escritos en el paradigma imperativo. Por lo tanto, ya tenemos cierta experiencia en usar este paradigma.
2	En este código, estamos usando dos variables: numero1 y numero2. La secuencia de lo que sucede dentro del código es lineal y se ejecuta paso a paso. Primero, se declara la variable numero1 y luego la variable numero2, lo que indica que está ocurriendo una secuencia.
3	Después de declarar ambas variables, el código indica que debemos mostrar en la pantalla el resultado de la operación, que es la suma de la suma de ambos valores y la multiplicación de los otros valores. Esto nos da como resultado 17, ya que 7 + 10 = 17.

Otro uso básico del paradigma funcional en Lua

Ahora, veremos el segundo paradigma que podemos utilizar en Lua. Este paradigma también es muy utilizado y ofrece muchas ventajas y posibilidades para nuestros programas y proyectos. Se trata del paradigma funcional.

Este paradigma puede ayudarnos a ahorrar tiempo y líneas de código en nuestros programas y proyectos. Aquí hay un ejemplo de código que demuestra su funcionamiento general:

```
1  function iva(valor)
2       return(valor*0.19)
3  end
4  producto = 10,000
5  print (“El impuesto IVA del producto es “ .. iva(producto))
```

Al ejecutar este código, obtendremos la siguiente salida en la terminal de nuestro programa:

Salida	El impuesto IVA del producto es 1900

Ahora, vamos a dar una breve explicación sobre nuestro código de ejemplo.

Explicación del código	
1	Creamos una función llamada iva() que recibe un parámetro llamado valor. Este parámetro representa el valor de un producto que queremos calcular el impuesto IVA.
2	La función iva() simplemente devuelve el 19% del valor recibido como parámetro.
3	Luego, creamos una variable llamada producto y le asignamos un valor numérico de 10,000. Esta variable es solo un ejemplo para ilustrar cómo podríamos usar la función iva() en un programa real.
4	Finalmente, imprimimos en pantalla el resultado de aplicar la función iva() a la variable producto.

Ejercicios de práctica	
1	Desarrollar una función para calcular la factorial de un número La factorial de un número es una operación matemática que implica multiplicar una secuencia de números naturales por debajo de ese número. En términos informáticos, podemos ver el cálculo del factorial como una iteración que disminuye hasta llegar a 1. Por ejemplo, el factorial de 6 es 720 (6 x 5 x 4 x 3 x 2 x 1 = 720), mientras que el factorial de 3 es 6 (3 x 2 x 1 = 6). Es importante tener en cuenta que debemos crear esta función utilizando funciones. La función debe recibir un solo parámetro y devolver el resultado de la operación.
2	Desarrollar un programa para calcular el valor del IVA de un producto Para este programa, debemos crear una función que reciba dos parámetros: el valor base del producto y el porcentaje de impuesto. La función debe devolver una cadena que indique el valor total a pagar, incluyendo el impuesto, por ejemplo: "El valor para pagar incluyendo el impuesto es $119". Podemos probar la solución con los siguientes ejemplos: >> print(iva(100, 19)) El valor para pagar incluyendo el impuesto es $119 >> print(iva(200, 100)) El valor para pagar incluyendo el impuesto es $400 >> print(iva(999, 0)) El valor para pagar incluyendo el impuesto es $999 Tenga en cuenta que este programa solo aceptará valores positivos naturales, por lo que debemos ignorar valores negativos, decimales y notación científica.
3	Crear un programa para obtener el n-ésimo número de la sucesión de Fibonacci La sucesión de Fibonacci es una secuencia de números donde cada nuevo número es la suma de los dos números anteriores. Debemos crear una función llamada "fibonacci" que reciba un solo parámetro, un número natural mayor que 2. Por ejemplo, el n-ésimo número de la sucesión de Fibonacci para n = 8 es 21 (1, 1, 2, 3, 5, 8, 13, 21), mientras que para n = 3 es 2 (1, 1, 2). La función debe devolver un valor numérico, no se debe mostrar toda la sucesión, solo el número en la posición indicada.
4	Crear un programa para calcular el área de un círculo. Para calcular el área de un círculo, debemos usar la siguiente fórmula: pi x r^2, donde pi es un número constante (aproximadamente igual a 3.14) y r es el radio del círculo. Debemos crear una función llamada "area_circulo" que reciba un solo parámetro, el radio del círculo. La función debe devolver el área del círculo como un valor numérico. Por ejemplo, si el radio es 5, el área del círculo sería: pi x 5^2 = 78.5.
5	Crear un programa para convertir grados Celsius a grados Fahrenheit Para convertir grados Celsius a grados Fahrenheit, debemos usar la siguiente fórmula: (C x 9/5) + 32, donde C es la temperatura en grados Celsius y F es la temperatura en grados Fahrenheit. Debemos crear una función llamada "celsius_a_fahrenheit" que reciba un solo parámetro, la temperatura en grados Celsius. La función debe devolver la temperatura en grados Fahrenheit como un valor numérico. Por ejemplo, si la temperatura en grados Celsius es 20, la temperatura en grados Fahrenheit sería: (20 x 9/5) + 32 = 68.

En conclusión

Comprender los paradigmas de programación es fundamental para poder elegir el estilo o metodología adecuados para resolver los diferentes tipos de problemas que se presentan en la programación. Lua es un lenguaje de programación multiparadigma capaz de manejar diferentes paradigmas de programación como el imperativo, el funcional y el orientado a objetos en un mismo programa, ofreciendo una experiencia híbrida y flexible. Es importante entender en profundidad cada uno de estos paradigmas antes de comenzar a escribir código. El paradigma de programación imperativo es uno de los más fáciles de entender, ya que utiliza un enfoque lineal y algorítmico para resolver problemas. Por otro lado, el paradigma de programación funcional utiliza funciones como la pieza clave, permitiéndonos ahorrar tiempo y líneas de código.

Capítulo 10

Uso de archivos con extensión .lua como librerías

Es hora de explorar una de las características poderosas de Lua: usar archivos externos para ejecutar funciones y usarlas como si estuvieran dentro de nuestro código. Podemos importar funciones de otros archivos .lua, y todas las funciones declaradas en ese archivo serán importadas y disponibles para ser llamadas. Esto es como crear nuestras propias librerías. Veamos un ejemplo gráfico:

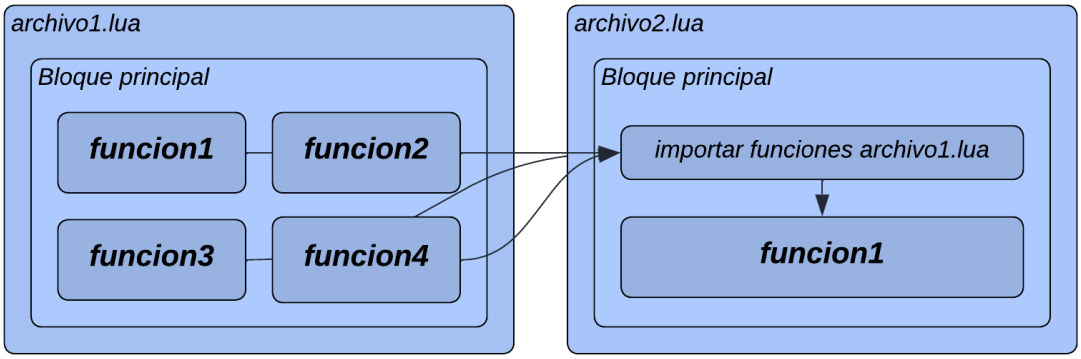


Ilustración 57 - Importar funciones entre documentos

Lua nos ofrece esta característica de manera fácil, aunque algunos otros lenguajes de programación también la tienen, pero con menos facilidad. Esto nos permite agregar más versatilidad a nuestros programas usando funciones de otros archivos, ahorrar tiempo escribiendo funciones que ya usamos anteriormente y tener un archivo con funciones básicas útiles para nuestros programas.

Sin embargo, para poder hacer uso de esta característica, debemos tener en cuenta que no podemos importar funciones de un archivo simple creado en Lua. Debemos asegurarnos de que el archivo sea un módulo, que contenga exclusivamente funciones. Explicaremos más sobre los módulos en Lua en el futuro.

La ventaja de usar módulos es que podemos tener un archivo con varias funciones, y trabajar con módulos en Lua es como crear nuestras propias librerías. Todo esto puede sonar complicado, pero con práctica y dedicación, lo podemos aprender. Verás muchos ejemplos prácticos de cómo usar esto en tus programas con bloques de código. Por eso, es hora de ver ejemplos de cómo usar módulos y qué tipo de funciones pueden tener en nuestros programas.

Importar funciones de un archivo externo en Lua con módulos

Es hora de aprender cómo utilizar los módulos en nuestros programas para crear nuestras propias bibliotecas personalizadas. Con estas bibliotecas, podremos aprovechar al máximo nuestras propias funciones. Antes de continuar, es importante que entendamos claramente qué son los módulos en nuestros programas. Usaremos estos módulos en nuestros ejemplos para crear diferentes funciones que luego podremos importar en diferentes archivos de Lua. Un módulo es un tipo de archivo escrito en Lua que tiene una estructura específica. Este tipo de archivos se caracteriza por su estructura, lo que los hace únicos y permite su funcionamiento. Veamos la siguiente ilustración para entender la estructura general de los módulos en Lua:

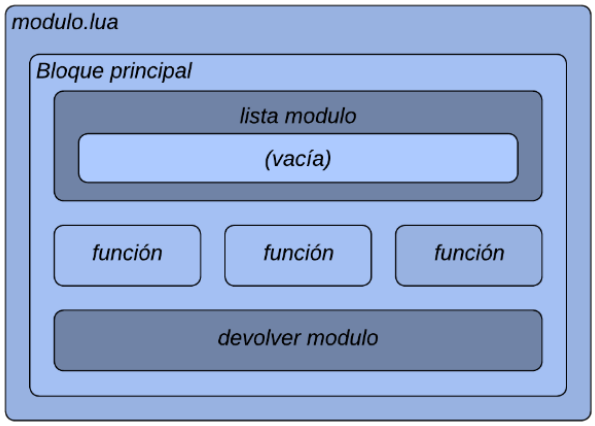


Ilustración 58 - Estructura de archivos de módulo en Lua

Es normal que al principio no entendamos completamente la ilustración, pero no hay que preocuparse. A continuación, explicaremos paso a paso con códigos ejemplos cómo funciona cada elemento que forma parte de un módulo en Lua. Veamos el siguiente ejemplo práctico para comprender mejor la estructura. Supongamos que creamos un módulo llamado archivo operaciones.lua. Aquí está el código de ejemplo que podríamos encontrar dentro de ese archivo:

```
1  local operaciones = {}
2  function operaciones.sumar(x, y)
3       return(x + y)
4  end
5  function operaciones.multiplicar(x, y)
6       return(x * y)
7  end
8  return operaciones
```

Como podemos ver, este archivo y su código siguen la estructura de un módulo en Lua. En este caso, tenemos un módulo con dos funciones: 'sumar' y 'multiplicar'. Estas funciones realizan las operaciones matemáticas que indican sus nombres. Es importante tener en cuenta que la nomenclatura de las funciones siempre comienza con el nombre del módulo, seguido de un punto y finalmente el nombre de la función junto a sus respectivos parámetros.

Es una buena práctica que el nombre de la lista interna del módulo y el nombre usado antes del nombre de la función correspondan al mismo nombre que el archivo donde almacenamos el código.

Veamos la siguiente ilustración para una mejor comprensión de los elementos de un módulo:

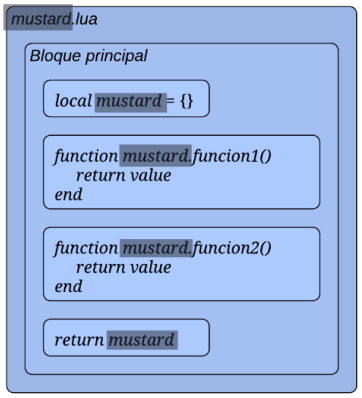


Ilustración 59 - Ejemplo de elementos de un módulo

Como se puede ver en la ilustración, la lista que forma parte del módulo debe ser de tipo local para evitar conflictos con las variables globales.

Ahora que entendemos cómo crear módulos y funciones dentro de ellos, es hora de ver cómo importar estas funciones a otros programas. Veremos cómo importar las funciones que definimos en el módulo al programa que estamos desarrollando y necesitamos utilizar estas funciones. Veamos un ejemplo de cómo importar un módulo en un programa con la siguiente ilustración:

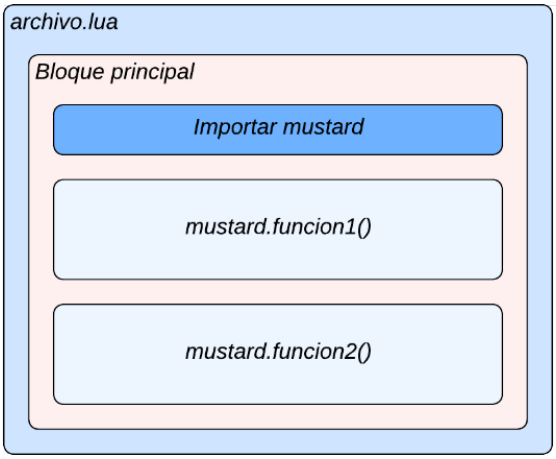


Ilustración 60 - Ejemplo de documento importando módulos

Como se puede ver, importar una biblioteca es muy sencillo. Solo tenemos que indicar que queremos importarla y luego llamar al módulo y especificar la función que queremos usar dentro de ella.

También es muy importante tener en cuenta que debemos seguir ciertas reglas para importar correctamente el módulo en nuestro programa y poder usar las funciones dentro de él. Una de estas reglas es que debemos usar la palabra reservada 'require' al momento de importar el módulo en nuestro programa. Además, es recomendable que ambos documentos estén en el mismo directorio para que el programa pueda acceder fácilmente al contenido del módulo y poder importarlo.

Veamos el siguiente gráfico para tener una mejor comprensión de lo que significa que ambos archivos estén en el mismo directorio:

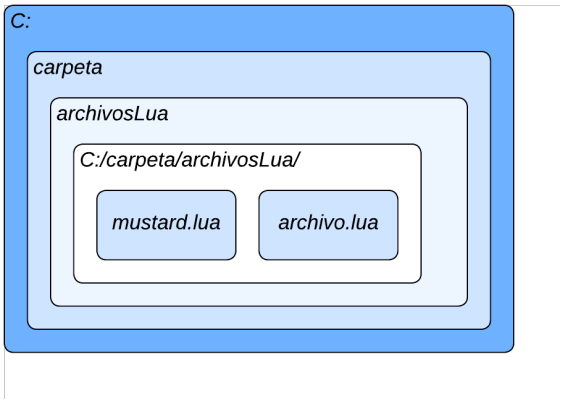


Ilustración 61 - Dos archivos dentro de un mismo directorio

Ahora que entendemos que ambos archivos deben estar en el mismo directorio, veamos un ejemplo práctico de código que podemos encontrar en nuestro segundo archivo, en este caso, llamado archivo.lua, donde exportaremos nuestras funciones. A continuación, veamos un ejemplo de código para importar funciones de un módulo:

```
1  require “mustard”
2  print(“Uso de la primera función: ” .. tostring(mustard.funcion1()))
3  print(“Uso de la segunda función: ” .. tostring(mustard.funcion2()))
```

Este ejemplo utiliza las funciones dentro del módulo 'operaciones.lua' que creamos anteriormente y las usa en nuestro programa. Aquí podemos ver una aplicación real de las funciones dentro de un módulo.

En resumen, importar funciones de un archivo externo con módulos en Lua es una técnica muy útil para crear librerías personalizadas y organizar el código de nuestro programa de una manera más eficiente. A través de esta técnica, podemos importar funciones de un módulo y usarlas en diferentes programas, lo que nos permite tener una estructura más clara y ordenada.

Podemos hacer lo mismo que mencionamos antes con el siguiente bloque de código, que nos sirve como ejemplo de cómo utilizar el módulo "operaciones.lua" que creamos anteriormente.

```
1  require “operaciones”
2  local valor1 = 17
3  local valor2 = 20
4  print(“Resultado de la suma: ” .. tostring(operaciones.sumar(valor1, valor2)))
5  print(“Resultado del producto: ” .. tostring(operaciones.multiplicar(valor1, valor2)))
```

Si ejecutamos este bloque de código desde el mismo directorio que nuestro módulo "operaciones.lua", entonces, obtendremos la siguiente salida en la pantalla:

```
Salida  Resultado de la suma: 37
        Resultado del producto: 340
```

Explicación del código	
1	En primer lugar, importamos el módulo que creamos anteriormente en nuestro programa. Lo hacemos con la palabra reservada "require" seguida del nombre del módulo escrito como una cadena de texto. De esta manera, importamos correctamente las funciones del módulo a nuestro programa.
2	Luego, creamos dos variables locales, llamadas "valor1" y "valor2". Estas variables contienen los valores numéricos 17 y 20, respectivamente. No hay una razón específica por la cual estas variables son locales, sólo que en este ejemplo relacionado con módulos, es una buena práctica seguir las buenas prácticas mencionadas en la sección sobre variables locales y globales.
3	Después, mostramos en la pantalla el resultado de usar las funciones dentro del módulo. Al analizar la salida del programa, podemos ver que la salida es la que debería ser, lo que demuestra que las funciones dentro del módulo se importaron correctamente a nuestro programa.

En conclusión

Aprendimos que el uso de archivos externos en Lua nos permite crear nuestras propias librerías, lo que nos ofrece más versatilidad en nuestros programas y nos permite ahorrar tiempo al no tener que volver a escribir funciones que ya hemos utilizado en el pasado. Sin embargo, para poder importar funciones de un archivo externo, debemos asegurarnos de que el archivo sea un módulo y contenga exclusivamente funciones. Además, es importante seguir ciertas reglas para importar correctamente el módulo en nuestro programa y poder utilizar las funciones dentro de él, como utilizar la palabra reservada "require" al momento de importar el módulo y asegurarnos de que ambos archivos estén en el mismo directorio.

Librerías y extensiones extra en Lua

Hasta ahora, hemos aprendido mucho sobre los conceptos básicos de Lua y la programación en general. ¡Es hora de dar un paso más y explorar aún más el mundo de Lua!

En esta sección, veremos qué son las librerías de Lua y cómo nos pueden ayudar a mejorar nuestro código. Las librerías nos brindan una sintaxis más rica y completa, con funciones adicionales que nos facilitan la programación y nos permiten realizar tareas complejas en pocas líneas de código.

Aunque ya hemos adquirido un conocimiento básico de Lua, la programación sigue avanzando y creciendo. Por ejemplo, ¿qué pasa si necesitamos un programa que genere un número aleatorio o que calcule el coseno de un ángulo determinado? Estas tareas podrían ser difíciles de realizar con solo los conceptos que hemos visto hasta ahora.

Pero ¡no te preocupes! Aquí es donde entran las librerías. Son paquetes incluidos por los desarrolladores de Lua para ayudarnos a realizar tareas complejas con pocas líneas de código. No vienen incluidas por defecto en el lenguaje porque podrían consumir mucha memoria y tener elementos que nunca se utilizarían.

Además, Lua nos permite usar librerías y paquetes creados por la comunidad, lo que significa que tenemos aún más posibilidades de desarrollo. En esta sección, tendremos la oportunidad de usar diferentes librerías y paquetes ofrecidos por el lenguaje para ampliar aún más nuestro conocimiento y realizar tareas más complejas. Estas librerías incluyen diferentes funcionalidades, como la capacidad de realizar operaciones matemáticas avanzadas.

Antes mencionamos que las librerías en Lua no vienen incluidas automáticamente. Debemos llamarlas y decir que las vamos a usar. Por suerte, los paquetes predeterminados de Lua suelen estar dentro de la descarga de los binarios, lo que significa que ya vienen con el lenguaje una vez que lo instalamos en nuestro sistema. Sin embargo, con librerías y paquetes de terceros, puede ser un poco más complicado. Primero debemos descargar la librería que queremos usar.

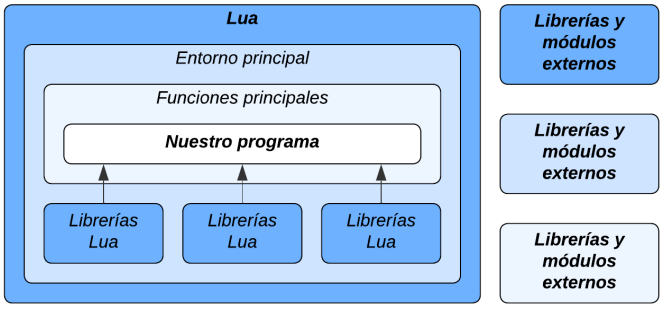


Ilustración 62- Papel de las librerías internas y externas al lenguaje

Como puedes ver en la ilustración anterior, las librerías internas y las externas tienen un papel diferente. Las librerías internas son aquellas que vienen con el lenguaje, mientras que las externas son aquellas que debemos descargar y usar.

Este concepto de usar paquetes para añadir funcionalidades adicionales al lenguaje de programación es bastante común en muchos lenguajes de programación diferentes. Cada lenguaje puede variar en cómo usar estas librerías, pero el concepto es el mismo.

Es hora de ver un ejemplo en acción de cómo usar una de las librerías de Lua. Vamos a empezar con la librería de manejo de listas y tablas. Como su nombre sugiere, esta librería nos permite hacer diferentes tipos de operaciones con nuestros datos en forma de listas que contienen múltiples otros datos.

Así que prepárate, porque vamos a escribir código. Esta sección es muy importante, así que asegúrate de prestar atención a todo lo que sucede en los siguientes ejemplos y explicaciones.

Capítulo 11

Uso de la librería table

Es hora de que nos adentremos en el uso de las librerías, para empezar, elegimos una que es fácil de usar: la librería table. Esta librería se encarga de manejar, controlar y ejecutar operaciones dentro de las listas o tablas en Lua.

Ya tuvimos la oportunidad de usar un poco la librería table, por ejemplo, cuando agregamos un nuevo elemento a una lista usando la función table.insert(). Esto significa que ya tenemos una idea de cómo funciona esta librería.

A lo largo de esta sección, nos enfocaremos en estudiar en detalle las diferentes operaciones que ofrece esta librería, incluyendo aquellas que consideramos fundamentales y las más comunes en la solución de problemas. Si no llegamos a ver alguna operación en particular, no hay problema, ya que también tendremos una tabla que incluye información, descripción y notación de las demás operaciones disponibles en la librería table.

Comencemos con el estudio de dos funciones dentro de esta librería. Ya hemos trabajado con estas funciones en ejemplos previos cuando estudiamos las listas y tablas en Lua, por lo que ya tenemos un poco de familiaridad con ellas.

El siguiente bloque de código es un ejemplo de cómo podemos hacer lo mencionado:

```
1  tablaDePrueba = {0, 100, 200, 400, 500, true}
2  table.remove(tablaDePrueba, 6)
3  table.insert(tablaDePrueba, 4, 300)
4  for i = 1, 6, do
5       print(tablaDePrueba[i])
6  end
```

De esta manera, al ejecutar este código, obtendremos la siguiente salida en la terminal de nuestro programa:

```
Salida  0
        100
        200
        300
        400
        500
```

Explicación del código	
1	Si, es posible que este tipo de código te resulte familiar, ya que hemos visto ejemplos de cómo trabajar con listas o tablas en Lua. Eso es porque estamos hablando de un tipo de datos importante.
2	¿Cómo esto se relaciona con las librerías que vamos a estudiar en Lua? Pues, aunque no lo parezca, está completamente relacionado. Ya que, cuando vimos ejemplos de operaciones con tablas en Lua, estábamos usando una librería, la librería table, que viene integrada con el lenguaje de programación para manejar estas listas.
3	En este código, creamos una tabla llamada "tablaDePrueba" y le decimos que tendrá 6 elementos: 5 números y un valor booleano.
4	Luego, usamos la librería table y su función "remove" para eliminar el valor booleano de la lista.
5	Agregamos un nuevo elemento a la tabla, un número, con el objetivo de que la tabla tenga solamente valores numéricos y que estos estén en un orden creciente.
6	Finalmente, mostramos los elementos de la tabla con una iteración for repetitiva. La salida será: 0, 100, 200, 300, 400 y 500.

Veamos otra operación que podemos hacer con la librería de manejo de tablas en Lua. En este caso, vamos a ver cómo podemos juntar los elementos de una lista en una sola cadena de texto.

Podemos hacer esto con la función "table.concat()". Esta función nos ayuda a generar una cadena de texto que incluya todos los valores que le indiquemos de la lista.

La estructura de la función es la siguiente:

Estructura table.concat([LISTA], [SEPARADOR], [INICIO], [FINAL])	
[LISTA]	La lista que queremos concatenar. Puede ser dada directamente o a través de una variable que contenga una lista.
[SEPARADOR]	Una cadena de texto que será el separador entre cada elemento de la lista al momento de generar la salida de la función.
[INICIO]	Un número que indica el índice de la lista desde el cual se comenzará a concatenar los elementos de la lista.
[FINAL]	Un número que indica el índice de la lista desde el cual se comenzará a concatenar los elementos de la lista.
Salida [CADENA]	
[CADENA]	Es la cadena de texto generada después de usar la función. Esta cadena es la concatenación de cada uno de los elementos de la lista.

Ahora, vamos a ver un ejemplo práctico de código donde podamos poner en práctica esta función de la librería de manejo de tablas en Lua.

Este bloque de código te puede ayudar como ejemplo:

```
1  tablaContenido = {“Tomate”, 17, 29, 101, -67, “Mundo”}
2  print(“Concatenación simple: ” .. table.concat(tablaContenido))
3  print(“Concatenación con separadores: ” .. table.concat(tablaContenido, “ – “))
4  print(“Concatenación con separadores y límites: ” .. table.concat(tablaContenido, “ – “, 2, 5))
```

Cuando corras este bloque de código, verás la siguiente salida en la terminal de tu programa:

```
Salida  Concatenación simple: Tomate1729101-67Mundo
        Concatenación con separadores: Tomate – 17 – 29 – 101 - -67 – Mundo
        Concatenación con separadores y límites: 17 – 29 – 101 - -67
```

Ahora, como hacemos en cada uno de los ejemplos de este libro, vamos a explicar lo que sucede en este bloque de código:

Explicación del código	
1	En primer lugar, declaramos una lista llamada tablaContenido en nuestro programa. Esta lista contiene los elementos que usaremos para crear la cadena de texto concatenada.
2	Luego, entra en acción la función table.concat() de la librería de manejo de tablas de Lua, indicando que queremos concatenar directamente todos los elementos de la lista.
3	Finalmente, usamos la función dos veces más, aplicando diferentes parámetros en cada ocasión. Una vez usamos separadores y en otra definimos un límite para los elementos de la lista que deben ser concatenados.

NOTA IMPORTANTE

Hay algunas limitaciones al usar la función table.concat(). Solo puedes concatenar elementos numéricos y de tipo texto en tu lista. Si necesitas usar otro tipo de datos, tendrás que convertirlos primero usando la función tostring().

Como mencionamos antes, hay muchas más funciones disponibles en esta librería, pero estudiarlas todas llevaría demasiado tiempo. Por eso, vamos a echar un vistazo a algunas de las otras funciones para que puedas verificar por ti mismo sus diferentes aplicaciones si es que te interesa.

La siguiente tabla te ayudará a entender mejor las funciones que tenemos en esta librería:

Librería *table*	
Función	Descripción
table.insert()	Agrega nuevos elementos a una lista.
table.remove()	Elimina elementos de la lista.
table.concat()	Concatena todos los elementos de una lista en una sola cadena de texto.
table.foreach()	Ejecuta una función en cada elemento de la lista.
table.sort()	Ordena los elementos de una tabla.

¡Y eso es todo! Con esto, podemos decir que hemos terminado de estudiar la librería de manejo de listas en Lua.

Capítulo 12

Uso de la librería de entrada y salida de datos

¡Es hora de conocer una librería que ya hemos visto antes! Se trata de la librería encargada de brindar funciones y métodos para la entrada y salida de datos en la terminal de nuestro programa.

Aunque podemos llamarla la librería de Entrada y Salida de Datos, es común encontrar que también se la conoce como la librería I/O. Esto se debe a que las letras "I" y "O" provienen de las palabras "input" y "output", que significan entrada y salida en inglés.

Aunque esto puede sonar confuso, no te preocupes. Ahora es el momento de ver algunos ejemplos prácticos.

```
1  texto1 = “Hola, ”
2  texto2 = “como te encuentras?”
3  io.write(texto1)
4  io.write(texto2)
```

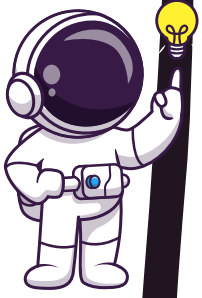
Al ejecutar este código, verás la siguiente salida en la terminal de tu programa:

```
Salida  Hola, como te encuentras?
```

Como de costumbre, vamos a revisar este ejemplo para asegurarnos de entenderlo bien. Repasar ejemplos como este es importante, incluso si ya los conocemos, para mantener nuestros conocimientos frescos y no olvidar las operaciones más básicas. Es importante destacar que, aunque no lo hayamos notado, ya hemos estado utilizando librerías de manera implícita en nuestros programas.

Explicación del código	
1	Creamos dos variables que almacenan valores de tipo texto, que son "Hola, " y "¿Cómo estás?". Luego, entra en acción nuestra librería principal, io.write(), que nos brinda una función para escribir en pantalla sin tener que hacer un salto de línea.
2	Al ejecutar este código, veremos como resultado en la terminal "Hola, ¿Cómo estás?".

¡Eso es todo! Con este ejemplo sencillo, hemos terminado de explorar cómo funciona la librería de Entrada y Salida de Datos en Lua.



En conclusión

Las librerías y extensiones extra en Lua son fundamentales para mejorar nuestro código y realizar tareas más complejas en menos líneas de código. Lua nos brinda la oportunidad de usar librerías creadas por la comunidad, lo que amplía aún más nuestras posibilidades de desarrollo. Es importante destacar que, aunque algunas librerías ya vienen incluidas en el lenguaje, otras deben ser descargadas y utilizadas por separado. En particular, la librería de manejo de tablas nos permite realizar diversas operaciones en listas y tablas, mientras que la librería de entrada y salida de datos nos brinda funciones para escribir y leer datos en la terminal de nuestro programa.

Capítulo 13

Lua y la librería matemática

Es hora de profundizar en una de las librerías más interesantes que Lua tiene para ofrecer. La librería matemática de Lua es una herramienta poderosa que permite a los programadores resolver problemas matemáticos complejos de manera sencilla. Esta librería es muy completa y cuenta con muchas funciones avanzadas y útiles. Sin embargo, para mantener las cosas simples, solo estudiaremos aquellas funciones que nos brinden un conocimiento general y nos permitan abordar diferentes escenarios.

Si deseas conocer todas las funciones que la librería matemática de Lua tiene para ofrecer, siempre puedes consultar la documentación oficial del lenguaje. De lo contrario, estudiar todas las funciones en profundidad podría resultar extenso. Ahora, es el momento de ver algunos ejemplos de lo que podemos hacer con esta librería. Descubrirás que es muy útil en muchas situaciones.

Uso de la librería matemática para generar número aleatorios

Una de las funciones más importantes de la librería matemática de Lua es la capacidad de generar números aleatorios. Podemos hacerlo fácilmente usando la función math.random(). A continuación, te muestro un ejemplo de código que ilustra su uso:

```
1  print(math.random())
2  print(math.random(100))
3  print(math.random(100))
4  print(math.random(10, 20))
```

Al ejecutar este código, obtendrás la siguiente salida en la terminal de tu programa:

```
Salida  0.60145073275656
        65
        83
        13
```

En este bloque de código, mostramos cómo generar números aleatorios en diferentes situaciones utilizando la librería matemática de Lua.

Explicación del código	
1	Primero, pedimos al programa que muestre números aleatorios sin ningún tipo de parámetro. De esta manera, obtenemos un número completamente aleatorio.
2	En el segundo y tercer caso, usamos la misma función con un solo parámetro numérico que indica el límite superior de los números aleatorios a generar. Por ejemplo, si usamos el valor 100, el programa generará un número aleatorio en el rango [0, 100].
3	En el último caso, hacemos uso de dos parámetros numéricos en la función para definir el rango de los números aleatorios a generar. Por ejemplo, si usamos los parámetros 10 y 20, el programa generará un número aleatorio en el rango [10, 20].

En este libro, aprenderemos sobre muchas más funciones y posibilidades que ofrece la librería matemática de Lua, pero, por ahora, esta es solo una simple introducción a esta poderosa herramienta.

Cálculo de valores absolutos con la librería matemática

Es hora de aprender una de las funciones más básicas que nos ofrece la librería matemática en este hermoso lenguaje de programación: el cálculo de valores absolutos.

Este libro está diseñado para que cualquier persona pueda entender los conceptos que se enseñan, por lo que dedicaremos una sección exclusiva a cada concepto que pueda resultar confuso para el público sin una formación académica extensa o para lectores jóvenes. La pregunta importante es: ¿Qué es el valor absoluto de un número? La respuesta es más sencilla y lógica de lo que parece. Como sabemos, el conjunto de los números es muy amplio e incluso infinito. Podemos dividir este tipo de conjuntos en dos categorías que serán útiles para este concepto: los números positivos y los números negativos.

Los números positivos son aquellos que son mayores que el número cero (0), mientras que los números negativos son aquellos que son menores que el cero (0). El valor absoluto es simplemente la distancia de un número desde el cero, pero como un número positivo. En otras palabras, la función de valor absoluto convierte cualquier número, ya sea positivo o negativo, en un número positivo de la misma magnitud. Por ejemplo, si calculamos el valor absoluto de los números -17, 0, 15, -4.5 y -100, sus valores absolutos serían 17, 0, 15, 4.5 y 100 respectivamente.

Este trozo de código puede ayudarnos a entender de qué estamos hablando:

```
1  numeros = {-5, 4, -17}
2  for i = 1, 3, 1 do
3  numeros[i] = math.abs(numeros[i])
4  Io.write(numeros[i] .. “ “)
5  end
```

Al correr este código, obtenemos lo siguiente como resultado:

```
Salida  5  4  17
```

Veamos una explicación más sencilla del código:

Explicación del código	
1	Creamos una lista llamada "numeros" con los números -5, 4 y -17.
2	Usamos un ciclo "for" para convertir cada número en su valor absoluto.
3	Finalmente, imprimimos los nuevos valores en pantalla.

Grados y radianes dentro de Lua para el uso de operaciones trigonométricas

Vamos a explorar un concepto clave cuando trabajamos con bibliotecas matemáticas y operaciones que requieren trigonometría en Lua.

Al medir ángulos, podemos usar diferentes formas de hacerlo, dependiendo de las unidades de medida de ángulo que utilicemos en nuestro programa. Estas unidades pueden afectar el resultado final, por lo que es importante especificar cuál estamos usando para que el programa pueda realizar los cálculos correctamente. Las dos unidades de medida de ángulo más populares e importantes son los grados y los radianes.

En Lua, dependiendo del tipo de datos que utilicemos, podemos obtener resultados diferentes. Sin embargo, también es posible usar funciones dentro del programa para hacer conversiones entre unidades de medida.

Es importante comprender cómo funcionan estas relaciones y cómo hacer estas conversiones. Este libro no cubre de manera detallada la explicación matemática y física detrás de estas relaciones y conversiones, pero usaremos funciones que nos facilitarán mucho el trabajo.

Como mencionamos antes, la mayor parte del proceso de conversión de datos para medir ángulos en Lua se puede realizar con funciones de bibliotecas matemáticas. Aquí hay un ejemplo sencillo donde podemos usar estas funciones para convertir grados a radianes en nuestro programa.

Conversión de unidades de grados a radianes

Veamos cómo convertir grados a radianes con un ejemplo.

```
1  grados = 90
2  print("La equivalencia de " .. grados .. "grados a radianes es igual a " .. math.rad(grados) .. " radianes.")
```

Al ejecutar este programa, veremos la siguiente salida en la pantalla de nuestra terminal:

```
Salida  La equivalencia de 90 grados a radianes es igual a 1.5707963267949 radianes.
```

Explicación del código	
1	En este código, establecemos que la cantidad de grados es de 90. Es importante saber que, en matemáticas, 90 grados equivale a pi/2 radianes. Por lo tanto, el resultado que muestra la salida de nuestro programa es correcto.

Conversión de unidades de radianes a grados

Veamos ahora un ejemplo para convertir radianes a grados.

```
1  radianes = math.pi
2  print("La equivalencia de " .. radianes .. " radianes a grados es igual a " .. math.deg(radianes) .. " grados.")
```

Al ejecutar este bloque de código, obtendremos la siguiente salida:

```
Salida  La equivalencia de 3.1415926535898 radianes a grados es igual a 180.0 grados.
```

Ahora, veamos una breve explicación de nuestro código:

Explicación del código	
1	En este código, creamos una variable y le asignamos el valor de pi. Luego, usamos la función math.deg() para convertir este valor a grados. Podemos verificar que el resultado corresponde a la equivalencia entre radianes y grados.

Usar el valor de pi con la librería matemática

La librería matemática en Lua nos permite trabajar con problemas trigonométricos y usar funciones trigonométricas. Para esto, necesitamos el valor de pi, que está disponible en la librería matemática como una función fácil de usar. Veamos un ejemplo:

```
1   print(math.pi)
```

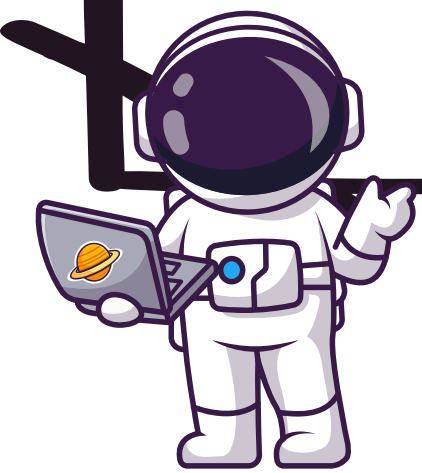
Al ejecutar este bloque de código, obtendremos la siguiente salida:

```
Salida   3.1415926535898
```

Como podemos ver, la función math.pi nos devuelve el valor de pi. Será de gran ayuda en los ejercicios relacionados con la trigonometría y problemas con círculos.

En conclusión

La librería matemática de Lua es una herramienta poderosa que nos permite resolver problemas matemáticos complejos de manera sencilla. Aprendimos que, si bien esta librería es muy completa y cuenta con muchas funciones avanzadas y útiles, solo necesitamos conocer aquellas que nos brinden un conocimiento general y nos permitan abordar diferentes escenarios. Vimos cómo la función math.random() nos permite generar números aleatorios, mientras que la función math.abs() nos permite calcular los valores absolutos de un número. También exploramos cómo Lua maneja las unidades de medida de ángulo y cómo podemos hacer conversiones entre grados y radianes usando las funciones math.rad() y math.deg(). Finalmente, aprendimos que la función math.pi nos permite trabajar con problemas trigonométricos y usar funciones trigonométricas.



Operadores trigonométricos en Lua

Es hora de estudiar y trabajar con problemas relacionados con la trigonometría en Lua. Aunque estos problemas pueden tener un enfoque matemático, los conceptos que usaremos son fáciles de entender. Antes de empezar, es importante conocer la trigonometría y su relación con Lua para poder resolver problemas. En resumen, la trigonometría es el estudio de los triángulos y las relaciones trigonométricas en diferentes situaciones. Estas relaciones son las que usaremos en nuestros programas en Lua. Veamos algunos ejemplos de cómo podemos aplicar estas funciones trigonométricas en Lua para tener una idea más clara de lo que podemos hacer con ellas.

Uso de las funciones trigonométricas principales en Lua

Es hora de echar un vistazo a algunos ejemplos de código que utilizan las diferentes funciones trigonométricas de la librería matemática de Lua. Aquí tienes un bloque de código de ejemplo:

```
1  pi = math.pi
2  print(math.sin(pi))
3  print(math.cos(pi))
4  print(math.tan(pi))
```

Y al ejecutar este código, se obtiene la siguiente salida:

```
Salida  1.2246063538224e-016
        -1.0
        -1.2246467991474e-016
```

Veamos una breve explicación del código:

Explicación del código	
1	Empezamos creando una variable llamada pi, que almacena el valor de math.pi. Esto nos permitirá usar pi en lugar de escribir math.pi cada vez que necesitemos su valor.
2	Luego, hacemos uso de las tres funciones trigonométricas principales: seno, coseno y tangente. Sin embargo, en la salida de nuestro programa, podemos ver que el valor de seno y tangente no es exactamente 0, pero es muy cercano.
3	La razón de que estos valores no sean exactamente 0 es debido a que math.pi no proporciona todos los decimales de pi. Ya que pi es un número con un número infinito de decimales que no siguen un patrón, el cálculo es un poco inexacto, pero obtenemos un resultado muy cercano al esperado.

Calcular longitudes de un lado del triángulo usando la librería matemática

Es hora de ver nuestro primer ejemplo de cómo aplicar las funciones matemáticas de trigonometría a diferentes problemas geométricos, numéricos y físicos.

Comenzaremos con un ejemplo sencillo, en el que usaremos operaciones para calcular la longitud de un lado de un triángulo equilátero.

Este es un ejemplo de código que te puede ayudar:

```
1  local angulo = 40
2  print("El ángulo del problema es de 40 grados.")
3  print("Introduce la longitud del cateto: ")
4  local cateto = io.read("*n")
5  local hipotenusa = cateto/(math.sin(math.rad(angulo)))
6  print("La hipotenusa es igual a " .. hipotenusa .. " unidades.")
```

Al ejecutar este código con la entrada de 17, obtendrás la siguiente salida en la pantalla de la terminal:

```
Salida  El ángulo del problema es de 40 grados.
        Introduce la longitud del cateto: 17
        La hipotenusa es igual a 26.447305056627 unidades.
```

Vamos a explicar brevemente el código para que lo entiendas mejor:

Explicación del código	
1	Al principio, creamos una variable numérica llamada angulo y le asignamos el valor de 40. Esto hace referencia a un caso hipotético en el que estamos tratando de visualizar un triángulo rectángulo con uno de sus ángulos de 40 grados.
2	Luego, mostramos en pantalla la información y pedimos al usuario que introduzca la longitud del cateto.
3	Finalmente, convertimos el valor del ángulo a radianes y usamos funciones trigonométricas para calcular la hipotenusa, mostrando el resultado en pantalla.

Con esto, concluimos nuestro primer ejemplo de código para aplicar las funciones trigonométricas en Lua.

En conclusión

En este capítulo vimos cómo podemos aplicar las funciones trigonométricas en Lua para resolver problemas relacionados con la trigonometría. Aprendimos que la trigonometría es el estudio de los triángulos y las relaciones trigonométricas en diferentes situaciones, y que estas relaciones son las que usamos en nuestros programas en Lua. También vimos algunos ejemplos de cómo podemos utilizar las funciones trigonométricas principales de la librería matemática de Lua, y aprendimos a calcular longitudes de un lado del triángulo utilizando la librería matemática.

Capítulo 14

Administración de documentos en Lua

Hemos trabajado con librerías antes, como la librería matemática para resolver problemas matemáticos avanzados. Ahora, vamos a ver una segunda librería útil en Lua: una librería que nos permite abrir, crear, guardar y modificar documentos. ¿Cómo es posible? Lua nos ofrece la posibilidad de interactuar con nuestro sistema, lo que nos da una gran ventaja sobre solo usar partes del propio lenguaje para solucionar problemas.

Este avance nos permite realizar muchas más funciones fuera del propio interpretador de Lua, como crear archivos dentro del programa y editarlos, eliminarlos, copiarlos, entre otras cosas.

A continuación, veremos un ejemplo de cómo usar estas funciones para manejar archivos dentro de Lua. Estas funciones son parte del manejo de archivos integrado en el lenguaje de programación Lua.

Librería I/O para manejo de archivos en Lua

Como mencionamos antes, es hora de usar la librería I/O para administrar y manejar archivos en Lua. Esta librería nos ofrece funciones como la creación de nuevos documentos, la lectura de documentos y la edición de documentos, entre otras cosas.

Además, usar esta librería será útil para construir un conocimiento básico sobre cómo funcionan las bases de datos en Lua.

Uso de la función io.open()

Continuemos con nuestro estudio sobre la librería I/O en Lua y, en particular, la función io.open(). Como su nombre indica, esta función principalmente abre documentos dentro de nuestro programa. Sin embargo, también podemos usarla para crear nuevos documentos y modificar documentos existentes.

Revisemos el bloque de código que usamos en la sección anterior para crear un nuevo documento:

```
1  documento = io.open(“prueba.txt”, “w”)
2  documento:write(“Hola mundo, esto es solamente una prueba”)
3  documento:close
```

Veamos más de cerca cómo funciona la función io.open() en nuestro código.

io.open(“prueba.txt”, “w”)	
io	Indica que estamos usando una función de la librería I/O de Lua.
open()	Este es el nombre de la función en cuestión.
“prueba.txt”	Este es el primer parámetro que usamos en la función, hace referencia al nombre que recibirá nuestro archivo cuando sea creado.

“w”	Este es el segundo parámetro, indica el método que usaremos para abrir el archivo mencionado en el primer parámetro.

Es importante tener en cuenta las diferentes opciones que tenemos con el uso de estos parámetros de cadena de texto y comprender lo que significan dentro de la función io.open() o cualquier otra función que use estos mismos parámetros.

En un futuro, profundizaremos más en los distintos métodos que podemos usar para abrir archivos con la función io.open().

La partícula de cadena de texto “w” en la función io.open() representa la acción de escribir en el archivo. Esta función creará un archivo con el nombre indicado en el primer parámetro en el directorio local donde se encuentra el archivo ejecutando las instrucciones de Lua. Si no existe un archivo con el mismo nombre en el directorio, entonces la función creará un archivo vacío y escribirá información en él posteriormente.

Es importante tener una comprensión más amplia de los diferentes métodos que se pueden usar dentro de la función io.open() y las tareas que estos métodos pueden realizar, como editar un archivo con información existente, solo leer un archivo, entre otras funciones. Con esta base de conocimiento, podemos aprovechar al máximo las capacidades de la librería I/O para manejar archivos en Lua.

En la siguiente tabla, podemos ver las diferentes cadenas de texto que podemos usar para indicar a nuestra función cómo abrir archivos con la función io.open(). Con estas opciones, tenemos una base sólida para elegir cómo queremos trabajar con archivos en nuestros programas.

Partículas de cadena de texto para io.open()	
Estas son pequeñas cadenas de texto que usamos dentro de la función para indicar el método o modo que queremos usar para abrir documentos. Son importantes porque forma parte de la librería de entrada y salida de datos de Lua. Estas partículas de cadena de texto se pasan como segundo parámetro en la función io.open() y nos permiten decirle a la función qué tarea queremos realizar, como crear documentos, leerlos, modificarlos o agregar nueva información.	
w – “Write”, escritura	Este modo indica que se usará para crear nuevos archivos.
r – “Read”, lectura	Este modo indica que se usará para leer la información dentro de un documento existente sin modificarlo.
a – “Append”, concatenación	Este modo indica que se mantendrá la información dentro del documento y se agregará nueva información como si fuera una concatenación.
b – “Binary”, binario	Este modo indica que se trabajará con el archivo en su formato binario dentro de nuestro código.
r+ - “Read update”, actualización conservante	Este modo indica que se actualizará el documento manteniendo la información anterior.
w+ - “Write update”, actualización no conservante	Este modo indica que se actualizará el documento, pero eliminando la información anterior.
a+, - “Append update”, actualización en concatenación	Este modo es similar al modo de apéndice que mencionamos anteriormente.

Ahora que conocemos las diferentes partículas de cadena de texto, podemos ver las diversas maneras en que podemos usar la función io.open en nuestros programas.

Aunque no nos detendremos en ejemplos específicos en las secciones siguientes, es útil tener estos métodos en mente cuando queramos realizar tareas similares en nuestros proyectos. Además, practicar con estos métodos en programas en Lua fortalecerá nuestra comprensión y habilidades como programadores, y nos ayudará a solucionar otros problemas y mejorar nuestro desempeño en el futuro.

Abrir archivos usando Lua

Para este ejemplo, crearemos dos archivos .lua en el mismo directorio. Uno de ellos contendrá la información que intentaremos leer, es decir, será el archivo que abriremos en nuestro código secundario. Normalmente, los archivos que podemos leer con esta librería son aquellos que contienen texto sin formato, como archivos .txt o archivos de código de Lua con extensión .lua. En este ejemplo, usaremos un archivo de extensión .lua para leer su contenido.

Creamos el primer documento con la siguiente información:

```
1    Hola, esto es solamente un texto de prueba
```

Este archivo solo contiene texto que leeremos posteriormente con el código que escribiremos en el segundo documento. Después de crear el documento, lo llamaremos "documento1.lua" y aseguraremos que ambos documentos estén en el mismo directorio para evitar problemas al buscar el archivo.

El segundo documento tendrá las instrucciones principales que ejecutaremos para leer el contenido del primer documento. Cuando llamamos un archivo con solo su nombre de forma indirecta, le estamos indicando al programa que este archivo se encuentra en la misma carpeta que el archivo con el código de Lua que debe ser ejecutado. Por esta razón, ambos documentos deben estar en el mismo directorio y tener nombres diferentes y sin caracteres especiales.

Ahora es el momento de crear el segundo archivo, que tendrá el código de Lua que nos permitirá leer la información del primer documento. Este documento tendrá código de Lua y utilizará la librería de entrada y salida de datos.

Este es el código que escribiremos en el segundo documento:

```
1    archivo = io.open(“documento1.lua”, “r”)
2    print(io.read(io.input(archivo)))
```

Así, al ejecutar este bloque de código obtenemos la siguiente salida en la pantalla de la terminal de nuestro programa:

```
Salida    Hola, esto es solamente un texto de prueba
```

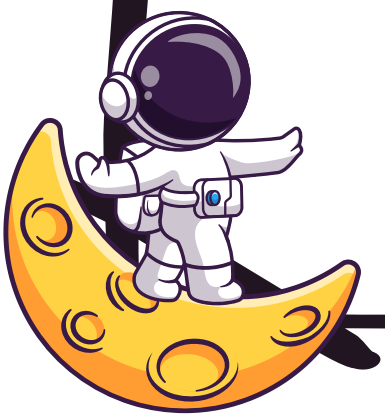
Es hora de profundizar en el código que acabamos de escribir. Hemos estado explicando el funcionamiento de este ejemplo a lo largo de su desarrollo, pero es importante profundizar en todo lo que está sucediendo en el programa. Aunque este tipo de ejemplos pueden ser simples, todavía es valioso hacer un análisis detallado para mejorar nuestra comprensión y habilidades como programadores.

Antes que nada, es importante mencionar que creamos un documento que contiene información relevante para nuestro código. Este documento es donde podemos encontrar la mayoría de la

información que necesitamos para agregar a nuestro programa. Aunque el documento tenga una extensión similar a un archivo de Lua, no significa que toda su información esté escrita en este lenguaje de programación. De hecho, puede haber cualquier tipo de información textual en el documento que nos resulte útil más adelante.

Si deseamos leer este documento y ejecutarlo con nuestro intérprete de lenguaje de programación, es probable que no obtengamos ningún resultado relevante, ya que solo hay información textual y no código escrito en el lenguaje de programación.

Sin embargo, si hay algún código escrito en Lua dentro del documento, el programa también leerá la información textual sin tener que ejecutar el código. La función de esta librería es solo manejar la entrada y salida de datos, por lo que, independientemente de lo que haya en el archivo, será tratado como solo un archivo de texto sin importar si hay órdenes escritas en Lua.



En conclusión

Vimos cómo la librería I/O nos permite administrar y manejar archivos en Lua de manera eficiente. La función io.open() es una herramienta esencial que nos permite abrir, crear y modificar documentos en nuestros programas. Es importante comprender las diferentes opciones de cadena de texto que podemos usar en la función io.open() para indicar cómo queremos abrir nuestros archivos. Además, hemos aprendido cómo leer información de un archivo existente y cómo podemos utilizarla en nuestro código de Lua.

Apertura de archivos en Lua que contienen instrucciones.

Como mencionamos antes, es hora de ver algunos ejemplos donde podemos tener un archivo que contenga tanto información textual como código escrito en Lua. Estos ejemplos nos ayudarán a entender que, independientemente del tipo de archivo y de la información que haya en él, el programa lo tratará como solo una entrada de texto y no hará caso a las instrucciones que pueda haber dentro del archivo.

Antes de empezar con estos ejemplos, es importante tener claro algunos conceptos sobre el manejo de archivos en Lua.

Empecemos creando un archivo que contenga instrucciones escritas en Lua. Lo llamaremos "documento.lua" y tendrá la siguiente información:

```
1    print(“Hola mundo”)
```

Como podemos ver, este archivo contiene información escrita como instrucciones de Lua, pero, como mencionamos antes, el objetivo no es que el programa lea este archivo y ejecute las instrucciones escritas en Lua. En su lugar, necesitamos que el programa lo trate como solo texto.

A continuación, crearemos otro archivo que contenga las instrucciones que debemos darle al programa para que lea la información en el primer archivo que creamos. Este archivo debe estar en el mismo directorio que el primer archivo y debe tener un nombre diferente. Lo llamaremos "instrucciones.lua".

Este archivo tendrá las siguientes instrucciones:

```
1    info = io.open(“documento.lua”, “r”)
2    print(io.read(io.input(info)))
```

Al ejecutar este código, obtendremos la siguiente salida en la terminal de nuestro programa:

```
Salida    print(“Hola mundo”)
```

Ahora, como de costumbre, vamos a explicar lo que sucede en nuestro código.

Explicación del código	
1	Primero, creamos un archivo que solo tiene una línea de "print("Hola mundo")", lo que en teoría le pide al programa que muestre "Hola mundo" en la pantalla. Sin embargo, esta instrucción no es procesada por el programa, ya que lo considera como parte del texto y no como una función a ejecutar.
2	Luego de crear este archivo, lo guardamos en un directorio donde también guardaremos el segundo archivo que crearemos. Esto lo hacemos para poder referirnos a este archivo sin tener que especificar su ruta completa. Solo necesitamos usar su nombre para hacer referencia a él.
3	Finalmente, creamos un segundo archivo donde escribiremos las instrucciones que queremos que ejecute el programa. En este caso, la tarea es simplemente leer el contenido del primer archivo que creamos sin problemas.

Leer información dentro de un documento sin extensión .lua

Como mencionamos antes, también podemos utilizar la misma función para leer información en documentos que no sean solo códigos escritos en el lenguaje de programación Lua o que no tengan la extensión .lua.

Esta es una de las características más importantes y útiles, ya que no es necesario que el documento tenga la extensión .lua para poder ser leído. Estos documentos pueden tener otras extensiones que contengan solo texto puro, es decir, que no sean otro tipo de documento como imágenes o audio.

Algunos ejemplos de otras extensiones que podemos usar en los documentos que leeremos con Lua incluyen .txt, .cpp, .py, entre otros.

Entender este concepto es sencillo, ya que se aplica no solo a Lua sino también a otros lenguajes de programación. Por ejemplo, el tipo de archivo universal para manejar texto sin formato es el .txt, por lo que es lógico que Lua también ofrezca la posibilidad de manejar archivos con esta extensión.

Es hora de ver algunos ejemplos prácticos en Lua del concepto que estamos discutiendo. A veces, ver algunos ejemplos es fundamental para entender por completo un tema en programación. Para este ejemplo, usaremos dos archivos. Es importante tener en cuenta que estos dos archivos deben estar en la misma carpeta para evitar problemas al momento de hacer referencia a uno de ellos.

Comenzaremos creando nuestro primer archivo. Este primer archivo no tendrá la extensión .lua, pero estará escrito en formato de texto universal estandarizado, es decir, con la extensión .txt.

A continuación, la información dentro de este archivo:

```
1    Esto es solamente un texto de prueba dentro de un documento
```

Después de escribir en este documento, es hora de guardarlo y continuar con la siguiente parte.

Ahora, crearemos un segundo documento. Este documento tendrá todo el código escrito en Lua, incluyendo todas las instrucciones que deseamos que la computadora ejecute. El código dentro de este documento será igual al que usamos antes:

```
1    archivo = io.open("info.txt", "r")
2    print(io.read(io.input(archivo)))
```

Al ejecutar este código, obtendremos la siguiente salida en la terminal de nuestro programa:

```
Salida    Esto es solamente un texto de prueba dentro de un documento
```

Ahora, siguiendo con nuestra costumbre en este libro, vamos a ver una explicación breve de lo que sucede en nuestro código:

Explicación del código	
1	Esta sección del programa tiene la misma explicación que la que vimos anteriormente.
2	Podemos ver que para leer la información dentro de un documento externo al que está ejecutando el código en Lua, no es necesario que este documento tenga una instrucción en Lua o tenga la extensión .lua.
3	Lua nos permite trabajar con diferentes tipos de documentos que contengan información textual, gracias a la librería de entrada y salida de datos.
4	Este concepto puede ser muy útil en el mundo real de la programación laboral, donde es común recibir un archivo previamente escrito con una extensión de texto distinta a .lua en diferentes situaciones."

Crear documentos usando Lua

Hemos aprendido y practicado cómo leer información en un documento, pero Lua ofrece muchas más posibilidades para manejar archivos. Es hora de ver cómo podemos usar Lua para crear nuevos documentos que contengan la información que queremos agregar en el programa. Esto puede ser muy útil cuando necesitamos crear archivos que contengan información generada internamente en nuestro programa según las instrucciones y condiciones especificadas por nosotros mismos.

Es importante tener en cuenta la importancia de los directorios y carpetas donde estemos al momento de crear el documento. El directorio en el que trabajemos con el documento que contiene las instrucciones para la computadora será considerado como un directorio predeterminado.

Si queremos crear un archivo en un directorio diferente al predeterminado, debemos asegurarnos de que el directorio exista en nuestra computadora local y conocer su ruta.

Este concepto puede ser trivial para algunas personas, así que es hora de hacer algunos ejemplos prácticos en Lua.

A diferencia de los ejemplos anteriores, esta vez no necesitamos crear dos documentos. Usaremos solo un documento que contenga toda la información e instrucciones escritas en Lua para crear el documento externo.

Este documento no debe tener un nombre específico, solo es importante tener en cuenta que el nombre que le asignemos debe ser diferente al nombre del nuevo documento que crearemos posteriormente en nuestro código. Por ejemplo, si le llamamos "instrucciones.lua", el nuevo archivo no debería llamarse "instrucciones.lua" para evitar errores de nombres duplicados en el mismo directorio.

Por ejemplo, crearemos un documento llamado "documento.lua" que contenga código escrito en Lua. Dentro de él escribiremos las instrucciones para que nuestro programa cree un nuevo documento usando las funciones que nos ofrece Lua.

El código escrito dentro de nuestro programa será el siguiente:

```
1  documento = io.open(“prueba.txt”, “w”)
2  documento:write(“Hola mundo, esto es solamente una prueba”)
3  documento:close
```

Este tipo de programa no genera una salida en la terminal de nuestro programa, ya que su función principal es solo crear un nuevo documento en el mismo directorio donde se encuentra el archivo que estamos ejecutando a través del intérprete de nuestro lenguaje.

La elección del nombre del archivo es responsabilidad del usuario, siempre y cuando no sea el mismo que el del archivo que lo está creando. Esto evita problemas como tener dos archivos con el mismo nombre en el mismo directorio.

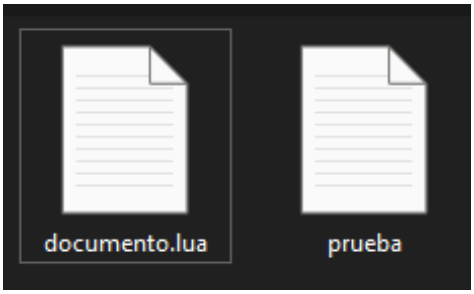
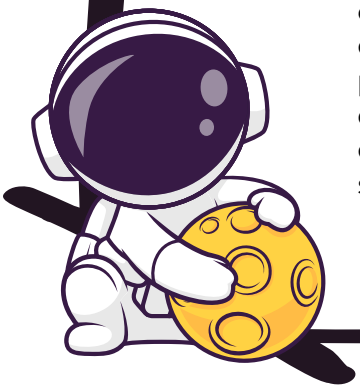


Ilustración 63- Creación del documento "prueba.txt" usando Lua

De esta manera, podemos confirmar que el documento "prueba.txt" se ha creado correctamente en nuestro directorio.

En conclusión

Aprendimos cómo abrir archivos en Lua que contienen instrucciones y cómo tratarlos como simples entradas de texto. También hemos visto cómo leer información dentro de documentos que no tienen la extensión .lua, como archivos de texto puro, y cómo crear nuevos documentos que contengan información generada internamente en nuestro programa. Además, comprendimos la importancia de los directorios y carpetas al trabajar con archivos en Lua. Esperamos que los ejemplos prácticos proporcionados en cada capítulo hayan sido de ayuda para entender estos conceptos.



Capítulo 15

Manejo de errores en Lua

La vida y la programación están llenos de desafíos que a veces nos impiden alcanzar nuestros objetivos. Durante el proceso de programación, es probable que te encuentres con errores.

Aunque es difícil programar sin tener que lidiar con errores, estos son una oportunidad para mejorar tus habilidades como programador. Identificar los errores en un programa y saber cómo solucionarlos es una habilidad importante para cualquier buen programador. No existe una fórmula mágica para aprender a manejar los errores en la programación, pero con práctica y experiencia, puedes desarrollar esta habilidad.

Hay muchas herramientas que nos ayudan a manejar los errores, como la función incorporada en el compilador de Lua que nos muestra directamente dónde ocurre el error. Además, también contamos con librerías que nos facilitan la identificación de errores y nos permiten realizar acciones específicas en base a ellos, en lugar de detener el programa por completo.

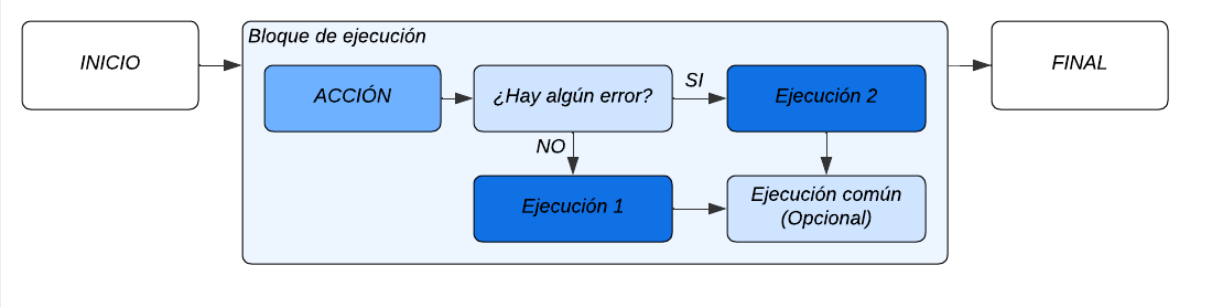


Ilustración 65 - Ejecución de bloques de código bajo errores

Usaremos la librería estándar de Lua para lograr este objetivo. Antes de aprender cómo accionar en base a los errores, es importante conocer los errores y las causas de estos. Por eso, dedicaremos esta sección a estudiar ejemplos de errores y las funciones que desarrollan dentro de nuestro programa, así como las razones por las que aparecen y cómo evitarlos.

Hay muchos errores y diferentes razones por las que un programa puede no funcionar correctamente, y la lista es suficientemente extensa como para dedicar una gran parte de este escrito a explicar cada posible error que puede aparecer. Además, es posible que una gran porción de los errores que aparecen en un programa se deba a una razón desconocida, lo que hace que sea imposible dar una explicación detallada y llegar a una solución generalizada para cada caso.

A continuación, para tener una referencia visual, podemos ver uno de los muchos errores que pueden aparecer al ejecutar un programa de Lua en la terminal de ejecución de nuestro programa.

```
PS C:\Users\LDTR> lua54 C:\Users\LDTR\error.lua
C:\Users\LDTR\AppData\Local\lua-5.4.2_Win64_bin\lua54.exe: C:\Users\LDTR\error.lua:1: ')' expected near <eof>
PS C:\Users\LDTR>
```

Hay varios tipos de errores comunes en la programación, como errores tipográficos o de sintaxis, errores de indexación al trabajar con elementos en una lista o iteración, y errores relacionados con el tiempo de

ejecución, cuando el programa requiere más tiempo del que nuestra computadora puede ofrecer. Es importante conocer estos errores y saber cómo identificarlos y solucionarlos para ahorrar tiempo en el desarrollo de nuestros proyectos.

En las siguientes secciones, veremos ejemplos de errores comunes en la programación, la salida que generan y cómo solucionarlos. Analizaremos lo que sucede durante la secuencia de ejecución de nuestro programa y cómo prevenir estos errores.

Errores de sintaxis

Vamos a comenzar con uno de los errores más comunes para los programadores: los errores de sintaxis. Es bastante frecuente que los programadores, tanto los experimentados como los principiantes, se encuentren con este tipo de error. A veces también se les llama "typos", ya que suelen ser causados por un carácter escrito incorrectamente en el código.

Estos errores se deben, como su nombre indica, a los errores humanos. No hay que avergonzarse de ellos o considerarlos una señal de ser un mal programador, ya que todos somos humanos y cometer errores al escribir es algo común. Puede ser que los cometamos cuando escribimos rápidamente o cuando pasamos por alto pequeños detalles al programar.

Afortunadamente, la mayoría de los entornos de programación tienen herramientas y facilidades para evitar estos errores, como el autocompletado de código o asistentes de programación basados en inteligencia artificial. Identificar este tipo de errores es sencillo, ya que en muchos casos el propio intérprete de Lua nos avisará de ellos mediante una advertencia de "error de sintaxis".

Estos errores son fáciles de solucionar, ya que la solución suele ser simplemente identificar dónde y cómo ocurre el error y corregir el typo que impide que el programa funcione correctamente.

Veamos un ejemplo de un código escrito en Lua con un error de sintaxis y cómo se refleja en la salida. Vamos a ver un ejemplo en el que escribimos incorrectamente una variable que ya habíamos definido. La salida reflejará el error y aparecerá en la pantalla, indicando que hay un error de sintaxis presente.

A continuación, podemos ver un ejemplo de esto:

```
1  variable = 17
2  print(vairable)
```

Al ejecutar este código, obtendríamos la siguiente salida en la terminal de nuestro programa:

```
Salida  nil
```

A pesar de que en la variable asignamos el valor numérico 17, la salida es nil. Esto se debe a un pequeño error en la escritura de la variable que usamos para imprimir en la pantalla. La variable que usamos no es la misma que creamos al principio del programa.

Explicación del código	
1	A pesar de que en la variable asignamos el valor numérico 17, la salida es nil. Esto se debe a un pequeño error en la escritura de la variable que usamos para imprimir en la pantalla. La variable que usamos no es la misma que creamos al principio del programa.
2	Esto hace que el intérprete de Lua intente referenciar una variable diferente, pero al no encontrar una variable con ese nombre exacto, se da cuenta de que la variable no existe. Al no existir la variable, muestra como salida el valor de nil, lo que significa que es una variable vacía o inexistente.

Ahora, vamos a ver otro ejemplo de error de sintaxis mucho más obvio. En este caso, se refiere a cuando intentamos referenciar a una función que no ha sido definida en el sistema o en nuestro código. Veamos un ejemplo.

A continuación, podemos ver un ejemplo de esto:

```
1    pritn(“Hola mundo”)
```

Al ejecutar este código, obtendríamos la siguiente salida en la terminal de nuestro programa:

```
Salida    attempt to call a nil value (global 'pritn' stack traceback:
          [RUTA DEL DOCUMENTO]:1: in main chunk
          [C]: in ?
```

Aquí vemos un mensaje de error mucho más claro y directo en nuestro programa. Este error ocurre porque hemos llamado a una función que no existe en nuestro código o en el sistema.

Explicación del código	
1	Aquí vemos un mensaje de error mucho más claro y directo en nuestro programa. Este error ocurre porque hemos llamado a una función que no existe en nuestro código o en el sistema.
2	En este caso, cometimos un error al escribir la función print(), escribiendo pritn() en su lugar. La solución es sencilla: buscar y corregir el error tipográfico en nuestro código.
3	En resumen, los errores sintácticos en Lua se refieren a los errores o fallos en el programa que ocurren cuando cometemos errores de escritura en el código. La solución a la mayoría de estos errores es corregirlos en el código.

Con esto, concluimos nuestro estudio de caso sobre errores de sintaxis en los programas.

Error de índice fuera del rango

Hemos aprendido un poco sobre qué son los índices en una lista.

Si recordamos cómo funciona una lista y lo importantes que son los índices dentro de los elementos, podemos decir que una lista es simplemente un tipo de datos que contiene varios otros tipos de datos y que los índices en nuestro programa brindan una dirección a estos elementos individuales para que no se mezclen o afecten entre sí. Los índices manejados por nuestros datos almacenados en nuestras variables son de tipo numérico y los usamos para acceder a un dato específico en nuestro programa.

La mayoría de estos errores ocurren durante las iteraciones o al intentar acceder a un índice de un elemento en una lista que no existe. Le decimos al programa que busque un elemento por su índice en una lista cuando dicho elemento simplemente no está en la estructura, haciendo que nuestro programa no sepa qué hacer debido a la información inexistente en nuestro programa.

Podemos entenderlo mejor usando un ejemplo hipotético de la vida cotidiana.

Imagina que te encuentras en una habitación con cinco cajas y cada caja contiene diferentes elementos dentro. No importa qué elementos en particular se encuentren dentro de las cajas.

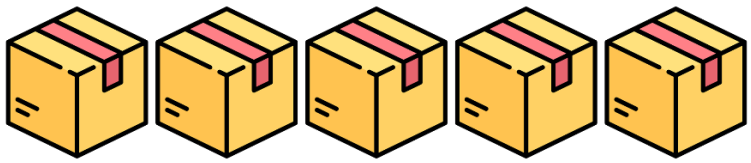


Ilustración 64 - "Nos encontramos en una habitación con cinco cajas, cada una de las cajas contiene diferentes elementos dentro"

Ahora, supongamos que eres el intérprete de un lenguaje de programación y te han asignado una tarea de abrir la caja número 7. ¿Cómo puedes abrirla si solo hay 5 cajas?

Este ejemplo hipotético nos ayuda a entender mejor qué es un error de índice fuera del rango y lo que significa en nuestro programa al intentar acceder a un índice de información inexistente en una estructura de datos.

Entonces, para explicar por qué ocurre este error, es importante entender cómo funcionan las listas en nuestros programas. Una lista es solo un tipo de elemento en Lua que puede contener otros tipos de elementos, ya sea otra lista, un valor booleano, numérico, etc. La principal característica de las listas en Lua es que pueden almacenar casi cualquier tipo de dato.

Cuando agregamos un elemento a una lista, ocupa un espacio en la memoria y para acceder a los elementos de la lista, es importante saber que cada elemento tiene una dirección única asignada como un valor numérico llamado índice. Este índice es un número que sirve como identificador único del elemento dentro de la lista y puede ser visto como la dirección de este elemento en la estructura de datos.

Estos números no comienzan en cero como en otros lenguajes de programación, sino que comienzan en 1, lo que hace a Lua un lenguaje de programación único. Estos números asignados a cada elemento en la lista comenzarán en 1 para el primer elemento, 2 para el segundo elemento, 3 para el tercer elemento, y así sucesivamente.

En resumen, supongamos que tenemos una lista con 7 elementos y cada elemento tiene un identificador único en la lista para acceder a los valores sin tener que recorrer cada elemento de la lista. Ahora, supongamos que tenemos un programa simple que solo desea imprimir el valor del dato en la posición 10 de la lista.

Esto es claramente un problema, ya que nuestra lista solo tiene 7 elementos y 7 posiciones, por lo que intentar acceder a un valor fuera de la lista resultará en un error de índice de lista.

Este bloque de código puede servir como ejemplo para entender mejor:

```
1  lista = {100, 200, 300, 400, 500, 600, 700}
2  print(lista[10])
```

Al ejecutar este código, obtendremos la siguiente salida en la terminal de nuestro programa:

```
Salida  nil
```

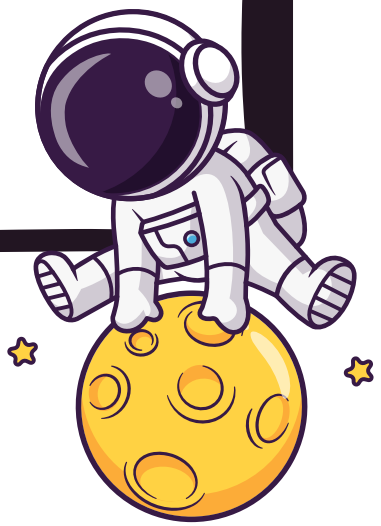
Finalmente, para entender mejor, veamos una explicación detallada de lo que ocurre en este código:

	Explicación del código
1	El resultado de este código es que la salida es "nil", ya que estamos intentando acceder a un elemento que no existe en la lista. Es interesante que para este tipo de errores, Lua solo muestre "nil" en lugar de un mensaje de error directamente. Esto es parte de cómo Lua maneja estos errores para evitar que detengan la ejecución del resto del programa.
2	La solución para este tipo de error es sencilla. Solo debemos tener en cuenta el índice que estamos intentando usar en la lista.
3	En este ejemplo específico, tenemos dos posibles soluciones. La primera es aumentar el número de elementos en la lista, para que tenga al menos 10 elementos para acceder a ellos. La segunda solución es cambiar el índice que estamos intentando usar para que corresponda a un elemento existente en la lista.

¡Y eso es todo! Ahora entendemos cómo se produce un error de índice fuera de rango.

En conclusión

El manejo de errores en la programación es una habilidad importante que todo programador debe tener. Aunque es difícil programar sin errores, estos son una oportunidad para mejorar nuestras habilidades. Lua nos ofrece herramientas como la función incorporada del compilador y las librerías para ayudarnos a identificar y solucionar errores. En particular, hemos visto que los errores de sintaxis y los errores de índice fuera del rango son dos de los errores más comunes en la programación.



Varios otros errores en Lua

Hemos adquirido una base de conocimiento sobre dos tipos de errores que pueden hacer que nuestro programa no funcione como debería. Estos dos errores son los más comunes que podemos cometer al programar. Sin embargo, como mencionamos en la introducción de la sección de manejo de errores en Lua, hay muchos más tipos de errores además de estos dos que acabamos de estudiar. Sería muy difícil explicar, investigar y solucionar cada uno de ellos en detalle. En esta corta sección, nos enfocaremos en darte una breve descripción de estos otros posibles errores que puedes encontrar en algún momento mientras programas. Es importante aclarar que el objetivo de ver estos errores es tener una idea general de lo que puede suceder en tus proyectos y tener una breve descripción de cada uno. Sin embargo, en esta sección no propondremos una explicación o solución detallada a este tipo de problemas.

Varios otros tipos de errores en Lua		
En esta tabla, podrás ver diferentes errores que puedes encontrar durante la ejecución de partes del código o durante el desarrollo del programa. Verás el nombre del error, un ejemplo de la salida que genera en la terminal de tu programa y, finalmente, una breve descripción del error.		
Nombre de error	Ejemplo	Descripción
Error de sintaxis	lua: prueba.lua:2: 'do' expected near 'print'	Este tipo de error ocurre cuando cometes errores humanos al escribir tu código.
Error de índice fuera de rango	No aplica.	Este es un error indirecto que ocurre cuando intentas referenciar un elemento dentro de una iteración o una lista con un índice que no existe en la lista o iteración.
Error de ejecución	lua: prueba.lua:2: attempt to perform arithmetic on local 'b' (a nil value) stack traceback: test2.lua:2: in function 'suma' test2.lua:5: in main chunk [C]: ?	Este tipo de error puede ocurrir por varias razones, como, por ejemplo, al indicar una entrada de datos errónea en tu programa. Son errores que detienen por completo la ejecución de tu programa sin ser errores de sintaxis.
Tiempo de ejecución indeterminado	No aplica.	Este también es un caso de un error indirecto. Se trata de errores cuando tu programa se queda ejecutando un bloque de código indefinidamente o cuando tu programa se queda estancado en la ejecución de una tarea.
Ejecución parcial no planeada	No aplica.	Este es un error indirecto que ocurre cuando un bloque de código no se ha ejecutado por completo de manera indeseada. Generalmente, este tipo de error ocurre cuando se ingresa un break dentro de un bloque de código y este se ejecuta en un momento erróneo.

De esta manera, podemos tener una base teórica y superficial de los diferentes tipos de errores que puedes encontrar en tus programas al momento de ejecutarlos.

Capítulo 16

Librería de comunicación con el sistema operativo

Lua también nos brinda la posibilidad de hacer que el lenguaje de programación se comunique con nuestro sistema operativo y obtenga información de él. Esto se logra gracias a las diferentes funciones que ofrece Lua para extraer información del sistema.

Es normal sentir cierto miedo al pensar en que nuestro lenguaje de programación pueda acceder a la información de nuestro sistema operativo, pero no hay de qué preocuparse. La información que se puede obtener es información general, como la fecha y hora en que el programa fue ejecutado, y aunque este proceso parece sencillo, en realidad es un proceso interno bastante complejo que requiere varios pasos. Sin embargo, Lua nos lo hace fácil, ya que podemos hacerlo con solo unas pocas líneas de código.

Además, Lua también nos permite interactuar con otros lenguajes de programación, como C. Pero incluso sin la ayuda de otros lenguajes, Lua es lo suficientemente potente para crear estas interacciones entre el programa y el sistema operativo y realizar diferentes tareas que requieran esta comunicación.

Para comprender mejor esta funcionalidad y tener una base de conocimiento más profunda sobre las funciones que podemos usar, es hora de ver algunos ejemplos prácticos de código escrito en Lua que realizan esta tarea.

Mostrar hora actual de nuestro sistema operativo en Lua

Como mencionamos anteriormente, es hora de ver algunos ejemplos prácticos de código que nos permitan comprender cómo podemos acceder a las funciones de la librería de control del sistema operativo en Lua. Comencemos con un ejemplo sencillo, que nos servirá como introducción para entender cómo podemos acceder a estas funciones.

El siguiente bloque de código de nuestro programa tiene la tarea de mostrar en la pantalla la hora actual de nuestro sistema al momento de ejecutar el programa.

El siguiente bloque de código es un ejemplo de cómo podemos mostrar la hora actual de nuestro sistema en Lua:

```
1  horaActual = os.date(“*t”)
2  print(“La hora actual es:”)
3  print(tostring(horaActual.hour) .. “ horas, “ .. tostring(horaActual.min) .. “ minutos y “ ..
   tostring(horaActual.sec) .. “ segundos.”)
```

Si el usuario ejecuta este archivo a las 2:43:50 p.m., la salida en la pantalla del terminal será la siguiente:

```
Salida  La hora actual es:
        14 horas, 43 minutos y 50 segundos.
```

Explicación del código	
1	En la primera línea, creamos una variable llamada "horaActual" que almacena la hora exacta en el momento en que se ejecuta esta línea de código.
2	Luego, usamos la librería "os", que se encarga de la comunicación entre nuestro programa y el sistema operativo, y en particular la función "date". Esta función almacena la fecha y hora actual de nuestro sistema en una lista. Es importante destacar que esta información es estática, es decir, no cambiará una vez que se ejecute, por lo que si necesitamos actualizarla, tendremos que volver a usar la función.
3	Además, vemos una cadena de texto desconocida como parámetro en la función "date", que le da un formato a la información de la fecha actual. No es necesario entender los diferentes métodos de formato, sino solo comprender la función y cómo podemos aprovecharla en nuestros programas.
4	Finalmente, después de almacenar toda la información en "horaActual", mostramos en pantalla los elementos correspondientes a la hora, los minutos y los segundos, convirtiéndolos en cadenas de texto y completando la ejecución de nuestro código.
5	Como resultado, podemos ver que el programa muestra la hora exacta en la que fue ejecutado.

En resumen, concluimos nuestro ejemplo práctico de código que muestra cómo usar la hora exacta de nuestro sistema operativo en nuestro programa.

Mostrar fecha completa de nuestro sistema operativo con Lua

En este caso, haremos uso de la función "date()" de la librería de relación con el sistema operativo de Lua. En el ejemplo anterior, usamos esta función para mostrar la hora en que se ejecutó el bloque de código en base a la información de nuestro sistema.

Ahora, veremos un ejemplo similar, pero en lugar de centrarnos solo en la hora, también tomaremos la información completa, como la fecha exacta. Este ejemplo será similar al anterior, usaremos la misma función y la misma cadena de texto para indicar el formato de extracción de los datos.

Echa un vistazo a este ejemplo para ver cómo mostrar la fecha completa en nuestro programa.

```
1  fechaActual = os.date(“*t”)
2  io.write(“El programa fue ejecutado el ”)
3  io.write(tostring(fechaActual.day) .. “ del mes “ .. tostring(fechaActual.month) .. “ del “ ..
   tostring(fechaActual.year))
```

Si ejecutamos el programa el 13 de enero de 2023, la salida en la pantalla de la terminal será:

```
Salida  El programa fue ejecutado el 13 del mes 1 del 2023
```

Como puedes ver, este ejemplo es muy similar al anterior, por lo que no es necesario dedicar mucho tiempo a explicarlo detalladamente.

Capítulo 17

Sección final: No hay límites para programar en Lua

Hemos llegado a la última sección de este libro, y en esta ocasión no abordaremos un tema en particular con detalle. En cambio, utilizaremos esta sección para dar un cierre formal a nuestro camino de aprendizaje.

Ahora podemos decir con orgullo que, al completar este libro, tenemos una sólida comprensión de las diferentes funciones básicas y conceptos fundamentales de este lenguaje de programación. Somos programadores de nivel básico en Lua.

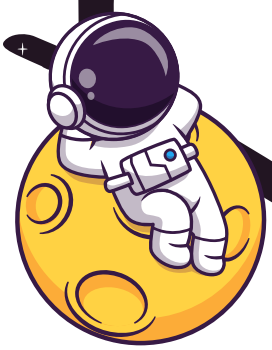
Es importante tener en cuenta que Lua es un lenguaje de programación impresionante con muchas posibilidades y aplicaciones que podemos aprovechar. Por supuesto, estudiar cada una de estas funcionalidades y aplicarlas a nuestros programas puede llevar tiempo, investigación y mucha práctica.

Además, podemos hacer muchas más cosas con Lua, como desarrollar videojuegos, crear redes neuronales, realizar simuladores complejos y mucho más. Lua es un lenguaje de programación poderoso con muchas funciones, solo necesitas saber cómo aprovechar su potencial y entender por qué la comunidad lo ama tanto.

Antes de finalizar este libro, encontrarás algunos ejercicios finales que puedes realizar para afianzar tus conocimientos sobre los temas que hemos cubierto.

En conclusión

Cubrimos los diferentes tipos de errores que podemos encontrar en Lua y hemos visto algunos ejemplos de cómo manejarlos. También exploramos la librería de comunicación con el sistema operativo y cómo podemos usarla para obtener información relevante de nuestro sistema. Además, hemos aprendido cómo mostrar la hora y la fecha actual de nuestro sistema operativo en nuestros programas Lua. Finalmente, destacamos la versatilidad de Lua y la amplia gama de aplicaciones que podemos explorar a medida que seguimos mejorando nuestras habilidades en programación en este lenguaje. Esperamos que los ejercicios finales te ayuden a consolidar tus conocimientos y te animamos a seguir explorando y creando con Lua. ¡Felicitaciones por completar este libro!



Ejercicios de práctica	
1	Desarrollar un programa que calcule el valor factorial de un número dado. Es importante tener en cuenta que debemos desarrollar este programa usando módulos, es decir, debemos crear dos documentos diferentes. Aquí están los pasos para hacerlo correctamente: 1 Crea un documento llamado "función.lua". Este documento será un módulo que contenga la función para calcular la factorial de un número. 2 Dentro del mismo documento, escribamos la función siguiendo la estructura de módulo adecuada. 3 Crea un segundo documento que usaremos para este ejemplo. Dentro de este documento, importamos el módulo que creamos antes. 4 Agregamos una línea de código para leer la entrada del usuario en nuestro programa y guardarla en una variable. Es importante tener en cuenta que esta entrada será un número. 5 Finalmente, usamos el módulo y la función dentro de él para calcular la factorial del número ingresado por el usuario. 6 Mostramos el resultado en la terminal de nuestro programa.
2	Realizar un programa que concatene los valores dentro de una lista con comas como separadores. Aquí están los pasos para hacerlo: 1 Creamos una variable llamada "lista" en nuestro programa. Esta lista incluirá los siguientes elementos: 17, "Mundo", -17, "Hola". 2 Luego, creamos una segunda variable que tendrá la concatenación de cada elemento de la lista en una cadena de texto. 3 Finalmente, mostramos en pantalla el contenido de esta segunda variable. La salida esperada es la siguiente: Salida 17, Mundo, -17, Hola
3	Desarrollar un programa que llene una lista de 10 elementos con números aleatorios. Para hacer esto, debemos usar la función "random" de la biblioteca matemática de Lua. Después de crear la lista, mostraremos en pantalla cada uno de los elementos en líneas separadas.
4	Desarrollar un programa que lea información de un documento con extensión .txt. La información dentro del documento puede ser cualquier cosa, siempre y cuando sea texto. Finalmente, debemos guardar esta información en una variable y luego crear un tercer documento que tenga el mismo contenido que el documento leído. Este tercer documento debe tener el mismo nombre que los dos primeros documentos.
5	Crear un programa que convierta una cantidad en dólares a euros. Para hacer esto, debemos seguir los siguientes pasos: 1 Leemos la cantidad en dólares que desea convertir el usuario. 2 Calculamos el equivalente en euros, teniendo en cuenta el tipo de cambio actual. 3 Mostramos el resultado en pantalla en un mensaje amigable, como "X dólares son Y euros".

Este libro fue escrito con pocas citas de otros escritos y material gráfico. Sin embargo, aquí encontrarás una lista de las referencias utilizadas y de los escritos que sirvieron de base teórica para el contenido textual y gráfico del libro. Además, también se mencionarán escritos que, aunque no están directamente relacionados con este libro y sus temas, se usaron durante el desarrollo de este escrito introductorio para aquellos que están empezando a aprender a programar en Lua.

Es relevante mencionar que las referencias en este libro se han gestionado siguiendo las normas APA, adaptándose al formato adecuado según el tipo de documento referenciado, ya sea un artículo de investigación, informe, página web u otros. Además, se ha puesto empeño en incluir referencias actualizadas y pertinentes a los temas abordados a lo largo del libro.

Capítulo 18

Bibliografía

Akoury, N., Salz, R., & Iyyer, M. (2023). Towards Grounded Dialogue Generation in Video Game Environments. *University of Massachusetts Amherst*.

Anónimo. (2023). *lua-users.org*. Obtenido de lua-users: http://lua-users.org/

Bakanov, V. M. (2022). Computational complexity when constructing rational plans for program execution in a given field of parallel computers. *Russian Technological Journal. 2022;10(6)*, 7–19.

Baszucki, D. (15 de Noviembre de 2021). *Introducing the Roblox Community Fund - Roblox Blog.* Obtenido de Roblox Blog: https://blog.roblox.com/2021/11/introducing-roblox-community-fund/

Deniau, L. (2023). *MAD-NG.* Meyrin: CERN.

Ellis, L. J. (2018). *GitHub - LewisJEllis/awesome-lua: A curated list of quality Lua packages and resources.* Obtenido de GitHub: https://github.com/LewisJEllis/awesome-lua

Ierusalimschy, R. (2004). *Programming in Lua (first edition).* Obtenido de The Programming Language Lua: https://www.lua.org/pil/contents.html#4

Ierusalimschy, R., Figueiredo, L. H., & Celes, W. (2001). *The Evolution of an Extension Language:.* TeCGraf, Department of Computer Science, PUC-Rio.

Ierusalimschy, R., Figueiredo, L. H., & Celes, W. (2005). The Implementation of Lua 5.0. *Journal of Universal Computer Science, vol. 11, no. 7*, 1159-1176.

Ierusalimschy, R., Figueiredo, L. H., & Filho, W. C. (Junio de 1996). Lua—An Extensible Extension Language. *Lua—An Extensible Extension Language*, págs. 635-652.

Jung, K., & Brown, A. (2011). *Beginning Lua Programming.* Indiniapolis: Wiley Publishing, Inc.

LuaRocks. (2023). *LuaRocks - The Lua package manager*. Obtenido de LuaRocks: https://luarocks.org/

Offringa, A. R., Adebahr, B., Kutkin, A., Adams, E. A., Oosterloo, T. A., Hulst, J. M., . . . Ziemke, J. (2023). An interference detection strategy for Apertif based on AOFlagger 3. *arXiv*, 15.

PUC-Rio. (13 de Enero de 2022). *Lua 5.4 Reference Manual*. Obtenido de The Programming Language Lua: https://www.lua.org/manual/5.4/

PUC-Rio. (24 de Octubre de 2022). *Lua: about.* Obtenido de The Programming Language Lua: https://www.lua.org/about.html

PUC-Rio. (25 de Enero de 2022). *Lua: download.* Obtenido de The Programming Language Lua: https://www.lua.org/download.html

Shirore Chetan, K. A. (2023). *Basic Mathematical Computations inside LaTeX using Lua.* Electronic Journal of Mathematics & Technology.

Skyrme, A., Rodriguez, N., & Ierusalimschy, R. (2008). Exploring Lua for Concurrent Programming. *Journal of Universal Computer Science, vol. 14, no. 21*, 3556-3572.

Weber, C. (2023). Generating Documents with FeatureIDE and pandoc. *VaMoS 2023: 17th International Working Conference on Variability Modelling of Software-Intensive Systems* (págs. 60-64). Odense: ACM Digital Library Home.