

# APRENDE PYTHON DESDE CERO HASTA AVANZADO

A C A D E M I A X

## Contenido

<b>1 Introducción</b>	<b>10</b>
1.1 Bienvenida . . . . .	10
1.1.1 Libro vivo . . . . .	11
1.1.2 Alcance . . . . .	11
1.2 Prerequisitos . . . . .	12
1.3 ¿Cómo evitar bloqueos? . . . . .	12
<b>2 Primeros pasos</b>	<b>13</b>
2.1 Visión general . . . . .	13
2.1.1 ¿Qué es y porqué debes aprenderlo? . . . . .	13
2.1.2 ¿En dónde se utiliza? . . . . .	14
2.1.3 ¿Qué trabajos puedes conseguir? . . . . .	14
2.1.4 ¿Cuánto puedes ganar? . . . . .	15
2.1.5 ¿Cuales son las preguntas más comunes? . . . . .	15
2.2 Historia, evolución, y versiones . . . . .	16
2.3 Características, ventajas, y diferencias . . . . .	16
2.4 Configuración . . . . .	18
2.4.1 IDE . . . . .	18
2.4.2 Entorno . . . . .	18
2.5 Hola Mundo . . . . .	19
2.6 Comentarios (una sola línea y multilínea) . . . . .	20
2.7 Sintaxis . . . . .	20
<b>3 Variables y tipos de datos</b>	<b>21</b>
3.1 Literales y tipos de datos . . . . .	21
3.2 Variables . . . . .	22
3.3 Reglas para nombrar variables . . . . .	24
3.4 Sensibilidad de mayúsculas y minúsculas . . . . .	25
3.5 Conjunto de caracteres . . . . .	26

## Contenido

---

3.6 Palabras clave e identificadores . . . . .	27
3.7 Cambiar el valor de una variable . . . . .	27
3.8 Asignar multiples variables . . . . .	28
3.9 Constantes . . . . .	29
3.10 Tipos de datos . . . . .	30
3.10.1 Números (int, float, complex) . . . . .	30
3.10.2 Texto (str) . . . . .	31
3.10.3 Booleanos (bool) . . . . .	31
3.10.4 Secuencias (list, tuple, range) . . . . .	32
3.10.5 Diccionarios (dict) . . . . .	34
3.10.6 Conjuntos (set, frozenset) . . . . .	35
3.10.7 Binarios (bytes, bytearray, memoryview) . . . . .	36
3.10.8 NoneType . . . . .	38
3.11 Formato de texto (f-Strings) . . . . .	39
3.12 Chequear tipo . . . . .	40
3.13 Sentencias, declaraciones, y tipado . . . . .	41
3.14 Conversión de tipos (casting) . . . . .	41
<b>4 Operadores</b> . . . . .	<b>42</b>
4.1 Operadores y expresiones . . . . .	42
4.2 Operadores aritméticos (+, -, *, /, %, **, //) . . . . .	43
4.3 Operadores de asignación (=, +=, -=, *=, /=, %=, **=, //=) . . . . .	45
4.4 Operadores comparativos (==, !=, >, <, >=, <=) . . . . .	48
4.5 Operadores lógicos (and, or, not) . . . . .	49
4.6 Operadores de membresía (in, not in) . . . . .	51
4.7 Operadores de identidad (is, is not) . . . . .	52
4.8 Operadores de bits (&,  , ~, ^, », «) . . . . .	53
<b>5 Estructuras de control de flujo</b> . . . . .	<b>57</b>
5.1 Estructuras de control de flujo y excepciones . . . . .	57
5.2 Bloques e indentación . . . . .	58

---

## Contenido

---

5.3	Condicionales . . . . .	59
5.3.1	if . . . . .	59
5.3.2	if...else . . . . .	59
5.3.3	if...elif...else . . . . .	60
5.3.4	Operador ternario . . . . .	61
5.3.5	Condicionales anidados . . . . .	62
5.3.6	match . . . . .	63
5.4	Bucles . . . . .	64
5.4.1	for . . . . .	64
5.4.2	continue . . . . .	65
5.4.3	break . . . . .	66
5.4.4	pass . . . . .	67
5.4.5	for...else . . . . .	68
5.4.6	while . . . . .	68
5.4.7	while...else . . . . .	69
5.5	Excepciones . . . . .	70
5.5.1	Tipos de excepciones . . . . .	70
5.5.2	raise . . . . .	72
5.5.3	try...except . . . . .	73
5.5.4	try...except...finally . . . . .	74
<b>6</b>	<b>Funciones</b> . . . . .	<b>75</b>
6.1	Declarar y llamar funciones . . . . .	75
6.2	Parámetros y argumentos . . . . .	76
6.3	Argumentos con valores predeterminados . . . . .	77
6.4	Argumentos arbitrarios . . . . .	78
6.5	Retorno de valores . . . . .	79
6.6	Ámbito global y local . . . . .	79
6.7	La palabra global . . . . .	80
6.8	La palabra nonlocal . . . . .	82
6.9	Funciones anidadas . . . . .	84

---

## Contenido

---

6.10 Cierres (closures) . . . . .	85
6.11 Recursividad . . . . .	86
6.12 Funciones anónimas o lambda . . . . .	87
6.13 Funciones incorporadas (built-in) . . . . .	89
6.13.1 range() . . . . .	89
6.13.2 print() . . . . .	89
6.13.3 len() . . . . .	90
6.13.4 pow() . . . . .	91
<b>7 Números</b>	<b>92</b>
7.1 Números binarios . . . . .	92
7.2 Números octales . . . . .	93
7.3 Números hexadecimales . . . . .	93
<b>8 Texto</b>	<b>94</b>
8.1 Formas de declarar texto . . . . .	94
8.2 Comillas simples vs comillas dobles . . . . .	95
8.3 Texto multilínea . . . . .	97
8.4 Secuencias de escape . . . . .	98
8.5 Más sobre f-strings . . . . .	98
8.6 Acceder a un carácter en texto . . . . .	100
8.7 Función len() en texto . . . . .	100
8.8 Métodos de texto . . . . .	101
8.8.1 Método de texto upper() . . . . .	101
8.8.2 Método de texto lower() . . . . .	102
8.8.3 Método de texto partition() . . . . .	103
8.8.4 Método de texto replace() . . . . .	103
8.8.5 Método de texto find() . . . . .	104
8.8.6 Método de texto strip() . . . . .	105
8.8.7 Método de texto split() . . . . .	106
8.8.8 Iterar a través de un texto . . . . .	106

## Contenido

---

8.8.9 Método de texto startswith() . . . . .	107
8.8.10 Método de texto isnumeric() . . . . .	108
<b>9 Listas . . . . .</b>	<b>109</b>
9.1 Crear una lista . . . . .	109
9.2 Los índices comienzan en cero . . . . .	109
9.3 Acceder a elementos de una lista . . . . .	110
9.4 Modificar elementos de una lista . . . . .	111
9.5 Borrar elementos de una lista . . . . .	112
9.6 Ordenar listas . . . . .	113
9.7 Comprensión de listas . . . . .	113
9.8 Función len() en listas . . . . .	115
9.9 Iterar a través de una lista . . . . .	115
9.10 Métodos de listas . . . . .	116
9.10.1 Método de lista append() . . . . .	116
9.10.2 Método de lista extend() . . . . .	117
9.10.3 Método de lista insert() . . . . .	117
9.10.4 Método de lista remove() . . . . .	118
9.10.5 Método de lista pop() . . . . .	119
9.10.6 Método de lista clear() . . . . .	120
9.10.7 Método de lista index() . . . . .	120
9.10.8 Método de lista count() . . . . .	121
<b>10 Tuplas . . . . .</b>	<b>122</b>
10.1 Crear una tupla . . . . .	122
10.2 Acceder a elementos de una tupla . . . . .	123
10.3 Índices negativos en una tupla . . . . .	123
10.4 Inmutabilidad . . . . .	124
10.5 Modificar elementos de una tupla . . . . .	125
10.6 Borrar elementos de una tupla . . . . .	126
10.7 Iterar a través de una tupla . . . . .	126

## Contenido

---

10.8 Métodos de tuplas . . . . .	127
10.8.1 Método de tupla count() . . . . .	127
10.8.2 Método de tupla index() . . . . .	128
<b>11 Sets</b>	<b>129</b>
11.1 Crear un set . . . . .	129
11.2 Acceder a elementos de un set . . . . .	129
11.3 Modificar elementos de un set . . . . .	130
11.4 Borrar elementos de un set . . . . .	131
11.5 Iterar a través de un set . . . . .	132
11.6 Métodos de sets . . . . .	133
11.6.1 Método de set union() . . . . .	133
11.6.2 Método de set intersection() . . . . .	134
11.6.3 Método de set difference() . . . . .	135
11.6.4 Método de set symmetric_difference() . . . . .	135
<b>12 Diccionarios</b>	<b>136</b>
12.1 Crear un diccionario . . . . .	136
12.2 Acceder a elementos de un diccionario . . . . .	137
12.3 Modificar elementos de un diccionario . . . . .	138
12.4 Borrar elementos de un diccionario . . . . .	139
12.5 Métodos de diccionario . . . . .	140
12.5.1 Método de diccionario get() . . . . .	140
12.5.2 Método de diccionario update() . . . . .	141
12.5.3 Método de diccionario keys() . . . . .	142
12.5.4 Método de diccionario values() . . . . .	143
12.5.5 Método de diccionario items() . . . . .	144
12.5.6 Iterar a través de un diccionario . . . . .	145
12.5.7 Método de diccionario setdefault() . . . . .	146
12.5.8 Método de diccionario popitem() . . . . .	147

## Contenido

---

<b>13 Programación orientada a objetos (POO)</b>	<b>148</b>
13.1 Paradigma . . . . .	148
13.2 Objetos . . . . .	149
13.2.1 Crear un objeto . . . . .	149
13.2.2 dir() . . . . .	150
13.2.3 Acceder a elementos de un objeto . . . . .	151
13.2.4 Borrar objetos . . . . .	151
13.3 Clases . . . . .	152
13.3.1 Crear una clase . . . . .	152
13.3.2 Función constructora . . . . .	154
13.3.3 El parámetro self . . . . .	155
13.3.4 Herencia y polimorfismo . . . . .	156
13.3.5 Encapsulamiento y abstracción . . . . .	157
<b>14 Módulos y paquetes</b>	<b>158</b>
14.1 Importación y uso de módulos externos . . . . .	158
14.2 Importar como alias . . . . .	159
14.3 from...import . . . . .	159
14.4 Importar todo (*) . . . . .	160
14.5 Exportar módulos . . . . .	161
14.6 Creación y uso de paquetes ( <code>__init__.py</code> ) . . . . .	162
14.7 Módulos comunes . . . . .	163
14.7.1 Módulo random . . . . .	163
14.7.2 Módulo math . . . . .	164
14.7.3 Módulo datetime (Fechas) . . . . .	165
14.7.4 Módulo re (Expresiones regulares) . . . . .	166
<b>15 Manejo de archivos</b>	<b>167</b>
15.1 Lectura de archivos . . . . .	167
15.2 with...open . . . . .	168
15.3 Crear y modificar archivos . . . . .	168

---

## Contenido

---

15.4 Borrar archivos . . . . .	170
15.5 Cambiar carpetas . . . . .	171
15.6 Listar carpetas y archivos . . . . .	171
15.7 Obtener carpetas . . . . .	172
15.8 Crear y modificar una carpeta . . . . .	173
15.9 Métodos de archivos . . . . .	174
15.9.1 Método de archivo readline() . . . . .	174
15.9.2 Método de archivo writelines() . . . . .	175
15.9.3 Método de archivo seek() . . . . .	176
15.9.4 Método de archivo tell() . . . . .	178
<b>16 Bibliotecas y frameworks</b>	<b>179</b>
16.1 Uso de bibliotecas (NumPy, Pandas, Matplotlib) . . . . .	179
16.2 Desarrollo de aplicaciones web con frameworks (Django o Flask) . . . . .	179
<b>17 Avanzado</b>	<b>180</b>
17.1 Sobrecarga de operadores . . . . .	180
17.2 Iteradores . . . . .	181
17.3 Generadores . . . . .	182
17.4 *args y **kwargs . . . . .	183
17.5 Programación funcional y decoradores . . . . .	185
17.6 Buffer de lectura de archivos . . . . .	186
17.7 Buenas prácticas y patrones de diseño . . . . .	188
<b>18 Siguientes pasos</b>	<b>189</b>
18.1 Herramientas . . . . .	189
18.2 Recursos . . . . .	189
18.3 ¿Que viene después? . . . . .	190
18.4 Preguntas de entrevista . . . . .	190

# 1 Introducción

## 1.1 Bienvenida

Bienvenid@ a este libro de Academia X en donde aprenderás Python práctico.

Hola, mi nombre es Xavier Reyes Ochoa y soy el autor de este libro. 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 este libro vas a aprender estos temas:

- Primeros pasos
- Variables y tipos de datos
- Operadores
- Estructuras de control de flujo
- Funciones
- Números
- Texto
- Listas
- Tuplas
- Sets
- Diccionarios
- Programación orientada a objetos (POO)
- Módulos y paquetes
- Manejo de archivos
- Bibliotecas y frameworks
- Avanzado

La motivación de este libro es darte todo el conocimiento técnico que necesitas para

## 1 INTRODUCCIÓN

---

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 este libro te dará los recursos que necesitas para que tengas éxito en este campo.

Empecemos!

### 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 (Feb 13 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 25 de julio 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 este libro 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 oficiales, populares, y avanzados en la lección de recursos.

## 1 INTRODUCCIÓN

---

### 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.
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 entendiendo los conceptos en este libro.

### 1.3 ¿Cómo evitar bloqueos?

Para sacarle el mayor provecho a este libro:

## 2 PRIMEROS PASOS

---

1. No solo leas este libro. 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 porqué 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 PRIMEROS PASOS

---

### 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.

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 PRIMEROS PASOS

---

### 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 ¿Cuáles 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 fácilmente.

### 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, ventajas, y diferencias

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 programas con una cantidad mínima de código.

Características:

- Lenguaje de alto nivel: Python es un lenguaje de alto nivel, lo que significa que está diseñado para ser fácil de leer y escribir. Esto hace que sea un lenguaje ideal para principiantes, ya que es más fácil de entender que otros lenguajes de programación.
- Interpretado: Python es un lenguaje interpretado, lo que significa que no necesita ser compilado antes de ser ejecutado. Esto hace que sea más fácil de

## 2 PRIMEROS PASOS

---

depurar y modificar, ya que los errores se pueden detectar y corregir más rápidamente.

- Multiparadigma: Python es un lenguaje multiparadigma, lo que significa que admite varios estilos de programación, como programación orientada a objetos, programación funcional y programación imperativa. Esto hace que sea un lenguaje versátil y flexible.
- Multiplataforma: Python es un lenguaje multiplataforma, lo que significa que se puede ejecutar en varias plataformas, como Windows, Mac OS X y Linux. Esto hace que sea un lenguaje ideal para desarrollar aplicaciones que se pueden ejecutar en varias plataformas.

Ventajas:

- Fácil de aprender: Python es un lenguaje fácil de aprender, ya que está diseñado para ser fácil de leer y escribir. Esto hace que sea un lenguaje ideal para principiantes.
- Flexible: Python es un lenguaje flexible, ya que admite varios estilos de programación. Esto hace que sea un lenguaje ideal para desarrollar aplicaciones que requieren una gran cantidad de flexibilidad.
- Potente: Python es un lenguaje potente, ya que admite una gran cantidad de bibliotecas y herramientas. Esto hace que sea un lenguaje ideal para desarrollar aplicaciones complejas.

Diferencias con otros lenguajes de programación:

- Python es un lenguaje de alto nivel, mientras que otros lenguajes de programación, como C y Java, son lenguajes de bajo nivel.
- Python es un lenguaje interpretado, mientras que otros lenguajes de programación, como C y Java, son lenguajes compilados.
- Python es un lenguaje multiparadigma, mientras que otros lenguajes de programación, como C y Java, son lenguajes de un solo paradigma.

## 2 PRIMEROS PASOS

---

- Python es un lenguaje multiplataforma, mientras que otros lenguajes de programación, como C y Java, son lenguajes de una sola plataforma.
- En Python, a diferencia de otros lenguajes, la indentación es importante. Esto lo verás más adelante.

### 2.4 Configuración

#### 2.4.1 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/>

#### 2.4.2 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

## 2 PRIMEROS PASOS

---

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:

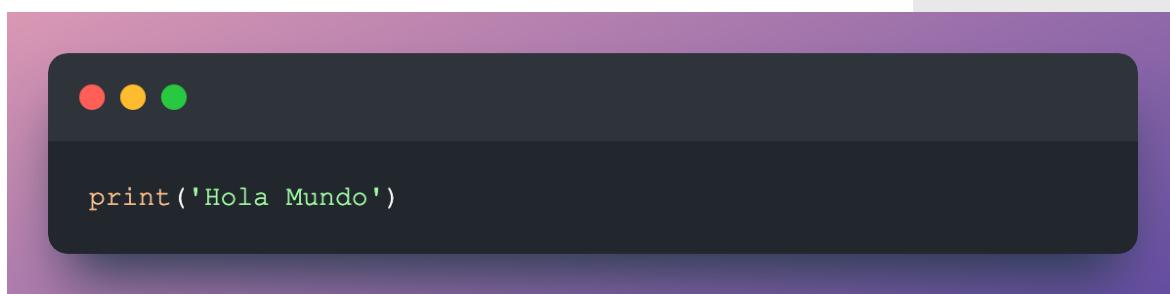
1. En la línea de comandos: Puedes abrir una terminal en tu sistema operativo y escribir **python3 nombre\_del\_archivo.py** Esto ejecutará el archivo de Python en cuestión.
2. Con un editor de código: Si estás utilizando un editor de código como PyCharm, Visual Studio Code, etc., puedes abrir tu archivo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

### 2.5 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.



A screenshot of a terminal window with a dark background. At the top, there are three small colored circles (red, yellow, green). Below them, the Python code `print('Hola Mundo')` is written in white text. The terminal window is set against a background that has a gradient from pink to purple.

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

Reto:

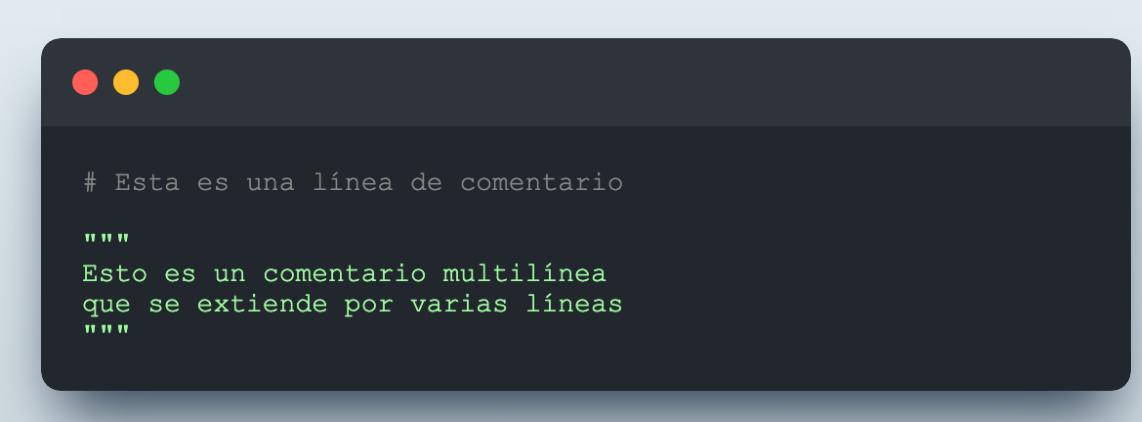
## 2 PRIMEROS PASOS

---

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

### 2.6 Comentarios (una sola línea y multilínea)

Los comentarios en Python son líneas de texto que se ignoran al momento de ejecutar el código. Se usan para documentar el código y explicar lo que hace. Los comentarios de una sola línea se escriben con el símbolo #, mientras que los comentarios multilínea se escriben entre 3 comillas dobles ( """ ). Ejemplo:



```
# Esta es una línea de comentario

"""
Esto es un comentario multilínea
que se extiende por varias líneas
"""
```

Reto:

Escribe un programa que imprima 3 en pantalla y documenta el código con el comentario “# Este código imprime 3” .

### 2.7 Sintaxis

Al igual que un idioma o lenguaje como el inglés, el francés, o el mandarin, un lenguaje de programación sigue reglas gramaticales y una palabra que vas a escuchar frecuentemente en el campo de la programación es la sintaxis.

La sintaxis de un lenguaje de programación es la estructura de un lenguaje de programación, que incluye reglas para la construcción de programas. Estas reglas se

### **3 VARIABLES Y TIPOS DE DATOS**

---

refieren a la forma en que los elementos de un programa se escriben, se ordenan, se interpretan, y se relacionan entre sí.

A lo largo de este libro vamos a aprender temas como variables, operadores, bucles, funciones, objetos, propiedades, métodos, etc, que son como fórmulas que permiten escribir funcionalidades comunes como sumar números, procesar texto, o representar objetos del mundo real en código.

También veremos conceptos gramaticales de este lenguaje como palabras claves, identificadores, sentencias, literales, expresiones, etc, para entender como se interpreta un programa y así escribirlo de manera correcta para evitar errores.

Finalmente, revisaremos buenas prácticas que son una manera de escribir código para crear programas que permitan un buen trabajo en equipo, mantenimiento a largo plazo, y la creación de aplicaciones escalables.

## **3 Variables y tipos de datos**

### **3.1 Literales y tipos de datos**

Literales y tipos de datos en Python son los elementos básicos para trabajar con el lenguaje. Los literales son valores que se escriben tal cual, como números, cadenas de texto, booleanos, etc. Los tipos de datos son los diferentes tipos de literales que se pueden usar en Python.

Ejemplo:

### 3 VARIABLES Y TIPOS DE DATOS

---

```
# Esto es un literal de tipo entero  
10  
  
# Esto es un literal de tipo cadena de texto  
"texto"  
  
# Esto es un literal de tipo booleano  
True
```

Reto:

Escribe el literal 22 en la consola.

## 3.2 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.

### 3 VARIABLES Y TIPOS DE DATOS

---

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

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.

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

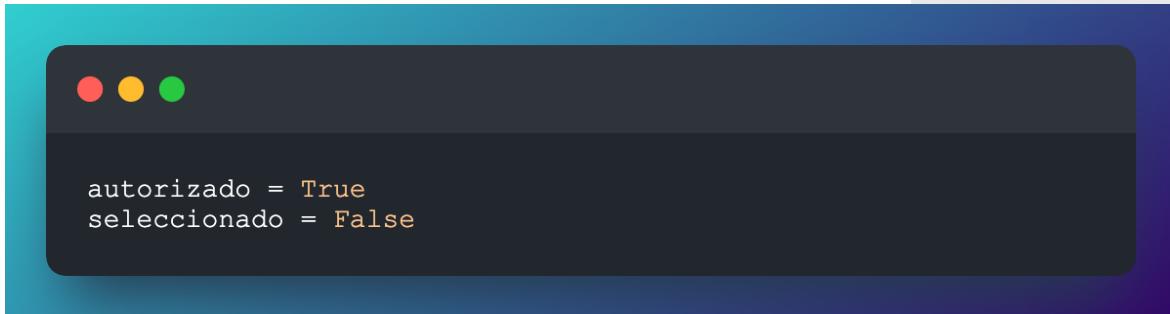
También podemos guardar datos como números enteros y decimales. Estos datos se usan para realizar operaciones matemáticas y representar valores de peso, dinero, entre otros. Este código asigna dos variables, una con un valor entero (100) y otra con un valor decimal (1.967857).

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

### 3 VARIABLES Y TIPOS DE DATOS

---

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.



```
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.

### 3.3 Reglas para nombrar variables

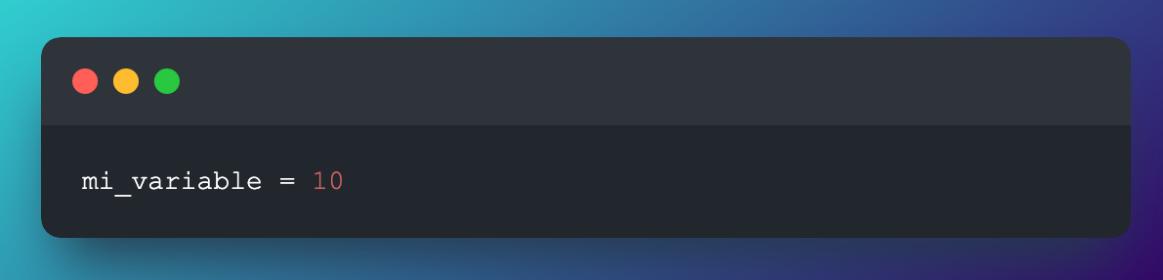
Las reglas para nombrar variables en Python son bastante sencillas:

1. Los nombres de variables deben comenzar con una letra o un guión bajo.
2. Los nombres de variables pueden contener letras, números y guiones bajos.
3. Los nombres de variables no pueden contener espacios.
4. Los nombres de variables no pueden comenzar con un número.
5. Los nombres de variables no pueden contener caracteres especiales, como !, @, #, \$, etc.

Ejemplo:

### 3 VARIABLES Y TIPOS DE DATOS

---



```
mi_variable = 10
```

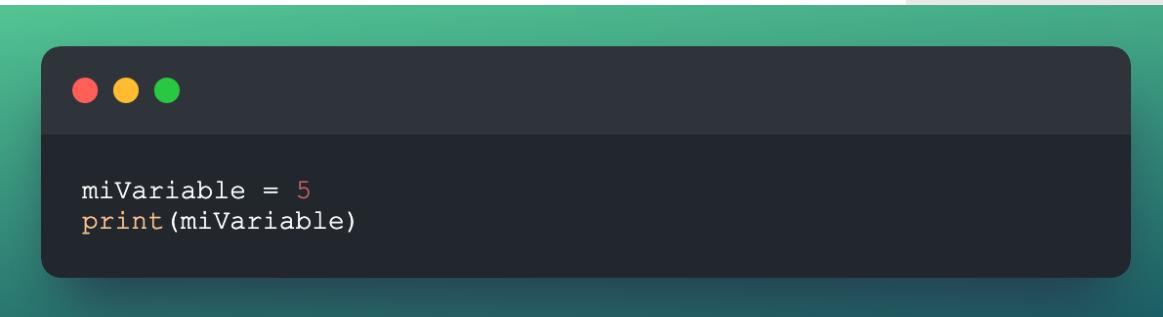
Reto:

Crea una variable llamada mi\_auto y asígnale el valor de “Tesla” .

#### 3.4 Sensibilidad de mayúsculas y minúsculas

Python es sensible a mayúsculas y minúsculas. Esto significa que los nombres de variables, funciones y clases deben escribirse exactamente como se definieron. Por ejemplo, si definimos una variable llamada “miVariable” , no podemos usar “Mivariable” o “mivariable” para referirnos a ella.

Ejemplo:



```
miVariable = 5
print(miVariable)
```

Reto:

Escribe un programa que declare el valor de 4 en la variable “Mivariable” e imprima el valor de una variable llamada “miVariable” para ver el error.

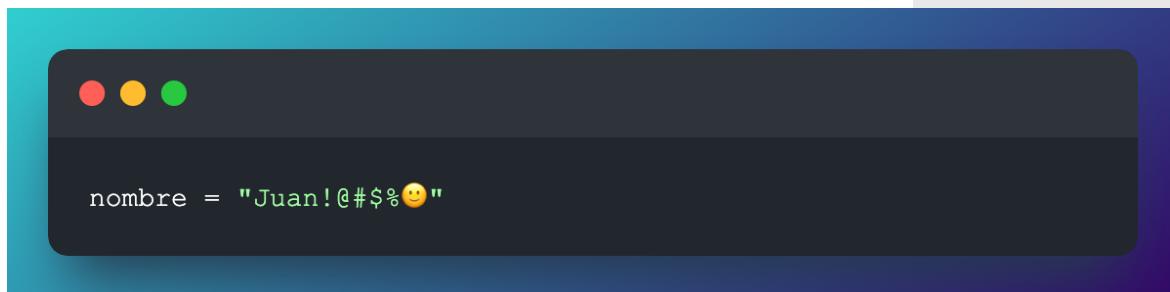
### 3 VARIABLES Y TIPOS DE DATOS

---

#### 3.5 Conjunto de caracteres

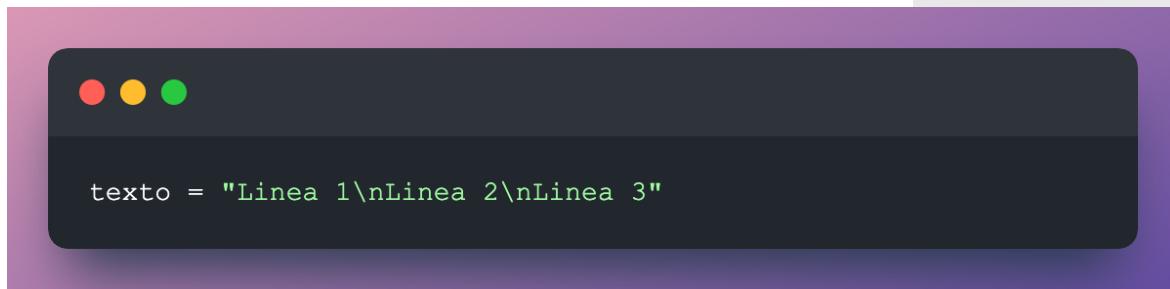
Python admite una amplia variedad de conjuntos de caracteres, incluyendo ASCII, Unicode y UTF-8. Por defecto, las cadenas en Python 3 son cadenas Unicode, lo que significa que pueden contener caracteres de muchos alfabetos diferentes e incluso emojis.

Ejemplo:



```
nombre = "Juan!@#$%😊"
```

También puedes incluir caracteres especiales en tus cadenas, como tabulaciones (`\t`) y saltos de línea (`\n`):



```
texto = "Linea 1\nLinea 2\nLinea 3"
```

Además, en Python también existe el tipo de datos **bytes**, que es una secuencia de números enteros en el rango de 0 a 255 que representan los valores ASCII de los caracteres.

Reto:

Crea una variable llamada **nombre** y asígnale el nombre Pablo Santana separado por un salto de línea.

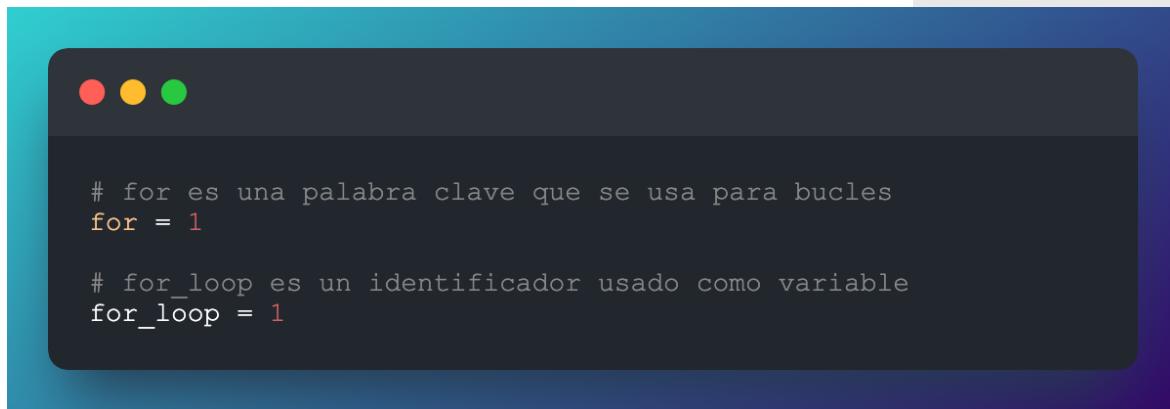
### 3 VARIABLES Y TIPOS DE DATOS

---

#### 3.6 Palabras clave e identificadores

Palabras clave e identificadores son los elementos básicos de la sintaxis de Python. Las palabras clave son palabras reservadas que tienen un significado especial en el lenguaje de programación. Estas palabras clave no pueden ser usadas como identificadores. Los identificadores son nombres asignados a variables, funciones, clases, etc.

Ejemplo:



The screenshot shows a terminal window with a dark background and three colored dots (red, yellow, green) at the top. The terminal displays the following Python code:

```
# for es una palabra clave que se usa para bucles
for = 1

# for_loop es un identificador usado como variable
for_loop = 1
```

Reto:

Escribe `for = 1` en la consola para ver que obtienes un error.

#### 3.7 Cambiar el valor de una variable

En Python, cambiar el valor de una variable es una tarea sencilla. Un ejemplo práctico y básico sería:

### 3 VARIABLES Y TIPOS DE DATOS

---

```
x = 5  
print(x)  
  
x = 10  
print(x)
```

El resultado de este código sería 5 y 10.

Reto:

Crea una variable llamada “y” con un valor inicial de 10. Luego, cambia el valor de “y” a 15 y muestra el resultado.

### 3.8 Asignar multiples variables

En Python, es posible asignar múltiples variables a la vez. Esto se hace mediante la asignación múltiple, que es una forma de asignar un valor a varias variables al mismo tiempo.

Ejemplo:

```
x, y, z = 10, 20, 30  
print(x, y, z) # Imprime 10 20 30
```

Reto:

### 3 VARIABLES Y TIPOS DE DATOS

---

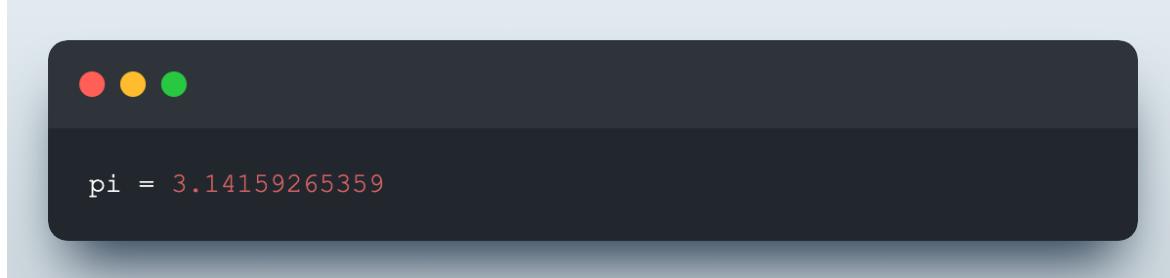
Asigna los valores 10, 20 y 30 a las variables a, b y c, respectivamente, y luego imprimelos en una sola línea.

#### 3.9 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. Python no tiene constantes como parte de su lenguaje pero es una convención declararlas como variables usando mayúsculas e indicar que el valor no debería ser cambiando, es decir, debería mantenerse constante.

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.



Reto:

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

### 3 VARIABLES Y TIPOS DE DATOS

---

#### 3.10 Tipos de datos

##### 3.10.1 Números (int, float, complex)

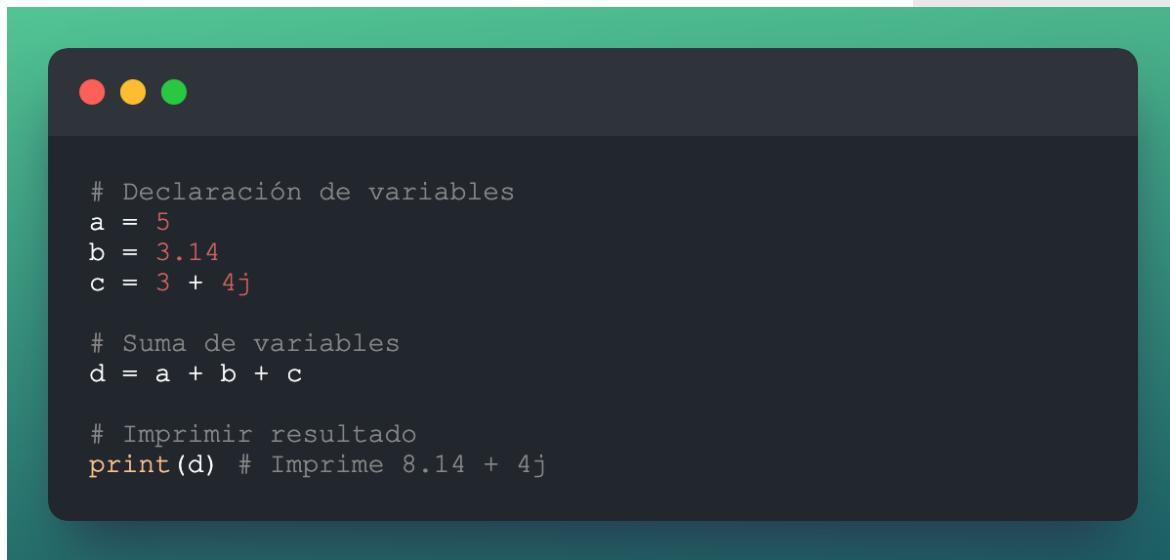
Los números en Python se dividen en tres tipos principales: enteros (int), flotantes (float) y complejos (complex).

Los enteros son números enteros, sin decimales, como 1, 2, 3, etc.

Los flotantes son números con decimales, como 1.5, 2.7, 3.14, etc.

Los complejos son números con partes reales e imaginarias, como  $3 + 4j$ .

Ejemplo:



```
# Declaración de variables
a = 5
b = 3.14
c = 3 + 4j

# Suma de variables
d = a + b + c

# Imprimir resultado
print(d) # Imprime 8.14 + 4j
```

Reto:

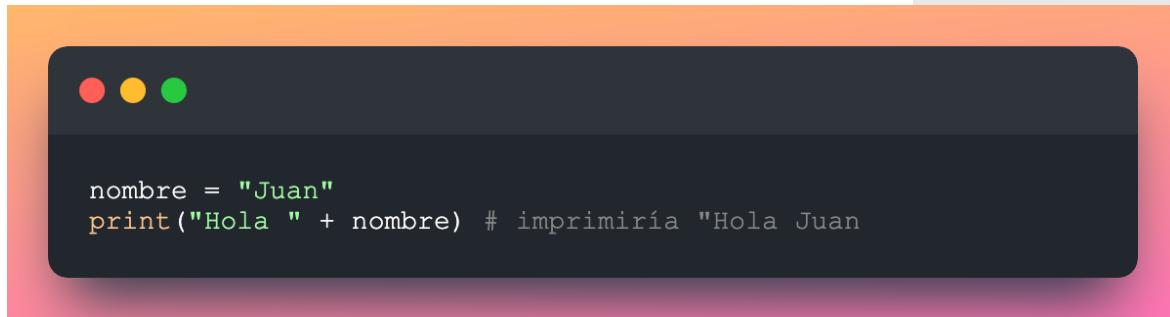
Escribe un programa que imprima la resta de 5 y 3.12.

### 3 VARIABLES Y TIPOS DE DATOS

---

#### 3.10.2 Texto (str)

Texto (str) en Python es un tipo de dato que se utiliza para representar cadenas de caracteres. Un ejemplo práctico y básico sería:



Reto:

Crea un programa que declare el nombre y edad de un usuario como “Ricardo” y 32 respectivamente, luego imprima una frase que diga “Hola [nombre], tienes [edad] años” .

#### 3.10.3 Booleanos (bool)

Los booleanos (bool) son un tipo de dato en Python que representan valores lógicos, es decir, verdadero (True) o falso (False). Estos son muy útiles para realizar comparaciones y tomar decisiones en un programa.

Ejemplo:

### 3 VARIABLES Y TIPOS DE DATOS

---

```
a = True  
b = False  
  
x = 5  
y = 10  
  
resultado = x > y  
print(resultado)
```

En este ejemplo, el resultado de la comparación será `False`, ya que 5 no es mayor que 10.

Reto:

Escribe un programa que imprima la variable “`esMayor`” que indique el resultado de la operación `20 > 18`.

#### 3.10.4 Secuencias (list, tuple, range)

Las secuencias en Python son una forma de almacenar una colección de elementos. Estas secuencias incluyen listas, tuplas y rangos.

Las listas son una secuencia de elementos separados por comas y encerrados entre corchetes. Por ejemplo:

### 3 VARIABLES Y TIPOS DE DATOS

---

```
● ● ●  
mi_lista = [1, 2, 3, 4, 5]  
# Acceder al primer elemento de la lista usando el índice  
print(mi_lista[0]) # Imprime 1
```

Los índices en las listas (y otros tipos de secuencias) comienzan en cero, lo que significa que el primer elemento de una lista tiene un índice de cero (0), el segundo elemento tiene un índice de uno (1), y así sucesivamente.

Las tuplas son similares a las listas, pero los elementos están encerrados entre paréntesis. Por ejemplo:

```
● ● ●  
mi_tupla = (1, 2, 3, 4, 5)  
# Acceder al primer elemento de la tupla  
print(mi_tupla[0]) # Imprime 1
```

Las listas son mutables y se utilizan comúnmente para almacenar colecciones de elementos que pueden cambiar o crecer en tamaño, mientras que las tuplas son inmutables y se utilizan comúnmente para almacenar colecciones de elementos que no cambian o que tienen un tamaño fijo.

Los rangos son una secuencia de números enteros consecutivos. Por ejemplo:

### 3 VARIABLES Y TIPOS DE DATOS

---

```
mi_rango = range(1, 6)
```

Reto:

Crea una lista, una tupla y un rango con los números del 1 al 10.

#### 3.10.5 Diccionarios (dict)

Los diccionarios (dict) en Python son un tipo de dato que almacena pares clave-valor. Estos pares se pueden usar para almacenar información de manera organizada.

Ejemplo:

```
# Crear un diccionario
mi_diccionario = {
    "nombre": "Juan",
    "edad": 25,
    "ciudad": "Madrid"
}

# Acceder a un elemento del diccionario
print(mi_diccionario["nombre"]) # Imprime "Juan"
```

Reto:

### 3 VARIABLES Y TIPOS DE DATOS

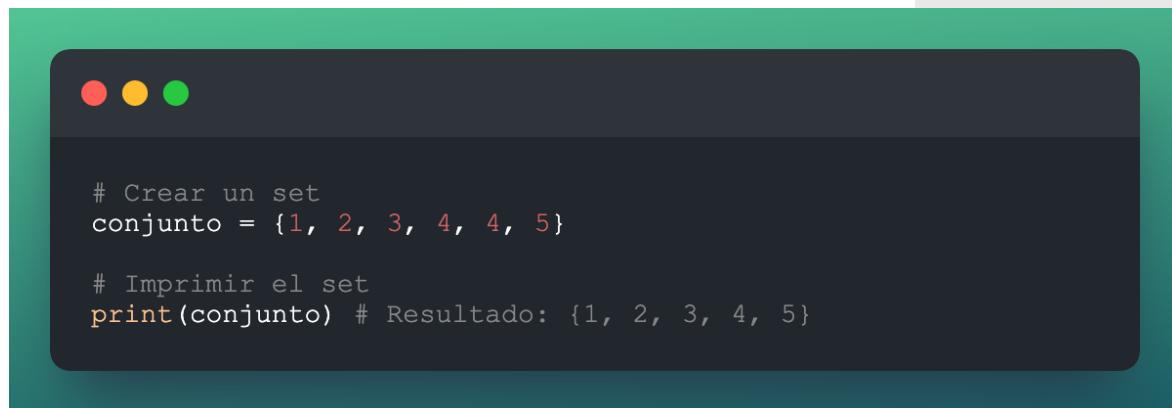
---

Crea un diccionario con los datos de una persona (nombre, edad, ciudad, país) y luego imprime los datos de esa persona.

#### 3.10.6 Conjuntos (set, frozenset)

Los conjuntos (set, frozenset) son un tipo de dato en Python que almacena elementos únicos, sin importar el orden en el que se ingresen. Esto significa que no hay elementos repetidos y que no hay índices.

Ejemplo:



```
# Crear un set
conjunto = {1, 2, 3, 4, 4, 5}

# Imprimir el set
print(conjunto) # Resultado: {1, 2, 3, 4, 5}
```

Los conjuntos (set) no tienen un orden específico, por lo que no podemos acceder a un elemento en un conjunto utilizando un índice como lo hacemos en una lista.

El tipo de conjunto llamado frozenset (conjunto congelado) es similar a un conjunto normal pero es inmutable, es decir, una vez creado, no se pueden agregar ni eliminar elementos. Los frozensets se definen utilizando la función “frozenset()” y se pueden utilizar en casos en los que necesitamos un conjunto inmutable.

Por ejemplo, podemos definir un frozenset de esta manera:

### 3 VARIABLES Y TIPOS DE DATOS

---



A screenshot of a Python code editor window. The title bar has three colored dots (red, yellow, green). The main area contains the following code:

```
mi_frozenset = frozenset([1, 2, 3, 4, 5])
```

Reto:

Crea un conjunto con los números del 1 al 10 y luego imprime el conjunto.

#### 3.10.7 Binarios (bytes, bytearray, memoryview)

Los binarios son una forma de almacenar datos en forma de bytes. En Python, hay tres formas principales de trabajar con binarios: bytes, bytearray y memoryview. Estos tipos de datos no son muy utilizados en programación web pero pueden ser útiles para usos avanzados como:

- Comunicación de datos en redes
- Procesamiento de datos binarios
- Interfaz con dispositivos de hardware
- Manipulación de datos científicos

**Bytes:** Los bytes son una secuencia inmutable de enteros no firmados de 8 bits. Esto significa que una vez creados, no se pueden modificar. Esto los hace útiles para almacenar datos binarios como imágenes, audio, etc.

Ejemplo:

### 3 VARIABLES Y TIPOS DE DATOS

---

```
# Crear una secuencia de bytes
mi_secuencia_de_bytes = b'Hola mundo'

# Imprimir la secuencia de bytes
print(mi_secuencia_de_bytes)

# Salida: b'Hola mundo'
```

**Bytarray:** Los bytarrays son como los bytes, pero son mutables. Esto significa que se pueden modificar después de su creación. Esto los hace útiles para realizar operaciones de bajo nivel en los datos binarios.

Ejemplo:

```
# Crear un bytarray
mi_bytarray = bytearray(b'Hello mundo')

# Modificar el bytarray
mi_bytarray[0] = ord('C')

# Imprimir el bytarray
print(mi_bytarray)

# Salida: bytearray(b'Cola mundo')
```

**Memoryview:** Los memoryviews son como los bytes y los bytarrays, pero son una vista de la memoria. Esto significa que no se crea una copia de los datos, sino que se crea una vista de los datos existentes. Esto los hace útiles para realizar operaciones

### 3 VARIABLES Y TIPOS DE DATOS

---

de bajo nivel en los datos binarios sin tener que crear una copia de los mismos.

Ejemplo:



```
# Crear un memoryview
mi_memoryview = memoryview(b'Hello mundo')

# Modificar el memoryview
mi_memoryview[0] = ord('C')

# Imprimir el memoryview
print(mi_memoryview)

# Salida: <memory at 0x7f9f9f9f9f9f>
```

Reto:

Crea una secuencia de bytes, un bytearray y un memoryview con los mismos datos y modifica uno de ellos. Luego, imprime los tres para ver si los cambios se han aplicado correctamente.

#### 3.10.8 NoneType

NoneType es un tipo de dato en Python que representa la ausencia de valor. Se usa para indicar que una variable no contiene ningún valor. Por ejemplo, si se declara una variable sin asignarle un valor o con el valor None, su tipo de dato será NoneType.

Ejemplo:

### 3 VARIABLES Y TIPOS DE DATOS

---

```
variable = None  
print(variable) # Imprime None que es de tipo NoneType
```

Reto:

Crea una variable llamada “mi\_variable” y asignale el valor None. Luego, imprime el tipo de dato de la variable.

#### 3.11 Formato de texto (f-Strings)

Los f-Strings son una forma de formatear cadenas de texto en Python. Esta forma de formateo es más simple y legible que la forma tradicional de formatear cadenas de texto. Esto se debe a que los f-Strings permiten la interpolación de variables directamente en la cadena de texto.

Ejemplo:

```
nombre = "John"  
edad = 20  
  
print(f"Hola, mi nombre es {nombre} y tengo {edad} años.")  
# Hola, mi nombre es John y tengo 20 años.
```

Reto:

### 3 VARIABLES Y TIPOS DE DATOS

---

Crea una variable llamada “pais” con el valor “Colombia” y usa un f-String para imprimir la frase “Vivo en Colombia” .

#### 3.12 Chequear tipo

Chequear tipo en Python es una forma de verificar el tipo de dato de una variable. Esto es útil para asegurarse de que los datos sean del tipo correcto para una operación específica. Por ejemplo, si se desea realizar una operación matemática con una variable, es importante asegurarse de que la variable sea un número antes de intentar realizar la operación.

Ejemplo:

```
# Definir una variable
variable = "Hola"

# Chequear el tipo de la variable
tipo_variable = type(variable)

# Imprimir el tipo de la variable
print(tipo_variable)

# Resultado
<class 'str'>
```

Reto:

Corre la función type() para cada tipo de dato que hemos visto.

### 3 VARIABLES Y TIPOS DE DATOS

---

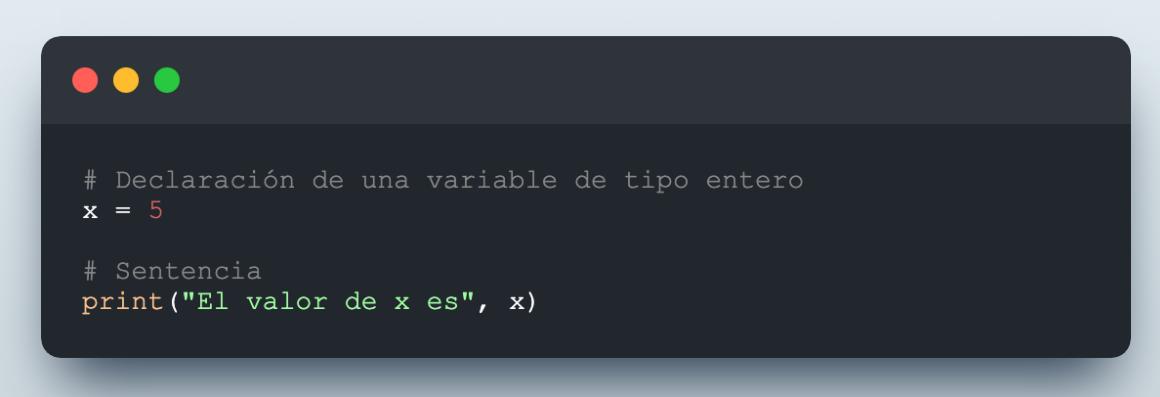
#### 3.13 Sentencias, declaraciones, y tipado

Una sentencia es una unidad de código que realiza una acción. Por ejemplo, la sentencia `print("Hola mundo")` imprime el texto “Hola mundo” en la pantalla.

Una declaración es una sentencia que asigna un valor a una variable. Por ejemplo, la declaración `x = 5` asigna el valor 5 a la variable x.

El tipado en Python es el proceso de asignar un tipo de datos a una variable. Por ejemplo, la declaración `x = 5` asigna el tipo de datos entero a la variable x. Python no es un lenguaje tipado por lo que el tipo es inferido implícitamente.

Ejemplo:



```
# Declaración de una variable de tipo entero
x = 5

# Sentencia
print("El valor de x es", x)
```

Reto:

Escribe un programa que declare una variable llamada num con el valor 10, imprima el valor de num en la pantalla y luego cambie el valor de num a 15.

#### 3.14 Conversión de tipos (casting)

El casting en Python es el proceso de convertir un tipo de dato a otro. Por ejemplo, convertir un entero a una cadena o una cadena a un entero.

Ejemplo:

## 4 OPERADORES

---

```
# Convertir un entero a una cadena  
  
numero = 10  
  
# Usamos la función str() para convertir número a texto  
cadena = str(numero)  
  
print(cadena) # Salida: "10"
```

Reto:

Escribe un programa que tome una cadena “432” como entrada y la convierta a un entero usando la función int().

## 4 Operadores

### 4.1 Operadores y expresiones

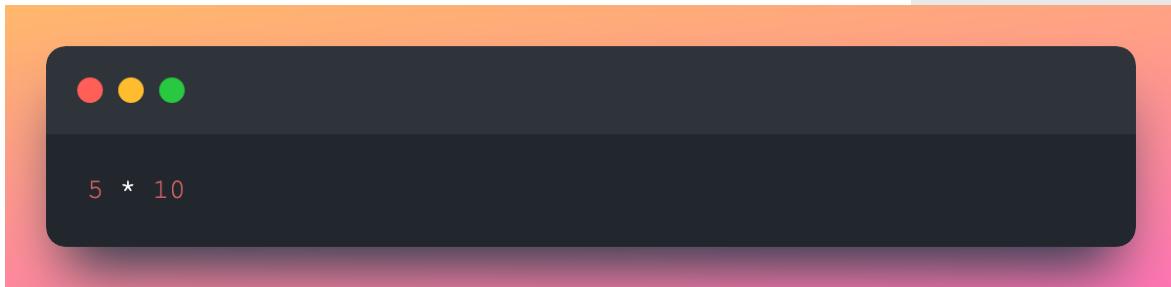
Los operadores y expresiones en Python son una parte importante de la programación. Los operadores son símbolos especiales que realizan operaciones matemáticas y lógicas sobre los valores. Las expresiones son combinaciones de operadores y valores que se evalúan para producir un resultado.

Ejemplo:

Supongamos que queremos calcular el área de un rectángulo con una base de 5 y una altura de 10. Podemos usar la siguiente expresión para calcular el área:

## 4 OPERADORES

---



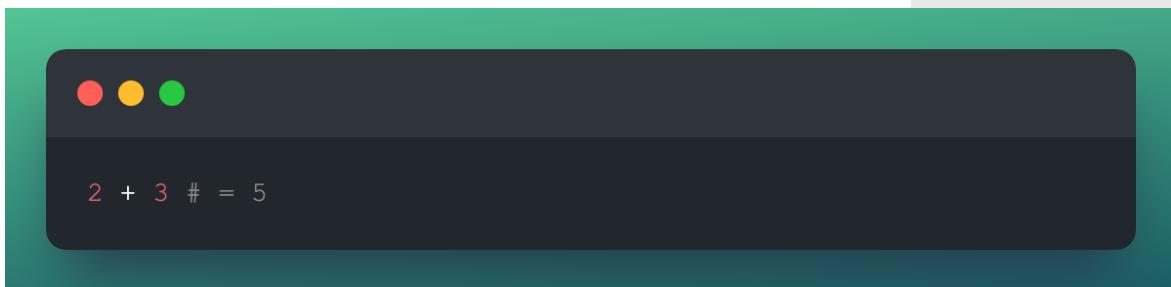
Reto:

Suma 3 y 8 y presiona enter para obtener 11 en la consola.

### 4.2 Operadores aritméticos (+, -, \*, /, %, \*\*, //)

Los operadores aritméticos son una parte importante de la programación en Python. Estos operadores se usan para realizar operaciones matemáticas básicas como sumar, restar, multiplicar, dividir y calcular el resto de una división. Estos operadores son:

- Suma (+): Este operador suma dos números. Por ejemplo:



- Resta (-): Este operador resta dos números. Por ejemplo:

## 4 OPERADORES

---



5 - 3 # = 2

- Multiplicación (\*): Este operador multiplica dos números. Por ejemplo:



2 \* 3 # = 6

- División (/): Este operador divide dos números. Por ejemplo:



6 / 3 # = 2

- Módulo (%): Este operador calcula la sobra de una división. Por ejemplo:

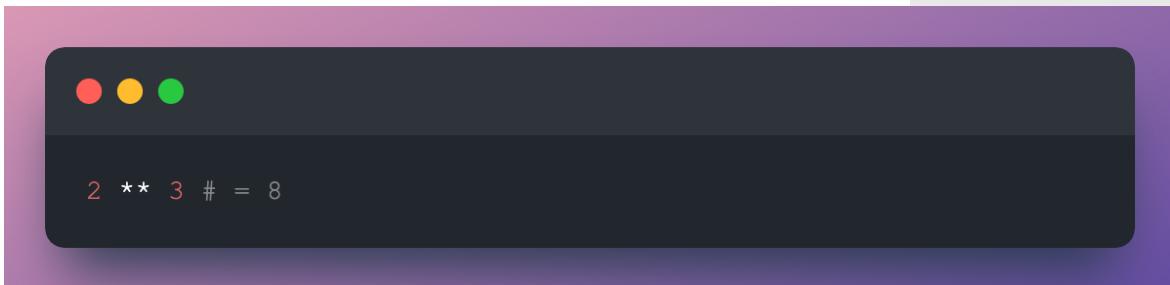


7 % 3 # = 1 (7 / 3 es 2 y la sobra es 1)

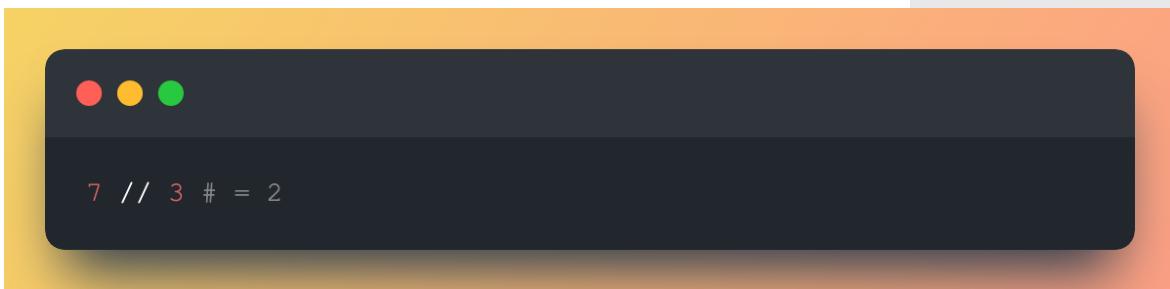
## 4 OPERADORES

---

- Exponenciación (\*\*): Este operador eleva un número a una potencia. Por ejemplo:



- División entera (/): Este operador divide dos números y devuelve el resultado como un número entero. Por ejemplo:



Reto:

Calcular el resultado de la siguiente operación  $2^{**} 3 * 5 + 7 // 3$ .

### 4.3 Operadores de asignación (=, +=, -=, \*=, /=, %=, \*\*=, //=)

Los operadores de asignación son una forma de asignar un valor a una variable. En Python, los operadores de asignación más comunes son:

- `=`: Asigna el valor a la derecha de la variable.

Ejemplo:

## 4 OPERADORES

---

```
x = 5 # x = 5
```

- **+=: Suma el valor a la derecha de la variable.**

Ejemplo:

```
x = 5  
x += 5 # x = 10
```

- **-=: Resta el valor a la derecha de la variable.**

Ejemplo:

```
x = 5  
x -= 5 # x = 0
```

- **\*=: Multiplica el valor a la derecha de la variable.**

Ejemplo:

## 4 OPERADORES

---

```
x = 5  
x *= 5 # x = 25
```

- /=: Divide el valor a la derecha de la variable.

Ejemplo:

```
x = 5  
x /= 5 # x = 1
```

- %=: Calcula el resto de la división del valor a la derecha de la variable.

Ejemplo:

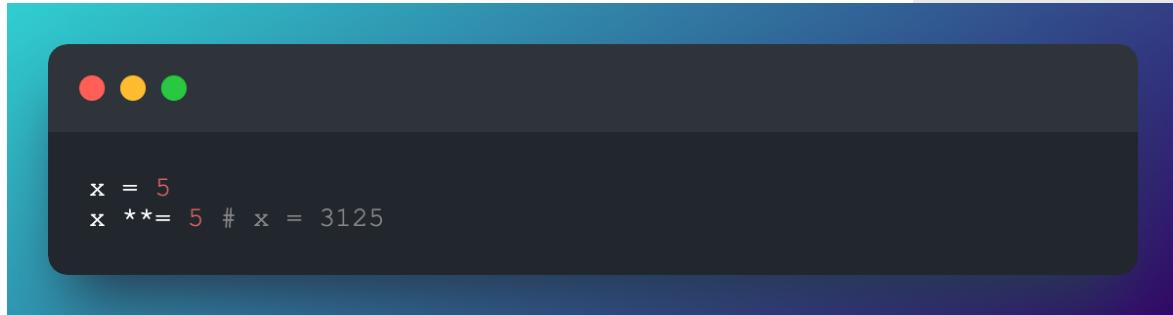
```
x = 5  
x %= 5 # x = 0
```

- \*\*=: Eleva el valor a una potencia la derecha de la variable.

Ejemplo:

## 4 OPERADORES

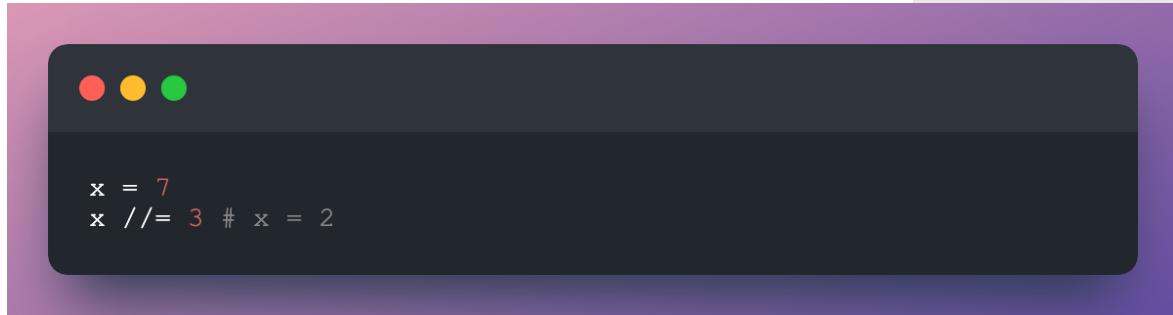
---



```
x = 5
x **= 5 # x = 3125
```

- //=: Calcula la división entera del valor a la derecha de la variable.

Ejemplo:



```
x = 7
x //= 3 # x = 2
```

Reto:

Crea un programa que asigne el valor 10 a la variable x, luego multiplique x por 5, luego reste 3 a x, luego divida x entre 2, luego calcule el resto de la división de x entre 4, luego eleve x a la potencia de 3, luego calcule la división entera de x entre 5 y finalmente imprima el resultado.

### 4.4 Operadores comparativos (==, !=, >, <, >=, <=)

Los operadores comparativos en Python son usados para comparar dos valores y determinar si son iguales, diferentes, mayores o menores entre sí. Estos operadores son:

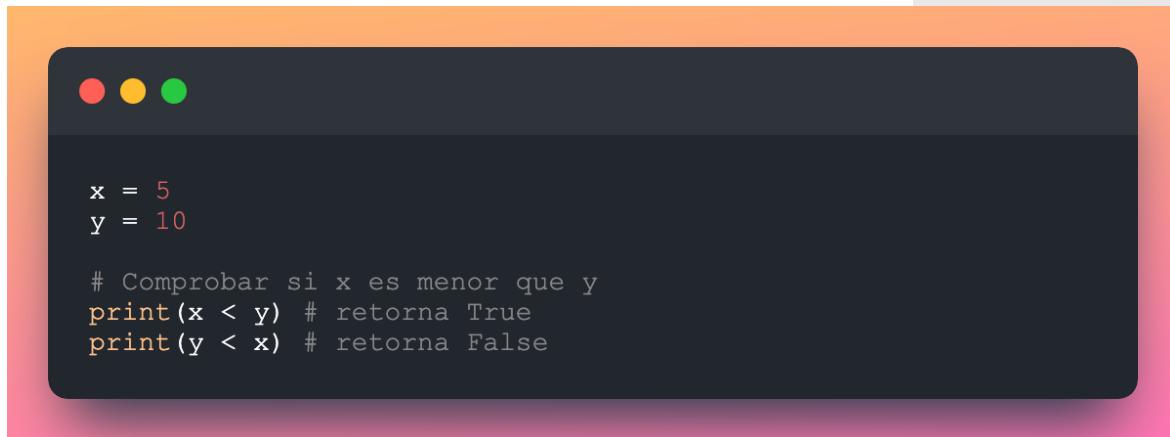
- == (igual a)

## 4 OPERADORES

---

- != (diferente de)
- > (mayor que)
- < (menor que)
- >= (mayor o igual que)
- <= (menor o igual que)

Ejemplo:



```
x = 5
y = 10

# Comprobar si x es menor que y
print(x < y) # retorna True
print(y < x) # retorna False
```

Reto:

Escribe un programa que determine si el primero de dos números enteros es mayor, menor o igual que el segundo.

### 4.5 Operadores lógicos (and, or, not)

El operador and se utiliza para evaluar dos condiciones y devuelve True si ambas son verdaderas. Por ejemplo:

## 4 OPERADORES

---

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

a = 5
b = 10

print(a > 0 and b > 0) # True
```

En este caso, la condición  $a > 0$  y la condición  $b > 0$  se evalúan y como ambas son verdaderas, True se imprime en la consola.

Por otro lado, el operador or se utiliza para evaluar dos condiciones y devuelve True si al menos una de ellas es verdadera. Por ejemplo:

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

a = 5
b = -10

print(a > 0 or b > 0) # True
```

En este caso, la condición  $a > 0$  es verdadera, pero la condición  $b > 0$  es falsa. Como al menos una de las condiciones es verdadera, True se imprime en la consola.

## 4 OPERADORES

---

Finalmente, el operador not se utiliza para negar una condición. Es decir, si la condición es verdadera, devuelve False, y si es falsa, devuelve True. Por ejemplo:

```
print(not True) # False
print(not False) # True

a = 5
b = -10

print(not (a > 0 and b > 0)) # True
```

En este caso, la condición  $a > 0$  es verdadera, pero la condición  $b > 0$  es falsa. Como estamos negando la condición completa, True se imprime en la consola.

Reto:

Escribe un pequeño programa que utilice los tres operadores lógicos en una sola expresión.

### 4.6 Operadores de membresía (in, not in)

Los operadores de membresía en Python son in y not in. Estos operadores se usan para verificar si un elemento se encuentra o no se encuentra dentro de una secuencia (lista, tupla, conjunto, etc) respectivamente.

Ejemplo:

## 4 OPERADORES

---

```
lista = [1, 2, 3, 4, 5]

print(3 in lista) # True
print(6 in lista) # False

print(3 not in lista) # False
print(6 not in lista) # True
```

Reto:

Crea una lista con los números del 1 al 10. Usa el operador `in` para verificar si el número 5 se encuentra en la lista.

### 4.7 Operadores de identidad (`is`, `is not`)

Los operadores de identidad en Python son `is` y `is not`. Estos operadores se usan para comparar si dos objetos son iguales o no son iguales en términos de identidad respectivamente. Esto significa que dos objetos son el mismo objeto si tienen el mismo valor de identidad.

Ejemplo:

## 4 OPERADORES

---

```
x = [1,2,3]
y = [1,2,3]

print(x is y) # False

z = x

print(x is z) # True
```

Reto:

Crea dos listas con los mismos elementos y usa los operadores de identidad para comprobar si son el mismo objeto.

### 4.8 Operadores de bits (&, |, ~, ^, », «)

Los operadores de bits son operadores binarios que trabajan con los bits de un número. Estos operadores se usan para realizar operaciones bit a bit en los números. En Python, los operadores de bits son &, |, ~, ^, » y «.

Estos operadores se utilizan en muchas tareas de programación, como la manipulación de bits, la cifrado de datos, etc.

En Python, los siguientes son los operadores binarios:

1. & (AND): Este operador binario compara dos números binarios bit a bit y si ambos bits son 1, entonces el bit correspondiente en el resultado también será 1. Si cualquiera de los bits es 0, el bit en el resultado será 0.

## 4 OPERADORES

---

```
● ● ●  
a = 0b10101010  
b = 0b11001100  
c = a & b  
print(bin(c)) # resultado: 0b10001000
```

2. | (OR): Este operador binario compara dos números binarios bit a bit y si alguno de los bits es 1, entonces el bit correspondiente en el resultado será 1. Si ambos bits son 0, el bit en el resultado será 0.

```
● ● ●  
a = 0b10101010  
b = 0b11001100  
c = a | b  
print(bin(c)) # resultado: 0b11101110
```

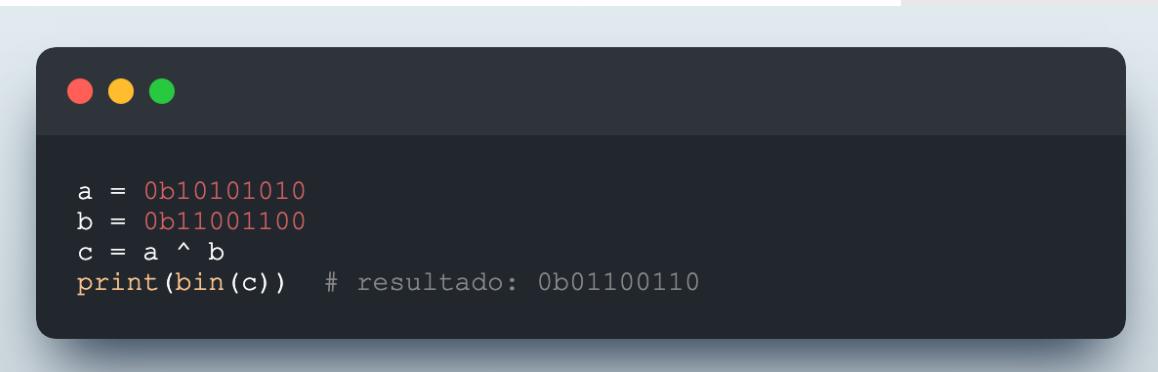
3. ~ (NOT): Este operador binario invierte todos los bits de un número binario. Por ejemplo, si el número es 100, el resultado será 011.

```
● ● ●  
a = 0b10101010  
b = ~a  
print(bin(b)) # resultado: -0b10101011
```

## 4 OPERADORES

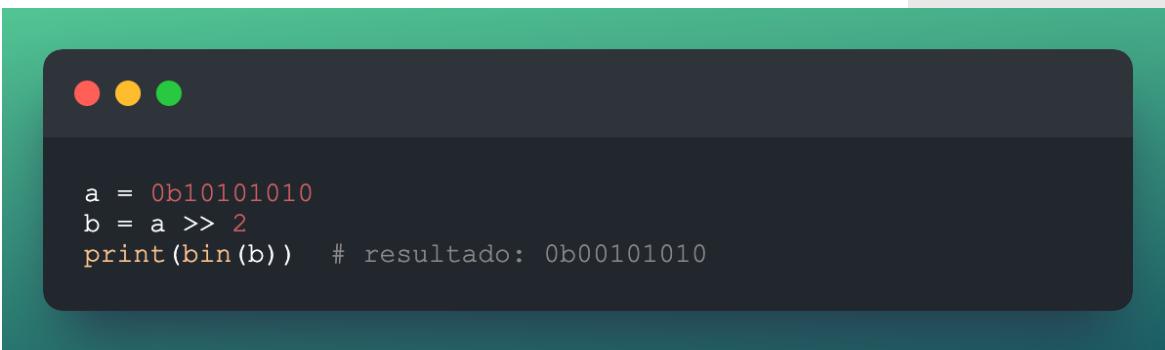
---

4. ^ (XOR): Este operador binario compara dos números binarios bit a bit y si los bits son diferentes, el bit correspondiente en el resultado será 1. Si los bits son iguales, el bit en el resultado será 0.



```
a = 0b10101010
b = 0b11001100
c = a ^ b
print(bin(c)) # resultado: 0b01100110
```

5. >> (SHIFT RIGHT): Este operador binario mueve los bits de un número binario hacia la derecha por una cantidad especificada de veces. Cada vez que se mueve hacia la derecha, se descarta el bit más a la derecha y se agrega un 0 a la izquierda.\*\*



```
a = 0b10101010
b = a >> 2
print(bin(b)) # resultado: 0b00101010
```

6. << (SHIFT LEFT): Este operador binario mueve los bits de un número binario hacia la izquierda por una cantidad especificada de veces. Cada vez que se mueve hacia la izquierda, se descarta el bit más a la izquierda y se agrega un 0 a la derecha.

## 4 OPERADORES

---

```
a = 0b10101010
b = a << 2
print(bin(b)) # resultado: 0b1010101000
```

Estos operadores también se pueden usar directamente con números. Por ejemplo:

Supongamos que tenemos dos números binarios: 1101 y 1010 que son 13 y 10 en sistema decimal.

Usando el operador `&`, podemos realizar una operación bit a bit entre estos dos números. El resultado sería 1000 que es 8 en sistema decimal. Esto se debe a que el operador `&` devuelve 1 si ambos bits son 1, de lo contrario devuelve 0.

```
print(13 & 10) # 8
```

Aunque no son tan comunes en programación web como en otros ámbitos de la programación, los operadores binarios pueden ser muy útiles en ciertos casos, como la manipulación de datos binarios (como imágenes o archivos binarios) y la criptografía (como la codificación y decodificación de datos en formatos binarios como Base64).

Reto:

Usando los operadores de bits `&`, `|`, `~`, `^`, `»` y `«`, realiza una operación bit a bit entre

## 5 ESTRUCTURAS DE CONTROL DE FLUJO

---

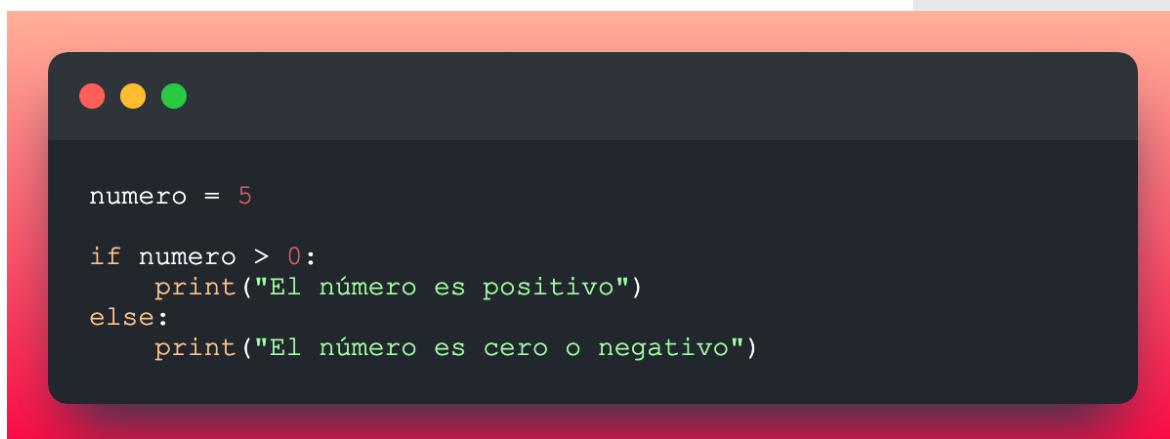
los números binarios 1101 y 1011.

# 5 Estructuras de control de flujo

## 5.1 Estructuras de control de flujo y excepciones

Las estructuras de control de flujo y excepciones en Python son herramientas fundamentales para controlar el flujo de un programa. Estas estructuras permiten a los programadores controlar el flujo de un programa, manejar errores y excepciones, y realizar acciones condicionales.

Ejemplo:



```
numero = 5

if numero > 0:
    print("El número es positivo")
else:
    print("El número es cero o negativo")
```

Este es un condicional. Nota que al correr este código solo una parte de este código corre. Hemos logrado controlar el flujo de nuestro código.

Reto:

Escribe el programa del ejemplo con el número 0 para ver un resultado distinto.

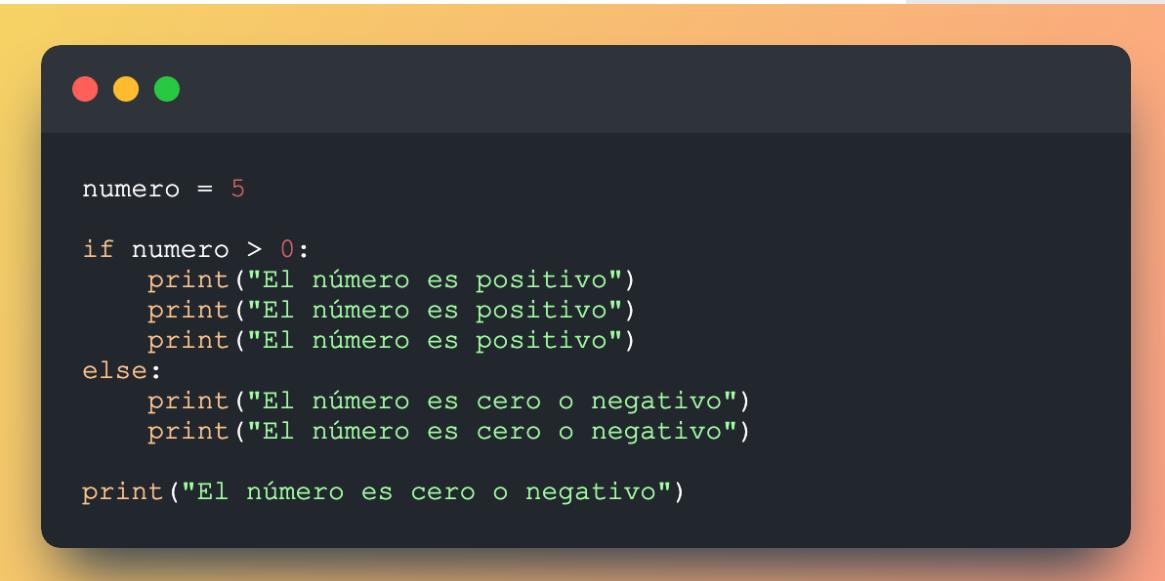
## 5 ESTRUCTURAS DE CONTROL DE FLUJO

---

### 5.2 Bloques e indentación

Los bloques y la indentación son fundamentales en Python para estructurar el código. Los bloques se definen por la indentación, que es el espacio en blanco al principio de una línea de código. Esto significa que todas las líneas de código que estén indentadas al mismo nivel pertenecen al mismo bloque.

Ejemplo:



```
numero = 5

if numero > 0:
    print("El número es positivo")
    print("El número es positivo")
    print("El número es positivo")
else:
    print("El número es cero o negativo")
    print("El número es cero o negativo")

print("El número es cero o negativo")
```

The screenshot shows a terminal window with a dark background and light-colored text. At the top, there are three colored window control buttons (red, yellow, green). The terminal displays the following Python code and its execution:

```
numero = 5

if numero > 0:
    print("El número es positivo")
    print("El número es positivo")
    print("El número es positivo")
else:
    print("El número es cero o negativo")
    print("El número es cero o negativo")

print("El número es cero o negativo")
```

The output of the code is:  
El número es positivo  
El número es positivo  
El número es positivo  
El número es cero o negativo  
El número es cero o negativo  
El número es cero o negativo

El código verifica si el valor de la variable “numero” es mayor que cero. Si es mayor que cero, imprime tres veces “El número es positivo”. Si no es mayor que cero (es cero o negativo), imprime dos veces “El número es cero o negativo”. Después de las comprobaciones, siempre imprime “El número es cero o negativo” sin importar la condición.

Reto:

Modifica el ejemplo removiendo la indención del segundo “print” para que veas que el resultado cambia. Este ejemplo multilínea es más fácil ejecutarlo como archivo con la extensión “.py” en lugar de usar la consola.

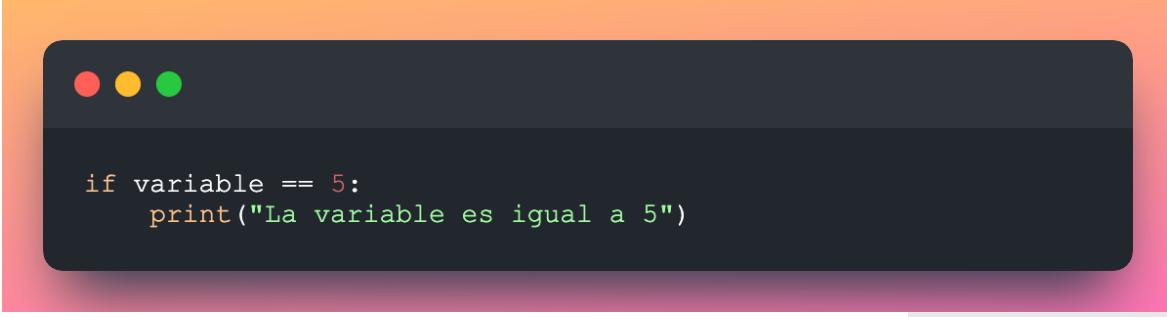
## 5 ESTRUCTURAS DE CONTROL DE FLUJO

---

### 5.3 Condicionales

#### 5.3.1 if

If en Python es una estructura de control de flujo que permite a los programadores ejecutar una acción específica si una condición es verdadera. Por ejemplo, si queremos imprimir un mensaje si una variable es igual a 5, podemos usar el siguiente código:



```
if variable == 5:  
    print("La variable es igual a 5")
```

Reto:

Escribe un programa revise el valor de una variable edad y luego imprima un mensaje diferente dependiendo de si la edad es menor o mayor que 18.

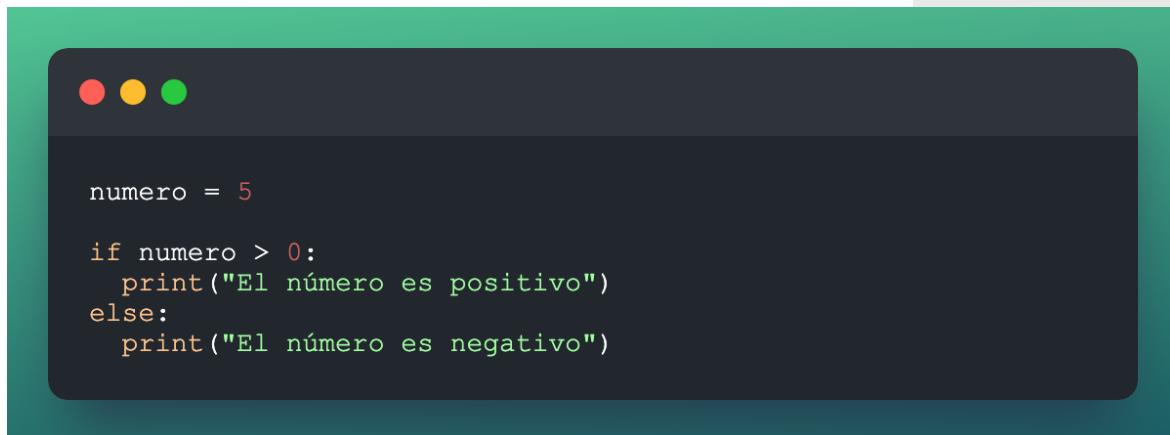
#### 5.3.2 if...else

If...else es una estructura de control de flujo en Python que permite a los programadores ejecutar una sección de código si una condición es verdadera, y otra sección de código si la condición es falsa.

Ejemplo:

## 5 ESTRUCTURAS DE CONTROL DE FLUJO

---



```
numero = 5

if numero > 0:
    print("El número es positivo")
else:
    print("El número es negativo")
```

Reto:

Escribe un programa revise un numero y luego imprima si el número es par o impar.

### 5.3.3 if…elif…else

If…elif…else es una estructura de control de flujo en Python que permite a los programadores tomar decisiones basadas en una o más condiciones. Esta estructura se compone de tres partes: if, elif y else.

Ejemplo:

## 5 ESTRUCTURAS DE CONTROL DE FLUJO

---

```
# input() pregunta al usuario y retorna texto
respuesta = input("Ingresa un número entero: ")

# int() convierte el texto en un numero
numero = int(respuesta)

if numero > 0:
    print("El número es positivo")
elif numero < 0:
    print("El número es negativo")
else:
    print("El número es cero")
```

En este caso se pueden incluir más bloques elif si se desean añadir más condiciones.

Reto:

Escribe un programa que pregunte al usuario por un número entero y luego imprima si el número es par o impar.

### 5.3.4 Operador ternario

El operador ternario (?) en Python es una forma abreviada de escribir una condición if-else. Esta sintaxis es útil para escribir código más conciso y legible. El operador ternario se compone de tres partes: una condición, una expresión si la condición es verdadera y una expresión si la condición es falsa.

Ejemplo:

## 5 ESTRUCTURAS DE CONTROL DE FLUJO

---

```
edad = 20
mensaje = "Eres mayor de edad" if edad >= 18 else "Eres menor
de edad"
print(mensaje)
```

Reto:

¿Puedes usar el operador ternario para imprimir un mensaje diferente dependiendo de si un número es par o impar?

### 5.3.5 Condicionales anidados

Los condicionales anidados en Python son una forma de anidar varios condicionales para evaluar una expresión. Esto significa que se pueden usar varios condicionales dentro uno de otro.

Ejemplo:

```
if x > 0:
    if x < 10:
        print("x es un número positivo entre 0 y 10")
    else:
        print("x es un número positivo mayor que 10")
else:
    print("x es un número negativo")
```

## 5 ESTRUCTURAS DE CONTROL DE FLUJO

---

A pesar de que los condicionales anidados pueden ser necesarios en varias ocasiones, es una buena práctica evitar código anidado ya que reduce la facilidad de lectura y puede introducir situaciones difíciles de depurar. Una mejor alternativa son las “claúsulas de guarda” que es una práctica avanzada.

Reto:

Escribe un programa que evalúe si un número es positivo, negativo o cero. Si el número es positivo, imprime si es par o impar. Si el número es negativo, imprime si es divisible por 3. Si el número es cero, imprime “el número es cero” .

### 5.3.6 match

Match, también conocido como switch, es una estructura de control de flujo en Python que permite seleccionar una de varias opciones de acción basadas en una variable. Esto se logra mediante la creación de una serie de casos, cada uno con una condición diferente. Si la condición se cumple, se ejecuta el código asociado a ese caso. Esta funcionalidad está disponible en Python desde la versión 3.10

Ejemplo:

## 5 ESTRUCTURAS DE CONTROL DE FLUJO

---

```
lenguaje = input("Qué lenguaje deseas aprender? ")

match lenguaje:
    case "JavaScript":
        print("Te puedes convertir en desarrollador web.")
    case "Python":
        print("Te puedes convertir en científico de datos")
    case "PHP":
        print("Te puedes convertir en desarrollador backend")
    case "Solidity":
        print("Te puedes convertir en desarrollador de
Blockchain")
    case "Java":
        print("Te puedes convertir en desarrollador móvil")
    case _:
        print("El lenguaje no importa. Tú puedes hacerlo!")
```

Cuando tengas comparaciones simples como el ejemplo usando if...elif...else, considera cambialo a match.

Reto:

Utiliza switch para seleccionar la acción a realizar al viajar a varios destinos.

## 5.4 Bucles

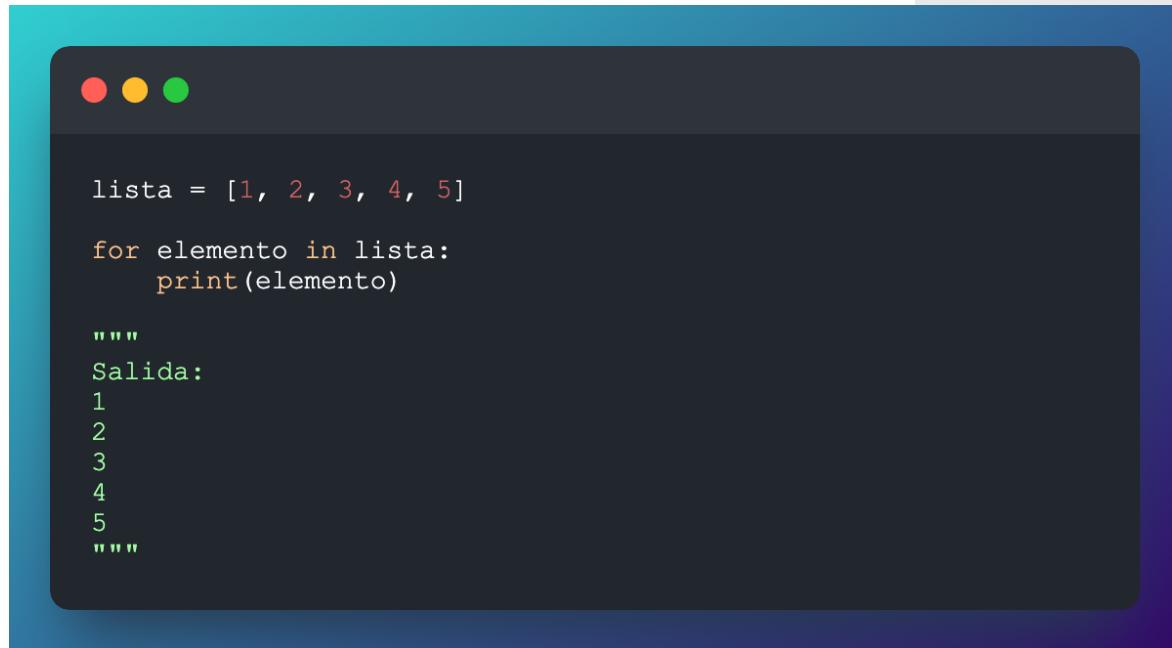
### 5.4.1 for

For en Python es una estructura de control de flujo que se utiliza para iterar sobre una secuencia (por ejemplo, una lista, una tupla o una cadena de caracteres). El bucle for itera sobre los elementos de la secuencia en el orden en que aparecen y ejecuta un bloque de código para cada elemento.

## 5 ESTRUCTURAS DE CONTROL DE FLUJO

---

Ejemplo:



The screenshot shows a terminal window with a dark background and three colored icons (red, yellow, green) at the top. The code is displayed in white text:

```
lista = [1, 2, 3, 4, 5]

for elemento in lista:
    print(elemento)

"""
Salida:
1
2
3
4
5
"""
```

The output of the program is shown below the code, also in white text:

```
1
2
3
4
5
```

Reto:

Escribe un programa que imprima los números del 1 al 10 utilizando un bucle for.

### 5.4.2 continue

Continue en Python es una palabra clave que se usa para saltar a la siguiente iteración de un bucle. Esto significa que cuando se encuentra una instrucción continue, el control se traslada inmediatamente a la siguiente iteración del bucle.

Ejemplo:

## 5 ESTRUCTURAS DE CONTROL DE FLUJO

---

```
● ● ●  
for i in range(10):  
    if i == 5:  
        continue  
    print(i)
```

En este ejemplo, el bucle se ejecuta 10 veces, pero cuando el contador de bucle alcanza el valor 5, la instrucción continue se ejecuta y el control se traslada a la siguiente iteración del bucle. Esto significa que el número 5 no se imprimirá.

Reto:

Escribe un programa que imprima los números del 1 al 10, excepto el 8.

### 5.4.3 break

Break es una palabra clave en Python que se usa para salir de un bucle. Un ejemplo práctico y básico sería el siguiente:

```
● ● ●  
for i in range(10): # rango de 0 a 10  
    if i == 5:  
        break  
    print(i)
```

En este ejemplo, el bucle se ejecutará desde 0 hasta 4, ya que cuando el contador llegue a 5, el bucle se romperá.

## 5 ESTRUCTURAS DE CONTROL DE FLUJO

---

Reto:

Escribe un programa que tome una lista de artistas y salga del bucle cuando encuentre tu artista favorito.

### 5.4.4 pass

Pass es una palabra clave en Python que se usa como una instrucción nula. Significa que el programa no hará nada cuando se encuentre con esta palabra clave.

Ejemplo:

```
for i in range(10):
    if i == 5:
        pass
    else:
        print(i)
```

En este ejemplo, el ciclo for se ejecutará 10 veces. Cuando el contador de bucle sea igual a 5, el programa pasará por la instrucción pass, lo que significa que no hará nada. En todos los demás casos, el programa imprimirá el valor de i.

Reto:

Escribe un programa que imprima los números del 1 al 10, excepto el 3. Utiliza la palabra clave pass para omitir el número 3.

## 5 ESTRUCTURAS DE CONTROL DE FLUJO

---

### 5.4.5 for...else

For...else en Python es una estructura de control de flujo que se usa para iterar sobre una secuencia (como una lista, tupla o cadena de caracteres) y ejecutar un bloque de código para cada elemento. Si el ciclo se completa sin encontrar un elemento que cumpla con una condición, el bloque de código en el else se ejecutará.

Ejemplo:

```
numeros = [1, 2, 3, 4, 5]

for numero in numeros:
    if numero % 2 == 0:
        # texto dinámico usando f-Strings
        print(f"El número {numero} es par")
        break
    else:
        print("No se encontró ningún número par")
```

Reto:

Escribe un programa que itere sobre una lista de números y determine si hay algún número divisible por 7. Si hay al menos un número divisible por 7, imprime el número. Si no hay ningún número divisible por 7, imprime un mensaje indicando que no hay ningún número divisible por 7.

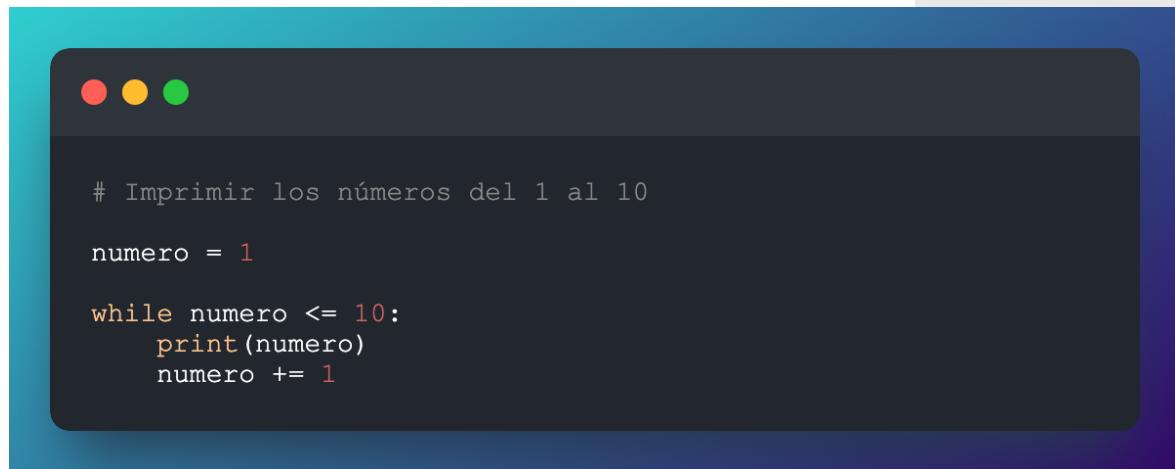
### 5.4.6 while

While es una estructura de control de flujo en Python que se usa para ejecutar un bloque de código mientras se cumpla una condición.

## 5 ESTRUCTURAS DE CONTROL DE FLUJO

---

Ejemplo:



```
# Imprimir los números del 1 al 10

numero = 1

while numero <= 10:
    print(numero)
    numero += 1
```

Ten cuidado al ejecutar este código con una condición que nunca pare de cumplirse ya que el código dentro del bloque correrá para siempre. Si esto sucede, el programa agotará los recursos de tu computador congelándolo o posiblemente requerirá terminar el programa de manera forzada.

Reto:

Imprimir los números del 10 al 1.

### 5.4.7 while...else

While...else en Python es una estructura de control de flujo que se ejecuta mientras una condición es verdadera. Si la condición se vuelve falsa, el bloque else se ejecuta.

Ejemplo:

## 5 ESTRUCTURAS DE CONTROL DE FLUJO

---

```
numero = 0

while numero < 10:
    print(numero)
    numero += 1
else:
    print("Número es igual o mayor a 10")
```

Reto:

Escribe un programa que imprima los números del 1 al 10. Si el número es divisible por 3, imprime “Fizz” en lugar del número. Si el número es divisible por 5, imprime “Buzz” . Si el número es divisible por 3 y 5, imprime “FizzBuzz” .

### 5.5 Excepciones

#### 5.5.1 Tipos de excepciones

Excepciones en Python son errores que se producen durante la ejecución de un programa. Estas excepciones pueden ser manejadas para evitar que el programa se detenga abruptamente. Existen varios tipos de excepciones en Python, entre ellas:

- Excepciones de sintaxis: Estas excepciones se producen cuando el código no sigue la sintaxis de Python. Por ejemplo, si se olvida un paréntesis en una línea de código, se generará una excepción de sintaxis.

## 5 ESTRUCTURAS DE CONTROL DE FLUJO

---

```
3 + (5 + 4 # SyntaxError: '(' was never closed
```

- Excepciones de valor: Estas excepciones se producen cuando un valor no es válido para una operación. Por ejemplo, si se intenta dividir un número entre cero, se generará una excepción de valor.

```
10 / 0 # ZeroDivisionError: division by zero
```

- Excepciones de nombre: Estas excepciones se producen cuando un nombre no es válido para una operación. Por ejemplo, si se intenta usar una variable que no ha sido definida, se generará una excepción de nombre.

```
print(variable) # NameError: name 'variable' is not defined.
```

## 5 ESTRUCTURAS DE CONTROL DE FLUJO

---

### 5.5.2 raise

“Raise” significa “elevar” en inglés, y es una forma de elevar una excepción en tu código. ¿Qué es una excepción? Imagina que estás caminando por el pasillo de la universidad y de repente tropiezas con una silla. ¡Eso es una excepción! En programación, una excepción es un error o un comportamiento inesperado que puede ocurrir durante la ejecución de un programa.

Ahora, cuando utilizamos raise en Python, estamos creando de forma intencional una excepción. Imagina que estás programando un juego y quieras asegurarte de que un usuario no pueda seleccionar una opción inválida. Puedes utilizar raise para elevar una excepción si el usuario ingresa una opción incorrecta.

Aquí tienes un ejemplo práctico:

```
def elegir_opcion(opcion):
    if opcion not in ["A", "B", "C"]:
        raise ValueError("Opción inválida!")

    print("¡Excelente elección!")

opcion_usuario = input("Ingresa tu opción (A, B o C): ")

try:
    elegir_opcion(opcion_usuario)
except ValueError as error:
    print(error)
```

En este ejemplo, la función elegir\_opcion verifica si la opción ingresada por el usuario está en la lista ["A", "B", "C"]. Si no lo está, utilizamos raise para elevar una excepción del tipo ValueError. Luego, en el bloque try-except, capturamos esa excepción y la imprimimos en pantalla.

## 5 ESTRUCTURAS DE CONTROL DE FLUJO

---

¿Listo para el reto? Aquí está:

Reto:

Escribe una función llamada dividir que tome dos números como argumentos. Si el segundo número es igual a cero, utiliza raise para elevar una excepción del tipo ZeroDivisionError. Si el segundo número es distinto de cero, la función debe devolver el resultado de la división.

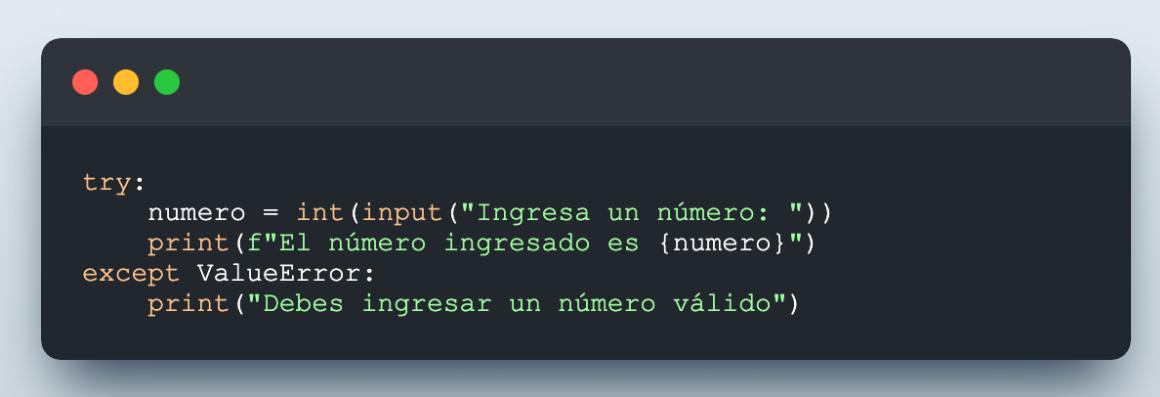
¡Buena suerte y diviértete programando!

### 5.5.3 try...except

Try en Python es una estructura de control de flujo que permite ejecutar un bloque de código y controlar errores que puedan ocurrir durante su ejecución. El bloque de código se encierra entre las palabras clave try y except.

Except en Python es una sentencia usada para manejar errores y excepciones. Esta sentencia permite que el programa siga ejecutándose a pesar de que se haya producido un error.

Ejemplos:



```
try:
    numero = int(input("Ingresa un número: "))
    print(f"El número ingresado es {numero}")
except ValueError:
    print("Debes ingresar un número válido")
```

## 5 ESTRUCTURAS DE CONTROL DE FLUJO

---

```
try:  
    x = 5 / 0  
except ZeroDivisionError:  
    print("No se puede dividir un número entre cero")
```

Reto:

Escribe un programa que solicite al usuario ingresar dos números y luego imprima el resultado de la división de los dos números. Si el usuario ingresa un valor no numérico, el programa debe mostrar un mensaje de error.

### 5.5.4 try···except···finally

Finally en Python es una palabra clave que se usa para indicar un bloque de código que se ejecutará siempre, sin importar si hay una excepción o no. Esto significa que el código dentro de un bloque finally se ejecutará siempre, incluso si hay una excepción.

Ejemplo:

## 6 FUNCIONES

---

```
try:  
    # Código que puede generar una excepción  
except:  
    # Código para manejar la excepción  
finally:  
    # Código que se ejecutará siempre
```

Reto:

Escribe un programa que solicite al usuario un número entero y luego imprima el resultado de dividir ese número entre 2. Utiliza un bloque finally para imprimir un mensaje de despedida al usuario.

## 6 Funciones

### 6.1 Declarar y llamar funciones

Las funciones en Python son bloques de código que realizan una tarea específica. Se declaran con la palabra clave def seguida del nombre de la función y los parámetros entre paréntesis.

Ejemplo:

## 6 FUNCIONES

---

```
def saludar():
    print("Hola!")

saludar()
```

Reto:

Crea una función que imprima la suma de 3 y 4.

### 6.2 Parámetros y argumentos

Los parámetros y argumentos son una parte importante de la programación en Python. Los parámetros son variables que se pasan a una función, mientras que los argumentos son los valores que se pasan a los parámetros.

Ejemplo:

```
def saludar(nombre):
    print("Hola, " + nombre + "!")

saludar("Juan")
```

En este ejemplo, la función saludar tiene un parámetro llamado nombre. Cuando se llama a la función saludar, se pasa el argumento “Juan” al parámetro nombre.

## 6 FUNCIONES

---

Reto:

Escribe una función que tome dos parámetros (num1 y num2) y devuelva la suma de los dos números. Luego, llama a la función con dos argumentos diferentes y muestra el resultado.

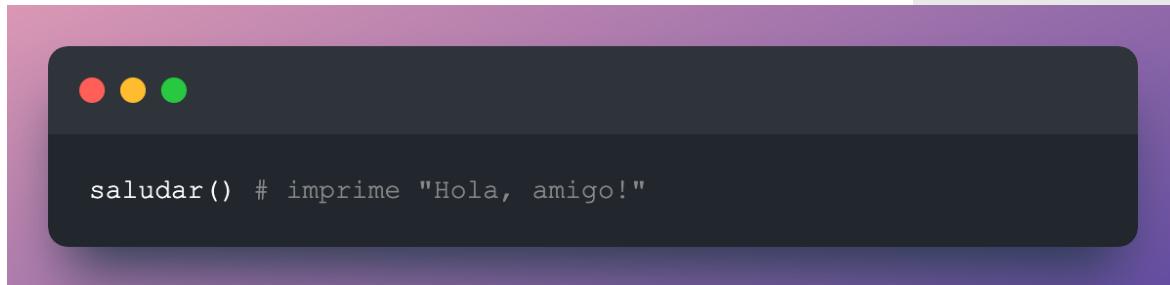
### 6.3 Argumentos con valores predeterminados

Los argumentos con valores predeterminados en Python son aquellos que se asignan a una función cuando se llama a la misma sin especificar un valor para el argumento. Esto significa que si una función tiene un argumento con un valor predeterminado, el usuario no necesita especificar un valor para ese argumento cuando llama a la función. Por ejemplo, considera la siguiente función:



```
def saludar(nombre="amigo"):
    print("Hola, " + nombre + "!")
```

En este ejemplo, el argumento nombre tiene un valor predeterminado de “amigo”. Esto significa que si el usuario llama a la función saludar sin especificar un valor para el argumento nombre, el valor predeterminado se usará. Por ejemplo:



```
saludar() # imprime "Hola, amigo!"
```

## 6 FUNCIONES

---

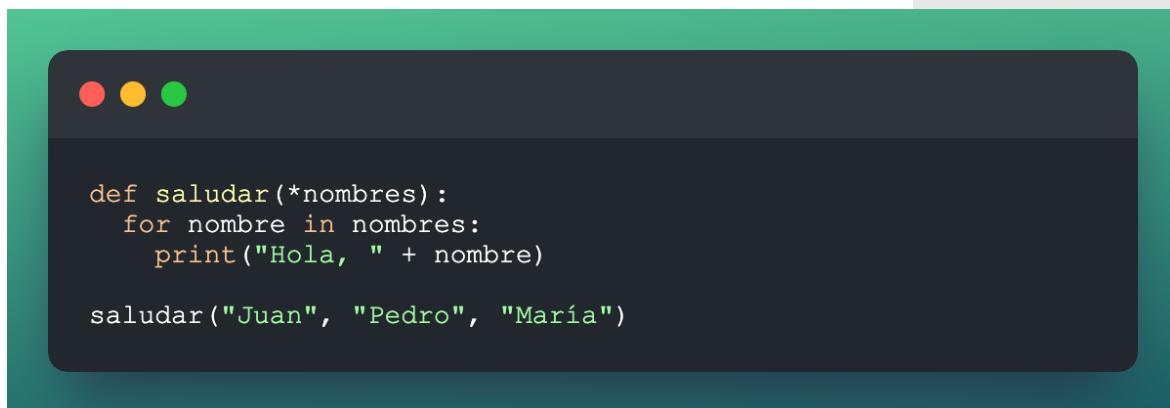
Reto:

Escribe una función que tome dos argumentos, un nombre y una edad, y que imprima un saludo con el nombre y la edad. El argumento de edad debe tener un valor predeterminado de 18.

### 6.4 Argumentos arbitrarios

Los argumentos arbitrarios son argumentos que se pasan a una función, pero que no se especifican en la definición de la misma. Estos argumentos se pasan como una tupla de argumentos con el nombre de la variable precedido por un asterisco (\*).

Ejemplo:



```
def saludar(*nombres):
    for nombre in nombres:
        print("Hola, " + nombre)

saludar("Juan", "Pedro", "María")
```

Reto:

Escribe una función que tome un número arbitrario de argumentos y devuelva la suma de todos ellos.

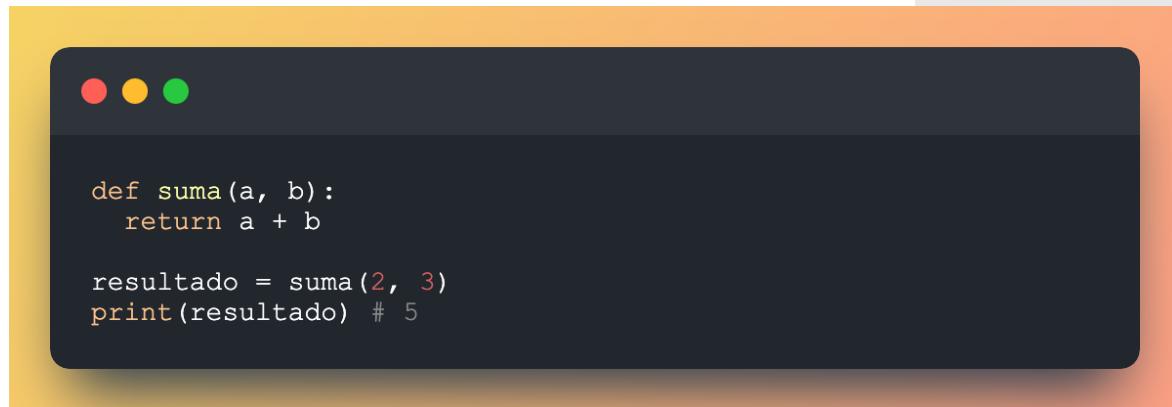
## 6 FUNCIONES

---

### 6.5 Retorno de valores

En Python, el retorno de valores es una forma de devolver un resultado de una función. Esto significa que una función puede devolver un valor que puede ser usado por el programa que la llamó.

Ejemplo:



```
def suma(a, b):
    return a + b

resultado = suma(2, 3)
print(resultado) # 5
```

Reto:

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

### 6.6 Ámbito global y local

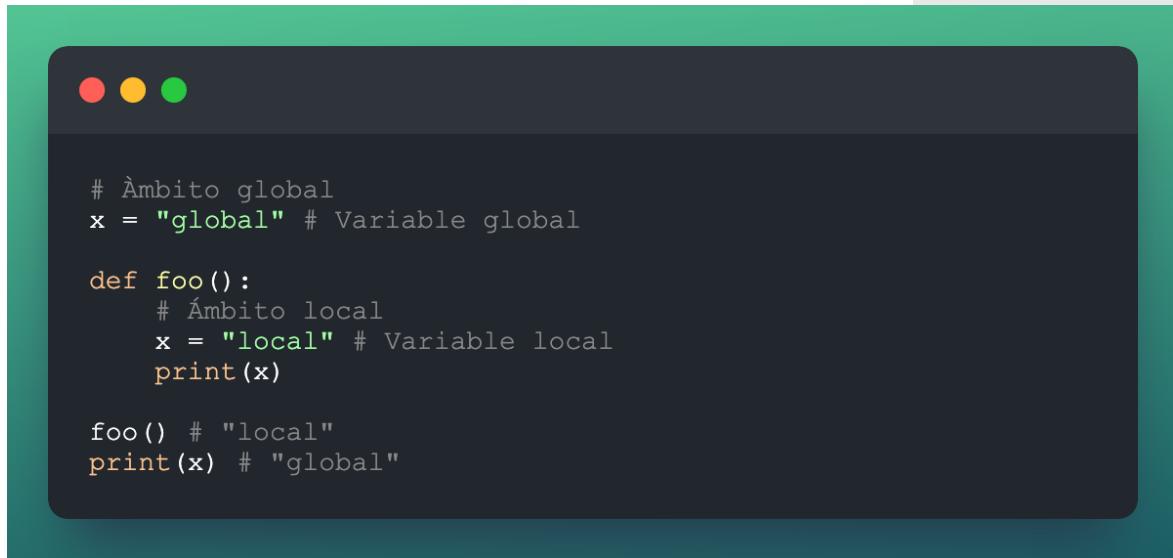
En Python, el ámbito global y local se refiere a la visibilidad de variables y funciones. El ámbito global se refiere a variables y funciones que están disponibles en todos los ámbitos, mientras que el ámbito local se refiere a variables y funciones que solo están disponibles dentro de un ámbito específico.

Ejemplo:

Ejemplo:

## 6 FUNCIONES

---



```
# Àmbito global
x = "global" # Variable global

def foo():
    # Àmbito local
    x = "local" # Variable local
    print(x)

foo() # "local"
print(x) # "global"
```

En este ejemplo, la variable x se define como global al principio del código. Luego, dentro de la función foo(), se define una variable local x con el mismo nombre. Cuando se llama a la función foo(), se imprime la variable local x y cuando se imprime la variable global x fuera de la función, se imprime la variable global.

Reto:

Escribe un programa en Python que defina una variable global y con el valor 3 y una variable local 3 con el valor 5 dentro de una función. Luego, imprime ambas variables.

### 6.7 La palabra global

En Python, la palabra clave **global** se utiliza para hacer referencia a una variable global en el ámbito actual de una función.

Cuando se declara una variable dentro de una función, esta variable se considera una variable local y solo está disponible dentro de la función. Sin embargo, si se necesita acceder a una variable global desde dentro de una función, es necesario

## 6 FUNCIONES

---

declarar esa variable como global usando la palabra clave **global**. De lo contrario, Python creará una nueva variable local con el mismo nombre en lugar de acceder a la variable global.

Aquí hay un ejemplo de cómo utilizar la palabra clave **global**:



```
x = 10

def foo():
    global x
    x = 20
    print("Dentro:", x)

foo()
print("Fuera:", x)
```

En este ejemplo, la variable **x** se declara como global en la función **foo()**, lo que significa que la función puede acceder y modificar el valor de **x** en el ámbito global. Por lo tanto, después de llamar a la función **foo()**, el valor de **x** cambia a 20 tanto dentro como fuera de la función.

Es importante tener en cuenta que el uso de variables globales a menudo se considera una mala práctica en la programación y se recomienda evitarlo siempre que sea posible. En su lugar, se recomienda utilizar argumentos de función y valores de retorno para pasar datos entre funciones.

Reto:

1. Crea una variable global llamada **contador** e inicialízala con 0.
2. Escribe una función llamada **incrementar\_contador** que aumente el valor de **contador** en 1 cada vez que se llame a la función.

## 6 FUNCIONES

---

3. Luego, escribe otra función llamada resetear\_contador que restablezca el valor de contador a '0'.
4. Finalmente, escribe un programa que llame a incrementar\_contador 5 veces y luego llame a resetear\_contador. Verifica que el valor de contador se restablezca correctamente a 0 después de llamar a resetear\_contador. No olvides usar global.

### 6.8 La palabra nonlocal

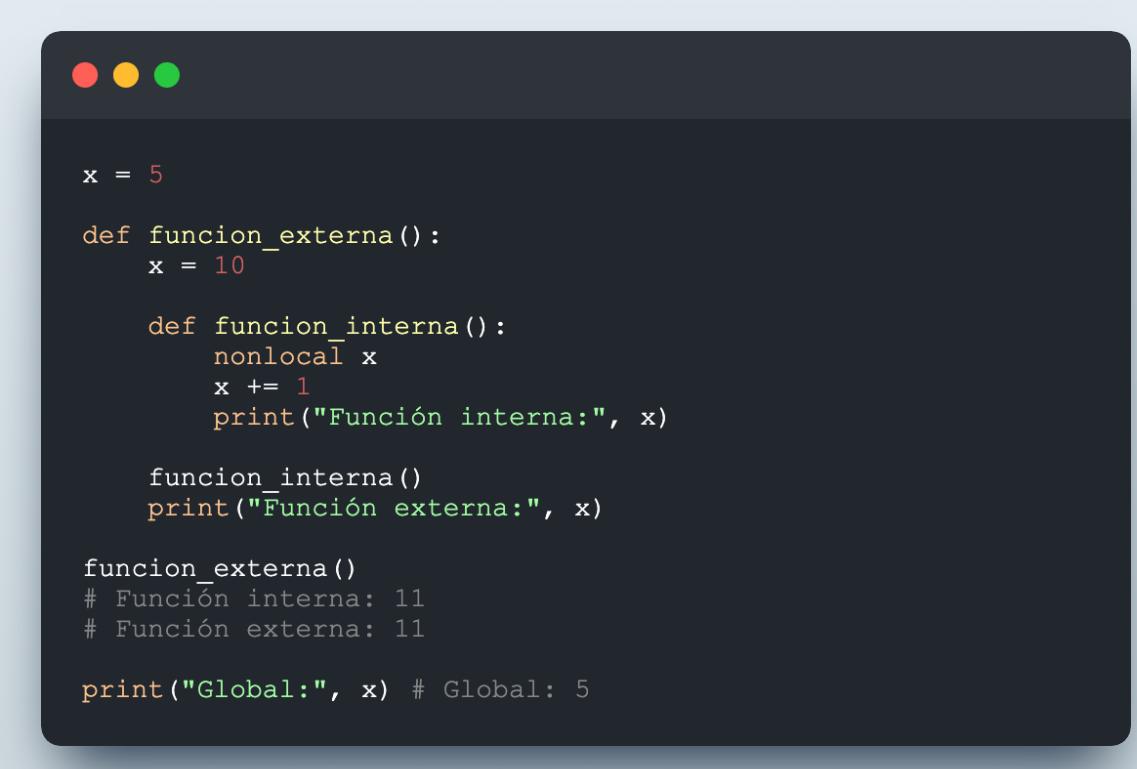
En Python, la palabra clave nonlocal se utiliza para hacer referencia a una variable en el ámbito que lo envuelve (enclosing), es decir, en un ámbito de nivel superior que no es el ámbito global.

Cuando se declara una variable dentro de una función, esta variable se considera una variable local y solo está disponible dentro de la función. Sin embargo, si se necesita acceder a una variable en un ámbito que lo envuelve (enclosing) desde dentro de una función anidada, es necesario declarar esa variable como no local usando la palabra clave nonlocal. De lo contrario, Python creará una nueva variable local con el mismo nombre en lugar de acceder a la variable no local.

Ejemplo:

## 6 FUNCIONES

---



```
x = 5

def funcion_externa():
    x = 10

    def funcion_interna():
        nonlocal x
        x += 1
        print("Función interna:", x)

    funcion_interna()
    print("Función externa:", x)

funcion_externa()
# Función interna: 11
# Función externa: 11

print("Global:", x) # Global: 5
```

En este ejemplo, la variable `x` se declara en el ámbito de la función `funcion_externa`, y luego se accede y modifica desde dentro de la función anidada `funcion_interna` utilizando la palabra clave `nonlocal`. De esta manera, la función `funcion_interna` puede acceder y modificar el valor de `x` en el ámbito que lo envuelve (enclosing) de `funcion_externa`.

Es importante tener en cuenta que la palabra clave `nonlocal` solo está disponible en Python 3 y no está disponible en versiones anteriores de Python. Además, el uso de variables no locales también a menudo se considera una mala práctica en la programación y se recomienda evitarlo siempre que sea posible. En su lugar, se recomienda utilizar argumentos de función y valores de retorno para pasar datos entre funciones.

Reto:

## 6 FUNCIONES

---

1. Crea una función llamada **contador** que lleve un registro de cuántas veces se ha llamado.
2. Dentro de la función **contador**, define una variable llamada **cuenta** y establece su valor inicial en 0.
3. Dentro de la función **contador**, define una función interna llamada **incremento** que incremente el valor de **cuenta** en 1 cada vez que se llame.
4. Usa la palabra clave **nonlocal** para hacer referencia a la variable **cuenta** dentro de la función **incremento**.
5. Llama a la función **contador** varias veces y verifica que el contador se incremente cada vez que se llama a la función **incremento**.

### 6.9 Funciones anidadas

Una función anidada en Python es una función definida dentro de otra función. Esto significa que la función interna está disponible en el ámbito local de la función externa, pero no está disponible fuera de ella.

Ejemplo:



The screenshot shows a Jupyter Notebook cell with three colored dots (red, yellow, green) at the top. The code in the cell is as follows:

```
def externa():
    mensaje = 'Hola'

    def interna():
        print(mensaje)

    interna()

externa()
```

## 6 FUNCIONES

---

Reto:

Escribe una función anidada que tome una lista como argumento y devuelva la suma de los elementos de la lista.

### 6.10 Cierres (closures)

Un cierre (closure) en Python es una función que recuerda el entorno en el que fue creada, es decir, los valores de las variables locales que estaban presentes en el momento de su creación. Esto significa que un cierre puede acceder a estas variables, incluso si el código que las creó ya no está presente en el ámbito actual.

Ejemplo:

```
def outer_func():
    message = 'Hola'

    def inner_func():
        print(message)

    return inner_func

my_func = outer_func()
my_func() # Hola
```

Reto:

Crea una función que tome una lista como argumento y devuelva una función que imprima el índice y el valor de cada elemento de la lista.

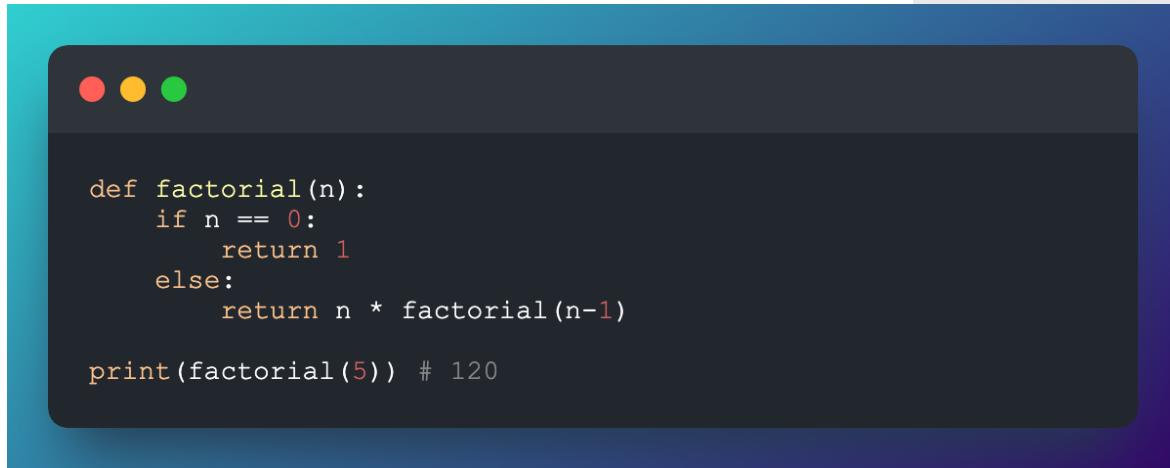
## 6 FUNCIONES

---

### 6.11 Recursividad

La recursividad en Python es una técnica de programación que se basa en la llamada a una función desde dentro de sí misma. Esto permite que una función se ejecute repetidamente hasta que se cumpla una condición de parada.

Ejemplo:



```
def factorial(n):
    if n == 0:
        return 1
    else:
        return n * factorial(n-1)

print(factorial(5)) # 120
```

El ejercicio define una función llamada “factorial” que toma un número entero “n” como argumento. La función calcula el factorial de “n” utilizando una técnica recursiva.

El factorial de un número entero “n” se define como el producto de todos los números enteros positivos desde 1 hasta “n”. Por ejemplo, el factorial de 5 se calcula como  $1 \times 2 \times 3 \times 4 \times 5 = 120$ .

En el código, la función “factorial” verifica si “n” es igual a cero. Si es así, devuelve 1, ya que el factorial de cero es 1 por definición. Si “n” no es cero, la función calcula el factorial de “n” multiplicando “n” por el factorial del número entero “n-1”. Luego, la función se llama a sí misma con el argumento “n-1” hasta que el valor de “n” llega a cero.

Finalmente, el código imprime el resultado del cálculo del factorial de 5 llamando a la función “factorial” con un argumento de 5 y devuelve el valor 120.

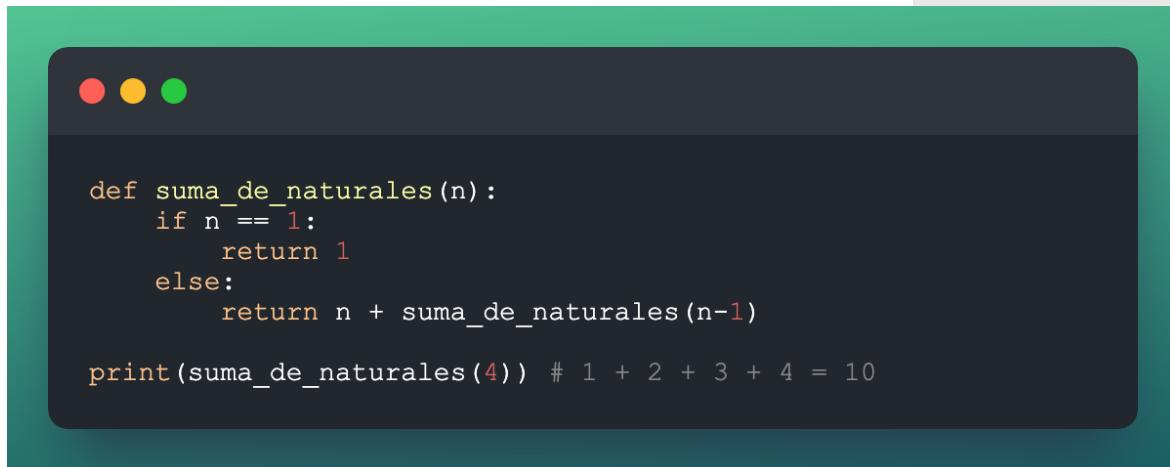
## 6 FUNCIONES

---

Reto:

Escribir una función recursiva que calcule la suma de los números naturales hasta un número n dado.

La recursividad puede ser algo compleja al iniciar y este ejercicio contiene el concepto de números naturales que no es específico a la programación así que te dejo la respuesta del reto para que lo revises.



A screenshot of a dark-themed Python code editor. At the top, there are three colored window control buttons (red, yellow, green). The code itself is as follows:

```
def suma_de_naturales(n):
    if n == 1:
        return 1
    else:
        return n + suma_de_naturales(n-1)

print(suma_de_naturales(4)) # 1 + 2 + 3 + 4 = 10
```

### 6.12 Funciones anónimas o lambda

Una función anónima o lambda en Python es una función que no tiene un nombre y se define en una sola línea. Estas funciones son útiles cuando se necesita una función simple y pequeña para una tarea específica.

Ejemplo:

## 6 FUNCIONES

---

```
# Función anónima para elevar al cuadrado un número
cuadrado = lambda x: x**2

# Usar la función
print(cuadrado(5)) # 25
```

Aquí hay otro ejemplo de una función lambda con varios parámetros:

```
# Definir una función lambda con dos parámetros
multiplicar = lambda x, y: x * y

# Llamar a la función multiplicar con dos argumentos
resultado = multiplicar(5, 7)

# Imprimir el resultado
print(resultado) # 35
```

En este ejemplo, se define una función lambda con dos parámetros “x” e “y” que multiplican los dos argumentos entre sí y devuelve el resultado. Luego, se llama a la función lambda con los argumentos 5 y 7, lo que da como resultado un valor de 35. Finalmente, se imprime el resultado utilizando la función print().

Reto:

Escribe una función anónima para calcular el área de un círculo dado su radio.

## 6 FUNCIONES

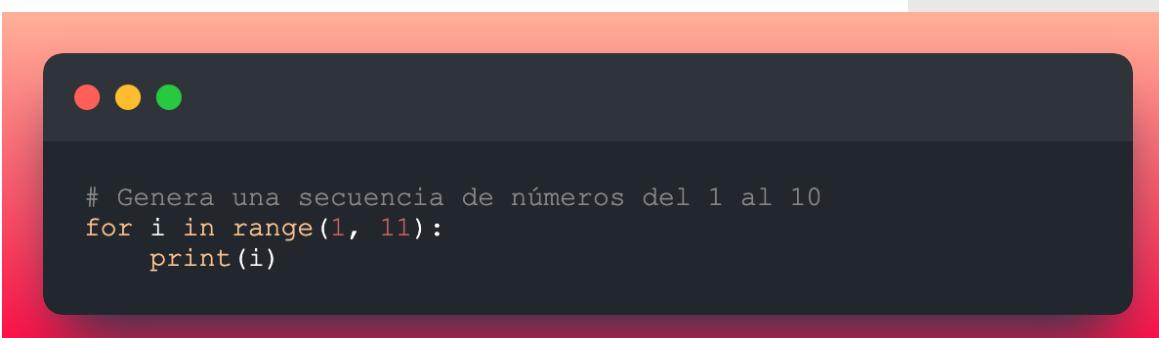
---

### 6.13 Funciones incorporadas (built-in)

#### 6.13.1 range()

range() es una función de Python que nos permite generar una secuencia de números. Esta función recibe como parámetros un número inicial, un número final y opcionalmente un paso. El número inicial es el primer número de la secuencia, el número final es el último número de la secuencia y el paso es el número de incremento entre cada número de la secuencia.

Ejemplo:



```
# Genera una secuencia de números del 1 al 10
for i in range(1, 11):
    print(i)
```

Reto:

Escribe un programa que imprima los números del 1 al 20, saltando de 2 en 2.

#### 6.13.2 print()

print() es una función en Python que se utiliza para imprimir una cadena de texto o un valor en la pantalla. Por ejemplo, si queremos imprimir la frase “Hola mundo” en la pantalla, podemos usar la función print() de la siguiente manera:

## 6 FUNCIONES

---

```
print("Hola mundo")
```

Reto:

Imprimir la frase “¡Hola mundo por enésima vez!” en la pantalla usando la función `print()`.

### 6.13.3 `len()`

`len()` es una función incorporada en Python que se utiliza para obtener la longitud de una secuencia, como una cadena de texto, lista o tupla. La longitud se refiere al número de elementos o caracteres que contiene la secuencia.

Ejemplo:

```
frutas = ['manzana', 'banana', 'cereza']
len(frutas) # 3
```

En este ejemplo, creamos una lista de frutas y llamamos a la función `len()` en la lista. La función devuelve 3, lo que significa que hay 3 elementos en la lista.

Reto:

- Crea una cadena de texto y usa la función `len()` para obtener su longitud.

## 6 FUNCIONES

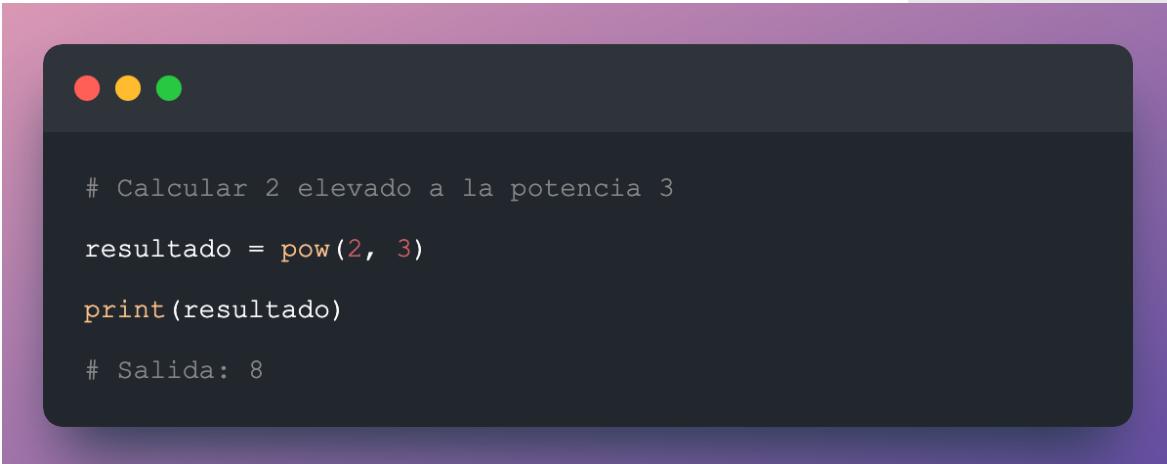
---

- Crea una lista o tupla y usa la función **len()** para obtener su longitud.
- Crea un diccionario y usa la función **len()** para obtener su longitud (número de pares clave-valor).

### 6.13.4 pow()

**pow()** es una función incorporada en Python que permite calcular la potencia de un número. Esta función toma dos argumentos, el primer argumento es el número base y el segundo argumento es el exponente.

Ejemplo:



```
# Calcular 2 elevado a la potencia 3
resultado = pow(2, 3)
print(resultado)

# Salida: 8
```

Reto:

Calcular el resultado de 5 elevado a la potencia 4.

Puedes encontrar la lista completa de todas las funciones incorporadas en Python en la documentación oficial de Python en el siguiente enlace:

<https://docs.python.org/3/library/functions.html>

Esta página incluye una descripción detallada de cada función incorporada, incluyendo su sintaxis, argumentos y valor de retorno. Además, también proporciona ejemplos prácticos de cómo usar cada función en el código.

# 7 Números

## 7.1 Números binarios

Los números binarios en Python son números con sólo dos dígitos 0 y 1 que se representan con un prefijo de 0b (cero y letra b). Estos números se utilizan en computación para representar información, ya que se pueden representar con una sola línea de código.

- 0b0 binario es 0 en decimal
- 0b1 binario es 1 en decimal
- 0b10 binario es 2 en decimal
- 0b11 binario es 3 en decimal
- 0b100 binario es 4 en decimal

Ejemplo:

```
numero_binario = 0b100

# Convertir un número decimal a binario

numero_decimal = 10

# Convertir el número decimal a binario
numero_binario = bin(numero_decimal)

# Imprimir el resultado
print(numero_binario) # 0b1010
```

Reto:

Escribe un programa en Python para convertir un número binario a decimal.

## 7 NÚMEROS

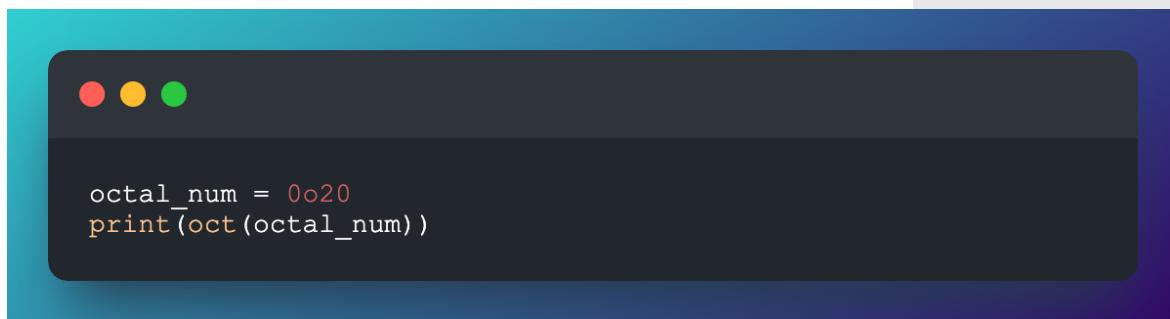
---

### 7.2 Números octales

Los números octales en Python se representan con un prefijo de 0o (cero y letra o). Estos números se usan para representar valores binarios en sistemas de computación. Por ejemplo, el número octal 0o17 se representa como 1111 en binario.

- 0o0 octales es 0 en decimal
- 0o1 octales es 1 en decimal
- 0o7 octales es 7 en decimal
- 0o10 octales es 8 en decimal
- 0o17 octales es 15 en decimal

Ejemplo:



```
octal_num = 0o20
print(oct(octal_num))
```

The screenshot shows a terminal window with a dark background and three colored dots (red, yellow, green) at the top. The code `octal\_num = 0o20` is entered on the first line, followed by `print(oct(octal\_num))` on the second line. The output of the program, `0o20`, is displayed below the code.

Reto:

Escribe un programa que tome un número octal como entrada y lo convierta a un número decimal.

### 7.3 Números hexadecimales

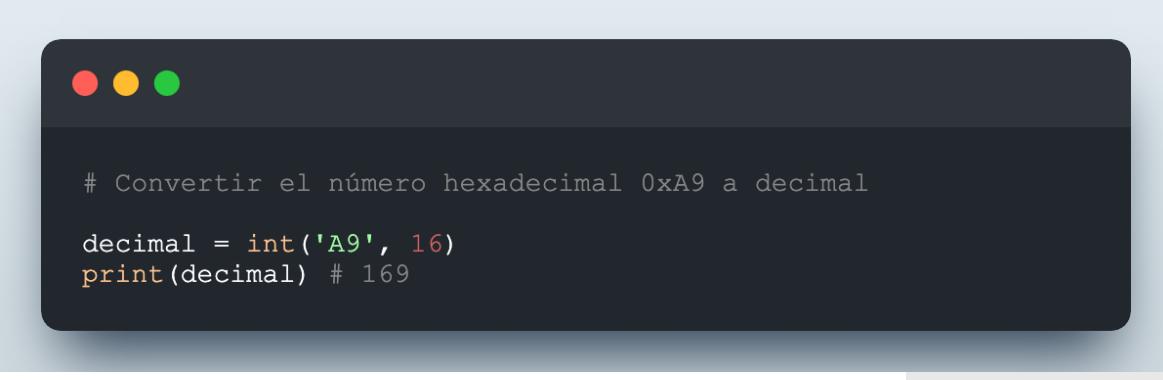
Los números hexadecimales en Python son una forma de representar números enteros usando una notación de base 16. Esta notación se usa para representar números binarios de 8 bits o más. Los números hexadecimales se escriben usando los dígitos 0-9 y las letras A-F.

## 8 TEXTO

---

- 0x0 hexadecimal es 0 en decimal
- 0x9 hexadecimal es 9 en decimal
- 0xA hexadecimal es 10 en decimal
- 0xF hexadecimal es 15 en decimal
- 0x10 hexadecimal es 16 en decimal

Ejemplo:



```
# Convertir el número hexadecimal 0xA9 a decimal
decimal = int('A9', 16)
print(decimal) # 169
```

Reto:

Escribe un programa que tome el número 17 en hexadecimal como entrada y lo convierta a decimal.

## 8 Texto

### 8.1 Formas de declarar texto

En Python, hay varias formas de declarar texto. Una forma es usar comillas simples (' ') o comillas dobles (""). Por ejemplo:

## 8 TEXTO

---

```
● ● ●  
texto_1 = 'Hola mundo'  
texto_2 = "Hola mundo"
```

También se puede usar la función str() para convertir un número a una cadena de texto. Por ejemplo:

```
● ● ●  
numero = 5  
texto_3 = str(numero)
```

Reto:

Crea una variable llamada texto\_4 que contenga el texto “Pink Floyd” . Pruébalo con comillas simples y comillas dobles.

### 8.2 Comillas simples vs comillas dobles

Ahora vamos a hablar sobre las diferencias entre las comillas simples y las comillas dobles. Y no, ¡no se trata de una batalla de quién es mejor! El uso de comillas simples o dobles en Python dependerá de la situación y de cómo necesitemos utilizar las comillas dentro del código.

Las comillas dobles son más comunes en Python porque se utilizan para definir cadenas de texto. Por ejemplo, si queremos imprimir la frase “Hola mundo” en la

## 8 TEXTO

---

consola, podríamos escribir:

```
print("Hola mundo")
```

Por otro lado, las comillas simples se utilizan con mayor frecuencia para definir cadenas de texto que contienen comillas dobles. Por ejemplo, si queremos imprimir la frase “Él dijo:” Hola mundo”” en la consola, podríamos escribir:

```
print('Él dijo: "Hola mundo"')
```

También lo podríamos hacerlo de forma reversa si lo necesitamos:

```
print("Él dijo: 'Hola mundo'")
```

En general se recomienda ser consistente con el uso de comillas en un proyecto. Y si escoges comillas simples para todos tus textos, puedes también escapar caracteres para usar el mismo tipo de comillas dentro del texto con barra oblicua \.

## 8 TEXTO

---

```
print("Él dijo: \"Hola mundo\")")
```

Reto:

¿Puedes imprimir en la consola la siguiente cadena de texto utilizando comillas simples y dobles al mismo tiempo?

“Caminante no hay camino, se hace camino al andar” - Antonio Machado

### 8.3 Texto multilínea

Texto multilínea en Python es una forma de escribir una cadena de texto que se extiende a través de varias líneas. Esto se logra usando tres comillas dobles ( """ ) al principio y al final de la cadena.

Ejemplo:

```
texto_multilinea = """Esta es una cadena de texto  
que se extiende a través de  
varias líneas"""
```

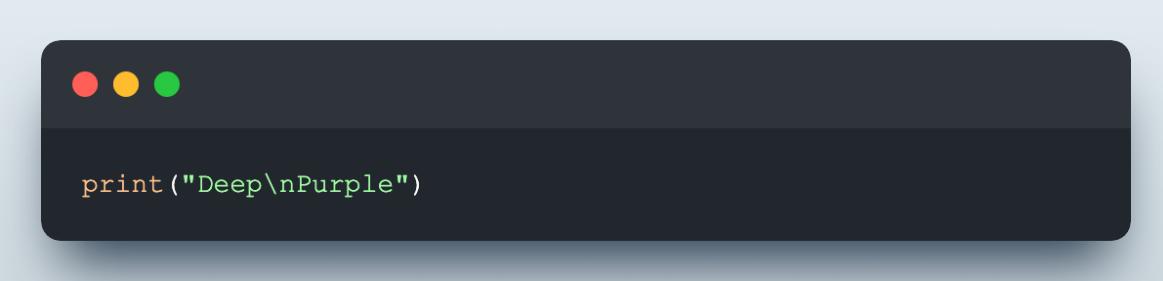
Reto:

Escribe una cadena de texto multilínea que contenga al menos 5 líneas.

## 8.4 Secuencias de escape

Las secuencias de escape en Python son caracteres especiales que se usan para representar caracteres no imprimibles o caracteres especiales. Estas secuencias de escape comienzan con una barra invertida (\) y seguida por un carácter especial. Por ejemplo, \n es una secuencia de escape que representa un salto de línea.

Ejemplo:



```
print("Deep\nPurple")
```

Esto imprimirá “Deep” en una línea y “nPurple” en la siguiente línea.

Reto:

Imprime una frase en la que la primera palabra esté en una línea y la segunda palabra en la siguiente línea.

## 8.5 Más sobre f-strings

Las f-strings en Python son una forma súper fácil y efectiva de imprimir variables en tus dentro de tu texto. Las f-strings en Python sustituyen la variable dentro de la cadena de texto al poner la letra f adelante y llaves {} dentro del texto.

Por ejemplo, si quieres imprimir el resultado de una suma de dos variables:

## 8 TEXTO

---

```
● ● ●  
num1 = 5  
num2 = 10  
print(f"La suma de {num1} y {num2} es igual a {num1 + num2}")  
# Output: "La suma de 5 y 10 es igual a 15"
```

O si quieres imprimir una cadena con un cierto número de decimales:

```
● ● ●  
pi = 3.14159265359  
print(f"El valor de pi es aproximadamente {pi:.2f}")  
# Output: "El valor de pi es aproximadamente 3.14"
```

¡Vaya! las f-strings son como una navaja suiza para tus programas en Python. Son eficientes, efectivas y ahorran tiempo. Pruébalas en tu próximo proyecto y verás cómo se vuelven adictivas.

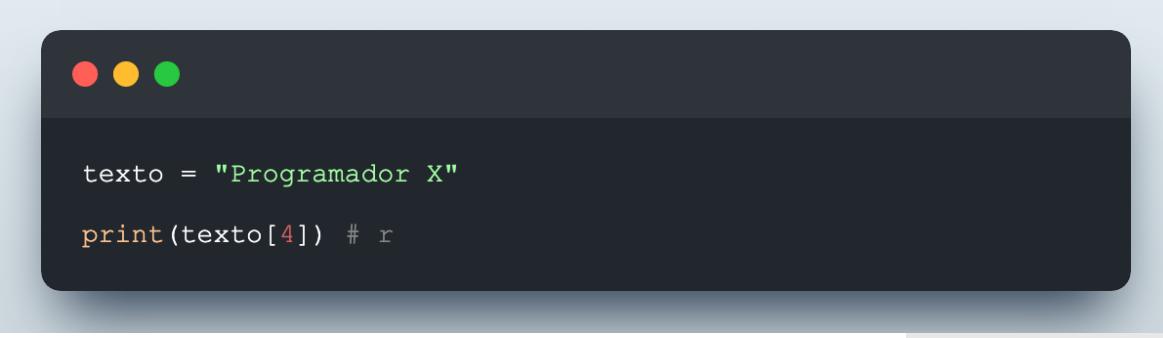
Reto:

Intenta imprimir un mensaje de bienvenida personalizado usando f-strings. Por ejemplo, “Hola Juan, ¡bienvenido a este programa en Python!” donde “Juan” es una variable que puedes modificar.

## 8.6 Acceder a un carácter en texto

Para acceder a un carácter en un texto en Python, se puede usar el índice de la cadena de texto. El índice es un número entero que se usa para identificar un carácter en una cadena de texto. Por ejemplo, si tenemos una cadena de texto como “Hola Mundo”, podemos acceder al carácter “M” usando el índice 4.

Ejemplo:



```
text = "Programador X"  
print(text[4]) # r
```

Reto:

Crea una cadena de texto y usa el índice para acceder a los caracteres de la cadena de texto.

## 8.7 Función len() en texto

La función len de un texto en Python es una función que nos permite conocer la cantidad de caracteres que contiene una cadena de texto. Esta función se puede utilizar para realizar operaciones como contar la cantidad de palabras en una frase, o para limitar la cantidad de caracteres que se pueden ingresar en un campo de un formulario.

Ejemplo:

## 8 TEXTO

---

```
● ● ●

texto = "Elefante"

# Obtener la cantidad de caracteres
longitud = len(texto)

# Imprimir la cantidad de caracteres
print(longitud)

# Resultado: 8
```

Reto:

Crea un programa que cuente la cantidad de caracteres en una frase.

### 8.8 Métodos de texto

#### 8.8.1 Método de texto upper()

Los métodos de texto en Python son funciones integradas que se pueden aplicar a una cadena de texto (una variable que contiene una secuencia de caracteres) para realizar diversas operaciones en ella.

upper() es un método de texto que convierte todas las letras de una cadena de texto a mayúsculas.

Ejemplo:

## 8 TEXTO

---

```
● ● ●  
texto = "Australia"  
texto_mayusculas = texto.upper()  
  
print(texto_mayusculas) # AUSTRALIA
```

Reto:

Crea un programa que tome una frase como entrada y devuelva la misma frase con todas las letras en mayúsculas.

### 8.8.2 Método de texto lower()

lower() es un método de texto que convierte una cadena de caracteres a minúsculas.

Ejemplo:

```
● ● ●  
texto = "Nueva Zelanda"  
texto_min = texto.lower()  
  
print(texto_min) # nueva zelanda
```

Reto:

Crea un programa que tome una frase como entrada y la imprima en mayúsculas y minúsculas.

### 8.8.3 Método de texto partition()

partition() es un método de texto que se usa para dividir una cadena de texto en una tupla de tres partes. Este método toma una cadena de texto y una cadena de separación como argumentos y devuelve una tupla que contiene la parte de la cadena antes de la cadena de separación, la cadena de separación misma y la parte de la cadena después de la cadena de separación.

Ejemplo:

```
# Definimos una cadena de texto
texto = "Python es un lenguaje de programación"

# Usamos la función partition() para dividir la cadena de
# texto
resultado = texto.partition("es un")

# Mostramos el resultado
print(resultado) # ('Python ', 'es un', ' lenguaje de
# programación')
```

Reto:

Escribe un programa que divida una cadena de texto en tres partes usando la función partition(). La cadena de texto debe contener una palabra clave que se usará como cadena de separación.

### 8.8.4 Método de texto replace()

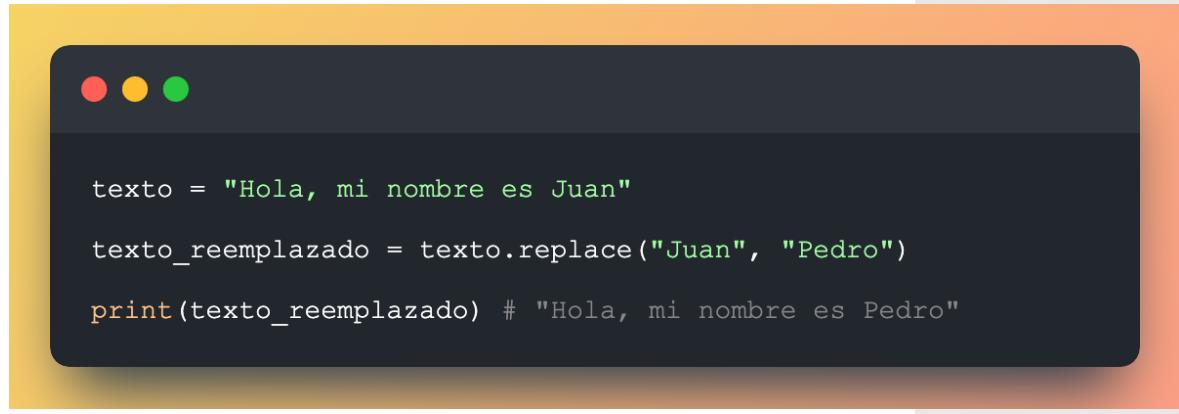
replace() es un método de texto que permite reemplazar una cadena de caracteres específica dentro de una cadena de caracteres más grande. Este método toma

## 8 TEXTO

---

dos argumentos: el primer argumento es la cadena de caracteres que se desea reemplazar, y el segundo argumento es la cadena de caracteres que se usará para reemplazarla.

Ejemplo:



```
● ● ●

texto = "Hola, mi nombre es Juan"
texto_reemplazado = texto.replace("Juan", "Pedro")
print(texto_reemplazado) # "Hola, mi nombre es Pedro"
```

Reto:

Escribe un programa que tome una frase como entrada y reemplace todas las letras “a” con la letra “e” .

### 8.8.5 Método de texto find()

find() es un método de texto que busca una cadena de caracteres dentro de otra cadena de caracteres. Devuelve la posición de la primera aparición de la cadena buscada, o -1 si no se encuentra.

Ejemplo:

## 8 TEXTO

---

```
● ● ●  
cadena = "Hola mundo"  
posicion = cadena.find("mundo")  
print(posicion) # 5
```

Reto:

Escribe un programa que busque la palabra “Python” en la cadena “¡Aprende Python hoy!” y devuelva la posición de la primera aparición de la palabra.

### 8.8.6 Método de texto strip()

strip() es un método de texto que se utiliza para eliminar los espacios en blanco especificados de la derecha e izquierda de una cadena.

Ejemplo:

```
● ● ●  
  
# Cadena con espacios en blanco al final  
mi_texto = "    Bangkok    "  
  
# Usamos strip() para eliminar los espacios en blanco  
mi_texto = mi_texto.rstrip()  
  
# Imprimimos la cadena  
print(mi_texto) # `Bangkok`
```

Reto:

## 8 TEXTO

---

Escribe un programa que lea una cadena del usuario y elimine los espacios en blanco de la cadena " Bad memories "

### 8.8.7 Método de texto split()

split() es un método de texto que se utiliza para dividir una cadena de texto en una lista de subcadenas. Este método toma un argumento opcional que es un delimitador, que es un carácter que se utiliza para separar las subcadenas. Por defecto, el delimitador es un espacio en blanco.

Ejemplo:

```
cadena = "Hola, soy una cadena de texto"
lista = cadena.split()
print(lista) # ['Hola,', 'soy', 'una', 'cadena', 'de',
'texto']
```

Reto:

Escribe un programa que tome una cadena de texto como entrada y divida la cadena en una lista de subcadenas usando el delimitador ',' . Imprime la lista resultante.

### 8.8.8 Iterar a través de un texto

Iterar a través de un texto en Python significa recorrer el texto palabra por palabra. Esto se puede hacer usando un bucle for. Por ejemplo, para imprimir cada palabra de una cadena de texto:

## 8 TEXTO

---

```
● ● ●  
texto = "Hola, esto es un ejemplo de iteración"  
  
for palabra in texto.split():  
    print(palabra)
```

Reto:

Itera a través de una cadena de texto y cuenta el número de palabras que contiene.

### 8.8.9 Método de texto startswith()

startswith() es un método de texto que permite verificar si una cadena de texto comienza con una subcadena específica. Este método devuelve un valor booleano (True o False).

Ejemplo:

```
● ● ●  
texto = "Hola, ¿cómo estás?"  
  
resultado = texto.startswith("Hola")  
  
print(resultado) # True
```

Reto:

## 8 TEXTO

---

Escribe un programa que verifique si una cadena de texto comienza con la palabra “Hola” o “Adiós” . Si comienza con “Hola” , imprime “Saludo” , y si comienza con “Adiós” , imprime “Despedida” .

### 8.8.10 Método de texto isnumeric()

isnumeric() es un método de texto que se utiliza para determinar si una cadena es numérica. Devuelve True si la cadena es numérica y False si no lo es.

Ejemplo:

```
# Cadena numérica
x = "12345"

# Comprobar si x es numérica
print(x.isnumeric()) # True
```

Reto:

Crea una cadena que contenga un número decimal y comprueba si es numérica usando isnumeric().

Los métodos que vimos vienen del objeto str() que envuelve a las cadenas de texto en Python. El enlace oficial de la documentación de Python donde se describen todos los métodos de texto es el siguiente:

<https://docs.python.org/3/library/stdtypes.html#text-sequence-type-str>

## 9 Listas

### 9.1 Crear una lista

Crear una lista en Python es una tarea sencilla. Una lista es una colección de elementos ordenados, que pueden ser de cualquier tipo de dato.

Ejemplo:

```
lista_ejemplo = [1, 2, 3, 4, 5]
```

Reto:

Crea una lista con los nombres de los meses del año y luego imprimela.

### 9.2 Los índices comienzan en cero

Si eres nuevo en Python, una de las cosas más confusas que puedes encontrarte es el hecho de que los índices comienzan en cero en este lenguaje. Pero, ¡no te preocupes! No eres el único que se ha confundido por esto.

¿Por qué los índices comienzan en cero en Python? Bueno, esta decisión está relacionada con la forma en que la memoria se almacena en los computadores.

Ahora, ¿cómo aplicamos esto en la práctica? Bueno, imagina que tienes una lista de ingredientes para hacer una hamburguesa y quieres acceder al primer elemento en la lista. En lugar de escribir índice 1, debes escribir índice 0:

## 9 LISTAS

---

```
● ● ●  
lista_ingredientes = ["pan", "lechuga", "tomate", "carne",  
"queso"]  
primer_ingrediente = lista_ingredientes[0]  
print(primer_ingrediente) # Output: "pan"
```

Reto:

Crea una lista de tus tres comidas favoritas y muestra el segundo elemento de la lista utilizando su índice. ¡No olvides que los índices comienzan en cero!

### 9.3 Acceder a elementos de una lista

Para acceder a los elementos de una lista en Python, se utiliza el índice de cada elemento. El índice de un elemento es su posición en la lista, comenzando desde 0. Por ejemplo, si tenemos la lista `mi_lista = [1, 2, 3, 4]`, el primer elemento de la lista es `mi_lista[0]` y el último elemento es `mi_lista[3]`.

Ejemplo:

## 9 LISTAS

---

```
● ● ●
```

```
mi_lista = [1, 2, 3, 4]

# Acceder al primer elemento
primer_elemento = mi_lista[0]
print(primer_elemento) # Imprime 1

# Acceder al último elemento
ultimo_elemento = mi_lista[3]
print(ultimo_elemento) # Imprime 4

# Accede a un rango desde el índice 1 hasta el índice 3
# sin incluir el elemento en el índice 3
rango = mi_lista[1:3]
print(rango) # Imprime [2,3]

# Accede a un rango desde el índice 0 hasta el índice 3
# sin incluir el elemento en el índice 3
rango = mi_lista[:3]
print(rango) # Imprime [1,2,3]

# Accede a un elemento desde el final
penultimo_elemento = mi_lista[-2]
print(penultimo_elemento) # Imprime [3]
```

Reto:

Crea una lista con 5 elementos y accede al tercer elemento.

### 9.4 Modificar elementos de una lista

Modificar elementos de una lista en Python es una tarea sencilla. Para modificar un elemento de una lista, simplemente asignamos un nuevo valor al índice del elemento que deseamos modificar. Por ejemplo, si tenemos una lista llamada “mi\_lista” con los

## 9 LISTAS

---

elementos [1, 2, 3, 4], podemos modificar el segundo elemento de la lista asignando un nuevo valor al índice 1:

```
● ● ●  
mi_lista = [1, 2, 3, 4]  
mi_lista[1] = 5
```

Ahora, la lista “mi\_lista” contiene los elementos [1, 5, 3, 4].

Reto:

Crea una lista con los números del 1 al 10. Luego, modifica el último elemento de la lista para que sea igual al número 15.

### 9.5 Borrar elementos de una lista

Para borrar elementos de una lista en Python, se puede usar la palabra clave `del` indicando el índice del elemento.

```
● ● ●  
frutas = ["plátano", "manzana", "naranja"]  
del frutas[1];  
print(frutas)
```

El resultado sería: ['plátano', 'naranja'].

Reto:

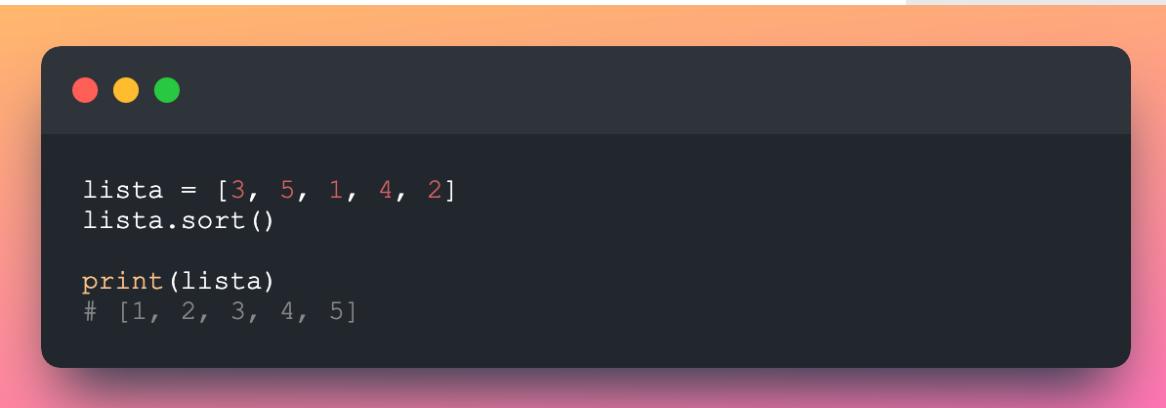
## 9 LISTAS

---

Crea una lista con 5 elementos y elimina el tercer elemento.

### 9.6 Ordenar listas

Ordenar listas en Python es una tarea sencilla. Para ordenar una lista, simplemente use el método `sort()`. Por ejemplo, para ordenar la lista [3, 5, 1, 4, 2]:



```
lista = [3, 5, 1, 4, 2]
lista.sort()

print(lista)
# [1, 2, 3, 4, 5]
```

Reto:

Cree una lista con números enteros aleatorios y ordénela de menor a mayor.

### 9.7 Comprensión de listas

Si tienes una lista y necesitas crear otra lista con algunos elementos seleccionados o modificados, la comprensión de listas se convierte en nuestra mejor amiga.

Imagínate que tienes una lista de números y quieres crear una nueva lista con el doble de cada número. En lugar de crear una nueva lista vacía y utilizar un `for` para recorrer la lista original y agregar cada número doble a la nueva lista, podemos simplificar todo con una comprensión de listas:

## 9 LISTAS

---



```
numeros = [1, 2, 3, 4, 5]
dobles = [numero * 2 for numero in numeros]
```

Ahí lo tienes, ahora tienes una nueva lista dobles con los números originales multiplicados por dos. La sintaxis es bastante sencilla: se inicia con los elemento que se van a agregar a la nueva lista (en este caso numero \* 2), seguido de for el nombre del item, seguido de in, y por último la lista original.

Pero eso no es todo lo que se puede hacer. Podemos agregar condicionales para seleccionar solo ciertos elementos o modificarlos. Por ejemplo, imaginemos que queremos crear una lista con solo los números pares del ejemplo anterior:



```
numeros = [1, 2, 3, 4, 5]
pares = [numero for numero in numeros if numero % 2 == 0]
```

¡Fantástico! Ahora tienes una nueva lista pares con solo los números pares de la lista original. La única diferencia en la sintaxis es agregar el condicional al final de la comprensión de la lista.

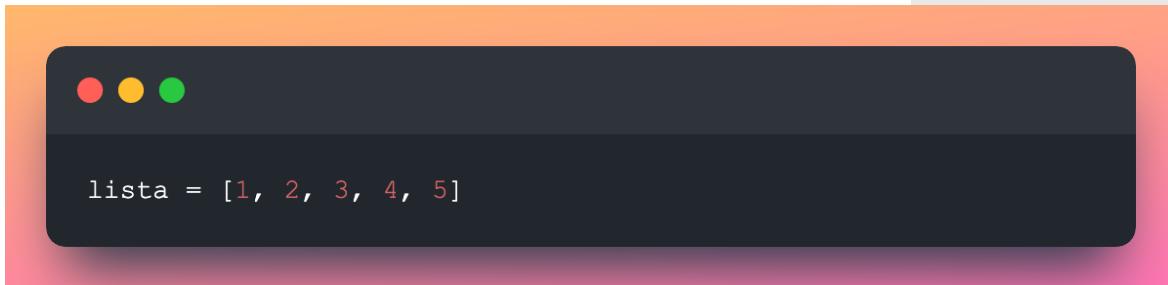
Aquí está el reto: crea una lista de palabras y luego crea una nueva lista solo con las palabras que empiecen con la letra “s” .

## 9 LISTAS

---

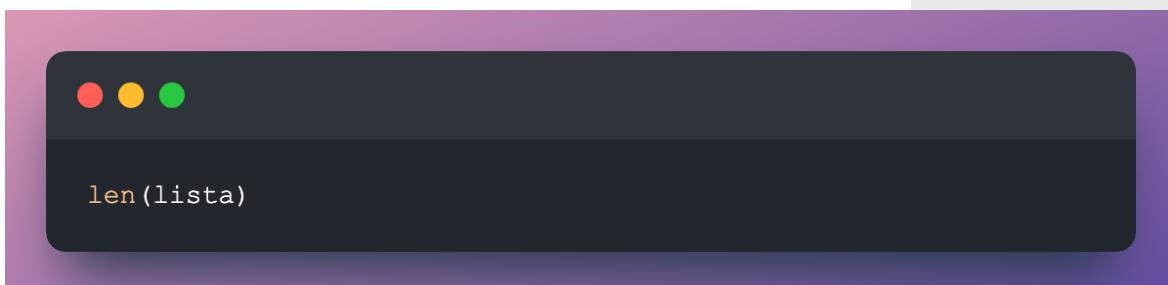
### 9.8 Función len() en listas

La función len() nos permite conocer la cantidad de elementos que contiene una lista. Por ejemplo, si tenemos la siguiente lista:



```
● ● ●
lista = [1, 2, 3, 4, 5]
```

Podemos conocer la cantidad de elementos que contiene la lista con la función len():



```
● ● ●
len(lista)
```

El resultado será 5, que es la cantidad de elementos que contiene la lista.

Reto:

Crea una lista con 5 elementos y usa la propiedad length para conocer la cantidad de elementos que contiene.

### 9.9 Iterar a través de una lista

Iterar a través de una lista en Python es una tarea común. Esto significa recorrer cada elemento de la lista uno por uno. Un ejemplo práctico y básico sería:

## 9 LISTAS

---

```
● ● ●  
lista = [1, 2, 3, 4, 5]  
  
for elemento in lista:  
    print(elemento)
```

Esto imprimirá cada elemento de la lista en una línea separada.

Reto:

Crea una lista con los números del 1 al 10 y usa un bucle for para imprimir cada elemento de la lista en una línea separada.

### 9.10 Métodos de listas

#### 9.10.1 Método de lista append()

append() es una función de Python que se usa para agregar un elemento al final de una lista. Por ejemplo, si tenemos una lista llamada `mi_lista` con los elementos [1, 2, 3], podemos usar `mi_lista.append(4)` para agregar el elemento 4 al final de la lista. Esto haría que `mi_lista` ahora contenga los elementos [1, 2, 3, 4].

```
● ● ●  
mi_lista = [1, 2, 3]  
mi_lista.append(4)  
print(mi_lista)
```

## 9 LISTAS

---

Reto:

Crea una lista vacía llamada mi\_lista y agrega el elemento "nuevo" al final de la lista.

### 9.10.2 Método de lista extend()

extend() es un método de listas en Python que permite agregar elementos a una lista existente. Por ejemplo, si tenemos una lista llamada lista\_1 con los elementos [1, 2, 3], podemos usar extend() para agregar los elementos [4, 5, 6] a la lista:

```
lista_1 = [1, 2, 3]
lista_2 = [4, 5, 6]
lista_1.extend(lista_2)

print(lista_1)
```

Salida: [1, 2, 3, 4, 5, 6]

Reto:

Crea una lista llamada lista\_3 con los elementos [7, 8, 9] y usa extend() para agregarlos a lista\_1.

### 9.10.3 Método de lista insert()

insert() es una función de Python que se usa para insertar un elemento en una lista en una posición específica. Por ejemplo, si tenemos una lista de números:

## 9 LISTAS

---

```
lista = [1, 2, 3, 4, 5]
```

Podemos usar la función `insert()` para insertar un número en una posición específica, por ejemplo, para insertar el número 6 en la posición 3:

```
lista.insert(3, 6)
```

La lista ahora se verá así: [1, 2, 3, 6, 4, 5]

Reto:

Crea una lista con 5 elementos y usa la función `insert()` para insertar un elemento en una posición específica.

### 9.10.4 Método de lista `remove()`

`remove()` es una función de Python que se utiliza para eliminar un elemento específico de una lista. Por ejemplo, si tenemos una lista de nombres:

Podemos usar `remove()` para eliminar el nombre “Ana” de la lista:

## 9 LISTAS

---

```
nombres = ["Juan", "Pedro", "Ana", "María"]
nombres.remove("Ana")
print(nombres) # ["Juan", "Pedro", "María"]
```

Reto:

Crea una lista con 5 elementos y elimina uno de ellos usando remove().

### 9.10.5 Método de lista pop()

pop() es una función de Python que se utiliza para eliminar el último elemento de una lista. Por ejemplo, si tenemos una lista llamada mi\_lista con los elementos [1, 2, 3, 4], podemos usar mi\_lista.pop() para eliminar el último elemento de la lista, que en este caso es el número 4. La lista quedaría entonces como [1, 2, 3].

```
mi_lista = [1, 2, 3, 4]
mi_lista.pop()

print(mi_lista)
```

Reto:

Crea una lista con 5 elementos y usa la función pop() para eliminar el último elemento

## 9 LISTAS

---

de la lista.

### 9.10.6 Método de lista clear()

clear() es una función de Python que se utiliza para eliminar todos los elementos de una lista. Esta función no toma ningún argumento y no devuelve ningún valor.

Ejemplo:

```
● ● ●  
lista = [1, 2, 3, 4, 5]  
lista.clear()  
print(lista) # []
```

Reto:

Crea una lista con 5 elementos y usa la función clear() para eliminar todos los elementos de la lista.

### 9.10.7 Método de lista index()

index() es una función de Python que devuelve el índice de la primera aparición de un elemento en una lista.

Ejemplo:

## 9 LISTAS

---

```
● ● ●  
lista = [1, 2, 3, 4, 5, 3]  
# Devuelve el índice de la primera aparición de 3  
indice = lista.index(3)  
  
print(indice) # 2
```

Reto:

Crea una lista con 5 elementos y usa la función index() para encontrar el índice de un elemento de tu elección.

### 9.10.8 Método de lista count()

count() es una función de Python que se utiliza para contar el número de veces que un elemento específico aparece en una lista, tupla o cadena de caracteres.

Ejemplo:

```
● ● ●  
lista = [1, 2, 3, 4, 5, 1, 2, 3, 4, 5]  
# Contar el número de veces que aparece el número 1 en la  
# lista  
print(lista.count(1)) # 2
```

Reto:

Crea una lista con 10 elementos y usa la función count() para contar el número de veces que aparece el número 5.

## 10 Tuplas

### 10.1 Crear una tupla

Una tupla en Python es una secuencia inmutable de elementos. Esto significa que una vez creada, no se puede modificar. Esto lo hace útil para almacenar datos que no se espera que cambien.

Ejemplo:

```
# crear tupla
tupla = (1, 2, 3, 4, 5)

# alternativamente se pueden crear a partir del objeto
tuple()
tupla = tuple([1, 2, 3, 4, 5])

# alternativamente se pueden obviar los paréntesis
tupla = 1, 2, 3, 4, 5

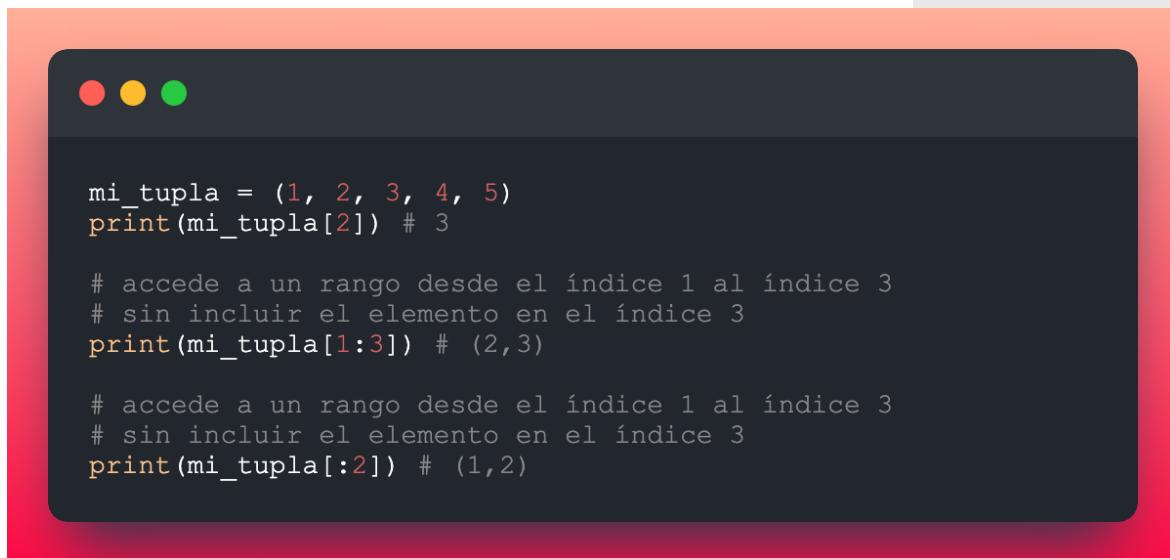
print(tupla)
```

Reto:

Crea una tupla con los nombres de los meses del año y luego imprimelos en orden alfabético.

## 10.2 Acceder a elementos de una tupla

Acceder a elementos de una tupla en Python es muy sencillo. Una tupla es una secuencia de valores separados por comas, entre paréntesis. Los elementos de una tupla se pueden acceder mediante su índice, el cual comienza en 0. Por ejemplo, si tenemos la tupla `mi_tupla = (1, 2, 3, 4, 5)`, para acceder al tercer elemento de la tupla, usaríamos `mi_tupla[2]`, el cual devolvería el valor 3.



```
● ● ●

mi_tupla = (1, 2, 3, 4, 5)
print(mi_tupla[2]) # 3

# accede a un rango desde el índice 1 al índice 3
# sin incluir el elemento en el índice 3
print(mi_tupla[1:3]) # (2,3)

# accede a un rango desde el índice 1 al índice 3
# sin incluir el elemento en el índice 3
print(mi_tupla[:2]) # (1,2)
```

Reto:

Crea una tupla con 5 elementos y accede al último elemento de la tupla.

## 10.3 Índices negativos en una tupla

Los índices negativos en una tupla en Python se usan para acceder a los elementos de la tupla desde el final hacia el principio. Esto significa que el índice -1 se refiere al último elemento de la tupla, el índice -2 se refiere al penúltimo elemento, y así sucesivamente.

Ejemplo:

```
tupla = ("a", "b", "c", "d", "e")
print(tupla[-1]) # imprime "e"
print(tupla[-2]) # imprime "d"
```

Reto:

Crea una tupla con 5 elementos y usa índices negativos para imprimir los 3 últimos elementos.

### 10.4 Inmutabilidad

Inmutabilidad en Python se refiere a la capacidad de un objeto de no cambiar su valor una vez que se ha asignado. Esto significa que una vez que un objeto se ha creado, su valor no puede ser modificado. Esto es útil para evitar errores de lógica en el código, ya que los valores no pueden cambiar sin que el programador lo sepa.

Ejemplo:

```
tupla = (1, 2, 3)
print(tupla) # imprime (1, 2, 3)

tupla[0] = 4 # Esto generará un error, ya que las tuplas son
inmutables
```

## 10 TUPLAS

---

Reto:

Crea una tupla con los números del 1 al 10. Luego, intenta cambiar el valor del número 5 a 15. ¿Qué sucede?

### 10.5 Modificar elementos de una tupla

Las tuplas son un tipo de dato inmutable en Python, lo que significa que una vez creada, no se pueden modificar sus elementos. Sin embargo, hay algunas formas de trabajar con ellas para lograr un resultado similar.

Ejemplo:

```
# Crear una tupla
mi_tupla = (1, 2, 3, 4, 5)

# Crear una lista a partir de la tupla
mi_lista = list(mi_tupla)

# Modificar un elemento de la lista
mi_lista[2] = 10

# Crear una nueva tupla a partir de la lista
mi_tupla_modificada = tuple(mi_lista)

# Imprimir la tupla modificada
print(mi_tupla_modificada) # (1, 2, 10, 4, 5)
```

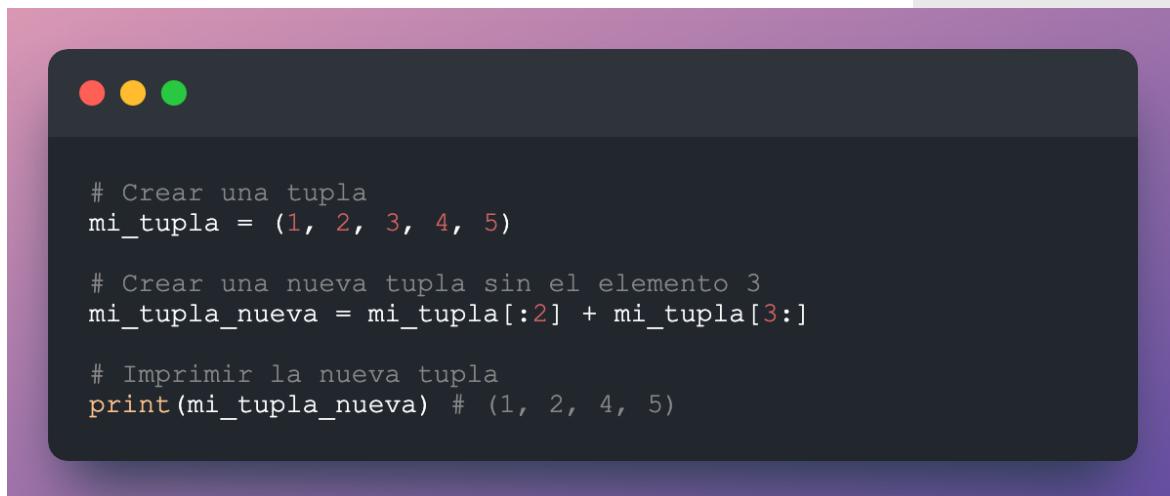
Reto:

Crea una tupla con 5 elementos, luego crea una lista a partir de la tupla, modifica un elemento de la lista y crea una nueva tupla a partir de la lista modificada. Imprime la tupla modificada para verificar el resultado.

## 10.6 Borrar elementos de una tupla

En Python, no es posible eliminar elementos de una tupla. Las tuplas son inmutables, lo que significa que una vez creada, no se pueden modificar. Sin embargo, es posible crear una nueva tupla que no contenga el elemento deseado.

Ejemplo:



```
# Crear una tupla
mi_tupla = (1, 2, 3, 4, 5)

# Crear una nueva tupla sin el elemento 3
mi_tupla_nueva = mi_tupla[:2] + mi_tupla[3:]

# Imprimir la nueva tupla
print(mi_tupla_nueva) # (1, 2, 4, 5)
```

Reto:

Crea una tupla con 5 elementos y luego crea una nueva tupla sin el tercer elemento.

## 10.7 Iterar a través de una tupla

Iterar a través de una tupla en Python es una forma de recorrer los elementos de una tupla. Esto se hace con un bucle for. Un ejemplo práctico y básico sería el siguiente:

```
● ● ●  
mi_tupla = "HTML", "CSS", "JavaScript", "Python"  
  
for elemento in mi_tupla:  
    print(f"Quiero aprender {elemento}!")
```

Reto:

Crea una tupla con los números del 1 al 10 y usa un bucle for para imprimir los números pares.

### 10.8 Métodos de tuplas

#### 10.8.1 Método de tupla count()

count() es una función de Python que se utiliza para contar el número de veces que un elemento específico aparece en una lista, tupla o cadena de caracteres.

Ejemplo:

```
● ● ●  
tupla = (1, 2, 3, 4, 5, 1, 2, 3)  
  
# Contar el número de veces que aparece el número 3  
print(tupla.count(3))  
  
# Salida: 2
```

Reto:

Crea una tupla con números enteros y utiliza la función count() para contar el número de veces que aparece el número 5.

### 10.8.2 Método de tupla index()

index() es una función de Python que devuelve el índice de un elemento específico en una tupla. Por ejemplo, si tenemos una tupla de números podemos usar index() para encontrar el índice de un elemento específico, como el número 3:

```
tupla = (1, 2, 3, 4, 5)  
print(tupla.index(3))
```

Esto devolverá el índice del elemento 3, que es 2.

Reto:

Crea una tupla con 5 elementos y usa index() para encontrar el índice de un elemento específico.

Los métodos que vimos vienen del objeto tuple() que envuelven a las tuplas en Python.

## 11 Sets

### 11.1 Crear un set

Un set o conjunto es una colección desordenada de elementos únicos. En Python, los sets se crean usando la palabra clave `set` o la función `set()`.

Ejemplo:

```
# Crear un set
mi_set = {"manzana", "plátano", "cereza", "cereza"}

# alternativamente se pueden crear a partir del objeto set()
mi_set = set(["manzana", "plátano", "cereza", "cereza"])

# Mostrar el set
print(mi_set) # {'cereza', 'manzana', 'plátano'}
```

Reto:

Crea un set con los números del 1 al 10 y luego imprímelo.

### 11.2 Acceder a elementos de un set

Para acceder a los elementos de un set en Python, se puede usar el método `.pop()` para extraer un elemento al azar del set. Por ejemplo:

```
mi_set = {1, 2, 3, 4, 5}  
elemento_azar = mi_set.pop()  
print(elemento_azar)
```

En este ejemplo, el elemento extraído al azar del set `mi_set` será impreso en pantalla.

Reto:

Crea un set con los números del 1 al 10, y usa el método `.pop()` para extraer un elemento al azar del set.

### 11.3 Modificar elementos de un set

Modificar elementos de un set en Python es una tarea sencilla. Un set es una colección de elementos únicos, por lo que no se pueden agregar elementos duplicados. Para modificar un set, se pueden usar los métodos `add()` y `remove()`.

Ejemplo:

```
# Crear un set
mi_set = {1, 2, 3, 4, 5}

# Agregar un elemento
mi_set.add(6)

# Verificar el set
print(mi_set)

# Salida: {1, 2, 3, 4, 5, 6}

# Remover un elemento
mi_set.remove(4)

# Verificar el set
print(mi_set)

# Salida: {1, 2, 3, 5, 6}
```

Reto:

Crea un set con los números del 1 al 10, luego agrega el número 11 y remueve el número 5.

### 11.4 Borrar elementos de un set

En Python, podemos borrar elementos de un set usando el método `discard()`. Este método elimina un elemento específico del set sin generar un error si el elemento no existe. El método `remove()` lanza un error si el elemento no existe.

Ejemplo:

```
# Declarar un set
mi_set = {1, 2, 3, 4, 5}

# Borrar el elemento 3
mi_set.discard(3)

# Imprimir el set
print(mi_set) # {1, 2, 4, 5}

# Borrar el elemento 3
mi_set.discard(3)

# Borrar el elemento 3
mi_set.remove(3) # error
```

Reto:

Crea un set con los números del 1 al 10, luego borra los números impares del set.

### 11.5 Iterar a través de un set

Iterar a través de un set en Python es una forma de recorrer los elementos de un conjunto. Esto se hace usando un bucle for. Un ejemplo básico sería:



```
codigos = {"EC", "US", "UK"}  
for codigo in codigos:  
    print(codigo)
```

Esto imprimiría cada elemento del conjunto en una línea separada.

Reto:

Crea un set con los números del 1 al 10 y usa un bucle for para imprimir los números pares.

### 11.6 Métodos de sets

#### 11.6.1 Método de set union()

union() es una función de Python que se usa para unir dos conjuntos. Esta función devuelve un nuevo conjunto que contiene los elementos de los dos conjuntos originales.

Ejemplo:

## 11 SETS

---

```
conjunto_1 = {1, 2, 3, 4}
conjunto_2 = {3, 4, 5, 6}

conjunto_unido = conjunto_1.union(conjunto_2)

print(conjunto_unido) # {1, 2, 3, 4, 5, 6}
```

Reto:

Crea dos conjuntos y usa la función union() para unirlos.

### 11.6.2 Método de set intersection()

intersection() es una función de Python que devuelve un conjunto con los elementos comunes entre dos o más conjuntos.

Ejemplo:

```
conjunto_1 = {1, 2, 3, 4, 5}
conjunto_2 = {3, 4, 5, 6, 7}

conjunto_interseccion = conjunto_1.intersection(conjunto_2)

print(conjunto_interseccion) # {3, 4, 5}
```

Reto:

Crea dos conjuntos con al menos 5 elementos cada uno y usa la función intersection() para encontrar los elementos comunes entre los dos conjuntos.

### 11.6.3 Método de set difference()

difference() es una función de Python que devuelve la diferencia entre dos conjuntos. Esta función devuelve un conjunto con los elementos que están en el primer conjunto, pero no en el segundo.

Ejemplo:

```
# Definimos dos conjuntos
conjunto_1 = {1, 2, 3, 4, 5}
conjunto_2 = {3, 4, 5, 6, 7}

# Usamos la función difference()
resultado = conjunto_1.difference(conjunto_2)

# Imprimimos el resultado
print(resultado) # {1, 2}
```

Reto:

Crea dos conjuntos y usa la función difference() para encontrar la diferencia entre ellos.

### 11.6.4 Método de set symmetric\_difference()

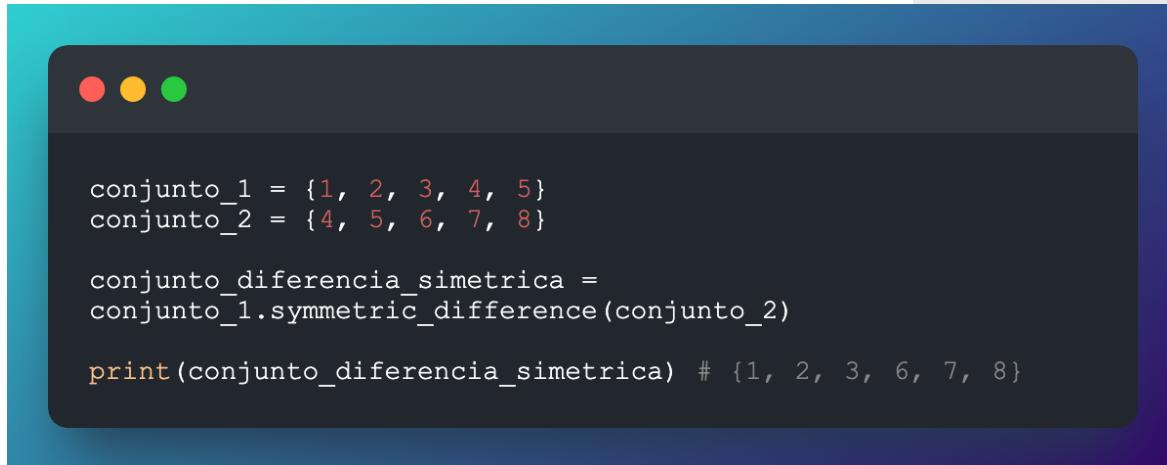
symmetric\_difference() es una función de Python que devuelve la diferencia simétrica entre dos conjuntos. Esto significa que devuelve los elementos que están en uno

## 12 DICCIONARIOS

---

de los conjuntos, pero no en ambos.

Ejemplo:



```
conjunto_1 = {1, 2, 3, 4, 5}
conjunto_2 = {4, 5, 6, 7, 8}

conjunto_diferencia_simetrica =
conjunto_1.symmetric_difference(conjunto_2)

print(conjunto_diferencia_simetrica) # {1, 2, 3, 6, 7, 8}
```

Reto:

Crea dos conjuntos y usa la función `symmetric_difference()` para encontrar la diferencia simétrica entre ellos.

Los métodos que vimos vienen del objeto `set()` que envuelve a los conjuntos en Python.

# 12 Diccionarios

## 12.1 Crear un diccionario

Crear un diccionario en Python es una tarea sencilla. Un diccionario es una colección de pares clave-valor, donde cada clave es única y se asocia con un valor.

Ejemplo:

## 12 DICCCIONARIOS

---



A screenshot of a dark-themed code editor window. At the top, there are three small colored circles (red, yellow, green) followed by a menu bar. The main area contains the following Python code:

```
mi_diccionario = {  
    "nombre": "Juan",  
    "edad": 25,  
    "ciudad": "Madrid"  
}
```

Reto:

Crea un diccionario con tres claves y valores diferentes.

### 12.2 Acceder a elementos de un diccionario

Acceder a elementos de un diccionario en Python es una tarea sencilla. Un diccionario es una estructura de datos que contiene claves y valores. Para acceder a un elemento específico, se debe usar la clave correspondiente.

Ejemplo:

## 12 DICCCIONARIOS

---

```
diccionario = {  
    "clave1": "valor1",  
    "clave2": "valor2",  
    "clave3": "valor3"  
}  
  
# Acceder al valor de la clave2  
valor2 = diccionario["clave2"]  
  
print(valor2) # valor2
```

Reto:

Crea un diccionario con 5 claves y valores, luego accede a uno de los valores usando la clave correspondiente.

### 12.3 Modificar elementos de un diccionario

Modificar elementos de un diccionario en Python es una tarea sencilla. Para modificar un elemento de un diccionario, simplemente asignamos un nuevo valor a la clave correspondiente. Por ejemplo, si tenemos un diccionario llamado “mi\_diccionario” con claves “clave1” y “clave2”, podemos modificar el valor de “clave1” de la siguiente manera:

## 12 DICCCIONARIOS

---

```
mi_diccionario["clave1"] = "nuevo valor"
```

Reto:

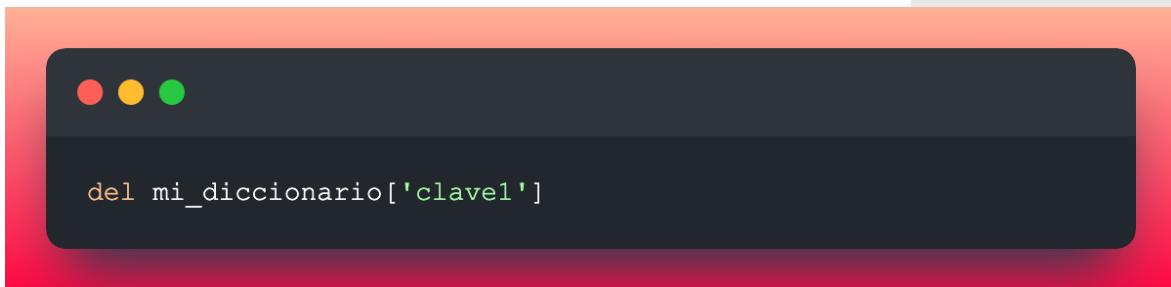
Crea un diccionario con tres claves y modifica el valor de una de ellas.

### 12.4 Borrar elementos de un diccionario

En Python, podemos borrar elementos de un diccionario usando la función `pop()`. Esta función toma una clave como argumento y elimina el elemento asociado a esa clave del diccionario. Por ejemplo, si tenemos un diccionario llamado `mi_diccionario` con claves 'a', 'b' y 'c' y valores 1, 2 y 3, respectivamente, podemos borrar el elemento con clave 'b' usando la siguiente línea de código:

```
mi_diccionario.pop('b')
```

También podemos borrar elementos de un objeto usando la función `del`. Esta función elimina un elemento de un objeto dado su nombre. Por ejemplo, si tenemos un diccionario llamado `mi_diccionario` con los elementos `clave1` y `clave2`, podemos borrar `clave1` de la siguiente manera:



del elimina uno o varios elementos por índice, pop() lo elimina por índice y también devuelve el valor.

Reto:

Crea un diccionario con al menos 5 elementos y borra uno de ellos usando la función pop().

### 12.5 Métodos de diccionario

#### 12.5.1 Método de diccionario get()

get() es un método de diccionario en Python que devuelve el valor de una clave específica. Si la clave no existe, devuelve un valor predeterminado.

Ejemplo:

## 12 DICCIONARIOS

---

```
● ● ●
```

```
diccionario = {'nombre': 'Juan', 'edad': 25}

# Devuelve el valor de la clave 'nombre'
valor = diccionario.get('nombre')
print(valor) # Juan

# Devuelve el valor predeterminado si la clave no existe
valor = diccionario.get('apellido', 'No existe')
print(valor) # No existe
```

Reto:

Crea un diccionario con al menos 3 claves y usa el método get() para obtener los valores de cada clave.

### 12.5.2 Método de diccionario update()

Update() es una función de Python que se utiliza para actualizar los valores de un diccionario. Esta función toma dos argumentos: un diccionario y una secuencia de pares clave-valor. Esta función actualiza el diccionario con los pares clave-valor especificados.

Ejemplo:

## 12 DICCCIONARIOS

---



```
diccionario = {'a':1, 'b':2, 'c':3}
diccionario.update({'b':4, 'd':5})
print(diccionario) # {'a': 1, 'b': 4, 'c': 3, 'd': 5}
```

Reto:

Crea un diccionario con algunos elementos y luego actualízalo con update() para agregar más elementos.

### 12.5.3 Método de diccionario keys()

keys() es un método en Python que devuelve una lista de las claves de un diccionario. Por ejemplo, si tenemos un diccionario llamado “mi\_diccionario” con claves “nombre” , “edad” y “ciudad” , podemos usar el método keys() para obtener una lista de estas claves:

## 12 DICCCIONARIOS

---

```
● ● ●  
mi_diccionario = {  
    "nombre": "Juan",  
    "edad": 25,  
    "ciudad": "Madrid"  
}  
  
claves = mi_diccionario.keys()  
  
print(claves)  
  
# Esto imprimirá: dict_keys(['nombre', 'edad', 'ciudad'])
```

Reto:

Crea un diccionario con al menos 3 claves y usa el método keys() para imprimir una lista de las claves.

### 12.5.4 Método de diccionario values()

values() es un método de diccionario en Python que devuelve una lista de los valores del diccionario.

Ejemplo:

## 12 DICCCIONARIOS

---

```
diccionario = {  
    "clave1": "valor1",  
    "clave2": "valor2",  
    "clave3": "valor3"  
}  
  
valores = diccionario.values()  
  
print(valores) # dict_values(['valor1', 'valor2', 'valor3'])
```

Reto:

Crea un diccionario con 5 claves y 5 valores, luego imprime los valores usando el método values().

### 12.5.5 Método de diccionario items()

items() es un método de diccionario en Python que devuelve una lista de tuplas, donde cada tupla contiene una clave y su valor correspondiente.

Ejemplo:

## 12 DICCCIONARIOS

---

```
diccionario = {  
    "nombre": "Juan",  
    "edad": 25,  
    "ciudad": "Madrid"  
}  
  
valores = diccionario.items()  
  
print(valores)  
  
# Salida:  
# dict_items([('nombre', 'Juan'), ('edad', 25), ('ciudad',  
'Madrid'))]
```

Reto:

Crea un diccionario con al menos 3 claves y sus valores correspondientes, luego imprime los valores usando el método items().

### 12.5.6 Iterar a través de un diccionario

Iterar a través de un diccionario en Python es una forma de recorrer todos los elementos de un diccionario. Esto se puede hacer usando el método items() para obtener una lista de tuplas de clave-valor. Por ejemplo:

## 12 DICCCIONARIOS

---

```
diccionario = {  
    "clave1": "valor1",  
    "clave2": "valor2",  
    "clave3": "valor3"  
}  
  
for clave, valor in diccionario.items():  
    print(clave, valor)  
  
"""  
Esto imprimirá:  
clave1 valor1  
clave2 valor2  
clave3 valor3  
"""
```

Reto:

Crea un diccionario con al menos 5 elementos y usa el método items() para iterar a través de él.

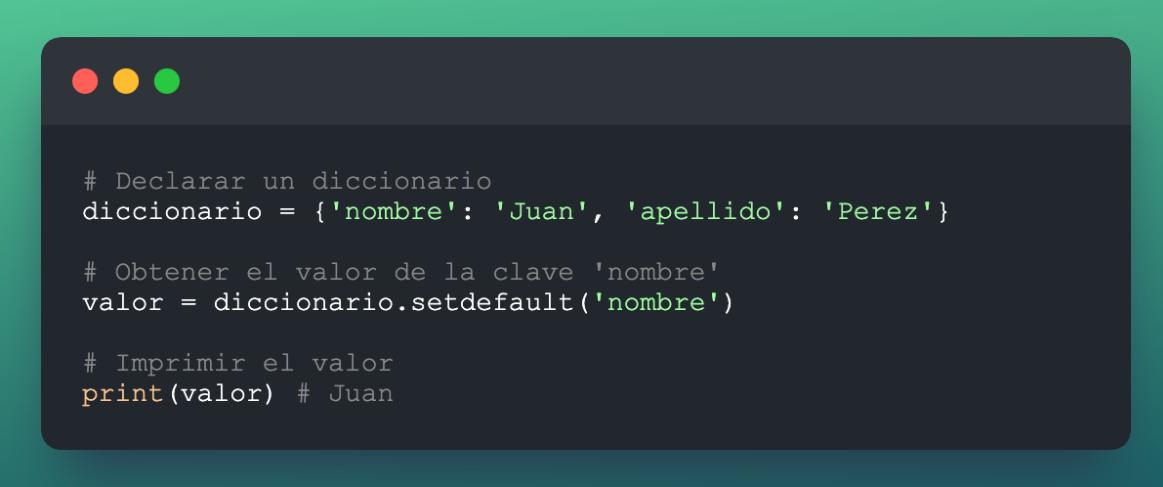
### 12.5.7 Método de diccionario setdefault()

setdefault() es un método de diccionario en Python que devuelve el valor de una clave específica. Si la clave no existe, el método crea la clave con el valor predeterminado especificado.

Ejemplo:

## 12 DICCIONARIOS

---



```
# Declarar un diccionario
diccionario = {'nombre': 'Juan', 'apellido': 'Perez'}

# Obtener el valor de la clave 'nombre'
valor = diccionario.setdefault('nombre')

# Imprimir el valor
print(valor) # Juan
```

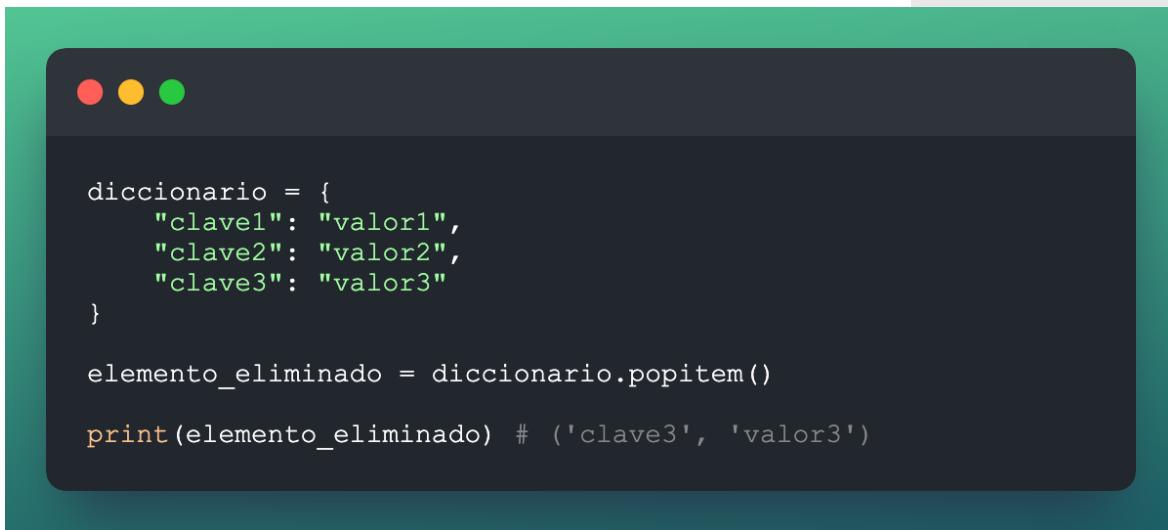
Reto:

Crea un diccionario con algunas claves y valores, luego usa `setdefault()` para obtener el valor de una clave específica.

### 12.5.8 Método de diccionario `popitem()`

`popitem()` es un método de diccionario en Python que elimina y devuelve un elemento aleatorio del diccionario.

Ejemplo:



```
diccionario = {
    "clave1": "valor1",
    "clave2": "valor2",
    "clave3": "valor3"
}

elemento_eliminado = diccionario.popitem()

print(elemento_eliminado) # ('clave3', 'valor3')
```

Reto:

Crea un diccionario con 5 elementos y usa popitem() para eliminar y devolver un elemento aleatorio.

Los métodos que vimos vienen del objeto dict() que envuelve a los diccionarios en Python.

# 13 Programación orientada a objetos (POO)

## 13.1 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. Python es un lenguaje de programación orientado a objetos, lo que significa que los programadores pueden crear objetos y usarlos para construir aplicaciones.

En Python, todo es un objeto debido a su naturaleza orientada a objetos. El paradig-

## 13 PROGRAMACIÓN ORIENTADA A OBJETOS (POO)

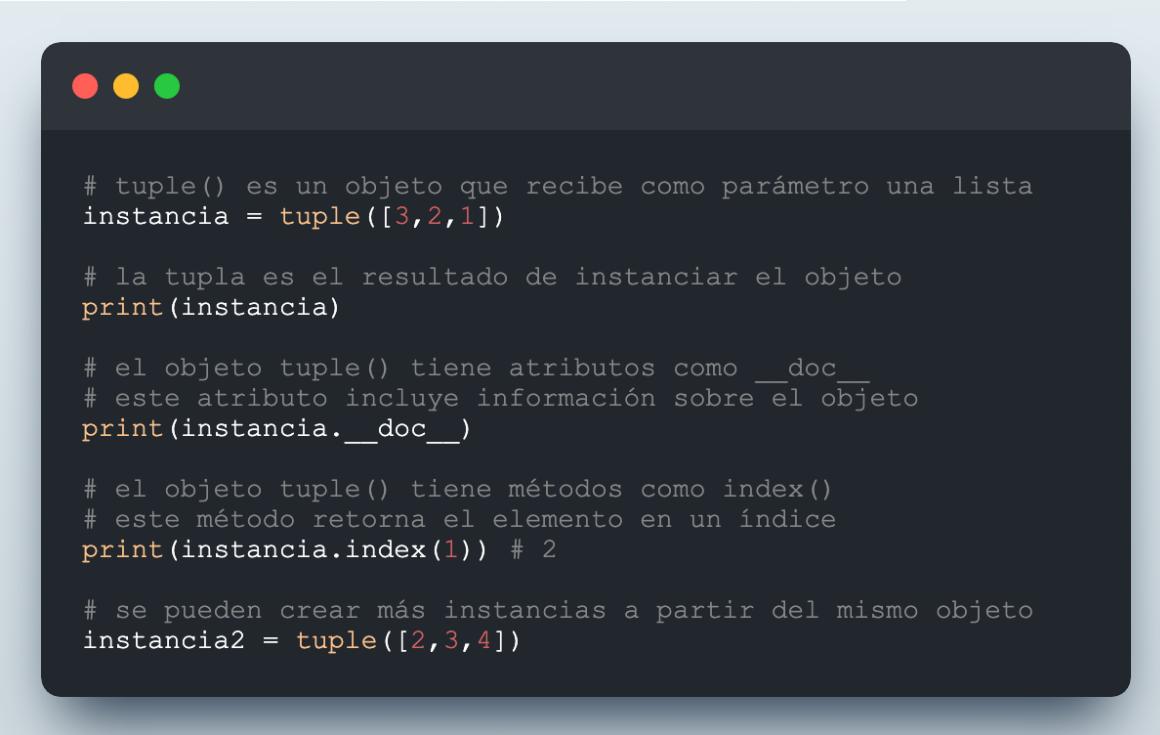
---

ma de programación orientada a objetos se basa en la idea de modelar entidades del mundo real como objetos con atributos y comportamientos asociados. En Python, incluso tipos de datos primitivos como números, cadenas y listas son objetos.

### 13.2 Objetos

#### 13.2.1 Crear un objeto

Un objeto en Python es una entidad que contiene variables y funciones. Estos variables y funciones se conocen como atributos y métodos, respectivamente. Un ejemplo básico de un objeto en Python es el siguiente:



```
# tuple() es un objeto que recibe como parámetro una lista
instancia = tuple([3,2,1])

# la tupla es el resultado de instanciar el objeto
print(instancia)

# el objeto tuple() tiene atributos como __doc__
# este atributo incluye información sobre el objeto
print(instancia.__doc__)

# el objeto tuple() tiene métodos como index()
# este método retorna el elemento en un índice
print(instancia.index(1)) # 2

# se pueden crear más instancias a partir del mismo objeto
instancia2 = tuple([2,3,4])
```

No se pueden alterar los atributos y métodos de los objetos que vienen en Python porque podría causar errores difíciles de depurar.

## 13 PROGRAMACIÓN ORIENTADA A OBJETOS (POO)

---

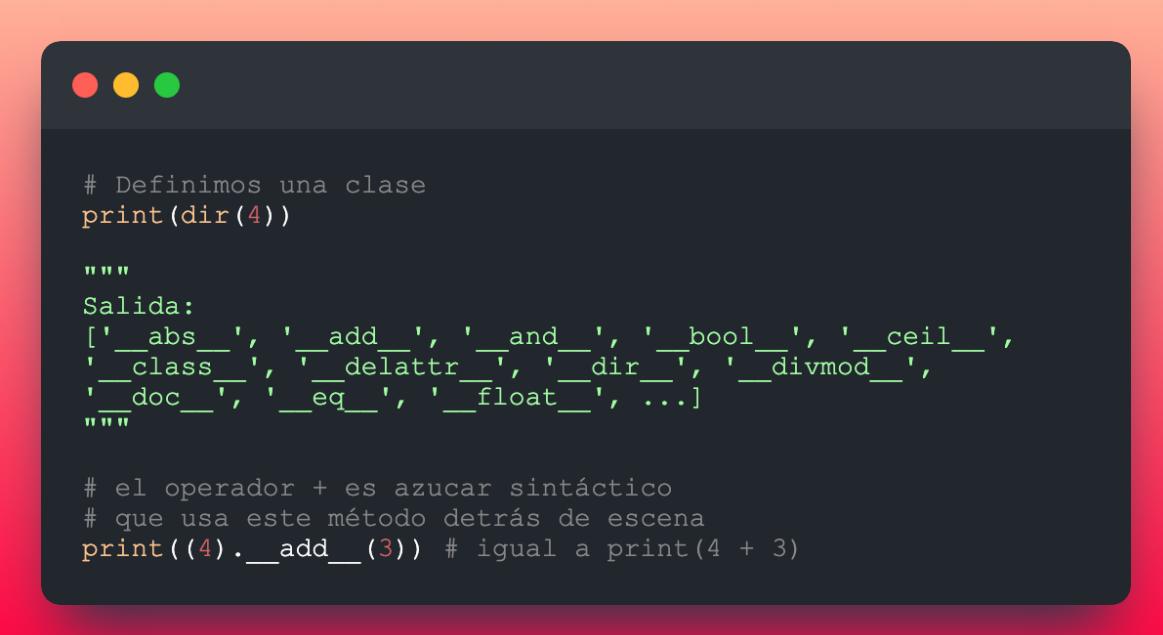
Reto:

Recrea este ejemplo con el objeto de Python set()

### 13.2.2 dir()

dir() es una función incorporada en Python que devuelve una lista de los nombres de los atributos y métodos de un objeto. Esta función es útil para explorar los atributos y métodos de un objeto, así como para obtener información sobre el objeto.

Ejemplo:



```
# Definimos una clase
print(dir(4))

"""
Salida:
['__abs__', '__add__', '__and__', '__bool__', '__ceil__',
'__class__', '__delattr__', '__dir__', '__divmod__',
'__doc__', '__eq__', '__float__', ...]

# el operador + es azucar sintáctico
# que usa este método detrás de escena
print((4).__add__(3)) # igual a print(4 + 3)
```

Reto:

Crea una función. Usa la función dir() para imprimir los atributos y métodos de la función.

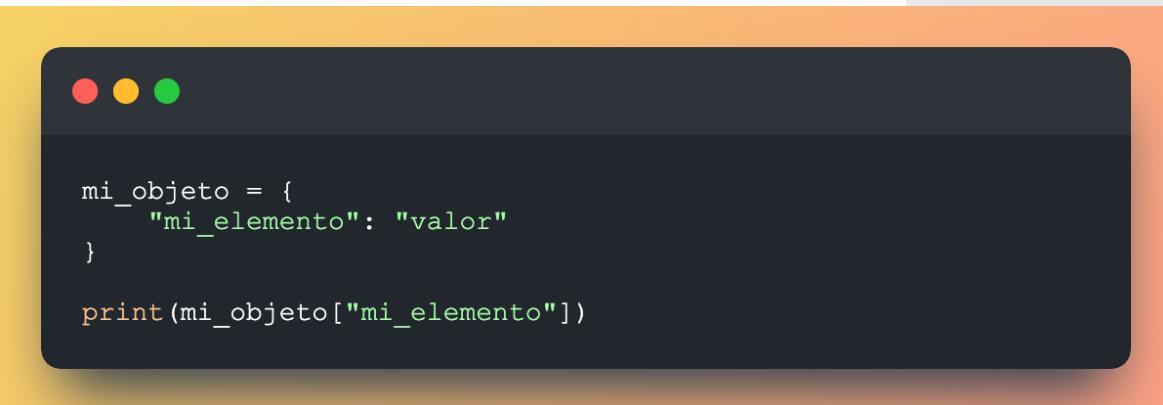
## 13 PROGRAMACIÓN ORIENTADA A OBJETOS (POO)

---

### 13.2.3 Acceder a elementos de un objeto

Para acceder a los elementos de un objeto en Python, se utiliza la notación []. Esto significa que para acceder a un elemento de un objeto, se escribe el nombre del objeto seguido de [] y dentro el nombre del elemento en texto.

Ejemplo:



```
mi_objeto = {  
    "mi_elemento": "valor"  
}  
  
print(mi_objeto["mi_elemento"])
```

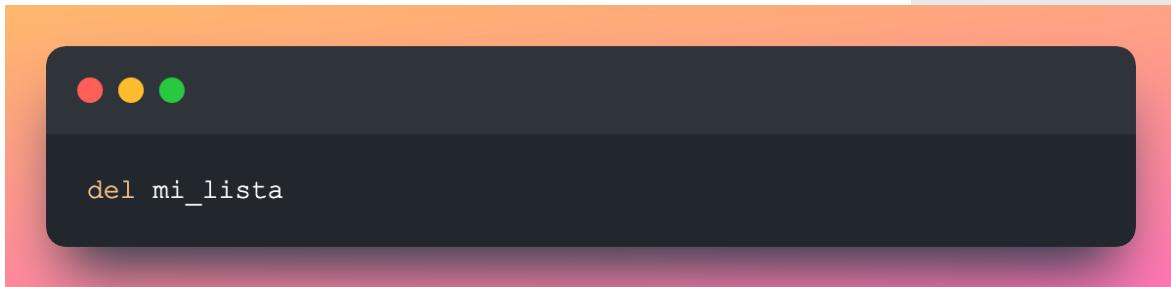
No se pueden alterar los atributos de los objetos que vienen en Python porque podría causar errores difíciles de depurar.

Reto:

Crea un objeto con dos elementos y accede a ellos.

### 13.2.4 Borrar objetos

Borrar objetos en Python es una tarea sencilla. Para borrar un objeto, simplemente usamos la palabra clave del. Por ejemplo, si queremos borrar una lista llamada mi\_lista, podemos hacerlo así:



Reto:

Cree una lista llamada `mi_lista` con algunos elementos y luego bórrela usando la palabra clave `del`.

### 13.3 Clases

#### 13.3.1 Crear una clase

Una clase en Python es una estructura de datos que contiene variables y funciones. Estas variables y funciones se conocen como atributos y métodos, respectivamente. Una clase es una plantilla para crear objetos.

Ejemplo:

## 13 PROGRAMACIÓN ORIENTADA A OBJETOS (POO)

---

```
● ● ●
```

```
class Persona:
    def __init__(self, nombre, edad):
        self.nombre = nombre
        self.edad = edad

    def saludar(self):
        print(f"Mi nombre es {self.nombre} y tengo {self.edad} años.")

persona = Persona("Juan", 20)
persona.saludar() # Mi nombre es Juan y tengo 20 años.

persona2 = Persona("José", 18)
persona2.saludar() # Mi nombre es José y tengo 18 años.
```

Este código define una clase llamada “Persona” que tiene dos atributos: “nombre” y “edad” . La clase también tiene un método llamado “saludar” que imprime un mensaje de saludo junto con el nombre de la persona.

En la parte inferior del código, se crea una instancia de la clase Persona llamada “persona” y se le asigna el nombre “Juan” y la edad “20” . Luego, se llama al método “saludar” de la instancia “persona” , lo que imprimirá el mensaje “Hola, mi nombre es Juan” .

Reto:

Crea una clase llamada “Libro” que tenga dos atributos: título y autor. Agrega un método llamado “mostrar\_informacion” que imprima el título y el autor del libro.

## 13 PROGRAMACIÓN ORIENTADA A OBJETOS (POO)

---

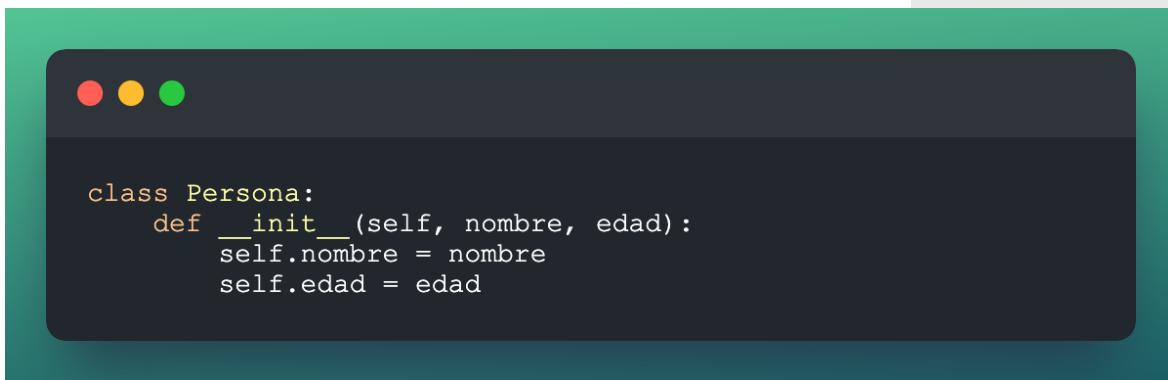
### 13.3.2 Función constructora

Una función constructora en Python es una función que crea un objeto y lo devuelve al usuario. Esta función se utiliza para inicializar un objeto con los valores predeterminados.

Para crear una función constructora en Python, es necesario seguir los siguientes requisitos:

1. La función debe tener el nombre `__init__`.
2. La función debe tener como primer parámetro el objeto `self`, que hace referencia a la instancia de la clase que se está creando.
3. La función puede tener otros parámetros opcionales para inicializar los atributos de la clase.
4. Dentro de la función, se deben asignar los valores de los parámetros a los atributos correspondientes de la instancia utilizando la sintaxis `self.nombre_atributo = valor`.

Por ejemplo, la siguiente función constructora crea una clase llamada “Persona” con dos atributos, “nombre” y “edad” :



```
class Persona:  
    def __init__(self, nombre, edad):  
        self.nombre = nombre  
        self.edad = edad
```

En este caso, la función constructora tiene dos parámetros (`nombre` y `edad`) además del parámetro `self`. Dentro de la función, se asignan los valores de los parámetros a los atributos correspondientes de la instancia (`self.nombre` y `self.edad`).

## 13 PROGRAMACIÓN ORIENTADA A OBJETOS (POO)

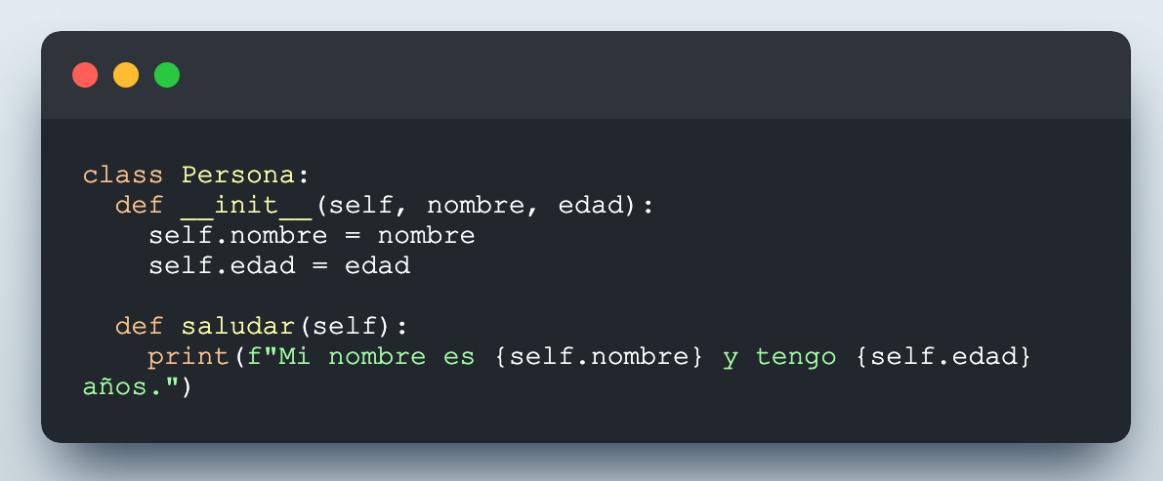
---

Reto:

Crea una función constructora en Python que cree un objeto de tipo “Libro” con los atributos “título” , “autor” y “año de publicación” .

### 13.3.3 El parámetro self

El parámetro self en Python es una referencia a la instancia actual de un objeto. Se usa para acceder a los atributos y métodos de la clase. Por ejemplo, si tenemos una clase llamada “Persona” , podemos definir un método llamado “saludar” que imprima un saludo a la persona:



```
● ● ●

class Persona:
    def __init__(self, nombre, edad):
        self.nombre = nombre
        self.edad = edad

    def saludar(self):
        print(f"Mi nombre es {self.nombre} y tengo {self.edad} años.")
```

En este ejemplo, el parámetro self se refiere a la instancia de la clase Persona, que es la variable “persona” .

Reto:

Crea una clase llamada “Cuenta” que tenga un atributo llamado “saldo” y un método llamado “depositar” que aumente el saldo en una cantidad dada. Luego, crea una instancia de la clase y usa el método para depositar una cantidad dada.

## 13 PROGRAMACIÓN ORIENTADA A OBJETOS (POO)

---

### 13.3.4 Herencia y polimorfismo

La Herencia en Python es un mecanismo que permite a una clase heredar los atributos y métodos de otra clase. Esto significa que una clase puede reutilizar el código de otra clase, simplificando el proceso de desarrollo. Para esto incluimos pasamos una clase a la definición de otra clase como parámetro, llamamos a la función super() que es la clase de que heredamos, y podemos pasar valores a su constructor.

Ejemplo:



```
● ● ●

class Animal:
    def __init__(self, nombre):
        self.nombre = nombre

    def hablar(self):
        print(f"Hola, soy un animal llamado {self.nombre}")

class Perro(Animal):
    def __init__(self, nombre):
        super().__init__(nombre)

fido = Perro("Fido")
fido.hablar()
```

El Polimorfismo en Python es una característica que permite a una clase comportarse de manera diferente dependiendo de la situación. Esto significa que una clase puede definir métodos con el mismo nombre, pero con diferentes parámetros.

## 13 PROGRAMACIÓN ORIENTADA A OBJETOS (POO)

---

```
class Animal:  
    def __init__(self, nombre):  
        self.nombre = nombre  
  
    def hablar(self):  
        print(f"Hola, soy un animal llamado {self.nombre}")  
  
class Perro(Animal):  
    def __init__(self, nombre):  
        super().__init__(nombre)  
  
    def hablar(self):  
        print("Guau, soy un perro y me llamo", self.nombre)  
  
fido = Perro("Fido")  
fido.hablar()
```

Reto:

Crea una clase Gato que herede de la clase Animal y que tenga un método hablar() que imprima “Miau, soy un gato y me llamo [nombre]” .

### 13.3.5 Encapsulamiento y abstracción

Encapsulamiento y abstracción son dos conceptos fundamentales en la programación orientada a objetos. El encapsulamiento se refiere a la capacidad de ocultar los detalles de implementación de un objeto, mientras que la abstracción se refiere a la capacidad de representar los objetos de una manera simplificada. En Python, esto se logra mediante la creación de clases.

Reto:

## 14 MÓDULOS Y PAQUETES

---

Crea una clase llamada “Coche” que tenga los atributos “marca” y “modelo”. Agrega un método llamado “mostrar\_datos” que imprima los datos del coche. Crea un objeto de la clase “Coche” y llama al método “mostrar\_datos” para verificar que funciona correctamente.

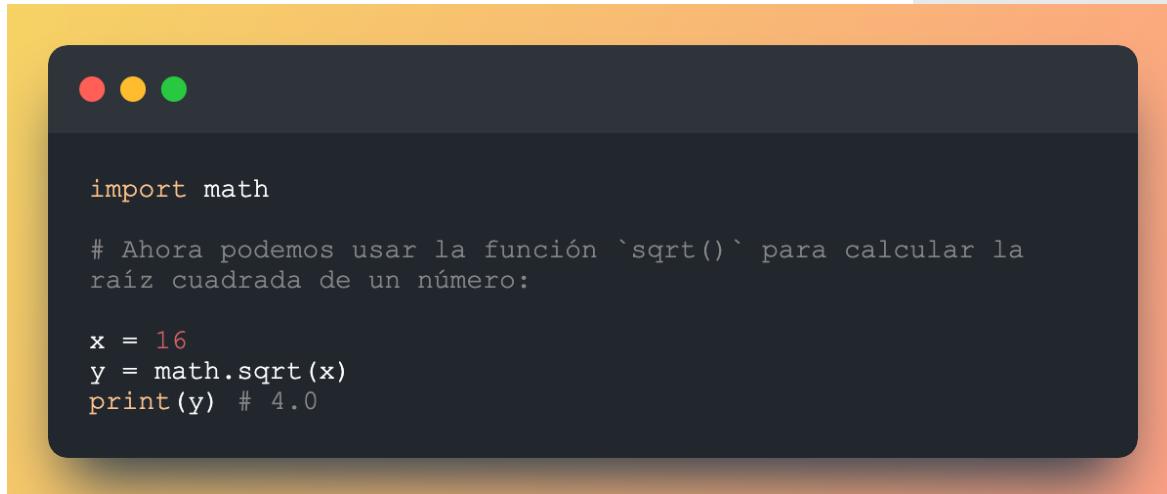
# 14 Módulos y paquetes

## 14.1 Importación y uso de módulos externos

La importación y uso de módulos externos en Python es una forma de extender la funcionalidad de un programa. Esto se logra mediante la importación de módulos externos, que son archivos de código Python que contienen funciones y clases adicionales. Estos módulos se pueden descargar desde la web o instalar desde una línea de comandos.

Ejemplo:

Supongamos que queremos usar la función `sqrt()` de la biblioteca estándar de Python para calcular la raíz cuadrada de un número. Primero, importamos el módulo `math`:



```
import math

# Ahora podemos usar la función `sqrt()` para calcular la
# raíz cuadrada de un número:

x = 16
y = math.sqrt(x)
print(y) # 4.0
```

## 14 MÓDULOS Y PAQUETES

---

Reto:

Escribe un programa que use la función pow() del módulo math para calcular el cuadrado de un número.

### 14.2 Importar como alias

Importar como alias en Python es una forma de importar un módulo o paquete en Python usando un alias en lugar de su nombre completo. Esto puede ser útil para evitar conflictos de nombres entre módulos o para hacer que el código sea más legible.

Ejemplo:

```
import math as m  
print(m.sqrt(16))
```

Reto:

Escribe un programa que importe el módulo math como mt y usa la función mt.sqrt() para calcular la raíz cuadrada de un número dado.

### 14.3 from...import

From...import es una sintaxis en Python que permite importar una o más definiciones de un módulo a un programa. Esto significa que puedes importar una función específica de un módulo, en lugar de tener que importar todo el módulo.

## 14 MÓDULOS Y PAQUETES

---

Ejemplo:

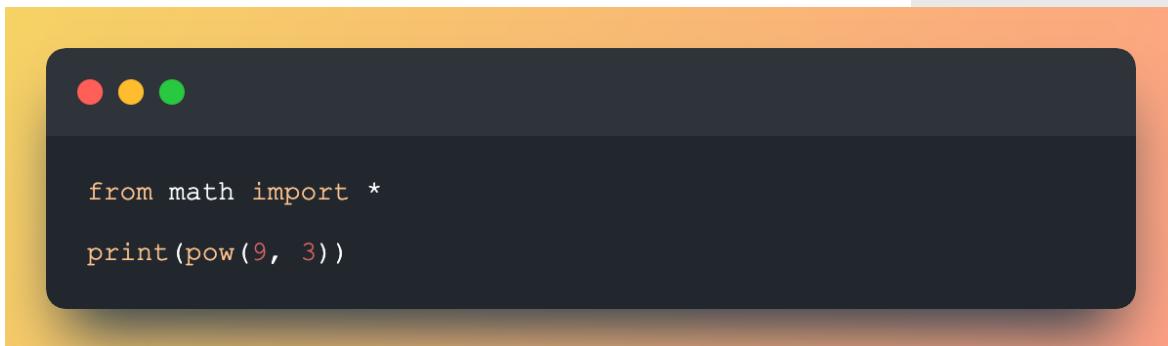
```
from math import sqrt  
  
num = 25  
  
# Calcular la raíz cuadrada de 25  
raiz_cuadrada = sqrt(num)  
  
print(raiz_cuadrada) # 5.0
```

Reto:

Escribe un programa que use from...import para importar la función pow() del módulo math y calcule el resultado de 2 elevado a la potencia de 10.

### 14.4 Importar todo (\*)

Importar todo (\*) en Python significa importar todos los módulos, clases y funciones de un paquete. Esto se hace con la sintaxis `from paquete import *`. Por ejemplo, si queremos importar todos los módulos de la biblioteca estándar de Python, podemos usar el siguiente código:



Esta sintaxis importa todos los nombres definidos en el paquete o módulo y los hace accesibles sin la notación de punto. Por ejemplo, si importa todas las funciones y variables de math, puede acceder a la función sqrt() simplemente escribiendo sqrt(). Sin embargo, esta sintaxis no es muy recomendada, ya que puede llevar a conflictos de nombres y dificultades para mantener la claridad del código.

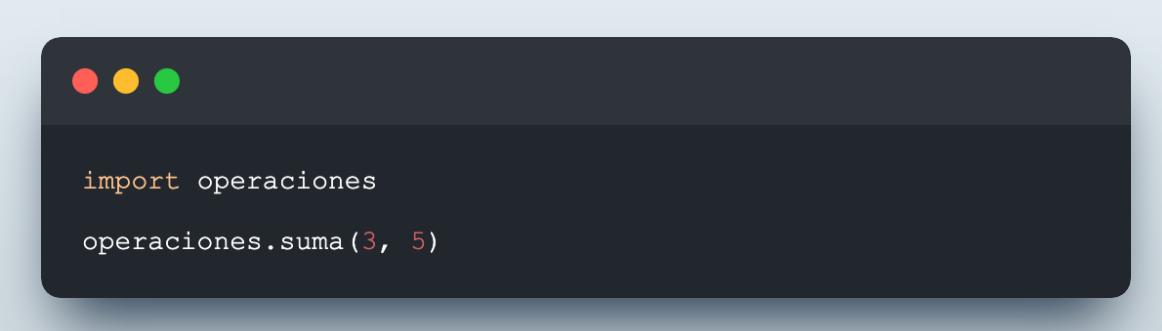
Reto:

Crea un programa que use la función random() del módulo random para generar un número aleatorio.

### 14.5 Exportar módulos

Primero, tenemos que entender que cuando escribimos código, es probable que queramos reutilizar ese código en diferentes partes de nuestro programa o incluso en diferentes programas. Una forma de hacer esto es crear un módulo de Python. En este caso, un módulo de Python es simplemente un archivo con extensión “.py” que contiene código que puede ser reutilizado.

Ahora bien, ¿cómo exportamos un módulo? Imagina que tienes un archivo llamado “operaciones.py” que contiene varias funciones que quieras reutilizar en diferentes programas. Para exportar este módulo simplemente tenemos que tener este archivo a lado del archivo que lo va a utilizar y usamos la palabra clave “import” de la siguiente manera:



```
import operaciones
operaciones.suma(3, 5)
```

Ahora para el reto simple. Crea un módulo de Python llamado “saludos.py” que contenga una función llamada “saludo” que toma un nombre como argumento y devuelve un saludo personalizado para ese nombre. Importa este módulo en un programa de Python diferente y utiliza la función “saludo” para dar la bienvenida a un usuario. ¿Fácil verdad? ¡Manos a la obra!

### 14.6 Creación y uso de paquetes (`__init__.py`)

Los paquetes en Python son una forma de organizar los módulos de una aplicación. Esto se logra mediante la creación de un archivo `init.py` en el directorio raíz del paquete. El archivo `init.py` se usa para inicializar el paquete y puede contener código de inicialización, variables, funciones y clases.

Ejemplo:

Supongamos que tenemos una aplicación que contiene tres módulos: `modulo1.py`, `modulo2.py` y `modulo3.py`. Para organizar estos módulos en un paquete, primero creamos un directorio llamado “`mi_paquete`” y luego creamos un archivo vacío `init.py` en el directorio. A continuación, movemos los módulos a este directorio. El directorio “`mi_paquete`” ahora contiene los tres módulos y el archivo `init.py` y lo podemos usar así:

```
# los archivos están ubicados así:  
# directorio/mi_paquete/__init__.py  
# directorio/mi_paquete/modulo.py  
# directorio/este_código  
  
import mi_paquete.modulo  
  
# o  
from mi_paquete import modulo  
  
# o  
import mi_paquete
```

Reto:

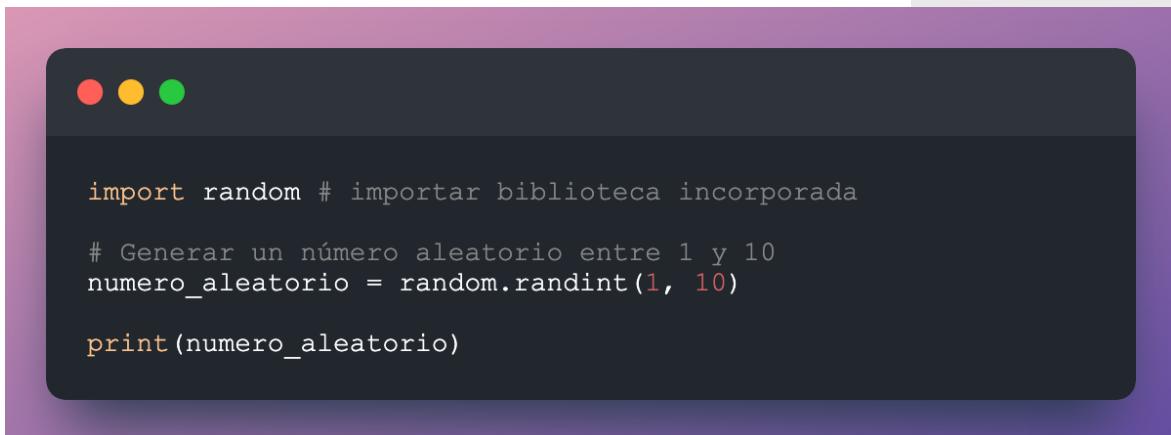
Crea un paquete llamado “mi\_paquete” con un archivo `init.py` y dos módulos llamados `modulo1.py` y `modulo2.py`. Luego, escribe un programa que importe los módulos del paquete y los use para realizar una tarea simple.

### 14.7 Módulos comunes

#### 14.7.1 Módulo random

El módulo `random` de Python es una biblioteca de funciones que nos permite generar números aleatorios. Esto es útil para crear programas que necesiten tomar decisiones aleatorias, como juegos de azar, simuladores, etc.

Ejemplo:



```
import random # importar biblioteca incorporada

# Generar un número aleatorio entre 1 y 10
numero_aleatorio = random.randint(1, 10)

print(numero_aleatorio)
```

Reto:

Crea un programa que genere un número aleatorio entre 1 y 100 y pregunte al usuario si el número generado es mayor o menor que 50. Si el usuario acierta, el programa debe imprimir un mensaje de felicitación. Si el usuario se equivoca, el programa debe imprimir un mensaje de error.

### 14.7.2 Módulo math

El módulo math de Python es una biblioteca de funciones matemáticas que nos permite realizar operaciones matemáticas básicas y avanzadas. Algunas de las funciones más comunes incluyen ceil(), floor(), log(), pow(), sqrt(), sin(), cos(), tan(), entre otras.

Ejemplo:

```
import math

# Calcular el valor absoluto de -5
abs_val = math.abs(-5)
print(abs_val) # 5
```

Reto:

Calcular el valor de la raíz cuadrada de 25 utilizando el módulo math y el método sqrt().

### 14.7.3 Módulo datetime (Fechas)

Python ofrece una variedad de herramientas para trabajar con fechas. Estas herramientas se encuentran en el módulo datetime. Por ejemplo, para obtener la fecha actual, se puede usar el método datetime.now():

```
import datetime

fecha_actual = datetime.datetime.now()
print(fecha_actual)
```

Salida:

2020-09-17 11:45:20.845862

## 14 MÓDULOS Y PAQUETES

---

Reto:

Crea un programa que imprima la fecha de mañana.

### 14.7.4 Módulo re (Expresiones regulares)

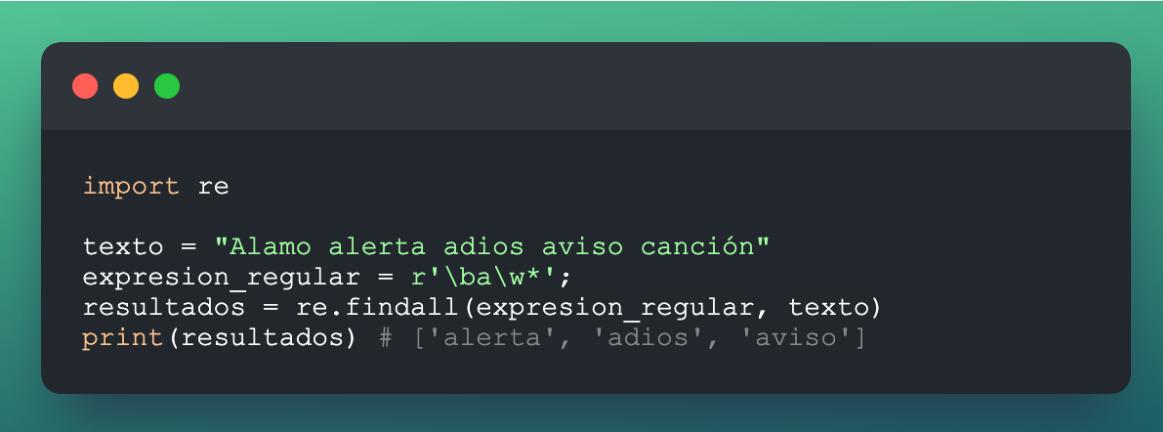
Las expresiones regulares en Python son una herramienta útil para encontrar patrones en cadenas de texto. Estas expresiones se escriben con una sintaxis especial que permite buscar y reemplazar patrones en una cadena de texto.

Ejemplo:

Supongamos que queremos encontrar todas las palabras que comienzan con la letra “a” en una cadena de texto. Podemos usar la siguiente expresión regular para hacer esto:

r'\ba\w\*'

Para realizar esta búsqueda necesitamos importar el módulo de expresiones regulares.



```
import re

texto = "Alamo alerta adios aviso canción"
expresion_regular = r'\ba\w*';
resultados = re.findall(expresion_regular, texto)
print(resultados) # ['alerta', 'adios', 'aviso']
```

Esta expresión regular buscará todas las palabras que comienzan con la letra “a” en una cadena de texto. Las expresiones regulares son un tema fuera del alcance de este libro y requieren su propio libro. Para aprender más puedes visitar este enlace: <https://docs.python.org/3/library/re.html>

Reto:

Escribe una expresión regular para encontrar todas las palabras que comienzan con la letra “b” y terminan con la letra “y” en una cadena de texto.

## 15 Manejo de archivos

### 15.1 Lectura de archivos

La lectura de archivos en Python es una tarea común que se realiza para procesar los datos contenidos en un archivo. Esto se puede hacer con la función `open()` de Python. Esta función devuelve un objeto de archivo que se puede usar para leer el contenido del archivo.

Ejemplo:

```
# Abrir el archivo
archivo = open("archivo.txt", "r")

# Leer el contenido del archivo
contenido = archivo.read()

# Imprimir el contenido
print(contenido)

# Cerrar el archivo
archivo.close()
```

Reto:

## 15 MANEJO DE ARCHIVOS

---

Escribe un programa que lea un archivo de texto y cuente el número de líneas que contiene.

### 15.2 with...open

with...open es una sentencia de control de contexto en Python que se usa para abrir un archivo, realizar operaciones sobre él y cerrarlo automáticamente. Esto significa que no es necesario cerrar el archivo manualmente.

Ejemplo:

```
with open('archivo.txt', 'r') as f:  
    contenido = f.read()  
    print(contenido)
```

Reto:

Escribe un programa que abra un archivo de texto, lea su contenido y luego escriba una línea al final del archivo.

### 15.3 Crear y modificar archivos

Python ofrece una variedad de métodos para crear y modificar archivos. Un ejemplo básico es usar la función open() para crear un archivo y escribir contenido en él.

## 15 MANEJO DE ARCHIVOS

---

```
# Abrir un archivo para escribir
f = open("archivo.txt", "w")

# Escribir contenido en el archivo
f.write("Hola, este es un archivo de prueba")

# Cerrar el archivo
f.close()
```

También se pueden usar métodos como `read()` y `write()` para leer y modificar el contenido de un archivo existente.

```
# Abrir un archivo para leer
f = open("archivo.txt", "r")

# Leer el contenido del archivo
contenido = f.read()

# Modificar el contenido
contenido_modificado = contenido + "\nAdiós!"

# Escribir el contenido modificado
f = open("archivo.txt", "w")
f.write(contenido_modificado)

# Cerrar el archivo
f.close()
```

Reto:

## 15 MANEJO DE ARCHIVOS

---

Crea un programa que lea un archivo de texto y cuente el número de palabras en él.

### 15.4 Borrar archivos

Borrar archivos en Python es una tarea sencilla. Para borrar un archivo, primero necesitamos importar la biblioteca os:

```
import os
```

Luego, usamos el método os.remove() para borrar el archivo. Por ejemplo, para borrar un archivo llamado myfile.txt:

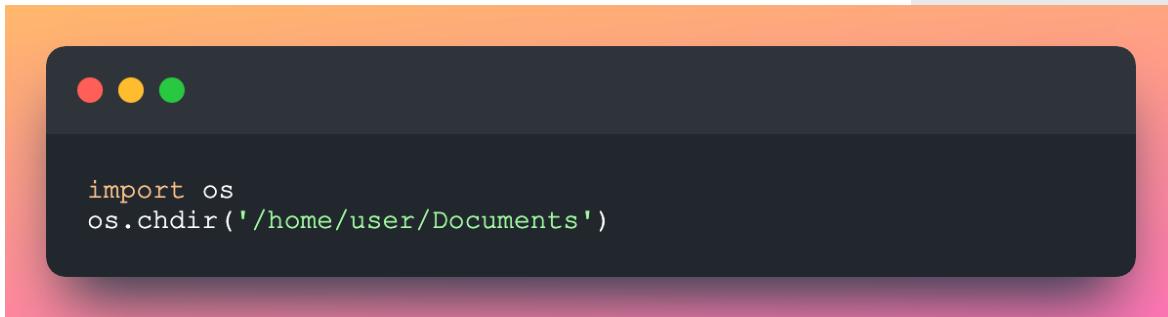
```
os.remove("myfile.txt")
```

Reto:

Crea un programa que borre todos los archivos con extensión .txt en un directorio específico.

## 15.5 Cambiar carpetas

Para cambiar de carpeta, se usa el método `os.chdir()`. Por ejemplo, para cambiar a la carpeta `/home/user/Documents`:



Reto:

Crea un programa que cambie a la carpeta `/home/user/Downloads` y luego imprima el directorio actual.

## 15.6 Listar carpetas y archivos

Para listar carpetas y archivos en Python, se puede usar el módulo `os` y la función `listdir()`. Esta función devuelve una lista de los nombres de los archivos y carpetas en un directorio específico.

Ejemplo:

```
import os

# Listar los archivos y carpetas en el directorio actual
archivos_y_carpetas = os.listdir('.')

# Imprimir los nombres de los archivos y carpetas
for nombre in archivos_y_carpetas:
    print(nombre)
```

Reto:

Escribe un programa que liste los archivos y carpetas en un directorio específico, y luego imprima los nombres de los archivos y carpetas que comienzan con una letra específica.

### 15.7 Obtener carpetas

En Python, podemos obtener carpetas usando la función `os.listdir()`. Esta función devuelve una lista de los nombres de los archivos y carpetas en un directorio específico. Por ejemplo, si queremos obtener todas las carpetas en el directorio actual, podemos usar el siguiente código:

```
import os

# Obtener todas las carpetas en el directorio actual

carpetas = [item for item in os.listdir('.') if
os.path.isdir(item)]

# Imprimir los nombres de las carpetas
for carpeta in carpetas:
    print(carpeta)
```

La lista se crea utilizando una expresión de lista. La sintaxis `os.listdir('.)` devuelve una lista de todos los archivos y subdirectorios en el directorio actual. La condición `os.path.isdir(item)` verifica si `item` es una carpeta o no. Por último, se utiliza una sintaxis compacta de listas de comprensión para crear una nueva lista (`carpetas`) que contiene solo los elementos que cumplen la condición especificada.

Reto:

Escribe un programa que recorra un directorio y todos sus subdirectorios y devuelva una lista con los nombres de todas las carpetas.

### 15.8 Crear y modificar una carpeta

Para crear, renombrar y remover una carpeta en Python, se puede usar la librería `os`.

Ejemplo:

```
import os

# Crear una carpeta
os.mkdir("mi_carpeta")

# Renombrar una carpeta
os.rename("mi_carpeta", "mi_nueva_carpeta")

# Remover una carpeta
os.rmdir("mi_nueva_carpeta")
```

Reto:

Crea una carpeta, renómbrala y luego remuévela usando la librería os.

### 15.9 Métodos de archivos

#### 15.9.1 Método de archivo readline()

`readline()` es una función de Python que lee una línea de un archivo de texto y la devuelve como una cadena. Esta función es útil para leer archivos de texto línea por línea.

Ejemplo:

## 15 MANEJO DE ARCHIVOS

---

```
# abrir el archivo
archivo = open("archivo.txt", "r")

# lee e imprime la primera linea
print(archivo.readline())

# lee e imprime la segunda linea
print(archivo.readline())

# cerrar el archivo
archivo.close()
```

Reto:

Escribe un programa que lea un archivo de texto y cuente el número de líneas que contiene línea por línea.

### 15.9.2 Método de archivo writelines()

writelines() en Python es un método de archivo que escribe una secuencia de cadenas a un archivo. Esta secuencia de cadenas debe ser una lista de cadenas, donde cada cadena es una línea de texto.

Ejemplo:

## 15 MANEJO DE ARCHIVOS

---

```
# Abrir un archivo para escribir
f = open("archivo.txt", "w")

# Escribir una lista de líneas
lines = ["Línea 1\n", "Línea 2\n", "Línea 3\n"]
f.writelines(lines)

# Cerrar el archivo
f.close()
```

Reto:

Escribe un programa que abra un archivo de texto, escriba una lista de líneas de texto en el archivo y luego cierre el archivo.

### 15.9.3 Método de archivo seek()

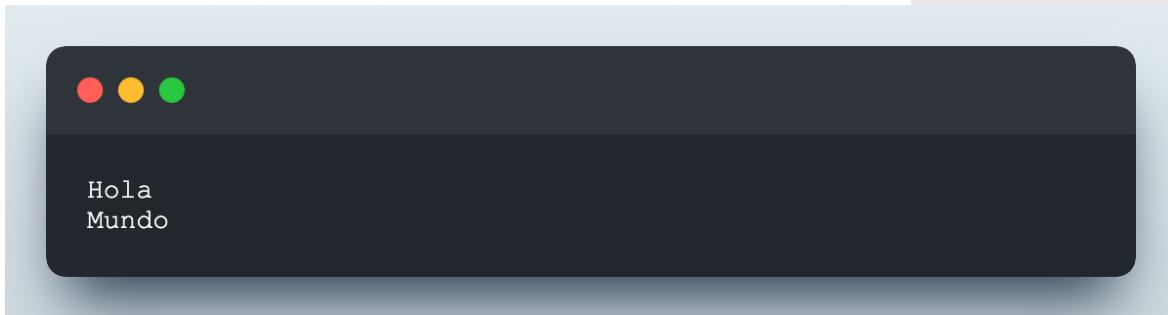
seek() es una función en Python que permite mover el puntero de un archivo a una posición específica. Esta función es útil para leer y escribir en archivos de manera eficiente.

Ejemplo:

Supongamos que tenemos un archivo de texto llamado “archivo.txt” con el siguiente contenido:

## 15 MANEJO DE ARCHIVOS

---



Podemos usar la función `seek()` para mover el puntero al principio del archivo y leer el contenido:

A screenshot of a Jupyter Notebook cell. The code is written in Python and reads from a file named "archivo.txt". It uses the `open()` function to open the file in reading mode ("r"). Then, it uses the `seek(2)` method to move the file pointer to the second character. Finally, it reads the entire file content with `read()` and prints it with `print()`. The output shows the text "la" and "Mundo" on separate lines, indicating that the file was read starting from the third character.

Reto:

Escribe un programa en Python que abra un archivo de texto, mueva el puntero al

## 15 MANEJO DE ARCHIVOS

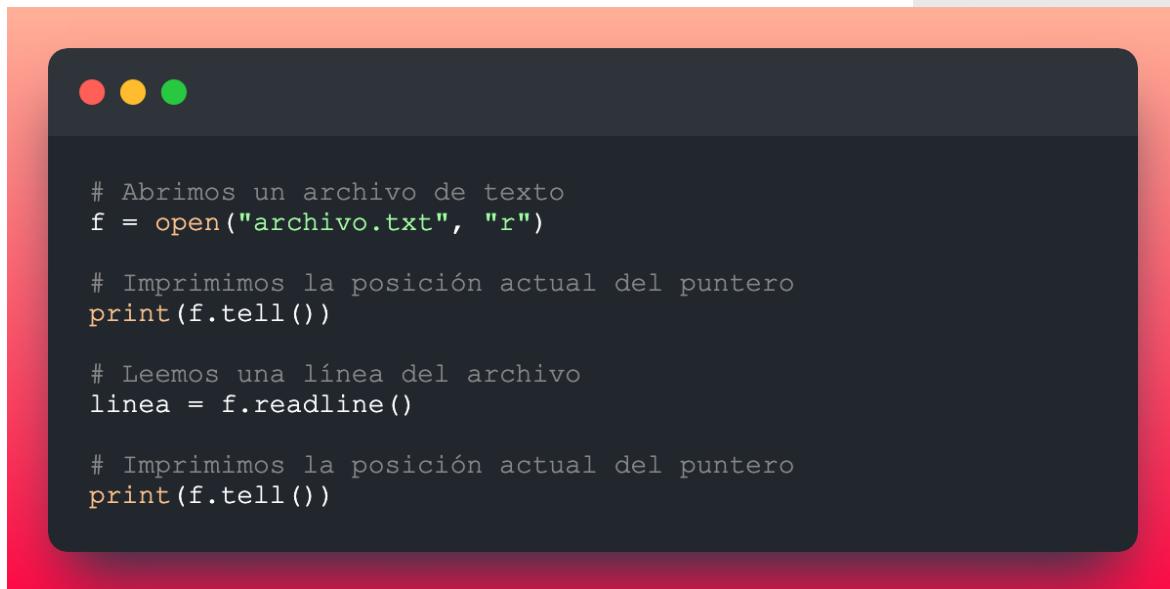
---

final del archivo y escriba una línea de texto.

### 15.9.4 Método de archivo tell()

tell() es un método de Python que devuelve la posición actual del puntero de un archivo. Esto significa que devuelve un número entero que indica la cantidad de bytes desde el inicio del archivo.

Ejemplo:



```
# Abrimos un archivo de texto
f = open("archivo.txt", "r")

# Imprimimos la posición actual del puntero
print(f.tell())

# Leemos una línea del archivo
linea = f.readline()

# Imprimimos la posición actual del puntero
print(f.tell())
```

Reto:

Escribe un programa que abra un archivo de texto, lea una línea y luego imprima la posición actual del puntero.

## 16 Bibliotecas y frameworks

### 16.1 Uso de bibliotecas (NumPy, Pandas, Matplotlib)

Las bibliotecas NumPy, Pandas y Matplotlib son herramientas fundamentales para el análisis de datos en Python. NumPy es una biblioteca de código abierto que proporciona una amplia variedad de funciones matemáticas y estructuras de datos para trabajar con vectores, matrices y arrays. Pandas es una biblioteca de análisis de datos de alto nivel que proporciona estructuras de datos y herramientas de análisis para manipular y analizar datos. Matplotlib es una biblioteca de gráficos 2D que proporciona una variedad de herramientas para crear gráficos y visualizaciones de datos.

Ejemplo:

Supongamos que tenemos un conjunto de datos que contiene información sobre el número de ventas de un producto en diferentes regiones. Podemos usar NumPy para crear un array con los datos, Pandas para crear un marco de datos con los datos y Matplotlib para crear un gráfico de barras que muestre el número de ventas por región.

Reto:

Utilizando datos creados por ti, crea un gráfico de líneas que muestre el número de ventas por región.

### 16.2 Desarrollo de aplicaciones web con frameworks (Django o Flask)

Desarrollar aplicaciones web con frameworks como Django o Flask en Python es una excelente forma de crear aplicaciones web escalables y robustas. Estos frameworks proporcionan una estructura para el desarrollo de aplicaciones web, permitiendo una mayor productividad y mantenibilidad.

tiendo a los desarrolladores centrarse en la lógica de la aplicación en lugar de la configuración de servidores.

Reto:

Aprende Django o Flask y crea una aplicación web usando Django o Flask que permita a los usuarios ver una lista de restaurantes cercanos. La aplicación debe permitir a los usuarios ver la ubicación de los restaurantes en un mapa, así como ver información sobre los restaurantes, como la dirección, el horario de apertura y cierre, y los platos disponibles.

## 17 Avanzado

### 17.1 Sobrecarga de operadores

La sobrecarga de operadores en Python es una característica que permite a los programadores definir comportamientos personalizados para los operadores existentes. Esto significa que los programadores pueden definir cómo se comportan los operadores cuando se usan con objetos personalizados.

Ejemplo:

Supongamos que queremos definir una clase llamada `Vector2D` que representa un vector en un espacio bidimensional. Podemos sobrecargar el operador `+` para que sume dos objetos `Vector2D`:

```
class Vector2D:  
    def __init__(self, x, y):  
        self.x = x  
        self.y = y  
  
    def __add__(self, other):  
        return Vector2D(self.x + other.x, self.y + other.y)  
  
v1 = Vector2D(1, 2)  
v2 = Vector2D(3, 4)  
v3 = v1 + v2  
print(v3.x, v3.y)  
  
# Salida: 4 6
```

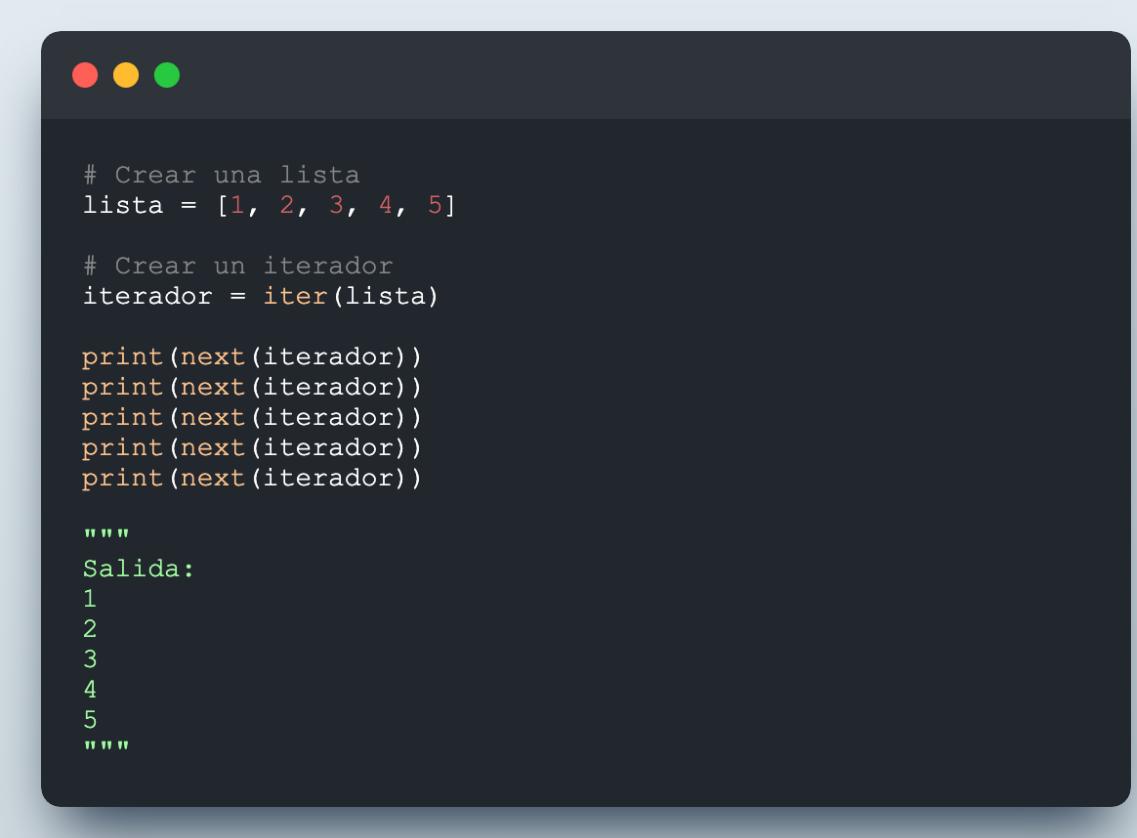
Reto:

Crea una clase llamada Vector3D que representa un vector en un espacio tridimensional. Sobrecarga el operador + para que sume dos objetos Vector3D.

### 17.2 Iteradores

Los iteradores en Python son objetos que permiten recorrer una secuencia de elementos. Estos objetos se comportan como contenedores, ya que contienen una secuencia de elementos, pero no permiten el acceso directo a los elementos. En su lugar, los iteradores permiten recorrer los elementos uno por uno usando la función next().

Ejemplo:



```
# Crear una lista
lista = [1, 2, 3, 4, 5]

# Crear un iterador
iterador = iter(lista)

print(next(iterador))
print(next(iterador))
print(next(iterador))
print(next(iterador))
print(next(iterador))

"""
Salida:
1
2
3
4
5
"""
```

Reto:

Crea un iterador que recorra una lista de números y devuelva el cuadrado de cada número.

### 17.3 Generadores

Los generadores son funciones que retornan un iterador y crean secuencias de valores que se pueden iterar, ahorrándote tiempo y memoria. Pueden salvar el día en situaciones en las que se desea trabajar con grandes cantidades de datos pero no deseamos guardarlos todos en la memoria al mismo tiempo.

¿Quieres ver un ejemplo? Mira esto:

```
# creo una función generador que tiene la palabra yield
def mi_generador(n):
    for i in range(n):
        yield i**2

# Crea un generador de los primeros 5 cuadrados
generador_cuadrados = mi_generador(5)

# Imprime los valores generados
print(next(generador_cuadrados)) # 0
print(next(generador_cuadrados)) # 1
print(next(generador_cuadrados)) # 4
print(next(generador_cuadrados)) # 9
print(next(generador_cuadrados)) # 16
print(next(generador_cuadrados)) # error StopIteration
```

¿Lo ves? En situaciones donde tienes millones de valores y la memoria es limitada, esto puede ser la diferencia entre éxito y fracaso.

Reto:

¿Qué tal si intentas crear un generador que devuelva sólo los números impares de una lista dada? No es nada complicado, pero te aseguro que te ayudará a fortalecer tus habilidades con generadores.

### 17.4 \*args y \*\*kwargs

¿Te has preguntado alguna vez cómo hacer que una función en Python pueda recibir un número variable de argumentos? Ya hemos visto \*args y ahora veremos \*\*kwargs.

## 17 AVANZADO

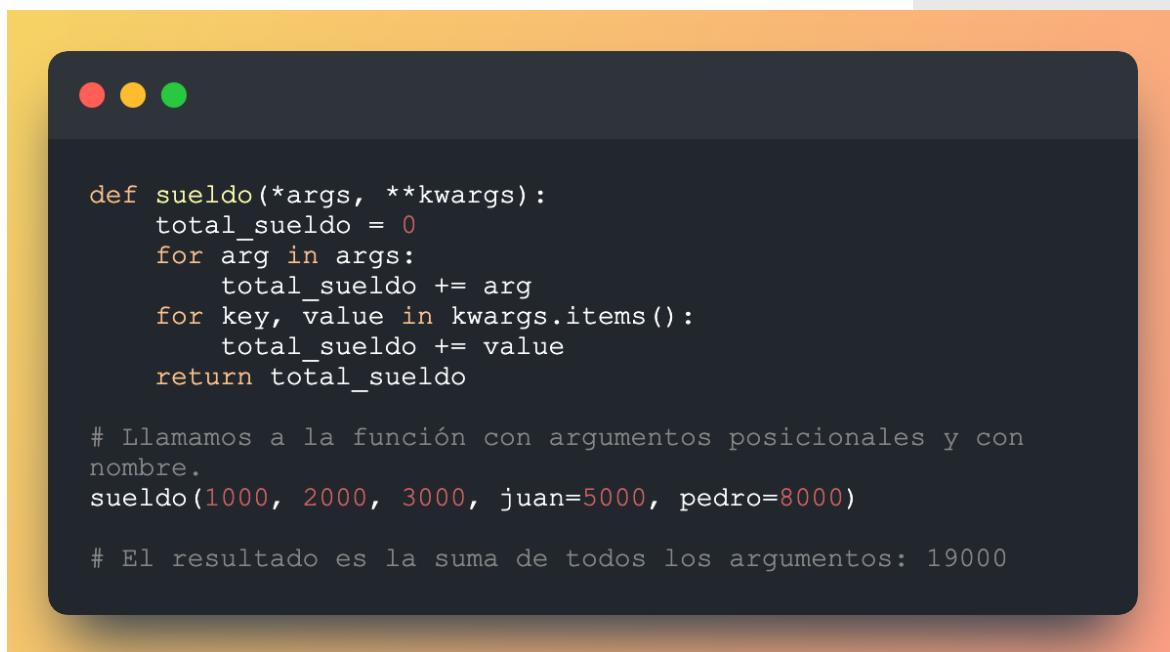
---

\*args y \*\*kwargs son la solución a tus problemas. Ahora, probablemente te estés preguntando ¿qué son esas cosas raras?

\*args es un parámetro especial que permite a una función recibir un número variable de argumentos posicionales. Es decir, puedes pasarle a una función tantos argumentos como quieras y esta los recibirá como una tupla.

Por otro lado, \*\*kwargs es también un parámetro especial, pero que permite a una función recibir un número variable de argumentos con nombre. Es decir, puedes pasarle a una función tantos argumentos con nombre como quieras y esta los recibirá como un diccionario.

Pero no te preocupes si esto aún te parece muy raro, porque con un ejemplo práctico todo se aclara:



```
def sueldo(*args, **kwargs):
    total_sueldo = 0
    for arg in args:
        total_sueldo += arg
    for key, value in kwargs.items():
        total_sueldo += value
    return total_sueldo

# Llamamos a la función con argumentos posicionales y con nombre.
sueldo(1000, 2000, 3000, juan=5000, pedro=8000)

# El resultado es la suma de todos los argumentos: 19000
```

¿Ves? ¡Así de fácil es utilizar \*args y \*\*kwargs! Ahora puedes hacer que tus funciones sean más versátiles y adaptables a distintas situaciones.

Reto:

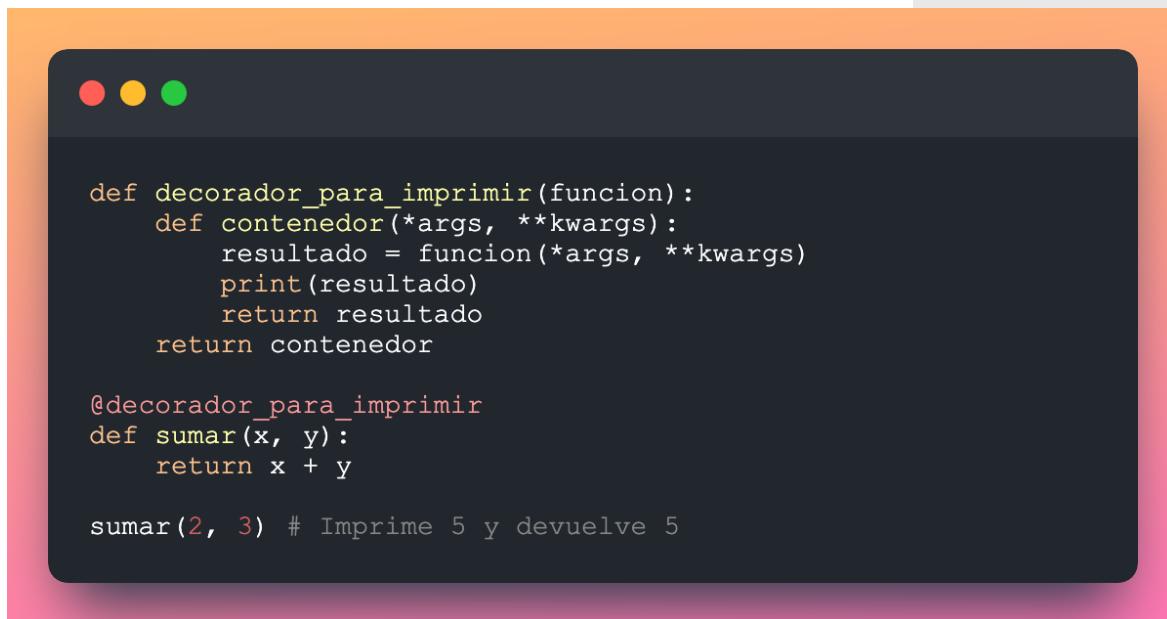
Crea una función que utilice \*args y \*\*kwargs para calcular la suma de los argumentos numéricos y \*\*kwargs.

## 17.5 Programación funcional y decoradores

La programación funcional es un paradigma de programación que se basa en la evaluación de funciones matemáticas para obtener resultados. Esto significa que los programas se escriben como una serie de funciones que se aplican a los datos de entrada para producir los resultados deseados. Los decoradores en Python son una forma de extender la funcionalidad de una función sin modificar el código de la función original. Esto se logra mediante la definición de una función decoradora que toma una función como argumento y devuelve una nueva función que contiene la funcionalidad adicional.

Ejemplo:

Supongamos que queremos agregar una función de impresión a una función existente. Esto se puede lograr con un decorador:



```
def decorador_para_imprimir(funcion):
    def contenedor(*args, **kwargs):
        resultado = funcion(*args, **kwargs)
        print(resultado)
        return resultado
    return contenedor

@decorador_para_imprimir
def sumar(x, y):
    return x + y

sumar(2, 3) # Imprime 5 y devuelve 5
```

Reto:

Escribe un decorador que tome una función y un número entero como argumentos y devuelva una nueva función que multiplique el resultado de la función original por el número entero.

## 17.6 Buffer de lectura de archivos

Cuando abrimos un archivo para leerlo en Python, el sistema operativo determina cuántos bytes se deben leer en cada operación de lectura. Esto significa que, cuando abrimos un archivo, no se lee todo de inmediato, sino que un trozo se carga en memoria con cada operación de lectura. En lugar de leer todas las líneas y llevarlas a la memoria, Python lee el archivo en fragmentos que caben en el buffer de lectura hasta que se llega al final del archivo.

El buffer permite al programa leer los datos del archivo en pequeñas porciones, lo que puede mejorar el rendimiento al minimizar los tiempos de espera. En Python, el buffer de lectura se habilita automáticamente cuando se lee un archivo utilizando la función `open()` en modo lectura (por defecto) o en modo lectura binaria (‘rb’).

Aquí hay un ejemplo práctico de cómo usar el buffer de forma manual para leer archivos en Python:

```
# Abrir archivo en modo lectura
archivo = open('ejemplo.txt', 'r')

# Loop para leer línea por línea
while True:
    # Lectura del archivo en un buffer
    data = archivo.read(1024)

    # Verificar si se ha llegado al final del archivo
    if not data:
        break

    # Procesamiento de los datos del buffer
    print(data)

# Cerrar archivo
archivo.close()
```

En este ejemplo, abrimos un archivo llamado ‘ejemplo.txt’ en modo lectura y luego creamos un bucle que seguirá leyendo el archivo hasta que se llegue al final. En cada iteración del bucle, se llama al método ‘read’ , que lee 1024 bytes del archivo en un buffer. Si el buffer está vacío, significa que hemos llegado al final del archivo y se rompe el bucle. Si el buffer contiene datos, se procesan y se imprimen en la consola utilizando la función ‘print’ .

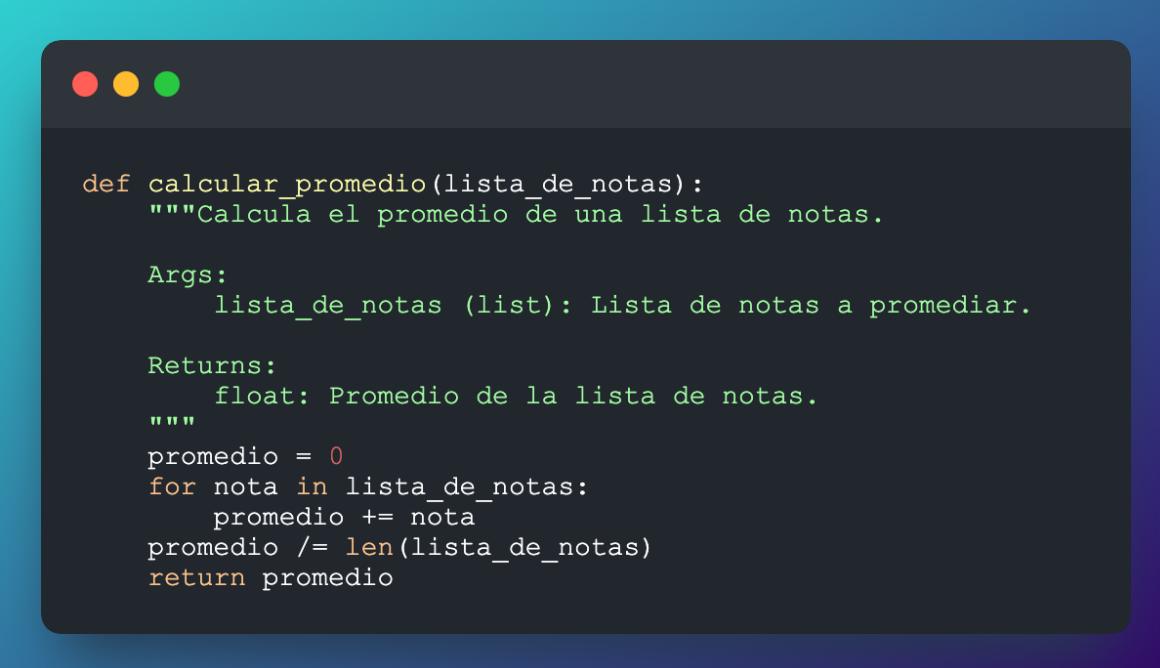
Por último, aquí viene tu reto:

Escribe un programa en Python que abra un archivo en modo escritura y escriba varios mensajes utilizando el buffer de escritura.

## 17.7 Buenas prácticas y patrones de diseño

Las buenas prácticas y patrones de diseño en Python son un conjunto de reglas y recomendaciones que se deben seguir para escribir código de calidad. Estas incluyen el uso de nombres descriptivos para variables y funciones, la indentación correcta, la documentación de código, el uso de excepciones para manejar errores, el uso de funciones para evitar la duplicación de código, etc.

Ejemplo:



```
def calcular_promedio(lista_de_notas):
    """Calcula el promedio de una lista de notas.

    Args:
        lista_de_notas (list): Lista de notas a promediar.

    Returns:
        float: Promedio de la lista de notas.
    """
    promedio = 0
    for nota in lista_de_notas:
        promedio += nota
    promedio /= len(lista_de_notas)
    return promedio
```

## 18 Siguientes pasos

### 18.1 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.
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.

### 18.2 Recursos

1. [Python.org](https://www.python.org): Esta es la página oficial de Python, que contiene documentación, descargas, tutoriales, foros de discusión y mucho más.

## 18 SIGUIENTES PASOS

---

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.

### 18.3 ¿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.

### 18.4 Preguntas de entrevista

1. ¿Cuál es la diferencia entre Python 2 y Python 3?

## 18 SIGUIENTES PASOS

---

- 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?

## 18 SIGUIENTES PASOS

---

- 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?

- 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()`.

## 18 SIGUIENTES PASOS

---

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' ).

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' .