



APRENDE **PYTHON** GUÍA PRÁCTICA

A C A D E M I A X

Contenido

1	Introducción	4
1.1	Bienvenida	4
1.1.1	Libro vivo	5
1.1.2	Alcance	5
1.2	Prerequisitos	5
1.3	¿Cómo evitar bloqueos?	6
2	Primeros pasos	7
2.1	Visión general	7
2.1.1	¿Qué es y por qué debes aprenderlo?	7
2.1.2	¿En dónde se utiliza?	7
2.1.3	¿Qué trabajos puedes conseguir?	8
2.1.4	¿Cuánto puedes ganar?	8
2.1.5	¿Cuales son las preguntas más comunes?	9
2.2	Historia, evolución, y versiones	9
2.3	Características y ventajas	10
2.4	Diferencias con otros lenguajes de programación	11
2.5	Configuración	12
2.5.1	IDE	12
2.5.2	Entorno	12
2.6	Hola Mundo	13
3	Gramática	14
3.1	Sintaxis	14
3.2	Comentarios	14
4	Tipos y variables	15
4.1	Variables	15
4.2	Listas	17

Contenido

4.3	Objetos	18
4.4	Constantes	20
5	Operadores	21
5.1	Operadores de aritméticos	21
5.2	Operadores comparativos	22
5.3	Operadores lógicos	22
6	Condicionales	23
6.1	Condición única	23
6.2	Múltiples condiciones	24
7	Bucles	26
7.1	Bucle for	26
7.2	Bucle while	27
8	Funciones	28
9	Programación orientada a objetos (POO)	30
9.1	Paradigma	30
9.2	Clases	31
10	Módulos	32
11	Siguientes pasos	33
11.1	Herramientas	33
11.2	Recursos	34
11.3	¿Que viene después?	34
11.4	Preguntas de entrevista	35

1 Introducción

1.1 Bienvenida

Bienvenid@ a esta guía de Academia X en donde aprenderás Python práctico.

Hola, mi nombre es Xavier Reyes Ochoa y soy el autor de esta guía. He trabajado como consultor en proyectos para Nintendo, Google, entre otros proyectos top-tier, trabajé como líder de un equipo de desarrollo por 3 años, y soy Ingeniero Ex-Amazon. En mis redes me conocen como Programador X y comparto videos semanalmente en YouTube desde diversas locaciones del mundo con el objetivo de guiar y motivar a mis estudiantes mientras trabajo haciendo lo que más me gusta: la programación.

En esta guía vas a aprender estos temas:

- Gramática
- Tipos y variables
- Operadores
- Condicionales
- Bucles
- Funciones
- Programación orientada a objetos (POO)
- Módulos

La motivación de esta guía es darte todo el conocimiento técnico que necesitas para elevar la calidad de tus proyectos y al mismo tiempo puedas perseguir metas más grandes, ya sea utilizar esta tecnología para tus pasatiempos creativos, aumentar tus oportunidades laborales, o si tienes el espíritu emprendedor, incluso crear tu propio negocio en línea. Confío en que esta guía te dará los recursos que necesitas para que tengas éxito en este campo.

Empecemos!

1 INTRODUCCIÓN

1.1.1 Libro vivo

Esta publicación fue planeada, editada, y revisada manualmente por Xavier Reyes Ochoa. La fundación del contenido fue generada parcialmente por inteligencia artificial usando ChatGPT (Mar 14 Version) de OpenAI. Puedes ver más detalles en <https://openai.com/>

Esto es a lo que llamo un trabajo **VIVO**, esto quiere decir que será actualizado en el tiempo a medida que existan cambios en la tecnología. La versión actual es 1.0.0 editada el 23 de marzo de 2023. Si tienes correcciones importantes, puedes escribirnos a nuestra sección de contacto en <https://www.academia-x.com>

1.1.2 Alcance

El objetivo de esta guía es llenar el vacío que existe sobre esta tecnología en Español siguiendo el siguiente enfoque:

- Se revizan los temas con un enfoque práctico incluyendo ejemplos y retos.
- Se evita incluir material de relleno ya que no es eficiente.
- Se evita entrar en detalle en temas simples o avanzados no-prácticos.

Si deseas profundizar en algún tema, te dejo varios recursos populares y avanzados en la lección de recursos como el estándar de esta tecnología (que puede ser difícil de leer si recién estás empezando).

1.2 Prerequisitos

Antes de aprender Python, necesitas lo siguiente:

1. Un computador: cualquier computador moderno tiene las capacidades de correr este lenguaje. Te recomiendo un monitor de escritorio o laptop ya que dispositivos móviles o ipads no son cómodos para programar.

1 INTRODUCCIÓN

2. Sistema operativo: conocimiento básico de cómo utilizar el sistema operativo en el que se ejecutará Python (por ejemplo, Windows, MacOS, Linux). Te recomiendo tener el sistema operativo actualizado.
3. Conocimiento básico de la línea de comandos: se utiliza para ejecutar programas en Python.
4. Un editor de texto: lo necesitas para escribir código de Python. Los editores de texto más populares son Visual Studio Code, Sublime Text, Atom y Notepad ++.
5. Un navegador web y conexión al internet: es útil para investigar más sobre esta tecnología cuando tengas dudas. Los navegadores más populares son Google Chrome, Mozilla Firefox, Safari y Microsoft Edge. Se recomienda tener el navegador actualizado.

Si ya tienes estos requisitos, estarás en una buena posición para comenzar a aprender Python y profundizar en sus características y aplicaciones.

Si todavía no tienes conocimiento sobre algunos de estos temas, te recomiendo buscar tutoriales básicos en blogs a través de Google, ver videos en YouTube, o hacer preguntas a ChatGPT. Alternativamente, puedes empezar ya e investigar en línea a medida que encuentres bloqueos enteniendo los conceptos en esta guía.

1.3 ¿Cómo evitar bloqueos?

Para sacarle el mayor provecho a esta guía:

1. No solo leas esta guía. La práctica es esencial en este campo. Practica todos los días y no pases de lección hasta que un concepto esté 100% claro.
2. No tienes que memorizarlo todo, solo tienes que saber donde están los temas para buscarlos rápidamente cuando tengas dudas.
3. Cuando tengas preguntas usa [Google](#), [StackOverFlow](#), y [ChatGPT](#)
4. Acepta que en esta carrera, mucho de tu tiempo lo vas utilizar investigando e innovando, no solo escribiendo código.

5. No tienes que aprender inglés ahora pero considera aprenderlo en un futuro porque los recursos más actualizados están en inglés y también te dará mejores oportunidades laborales.
6. Si pierdas la motivación, recuerda tus objetivos. Ninguna carrera es fácil pero ya tienes los recursos para llegar muy lejos. Te deseo lo mejor en este campo!

2 Primeros pasos

2.1 Visión general

2.1.1 ¿Qué es y por qué debes aprenderlo?

Python es un lenguaje de programación de alto nivel, interpretado, multiparadigma y multiplataforma. Está diseñado para ser fácil de leer y escribir, y su sintaxis permite a los programadores expresar conceptos en pocas líneas de código.

Debes aprender Python porque es un lenguaje de programación muy versátil y poderoso. Puede usarse para desarrollar aplicaciones de escritorio, servidores web, aplicaciones móviles, juegos, inteligencia artificial, análisis de datos y mucho más. Además, es un lenguaje de programación muy popular, por lo que hay una gran cantidad de recursos disponibles para ayudarte a aprender.

2.1.2 ¿En dónde se utiliza?

Python se utiliza en una amplia variedad de aplicaciones, desde la programación web hasta la ciencia de datos, la inteligencia artificial, la visualización de datos y la automatización.

También se utiliza en la programación de videojuegos, la creación de aplicaciones móviles, la creación de aplicaciones de escritorio, la creación de scripts para automatizar tareas y la creación de aplicaciones web.

2 PRIMEROS PASOS

Además, Python se utiliza en la creación de aplicaciones de Internet de las cosas (IoT), la creación de aplicaciones de aprendizaje automático, la creación de aplicaciones de análisis de datos, la creación de aplicaciones de seguridad informática y la creación de aplicaciones de administración de bases de datos.

2.1.3 ¿Qué trabajos puedes conseguir?

- Desarrollador de software
- Desarrollador web
- Analista de datos
- Científico de datos
- Desarrollador de aplicaciones móviles
- Desarrollador de videojuegos
- Desarrollador de inteligencia artificial
- Desarrollador de sistemas
- Desarrollador de automatización
- Desarrollador de IoT
- Desarrollador de blockchain
- Desarrollador de DevOps

2.1.4 ¿Cuánto puedes ganar?

El salario que puedes ganar usando Python depende de muchos factores, como tu experiencia, el lugar donde trabajas, el tipo de trabajo que estás haciendo, etc. En general, los desarrolladores de Python pueden ganar entre \$50,000 y \$150,000 al año en los Estados Unidos, dependiendo de la ubicación y el nivel de experiencia.

Es importante tener en cuenta que estos son solo promedios y que el salario real que puedes ganar puede ser mayor o menor, dependiendo de los factores mencionados anteriormente. Además, siempre es una buena idea investigar y hacer preguntas sobre los salarios y las condiciones laborales antes de aceptar un trabajo.

2.1.5 ¿Cuales son las preguntas más comunes?

1. ¿Cómo instalo Python?
2. ¿Cómo puedo aprender Python?
3. ¿Qué es una variable en Python?
4. ¿Cómo puedo leer y escribir archivos en Python?
5. ¿Cómo puedo ejecutar un script de Python?
6. ¿Cómo puedo usar funciones en Python?
7. ¿Cómo puedo usar bucles en Python?
8. ¿Cómo puedo usar condicionales en Python?
9. ¿Cómo puedo usar módulos en Python?
10. ¿Cómo puedo usar excepciones en Python?

Al finalizar este recurso, tendrás las habilidades necesarias para responder o encontrar las respuestas a estas preguntas.

2.2 Historia, evolución, y versiones

Python fue creado por Guido van Rossum en 1991. Fue diseñado para ser un lenguaje de programación fácil de usar, con un enfoque en la legibilidad del código. La primera versión de Python fue lanzada en 1994. Desde entonces, se han lanzado varias versiones de Python, cada una con nuevas características y mejoras.

Las versiones principales de Python incluyen Python 2.0, lanzado en 2000, y Python 3.0, lanzado en 2008. Python 2.0 fue una actualización significativa del lenguaje, con mejoras en la sintaxis, la seguridad y la portabilidad. Python 3.0 fue una actualización aún mayor, con una nueva sintaxis, mejoras en la seguridad y una mayor compatibilidad con otros lenguajes.

Desde entonces, se han lanzado varias versiones de Python, cada una con nuevas características y mejoras. Estas versiones incluyen Python 3.1, 3.2, 3.3, 3.4, 3.5, 3.6, 3.7, 3.8, 3.9, 3.10, y 3.11. Estas versiones han añadido nuevas características

como mejoras en la sintaxis, mejoras en la seguridad, mejoras en la eficiencia y mejoras en la compatibilidad con otros lenguajes.

2.3 Características y ventajas

Python es un lenguaje de programación de alto nivel, interpretado, multiparadigma y multiplataforma. Está diseñado para ser fácil de leer y escribir, y para permitir a los programadores crear código de forma rápida y eficiente.

Python ofrece una variedad de características y ventajas que lo hacen un lenguaje de programación ideal para principiantes y profesionales. Estas características incluyen:

- Una sintaxis simple y clara: Python tiene una sintaxis simple y clara que es fácil de leer y escribir. Esto hace que sea un lenguaje ideal para principiantes, ya que es fácil de aprender.
- Una gran biblioteca estándar: Python viene con una gran biblioteca estándar que contiene una variedad de funciones y módulos que pueden ser utilizados para desarrollar aplicaciones. Esto significa que los programadores no tienen que escribir todo el código desde cero.
- Soporte para múltiples paradigmas de programación: Python es un lenguaje multiparadigma, lo que significa que soporta varios paradigmas de programación, como programación orientada a objetos, programación funcional y programación imperativa. Esto significa que los programadores pueden elegir el paradigma de programación que mejor se adapte a sus necesidades.
- Multiplataforma: Python es un lenguaje multiplataforma, lo que significa que se puede ejecutar en diferentes sistemas operativos, como Windows, Mac OS X y Linux. Esto significa que los programadores pueden escribir código que se ejecutará en diferentes plataformas sin tener que cambiar el código.

- Alto rendimiento: Python es un lenguaje de alto rendimiento, lo que significa que los programas escritos en Python se ejecutan más rápido que los programas escritos en otros lenguajes. Esto significa que los programadores pueden crear aplicaciones más rápidas y eficientes.

2.4 Diferencias con otros lenguajes de programación

Python es un lenguaje de programación interpretado, lo que significa que el código se ejecuta directamente sin necesidad de compilarlo. Esto lo hace más fácil de usar y más rápido de escribir que otros lenguajes de programación.

Python también es un lenguaje de programación orientado a objetos, lo que significa que los programas se escriben usando objetos y clases. Esto hace que sea más fácil de entender y mantener el código.

Otra diferencia importante entre Python y otros lenguajes de programación es que Python es un lenguaje de alto nivel, lo que significa que el código es más legible para los humanos. Esto hace que sea más fácil de aprender y usar.

Python también es un lenguaje de programación dinámico, lo que significa que los programas se pueden cambiar en tiempo de ejecución. Esto hace que sea más fácil de depurar y probar.

En Python, a diferencia de otros lenguajes, la indentación es importante. Esto lo verás más adelante.

Finalmente, Python es un lenguaje de programación multiparadigma, lo que significa que se pueden usar diferentes estilos de programación, como programación orientada a objetos, programación funcional y programación imperativa. Esto hace que sea más versátil y flexible.

2.5 Configuración

2.5.1 IDE

Los archivos de Python son archivos de texto. Puedes editarlos con **editores de texto** como Notepad en Windows o Notes en MacOS pero es recomendado utilizar un **IDE** (Integrated Development Environment) que es una aplicación de edición de código más avanzado que le da colores a tu código para que sea más fácil de leer y tengas funciones de autocompletado, entre otras. Algunos IDEs populares son [Brackets](#), [Atom](#), [Sublime Text](#), [Vim](#), y [Visual Studio Code](#).

El editor recomendado para practicar el código que vamos a ver es Visual Studio Code (o VSCode) que puedes bajar desde <https://code.visualstudio.com/>

2.5.2 Entorno

Para usar Python en tu computador, primero debes instalar el intérprete de Python. Puedes descargar la última versión de Python desde el sitio web oficial de Python en <https://www.python.org/downloads/>. Una vez descargado, sigue las instrucciones de instalación para completar la instalación.

Una vez instalado, puedes verificar la instalación abriendo una ventana de línea de comandos y escribiendo **python3 --version**. Esto mostrará la versión de Python que has instalado.

Puedes correr Python abriendo una terminal o línea de comandos y escribiendo **python3** y presionando enter. Si la instalación fue exitosa, verás un prompt de Python como **>>>** donde puedes escribir tu código. Puedes salir de la consola de Python escribiendo **exit()**.

También hay algunas maneras de ejecutar archivos de Python. Aquí hay algunos métodos comunes:

2 PRIMEROS PASOS

1. En la línea de comandos: Puedes abrir una terminal en tu sistema operativo y escribir **python3 nombre_del_archivo.py** Esto ejecutará el archivo de Python en cuestión.
2. Con un editor de código: Si estás utilizando un editor de código como PyCharm, Visual Studio Code, etc., puedes abrir tu archivo de Python en el editor, asegurarte de que esté activo y presionar “Ejecutar” o utilizar un atajo de teclado como F5.

2.6 Hola Mundo

“Hola Mundo” es un ejemplo clásico que se utiliza para mostrar el funcionamiento básico de un lenguaje de programación.

Ejemplo:

En este ejemplo, se imprime el texto “Hola Mundo” en la consola.



También podrías modificar este código con tu nombre. Si es “Juan”, debería imprimir en la consola “Hola, Juan” .

Reto:

Modifica el ejemplo anterior para imprimir “Hola Universo” en la consola.

3 Gramática

3.1 Sintaxis

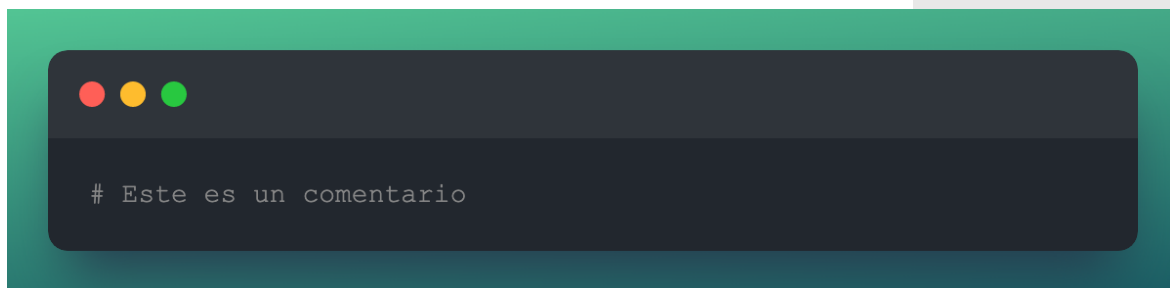
Python es un lenguaje de programación de alto nivel con una sintaxis clara y concisa. Utiliza palabras clave para indicar acciones, y los bloques de código se organizan mediante la indentación. El conjunto de caracteres de Python incluye letras, números, signos de puntuación y otros caracteres especiales. Python es sensible a mayúsculas y minúsculas, por lo que los nombres de variables, funciones y otros elementos deben escribirse de manera consistente.

3.2 Comentarios

Los comentarios en Python son líneas de texto que no son interpretadas como parte del código. Se usan para proporcionar información adicional sobre el código, como explicaciones, notas, o instrucciones para el desarrollador. Los comentarios también se pueden usar para deshabilitar código temporalmente, sin tener que eliminarlo completamente.

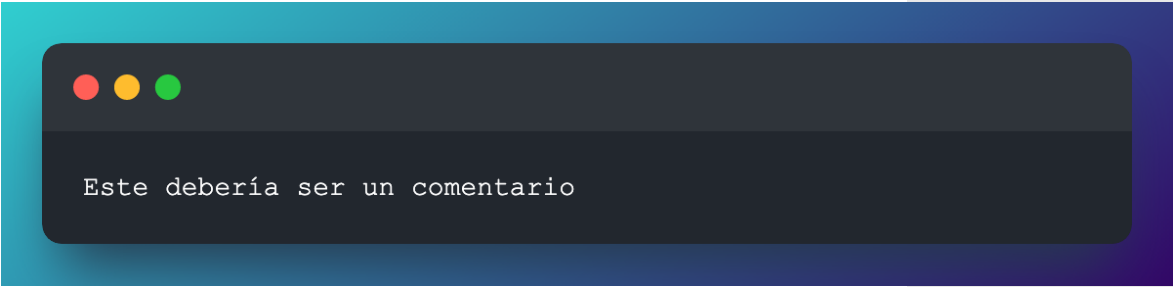
Ejemplo:

Este código es un comentario.



Reto:

Comenta esta línea para que no cause errores y se lea como comentario:

A terminal window with a dark background and three colored window control buttons (red, yellow, green) in the top left corner. The text "Este debería ser un comentario" is displayed in a monospaced font.

```
Este debería ser un comentario
```

4 Tipos y variables

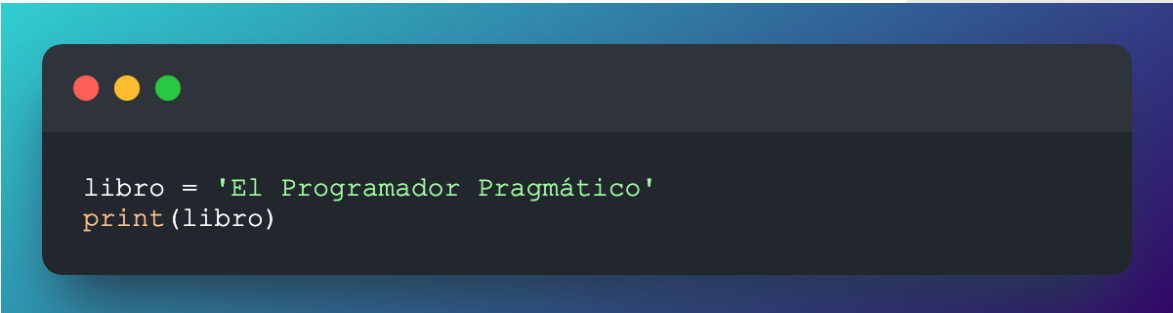
4.1 Variables

La manipulación de datos es una tarea fundamental de un lenguaje de programación. Para trabajar con datos necesitamos guardarlos en variables.

Una variable es un contenedor para almacenar datos que retiene su nombre y puede cambiar su valor a lo largo del tiempo. En los siguientes ejemplos vamos a ver varios tipos de datos que puedes guardar en variables.

Ejemplo:

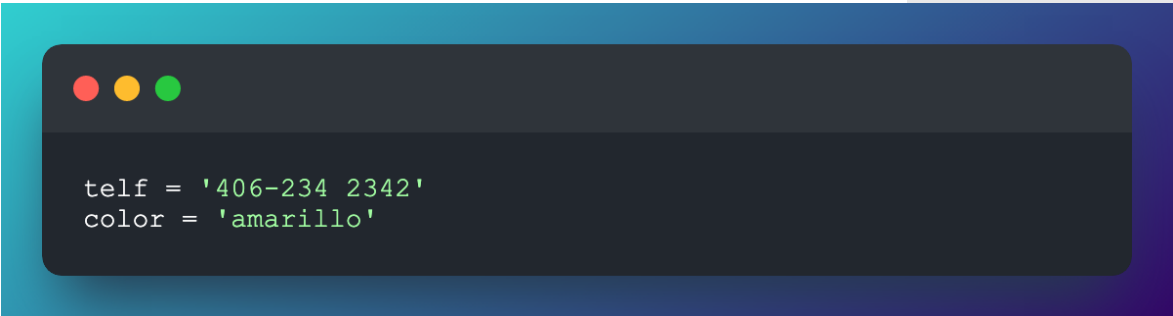
Este código declara una variable llamada “libro” y le asigna el valor de una cadena de texto que contiene el título de un libro. Luego, imprime el valor de la variable “libro” en la consola.

A terminal window with a dark background and three colored window control buttons (red, yellow, green) in the top left corner. The code "libro = 'El Programador Pragmático'" and "print(libro)" is displayed in a monospaced font.

```
libro = 'El Programador Pragmático'  
print(libro)
```

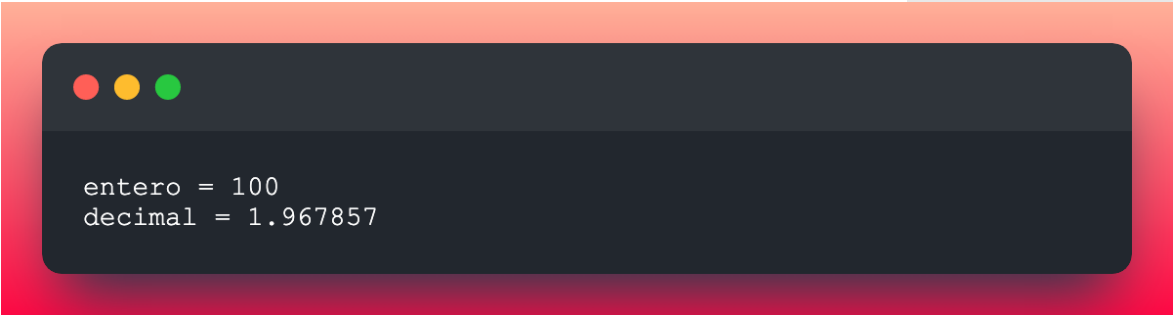
4 TIPOS Y VARIABLES

Texto es un tipo de dato útil para guardar información como números telefónicos y colores, entre otros. Este código asigna dos variables, una llamada `telf` que contiene un número de teléfono como una cadena de caracteres, y otra llamada `color` que contiene el color amarillo como una cadena de caracteres.

A terminal window with a dark background and three colored window control buttons (red, yellow, green) in the top-left corner. The background of the terminal has a blue-to-purple gradient. The code is written in a light green monospace font.

```
telf = '406-234 2342'
color = 'amarillo'
```

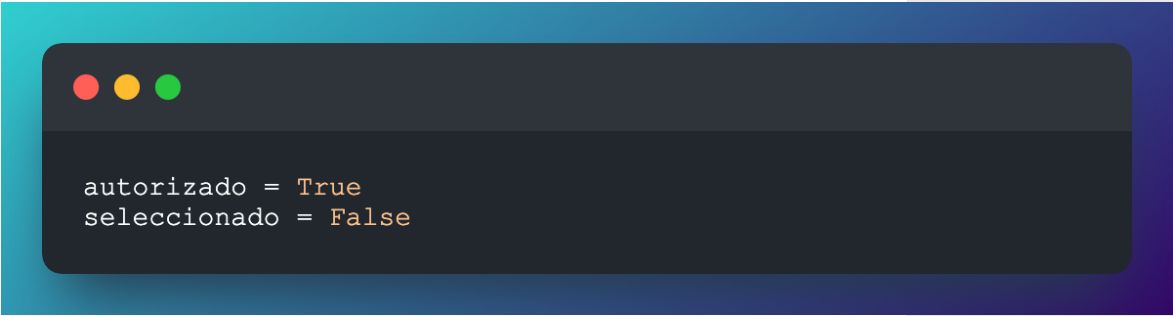
También podemos guardar datos como números enteros y decimales. Estos datos se usan para realizar operaciones matemáticas y representar valores de peso, dinero, entre otros. Este código asigna dos variables, una con un valor entero (100) y otra con un valor decimal (1.967857).

A terminal window with a dark background and three colored window control buttons (red, yellow, green) in the top-left corner. The background of the terminal has a red-to-orange gradient. The code is written in a light green monospace font.

```
entero = 100
decimal = 1.967857
```

El tipo de dato booleano representa los valores de verdadero y falso. Este tipo de datos es útil, por ejemplo, para indicar si un usuario está autorizado a acceder a una app o no, entre varios usos. Este código crea una variable llamada “`autorizado`” que es verdadera y otra variable llamada “`seleccionado`” que es falsa.

4 TIPOS Y VARIABLES

A terminal window with a dark background and three colored window control buttons (red, yellow, green) in the top left corner. The terminal displays two lines of Python code: `autorizado = True` and `seleccionado = False`.

```
autorizado = True
seleccionado = False
```

Es importante saber que, en el mundo del código binario, el número 1 representa verdadero y 0 representa falso.

Reto:

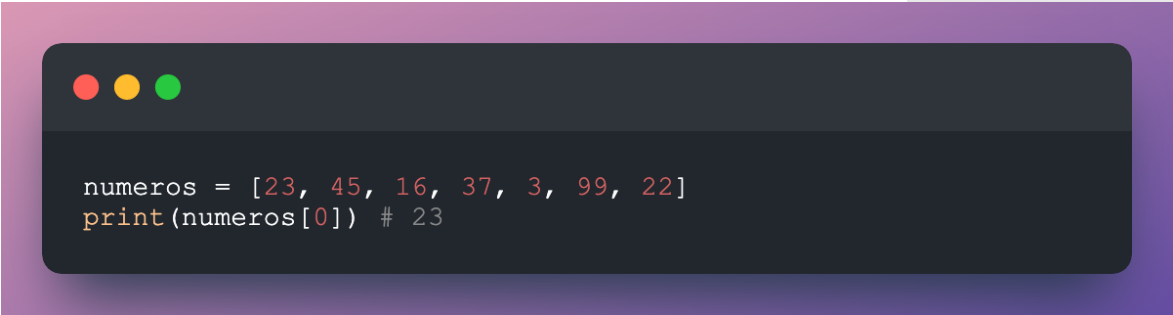
Crea una variable “a” con 33 como texto e imprímela a la consola.

4.2 Listas

Las listas en Python son una estructura de datos que nos permite almacenar una colección ordenada de elementos. Estos elementos pueden ser de cualquier tipo, desde números hasta sublistas. También los vas a encontrar con otros nombres como arreglos, matrices, arrays, etc.

Ejemplo:

Este código está imprimiendo el primer elemento de una matriz de números a la consola. En este caso, el primer elemento es 23.

A terminal window with a dark background and three colored window control buttons (red, yellow, green) in the top left corner. The terminal displays two lines of Python code: `numeros = [23, 45, 16, 37, 3, 99, 22]` and `print(numeros[0]) # 23`.

```
numeros = [23, 45, 16, 37, 3, 99, 22]
print(numeros[0]) # 23
```


4 TIPOS Y VARIABLES

También existen listas de texto. Este código crea una variable llamada “animales” que contiene una lista de elementos, en este caso, tres animales: perro, gato y tigre.



```
animales = ['perro', 'gato', 'tigre']
```

Y esta es una lista de datos mixtos. Este código crea una variable llamada “datosMixtos” que contiene una lista de elementos de diferentes tipos, como texto, números, booleanos y otra lista.



```
datosMixtos = ['texto', 69, True, ['lista dentro de otra lista']]
```

Reto:

Crea una lista en Python que contenga los nombres de los meses del año. Accede al 3er índice e imprímelo en la consola.

4.3 Objetos

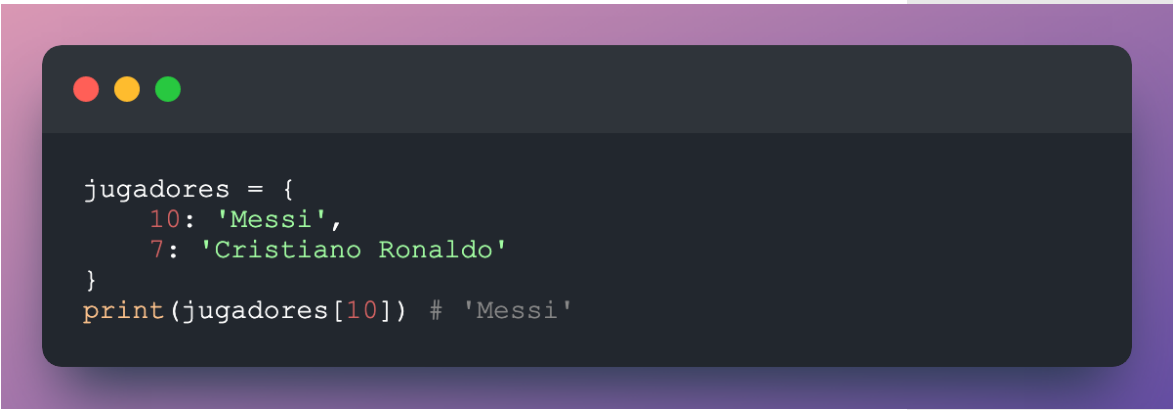
Los objetos en Python son una forma de almacenar y organizar datos mapeándolos de uno a uno. Estos datos se almacenan en forma de pares clave-valor, donde la

4 TIPOS Y VARIABLES

clave suele ser una cadena de texto y el valor puede ser cualquier tipo de dato. También los vas a encontrar con otros nombres como mapas, diccionarios, etc.

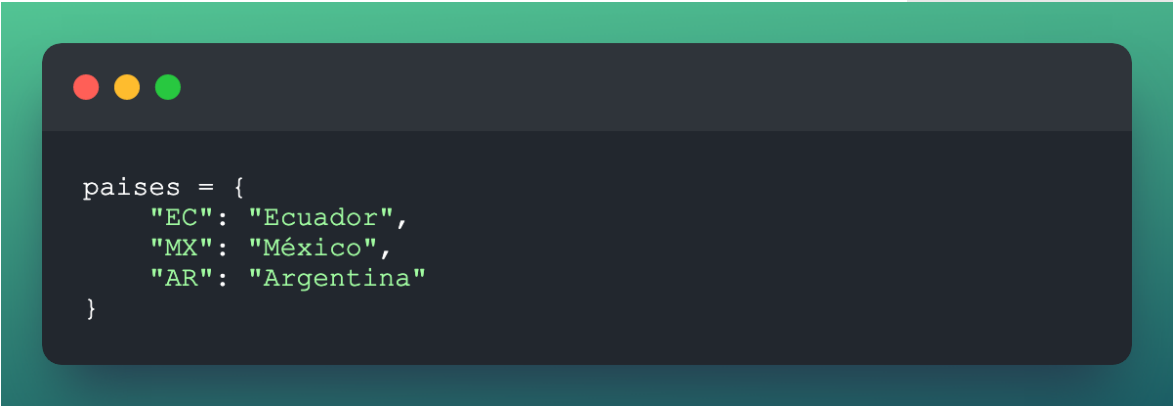
Ejemplo:

Este código crea un objeto llamado jugadores que contiene dos pares clave-valor. El primer par clave-valor es 10: 'Messi' y el segundo es 7: 'Cristiano Ronaldo' . Luego, se imprime el valor asociado con la clave 10, que es 'Messi' .



```
jugadores = {  
    10: 'Messi',  
    7: 'Cristiano Ronaldo'  
}  
print(jugadores[10]) # 'Messi'
```

También podemos mapear de texto a texto. Este código crea un objeto llamado “países” que contiene claves y valores. Las claves son códigos de países (EC, MX, AR) y los valores son los nombres de los países (Ecuador, México, Argentina).



```
países = {  
    "EC": "Ecuador",  
    "MX": "México",  
    "AR": "Argentina"  
}
```

E incluso podemos mapear de texto a listas de texto, entre muchas otras opciones. Este código establece un objeto llamado “emails” que contiene dos pares clave-

4 TIPOS Y VARIABLES

valor. Cada clave es un nombre de persona y el valor es un arreglo de direcciones de correo electrónico asociadas con esa persona.

```
emails = {  
    'Juan': ['juan@gmail.com'],  
    'Ricardo': ['ricardo@gmail.com', 'rick@aol.com']  
}
```

Reto:

Crea un objeto en Python que represente una computadora, con sus respectivas características (marca, modelo, procesador, memoria RAM, etc).

4.4 Constantes

Las constantes son variables que no pueden ser reasignadas. Esto significa que una vez que se asigna un valor a una constante, este no puede ser cambiado.

Ejemplo:

Esta línea de código establece una constante llamada “pi” con un valor de 3.14159265359. Esta constante se puede usar para almacenar un valor numérico que no cambiará a lo largo del programa.

```
pi = 3.14159265359
```

5 OPERADORES

Reto:

Crea una constante llamada 'SALUDO' y asígnale el valor 'Hola Planeta'. Luego, imprime el valor de la constante en la consola.

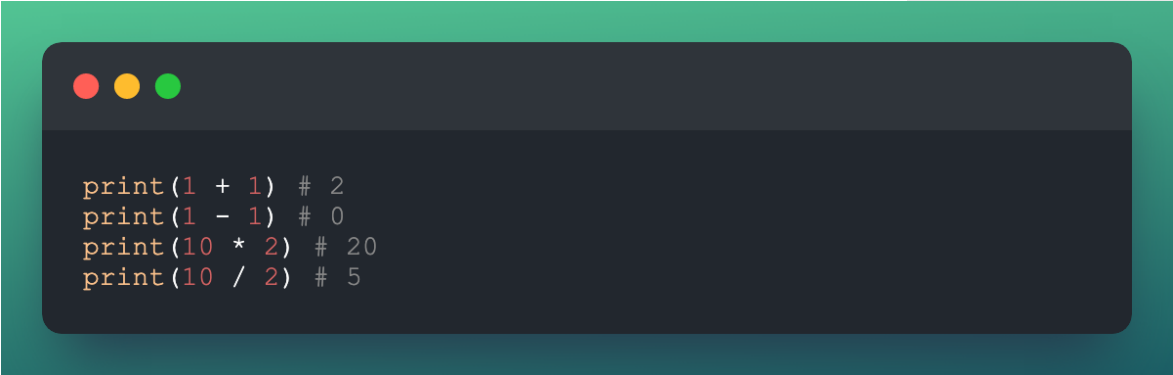
5 Operadores

5.1 Operadores de aritméticos

Los operadores aritméticos en Python son usados para realizar operaciones matemáticas básicas. Estos operadores incluyen sumar, restar, multiplicar, dividir, entre otros.

Ejemplo:

Este código realiza operaciones matemáticas básicas, imprimiendo los resultados en la consola. Estas operaciones incluyen suma, resta, multiplicación y división.

A terminal window with a dark background and a green title bar. It contains four lines of Python code, each followed by a comment showing the result. The code performs addition, subtraction, multiplication, and division.

```
print(1 + 1) # 2
print(1 - 1) # 0
print(10 * 2) # 20
print(10 / 2) # 5
```

Reto:

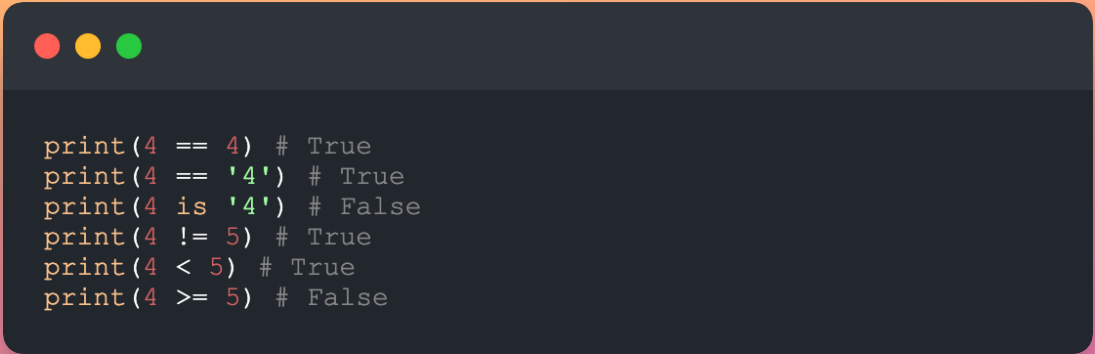
Usa los operadores aritméticos para calcular el área de un círculo con radio 5.

5.2 Operadores comparativos

Los operadores comparativos en Python son usados para comparar dos valores y determinar si son iguales o diferentes. Estos operadores son útiles para determinar si dos valores son iguales, si dos valores no son iguales, si un valor es mayor o menor que otro, entre otros.

Ejemplo:

Este código muestra cómo se usan los operadores para comparar valores.



```
print(4 == 4) # True
print(4 == '4') # True
print(4 is '4') # False
print(4 != 5) # True
print(4 < 5) # True
print(4 >= 5) # False
```

Reto:

Escribe un programa que compare dos números 'a' y 'b' y determina si 'a' con el valor de 4 es mayor, menor o igual a 'b' con un valor de 2.

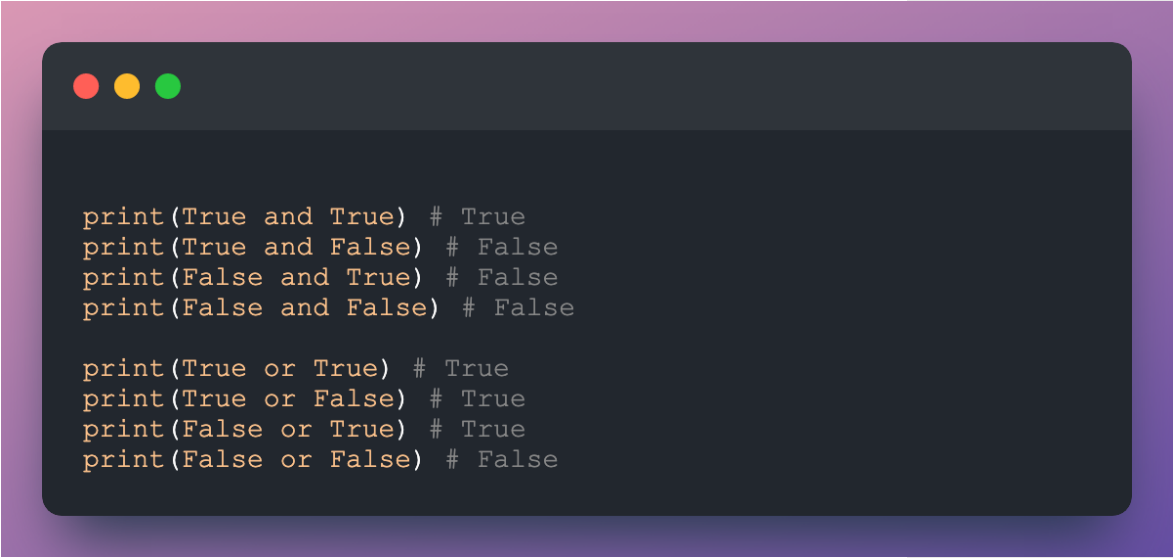
5.3 Operadores lógicos

Los operadores lógicos en Python se usan para realizar comparaciones entre valores y devolver un valor booleano (verdadero o falso). Estos operadores pueden ser útiles, por ejemplo, si deseamos saber si un 'animal' es un gato y es un mamífero (al mismo tiempo).

Ejemplo:

6 CONDICIONALES

Este código muestra el uso de los operadores lógicos 'y' y 'o' para evaluar expresiones booleanas. El operador 'y' devuelve verdadero si ambas expresiones son verdaderas, de lo contrario devuelve falso. El operador 'o' devuelve verdadero si al menos una de las expresiones es verdadera, de lo contrario devuelve falso.



```
print(True and True) # True
print(True and False) # False
print(False and True) # False
print(False and False) # False

print(True or True) # True
print(True or False) # True
print(False or True) # True
print(False or False) # False
```

Reto:

Escribe una línea de código que devuelva verdadero si 'x' es mayor que 0 y 'y' es menor que 0.

6 Condicionales

6.1 Condición única

Los condicionales son estructura de control de flujo en Python, es decir, controlan el flujo del código. Permiten ejecutar una sección de código si una condición es verdadera. También permiten ejecutar otra sección de código, si la condición es falsa.

6 CONDICIONALES

Ejemplo:

Este código comprueba si una variable booleana (autorizado) es verdadera. Si es así, imprime un mensaje en la consola indicando que el usuario puede ingresar. Si no es así, imprime un mensaje indicando que el usuario no puede ingresar.



```
autorizado = True

if autorizado:
    print('Puede ingresar')
else:
    print('No puede ingresar')
```

Reto:

Escribe código condicional que revise si un número 'a' es par o impar. La variable 'a' tiene el valor de 17 y debe imprimir a la consola 'Es par' si es par o 'Es impar' si es impar.

6.2 Múltiples condiciones

Adicionalmente, los condicionales permiten ejecutar varias secciones de código de acuerdo a varias condiciones.

Ejemplo:

En este caso podemos ver que el código que se ejecuta depende de varios valores. Este código comprueba si una variable llamada entero es igual a 99 o 100. Si es igual a 99, imprime "Es 99" en la consola. Si es igual a 100, imprime "Es 100" en la consola. Si no es ni 99 ni 100, imprime "No 99 ni 100" en la consola.

6 CONDICIONALES

```
entero = 100

if entero == 99:
    print('Es 99')
elif entero == 100:
    print('Es 100') # ✓
else:
    print('No 99 ni 100')
```

Este condicional es útil cuando hay una gran cantidad de posibles resultados para una expresión. Este código compara una variable (color) con diferentes valores y ejecuta una acción dependiendo del resultado de la comparación. En este caso, si la variable color es igual a 'amarillo', se imprimirá 'Advertencia' en la consola.

```
color = 'amarillo'

if color == 'verde':
    print('Éxito')
elif color == 'amarillo':
    print('Advertencia') # ✓
else:
    print('Error')
```

Reto:

Crea un programa que evalúe una variable 'numero' y dependiendo del resultado, imprima un mensaje diferente. Si el número es mayor a 10, imprime "El número es

7 BUCLES

mayor a 10” . Si el número es menor a 10, imprime “El número es menor a 10” . Si el número es igual a 10, imprime “El número es igual a 10” .


7 Bucles

7.1 Bucle for

Los bucles son otra estructura de control de flujo que se utilizan para iterar sobre una secuencia de elementos. También se los llama loops o ciclos.

Ejemplo:

Este código itera sobre una lista de animales e imprime cada elemento de la lista en la consola.



```
animales = ['perro', 'gato', 'tigre']  
  
for animal in animales:  
    print(animal)  
  
# 'perro'  
# 'gato'  
# 'tigre'
```

Reto:

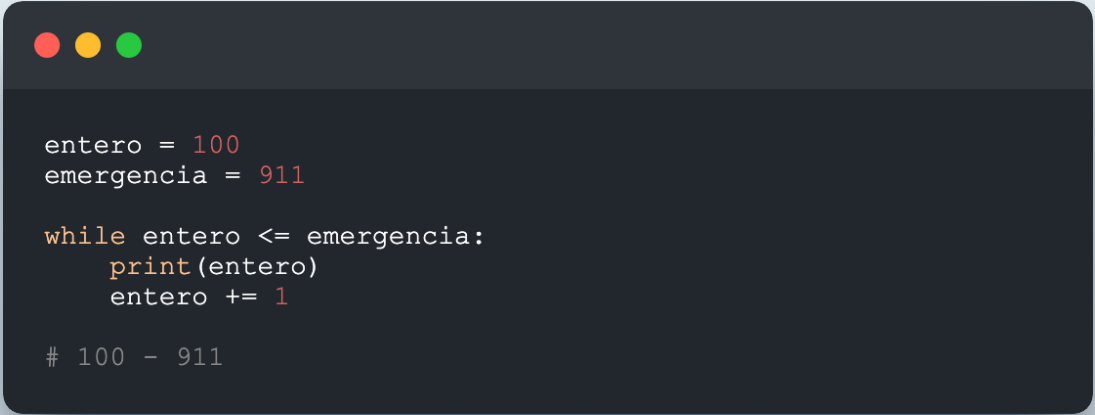
Escribe un bucle que imprima los números del 1 al 10 en orden ascendente.

7.2 Bucle while

while es un tipo de bucle en Python que se usa para ejecutar una serie de instrucciones mientras se cumpla una condición. Esta condición se evalúa antes de cada iteración del bucle.

Ejemplo:

Este código establece un bucle while que imprime los números enteros desde 100 hasta 911 en la consola. El bucle while se ejecutará mientras la variable entero sea menor o igual a la variable emergencia. Cada vez que el bucle se ejecuta, la variable entero se incrementa en 1.



```
entero = 100
emergencia = 911

while entero <= emergencia:
    print(entero)
    entero += 1

# 100 - 911
```

Ten cuidado de no incluir una condición para parar el bucle. Esto podría seguir corriendo tu código indefinidamente hasta congelar tu computador.

Reto:

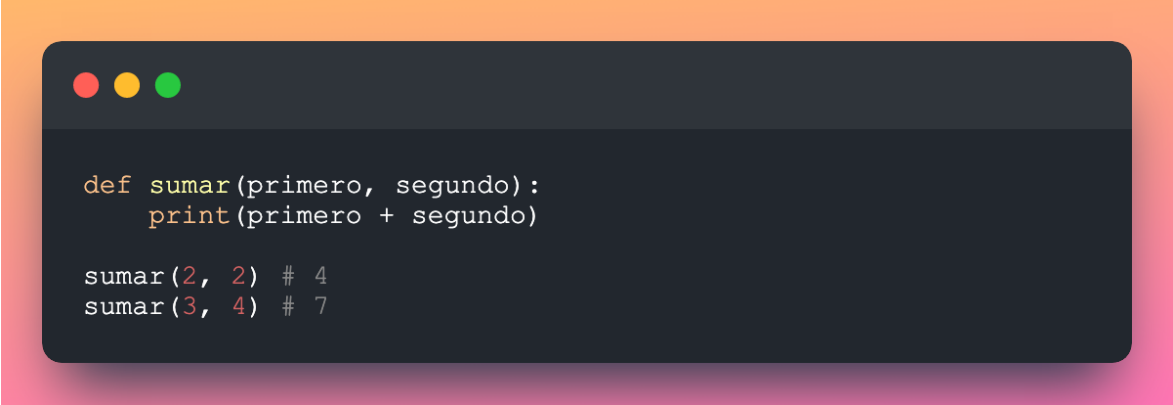
Crea un bucle while que imprima los números del 1 al 10 en la consola.

8 Funciones

En Python, una función es un bloque de código diseñado para realizar una tarea específica y se puede reutilizar en varias partes el código. Las funciones se pueden definir para realizar cualquier tarea, desde realizar cálculos hasta mostrar mensajes en la pantalla.

Ejemplo:

Esta función toma dos argumentos (primero y segundo) de forma dinámica y los suma. Luego, imprime el resultado en la consola. Esta función se llama dos veces con diferentes argumentos para mostrar los resultados de la suma.



```
def sumar(primero, segundo):  
    print(primero + segundo)  
  
sumar(2, 2) # 4  
sumar(3, 4) # 7
```

También puedes crear funciones que retornen un valor. Esto significa que una vez que se ejecuta una función, el valor devuelto se puede usar en otra parte del código. Esta función toma dos argumentos (primero y segundo) y los multiplica para devolver el resultado. En este caso, el resultado es 6.

8 FUNCIONES

```
def multiplicar(primer, segundo):  
    return primero * segundo  
  
resultado = multiplicar(3, 2)  
print(resultado) # 6
```

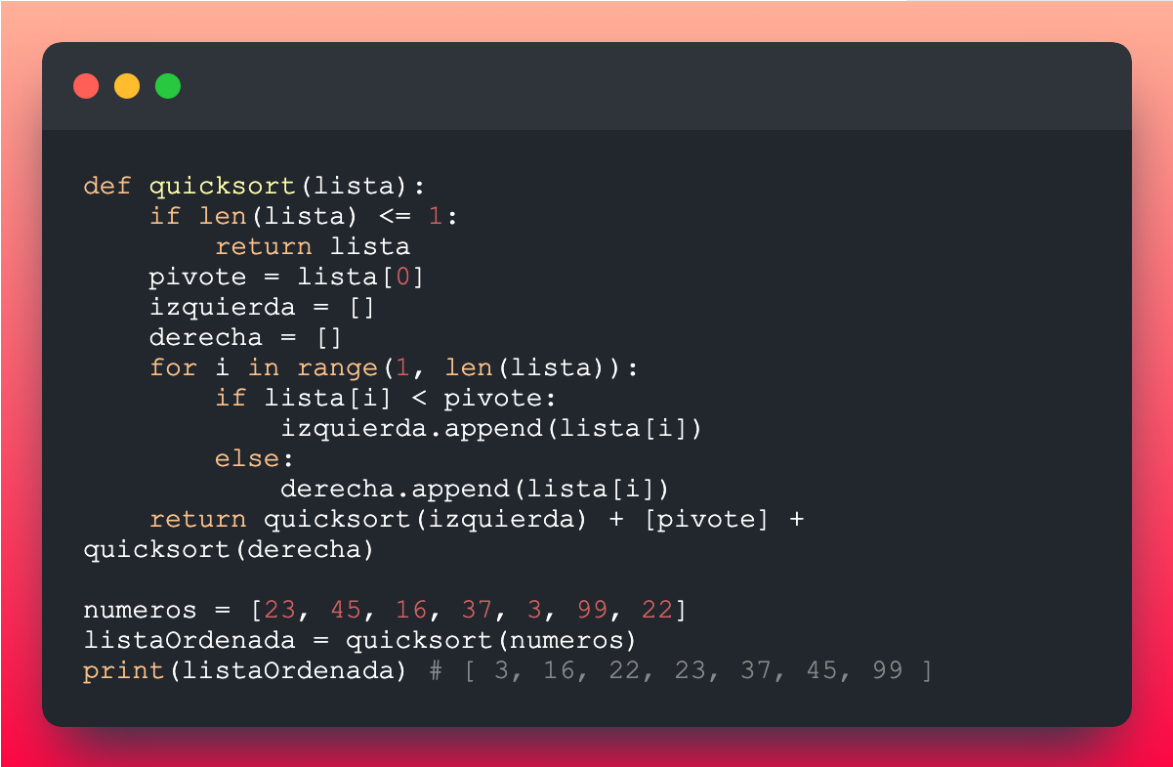
Esta función imprime el primer elemento de una lista. En este caso, la lista es una lista de animales, y la función imprime el primer elemento de la lista, que es 'perro'. Funciones como esta son útiles para evitar la repetición de código y para ahorrar tiempo.

```
def imprimirPrimerElemento(lista):  
    print(lista[0])  
  
animales = ['perro', 'gato', 'tigre']  
imprimirPrimerElemento(animales) # 'perro'
```

Finalmente, puedes ver otro ejemplo de una función bastante compleja. Esta función implementa el algoritmo de ordenamiento QuickSort para ordenar una lista de elementos. El algoritmo comienza tomando un elemento de la lista como pivote y luego coloca los elementos menores que el pivote a su izquierda y los elementos mayores a su derecha. Luego, se aplica el mismo algoritmo a las sublistas izquierda y derecha hasta que la lista esté ordenada. No tienes que entender todo lo que hace internamente pero te dará una idea de la complejidad que pueden tener programas

9 PROGRAMACIÓN ORIENTADA A OBJETOS (POO)

como apps con miles de funciones.



```
def quicksort(lista):  
    if len(lista) <= 1:  
        return lista  
    pivote = lista[0]  
    izquierda = []  
    derecha = []  
    for i in range(1, len(lista)):  
        if lista[i] < pivote:  
            izquierda.append(lista[i])  
        else:  
            derecha.append(lista[i])  
    return quicksort(izquierda) + [pivote] +  
    quicksort(derecha)  
  
numeros = [23, 45, 16, 37, 3, 99, 22]  
listaOrdenada = quicksort(numeros)  
print(listaOrdenada) # [ 3, 16, 22, 23, 37, 45, 99 ]
```

Reto:

Escribe una función que tome dos números como argumentos y devuelva el mayor de los dos.

9 Programación orientada a objetos (POO)

9.1 Paradigma

La Programación Orientada a Objetos (POO) es un paradigma de programación que se basa en la creación de objetos que contienen datos y funcionalidades. Estos

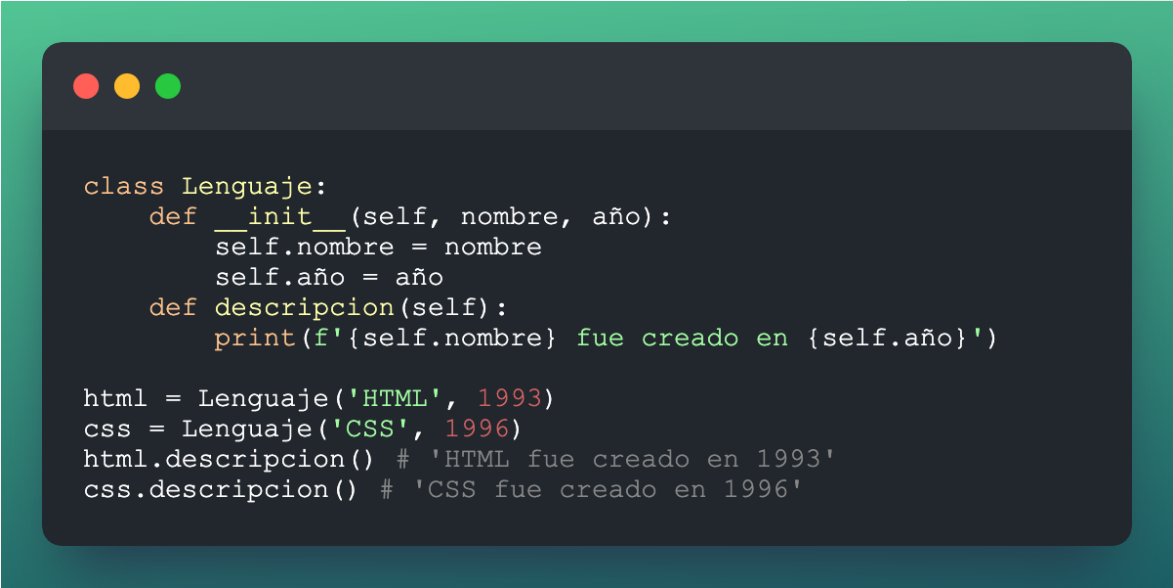
objetos se relacionan entre sí para formar una estructura de datos compleja. Los lenguajes de programación orientados a objetos permiten que los programadores puedan crear objetos y usarlos para construir aplicaciones.

9.2 Clases

Las clases en Python son una forma de definir objetos creando una plantilla. Se pueden usar para crear objetos con propiedades y métodos similares. Las propiedades son valores que pertenecen al objeto y los métodos son funciones que pertenecen al objeto.

Ejemplo:

En este ejemplo creamos una clase llamada 'Lenguaje' que inicializamos con las propiedades 'nombre' y 'año'. Adicionalmente creamos el método 'descripción' que imprime un texto usando las propiedades previas. Con esa clase podemos crear una instancia para el lenguaje 'HTML' y otra para 'CSS'. Ahora podemos crear, en teoría, un número infinito de lenguajes sin reescribir los objetos manualmente.



```
class Lenguaje:
    def __init__(self, nombre, año):
        self.nombre = nombre
        self.año = año
    def descripcion(self):
        print(f'{self.nombre} fue creado en {self.año}')

html = Lenguaje('HTML', 1993)
css = Lenguaje('CSS', 1996)
html.descripcion() # 'HTML fue creado en 1993'
css.descripcion() # 'CSS fue creado en 1996'
```

10 MÓDULOS

Reto:

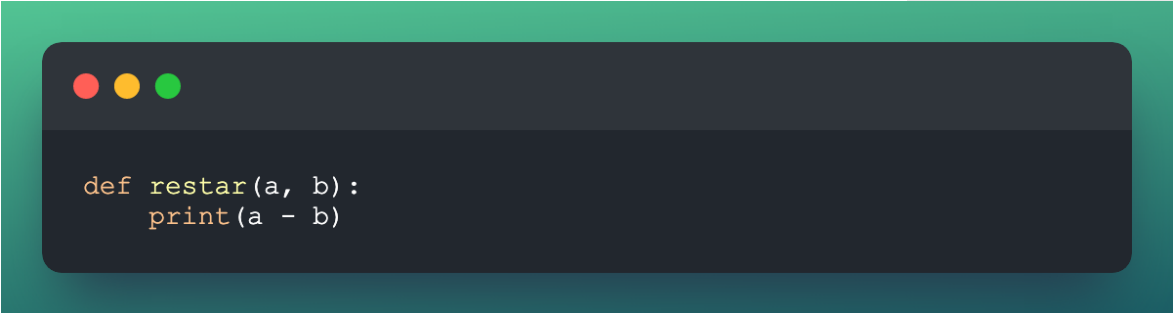
Crea una clase llamada Auto que tenga propiedades como marca, modelo y año. Luego, crea un par de instancias de esta clase.

10 Módulos

Los módulos son una forma de incluir código externo de otro archivo. Esto permite a los desarrolladores reutilizar código y ahorrar tiempo al escribir código.

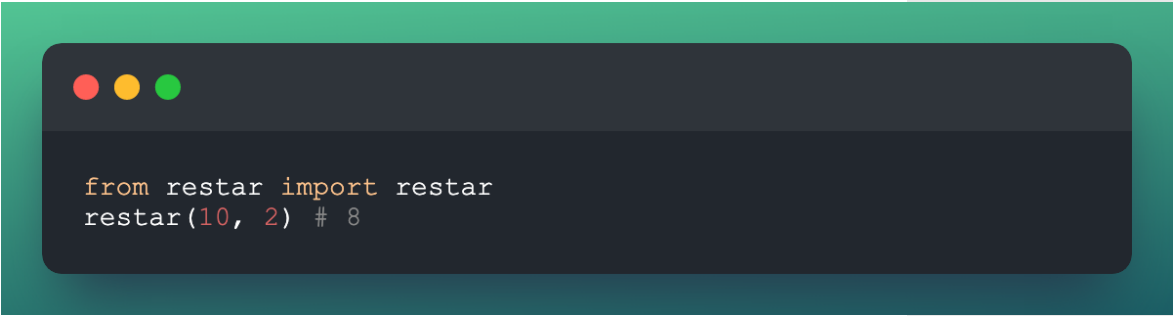
Ejemplo:

Este archivo define una función que toma dos argumentos (a y b) y luego imprime el resultado de la resta de a y b en la consola. El código exporta la función para que pueda ser usada en otros archivos. Exportar es una forma de compartir código entre archivos. Esto significa que un archivo puede exportar variables, funciones y objetos para que otros archivos los puedan usar.



```
def restar(a, b):  
    print(a - b)
```

Y en otro archivo podemos importar y utilizar esta función. Este código importa una función desde el archivo externo y luego la usa para restar 10 menos 2, resultando en 8.



```
from restar import restar
restar(10, 2) # 8
```

Reto:

Crea un archivo llamado que contenga una función para calcular el área de un rectángulo. Luego, importa la función en otro archivo y usala para calcular el área de un rectángulo con lados de 3 y 4.

11 Siguientes pasos

11.1 Herramientas

1. **IPython:** Es una herramienta de línea de comandos interactiva para Python que proporciona un entorno de desarrollo rico y productivo para trabajar con código Python.
2. **Jupyter Notebook:** Es una aplicación web de código abierto que permite crear y compartir documentos que contienen código, ecuaciones, visualizaciones y texto explicativo.
3. **PyCharm:** Es un entorno de desarrollo integrado (IDE) para Python. Ofrece una variedad de herramientas para ayudar a los desarrolladores a escribir, depurar, optimizar y refactorizar código.
4. **Anaconda:** Es una distribución de Python que incluye una amplia gama de paquetes de ciencia de datos, herramientas de análisis y herramientas de desarrollo.

11 SIGUIENTES PASOS

5. **Pytest:** Es un marco de pruebas para Python que permite a los desarrolladores escribir pruebas unitarias y de integración para sus aplicaciones.
6. **Flask:** Es un marco web de Python que permite a los desarrolladores crear aplicaciones web rápidamente.
7. **Django:** Es un marco web de Python de alto nivel que permite a los desarrolladores crear aplicaciones web complejas de forma rápida y sencilla.

11.2 Recursos

1. [Python.org](#): Esta es la página oficial de Python, que contiene documentación, descargas, tutoriales, foros de discusión y mucho más.
2. [Stack Overflow](#): Stack Overflow es una comunidad de desarrolladores que comparten conocimientos y ayudan a otros a resolver problemas de programación.
3. [GitHub](#): GitHub es una plataforma de desarrollo colaborativo que alberga una gran cantidad de proyectos de código abierto escritos en Python.
4. [PyPI](#): PyPI es el repositorio oficial de paquetes de Python, que contiene miles de paquetes de software de terceros que pueden ser descargados e instalados en su proyecto.
5. [RealPython](#): RealPython es un sitio web de recursos de aprendizaje de Python que contiene tutoriales, artículos y ejemplos de código.

11.3 ¿Que viene después?

1. Familiarizarse con los conceptos básicos de programación, como variables, estructuras de datos, bucles, funciones y clases.
2. Practicar escribiendo código en Python.

3. Aprender sobre el uso de bibliotecas y frameworks específicos de Python.
4. Desarrollar proyectos reales con Python.
5. Aprender sobre la seguridad de la información y la seguridad de la aplicación.
6. Investigar sobre la optimización de código y la eficiencia de los algoritmos.
7. Aprender sobre la integración de Python con otros lenguajes de programación.
8. Participar en la comunidad de Python para compartir conocimientos y obtener ayuda.

11.4 Preguntas de entrevista

1. ¿Cuál es la diferencia entre Python 2 y Python 3?
 - Python 3 es la última versión de Python, con mejoras significativas en relación a Python 2. Estas mejoras incluyen una sintaxis más limpia, mejoras en la gestión de memoria, mejoras en la seguridad y una mejor integración con otros lenguajes.
2. ¿Qué es una tupla en Python?
 - Una tupla es una secuencia inmutable de elementos. Está formada por una serie de valores separados por comas, entre paréntesis. Las tuplas son similares a las listas, pero son inmutables, lo que significa que una vez creadas, no se pueden modificar.
3. ¿Cómo se crea una lista en Python?
 - Una lista se puede crear usando corchetes []. Los elementos de la lista se separan por comas. Por ejemplo: `mi_lista = [1, 2, 3, 4]`.
4. ¿Cómo se puede acceder a los elementos de una lista en Python?
 - Los elementos de una lista se pueden acceder usando su índice. Los índices comienzan en 0. Por ejemplo, para acceder al primer elemento de la lista, se usaría `mi_lista[0]`.

5. ¿Cómo se puede agregar un elemento a una lista en Python?

- Se puede agregar un elemento a una lista usando el método `append()`. Por ejemplo, para agregar el número 5 a la lista anterior, se usaría `mi_lista.append(5)`.

6. ¿Cómo se puede eliminar un elemento de una lista en Python?

- Se puede eliminar un elemento de una lista usando el método `remove()`. Por ejemplo, para eliminar el número 5 de la lista anterior, se usaría `mi_lista.remove(5)`.

7. ¿Cómo se puede ordenar una lista en Python?

- Se puede ordenar una lista usando el método `sort()`. Por ejemplo, para ordenar la lista anterior de menor a mayor, se usaría `mi_lista.sort()`.

8. ¿Qué es un diccionario en Python?

- Un diccionario es una estructura de datos que almacena pares clave-valor. Estos pares se separan por comas y están entre llaves `{}`. Por ejemplo: `mi_diccionario = { 'clave1' : 'valor1' , 'clave2' : 'valor2' }`.

9. ¿Cómo se puede acceder a los elementos de un diccionario en Python?

- Los elementos de un diccionario se pueden acceder usando su clave. Por ejemplo, para acceder al primer elemento del diccionario anterior, se usaría `mi_diccionario['clave1']`.

10. ¿Cómo se puede agregar un elemento a un diccionario en Python?

- Se puede agregar un elemento a un diccionario usando la sintaxis de asignación. Por ejemplo, para agregar el par clave-valor `'clave3' : 'valor3'` al diccionario anterior, se usaría `mi_diccionario['clave3'] = 'valor3' .`

11. ¿Cómo se puede eliminar un elemento de un diccionario en Python?

11 SIGUIENTES PASOS

- Se puede eliminar un elemento de un diccionario usando el método `pop()`. Por ejemplo, para eliminar el par clave-valor `'clave3' : 'valor3'` del diccionario anterior, se usaría `mi_diccionario.pop('clave3')`.

12. ¿Qué es una función en Python?

- Una función es un bloque de código que se puede reutilizar para realizar una tarea específica. Las funciones se definen usando la palabra clave `def`. Por ejemplo: `def mi_funcion(): # código aquí`.

13. ¿Cómo se puede llamar a una función en Python?

- Una función se puede llamar usando su nombre seguido de paréntesis. Por ejemplo, para llamar a la función anterior, se usaría `mi_funcion()`.

14. ¿Qué es una clase en Python?

- Una clase es una plantilla para crear objetos. Las clases se definen usando la palabra clave `class`. Por ejemplo: `class MiClase: # código aquí`.

15. ¿Cómo se puede crear un objeto a partir de una clase en Python?

- Un objeto se puede crear a partir de una clase usando la sintaxis de asignación. Por ejemplo, para crear un objeto a partir de la clase anterior, se usaría `mi_objeto = MiClase()`.

16. ¿Qué es una excepción en Python?

- Una excepción es una situación en la que un programa no puede completar una tarea. Las excepciones se manejan usando la palabra clave `try`. Por ejemplo: `try: # código aquí except Exception as e: # código aquí`.

17. ¿Cómo se puede leer un archivo en Python?

- Se puede leer un archivo usando el método `open()`. Por ejemplo, para leer un archivo llamado `mi_archivo.txt`, se usaría `mi_archivo = open('mi_archivo.txt' , 'r')`.

11 SIGUIENTES PASOS

18. ¿Cómo se puede escribir en un archivo en Python?

- Se puede escribir en un archivo usando el método `write()`. Por ejemplo, para escribir en el archivo anterior, se usaría `mi_archivo.write('mi texto aquí')`.

19. ¿Qué es una cadena de caracteres en Python?

- Una cadena de caracteres es una secuencia de caracteres entre comillas. Por ejemplo: `mi_cadena = 'Hola mundo' .`

20. ¿Cómo se puede concatenar cadenas de caracteres en Python?

- Se pueden concatenar cadenas de caracteres usando el operador `+`. Por ejemplo, para concatenar la cadena anterior con otra cadena, se usaría `mi_cadena + ' otra cadena' .`