

Tabla de contenido

| | |
|--|--------|
| 1. Objetivo de la unidad 2..... | pág 2 |
| 2. Tema 2: Introducción a la Programación en Python..... | pág 2 |
| 2.1 ¿Qué es Python?..... | pág 2 |
| 2.2 ¿Qué es Google Colaboratory?..... | pág 3 |
| 2.3 Notebooks y tipos de Celdas..... | pág 5 |
| 2.4 Sintaxis de Python..... | pág 7 |
| 2.5 Tipos de datos en Python..... | pág 8 |
| 2.6 Operaciones Aritméticas..... | pág 11 |
| 2.7 Estructuras de datos en Python..... | pág 15 |
| 2.8 Autoevaluación..... | pág 19 |
| 3. Bibliografía..... | pág 20 |

1. Objetivo de la unidad 2

Reconocer la sintaxis de Python, incluyendo la declaración de variables, tipos de datos, expresiones, estructuras de datos, control de flujo y bucles, para implementar programas simples y realizar tareas repetitivas. Asimismo, comprender la creación y el uso de funciones en Python, para desarrollar programas con código más modular, eficiente y reutilizable.

2. Tema 2: Introducción a la Programación en Python.

2.1 ¿Qué es Python?

Python es un lenguaje de programación de alto nivel, interpretado, multiparadigma y de propósito general. Fue creado por Guido van Rossum y lanzado por primera vez en 1991. Python es conocido por su sintaxis simple y legible, lo que lo hace fácil de aprender y utilizar. Python se utiliza en una variedad de campos, incluyendo desarrollo web, análisis de datos, inteligencia artificial, aprendizaje automático, desarrollo de juegos y más. Es una herramienta poderosa y versátil que se adapta a una amplia gama de necesidades de programación.

Características de Python

- **Fácil de aprender y usar:** Python se destaca por su sintaxis clara y legible, lo que lo hace ideal para principiantes en programación y para aquellos que desean prototipar rápidamente ideas.
- **Multiparadigma:** Python soporta múltiples estilos de programación, incluyendo programación imperativa, orientada a objetos y funcional.
- **Interpretado:** Python es un lenguaje interpretado, lo que significa que el código Python se ejecuta línea por línea por un intérprete. Esto facilita la escritura y la depuración del código, pero puede ser más lento en comparación con los lenguajes compilados.

- **Amplia biblioteca estándar:** Python viene con una biblioteca estándar extensa que proporciona módulos y funciones para una amplia gama de tareas, desde manipulación de archivos hasta desarrollo web y procesamiento de datos.
- **Comunidad activa:** Python tiene una comunidad grande y activa de desarrolladores que contribuyen con bibliotecas, herramientas y recursos educativos. Esto hace que sea fácil encontrar ayuda y recursos adicionales cuando se trabaja con Python.

¿Para quién es Python?

Python es para cualquier persona interesada en aprender a programar o en utilizar la programación como una herramienta para resolver problemas en diversos campos. Es adecuado para una amplia gama de personas, desde principiantes absolutos en programación hasta desarrolladores experimentados. Por ejemplo,

- Principiantes en programación
- Estudiantes
- Desarrolladores de Software
- Científicos de datos e investigadores
- Ingenieros de software y más...

Enlace a Python: <https://www.python.org/>

¡Descubre el poder de Python y desata tu creatividad en la programación!

2.2 ¿Qué es Google Colaboratory?

También conocido como Colab, es una plataforma en línea gratuita ofrecida por Google que permite a los usuarios escribir y ejecutar código en lenguaje Python de manera colaborativa. Se basa en el entorno de desarrollo Jupyter Notebook y se ejecuta en la nube de Google.

Características de Google Colaboratory

Acceso gratuito a recursos de GPU: Colab proporciona acceso gratuito a unidades de procesamiento gráfico (GPU), lo que es beneficioso para ejecutar tareas intensivas en cómputo, como entrenamiento de modelos de aprendizaje profundo.

Bibliotecas preinstaladas: Colab viene preinstalado con muchas bibliotecas de Python populares, como TensorFlow, PyTorch y matplotlib,

lo que facilita el desarrollo de proyectos de aprendizaje automático y análisis de datos.

Colaboración en tiempo real: Los usuarios pueden compartir sus cuadernos de Colab con otros, lo que facilita la colaboración en tiempo real. Los colaboradores pueden ver y editar el código simultáneamente.

Acceso a servicios de Google Cloud: Colab permite acceder a los servicios en la nube de Google, como BigQuery y Google Cloud Storage, para realizar análisis de datos a gran escala.

Integración con Google Drive: Los Notebooks de Colab se pueden guardar en Google Drive, lo que facilita el almacenamiento y la colaboración en proyectos.

Desventajas de Colab

Limitaciones de recursos: Aunque Colab proporciona acceso gratuito a recursos de GPU, estos recursos no son ilimitados. Las sesiones de Colab pueden ser interrumpidas después de un tiempo o si se supera un límite de uso.

Bibliotecas desactualizadas: Aunque Colab suele tener instaladas muchas bibliotecas populares, a veces estas no están en sus versiones más recientes. Esto puede ser un problema si se requiere una característica específica de una versión más nueva.

Dependencia de la conexión a internet: Colab requiere una conexión a Internet activa para su uso. Si la conexión es inestable o se pierde, puede afectar la capacidad para ejecutar o guardar Notebooks.

Restricción de uso intensivo: Google Colab puede aplicar restricciones en el uso intensivo de recursos, como el uso de GPU, para evitar abusos. Esto puede limitar la ejecución de tareas computacionalmente intensivas durante largos períodos.

Máquinas virtuales temporales: Las máquinas virtuales de Colab son temporales y se reinician después de un período de inactividad prolongado. Esto puede llevar a la pérdida de datos no guardados.

Nota: Para acceder a Google Colab requieres de una cuenta de Gmail personal.

Enlace a Google Colaboratory: <https://colab.research.google.com/>



2.3 Notebooks y tipos de Celdas

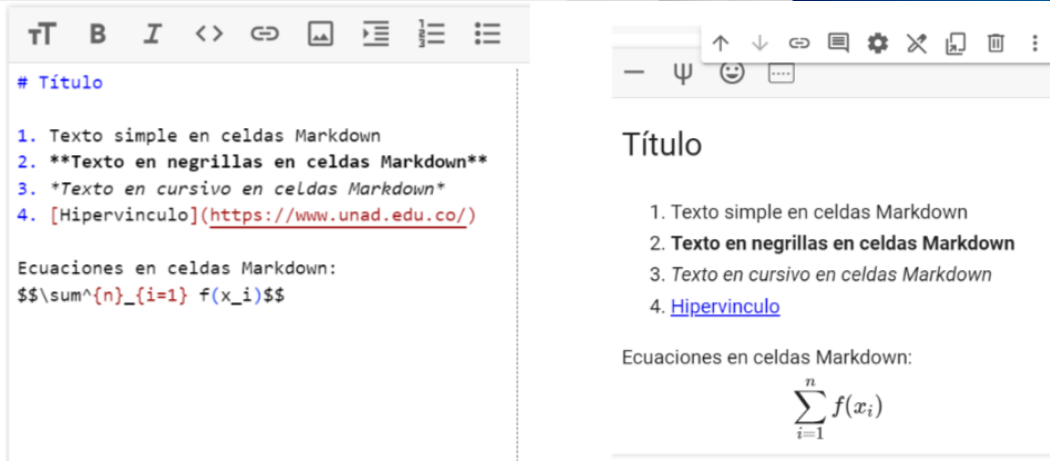
¿Qué son los Notebooks?

Son entornos interactivos que permiten combinar texto, código y resultados visuales en un solo documento. Están diseñados para facilitar la creación, colaboración y presentación de trabajos que involucren programación y análisis de datos.

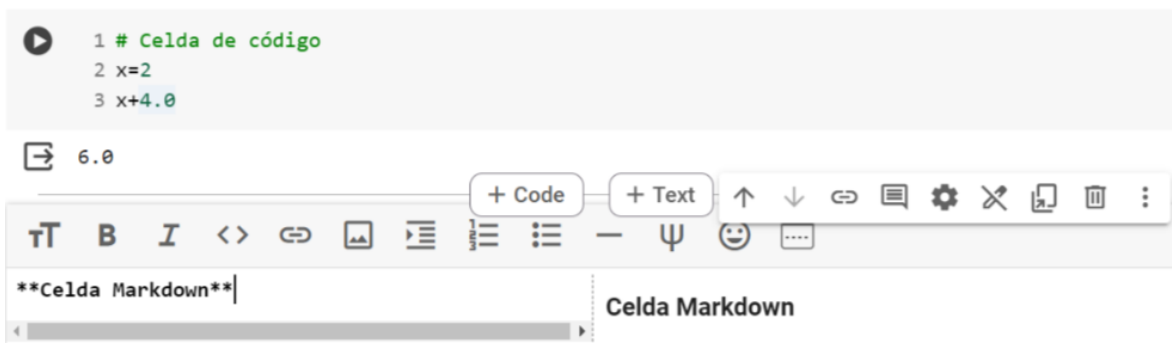
Uno de los formatos de Notebooks más comunes es el formato Jupyter Notebook, que se utiliza en herramientas como Google Colab, JupyterLab y otros entornos similares. Estos archivos tienen la extensión .ipynb.

Celdas Markdown

Estas celdas contienen texto formateado utilizando la sintaxis Markdown. Puedes usar Markdown para agregar encabezados, listas, enlaces, imágenes y otros elementos de formato enriquecido al texto. Este tipo de celdas se utiliza para agregar documentación, explicaciones, y comentarios a tu código. En Colab, las celdas de texto o markdown cuentan con botones de un editor de texto común, con shift + enter puedes ver el resultado final de la edición.



Shift + Enter



Celdas de código

Estas celdas contienen fragmentos de código en un lenguaje de programación específico (como Python, R, Julia, etc.). Puedes escribir, ejecutar y editar código directamente en estas celdas. El resultado de la ejecución del código se muestra inmediatamente debajo de la celda de código.

En Colab, las celdas de código cuentan con un botón de **play** para ejecutar o correr su código, dicha operación también se puede ejecutar con la combinación de teclas **shift + enter**.

```
✓ [1] 1 x=2
0 s    2 y=3.0

✓ [2] 1 x+y
0 s    5.0

✓ [3] 1 hola="Hola Mundo!"
0 s
```

2.4 Sintaxis de Python

Python es conocido por su sintaxis clara y legible, lo que lo hace especialmente atractivo tanto para principiantes como para programadores experimentados. Aquí hay un vistazo rápido a algunos aspectos clave de la sintaxis en Python:

Indentación significativa: En Python, la indentación (sangría) se utiliza para delimitar bloques de código en lugar de utilizar llaves como en otros lenguajes. Esto promueve la legibilidad y la coherencia en el código.

```
1 nombre = "Miguel"
2 apellido = "Vargas"
3
4 if nombre == "Miguel":
5     if apellido == "Vargas":
6         print("Hola, te estaba buscando")
7     else:
8         print("Te llamas Miguel, pero no a quién busco")
9 else:
10    print("Hola, estoy buscando a Miguel")
```

Tipado dinámico: Python es un lenguaje de tipado dinámico, lo que significa que no es necesario declarar explícitamente el tipo de una variable antes de usarla. El tipo de una variable se infiere automáticamente en función del valor asignado.

Comentarios: Los comentarios en Python se crean precediendo el texto con el símbolo **#**. Estos comentarios son útiles para documentar el código y hacerlo más comprensible para otros desarrolladores.

```
# Este programa suma dos números y muestra el resultado.  
  
# Definimos la función sumar  
def sumar(a, b):  
    return a + b  
  
# Llamamos a la función sumar con los números 3 y 5  
resultado = sumar(3, 5)  
  
# Mostramos el resultado de la suma  
print("La suma de 3 y 5 es:", resultado)
```

→ La suma de 3 y 5 es: 8

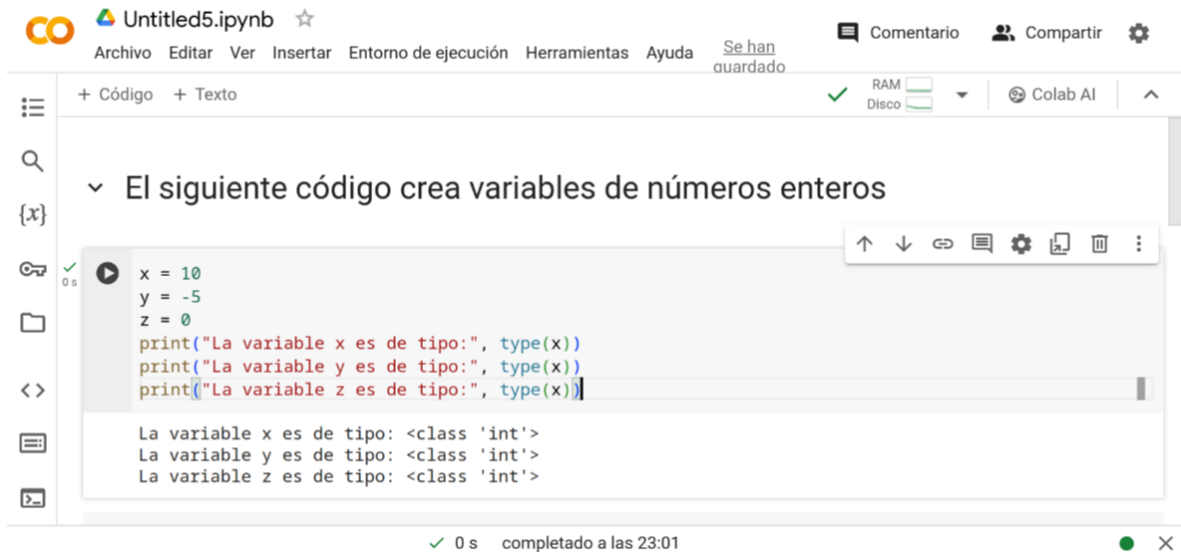
2.5 Tipos de datos en Python

Los tipos de datos se refieren a las categorías o clases de datos que se pueden utilizar en un programa. Los tipos de datos determinan qué operaciones se pueden realizar en los datos y cómo se almacenan en la memoria. Algunos de los tipos de datos más comunes en Python.

En Python, los datos se clasifican en diferentes tipos, desde los básicos como enteros y cadenas de caracteres, hasta estructuras más complejas como listas, tuplas, conjuntos y diccionarios:

Enteros Los enteros (int) en Python son un tipo de datos que representa números enteros, es decir, números sin parte decimal. Los enteros pueden ser positivos, negativos o cero. Por ejemplo: -2, -1, 0, 1, 2, 3. 'int'.

Ejemplo:



Untitled5.ipynb

Archivo Editar Ver Insertar Entorno de ejecución Herramientas Ayuda [Se han guardado](#)

+ Código + Texto

✓ RAM
Disco

Colab AI

El siguiente código crea variables de números enteros

```
x = 10
y = -5
z = 0
print("La variable x es de tipo:", type(x))
print("La variable y es de tipo:", type(y))
print("La variable z es de tipo:", type(z))
```

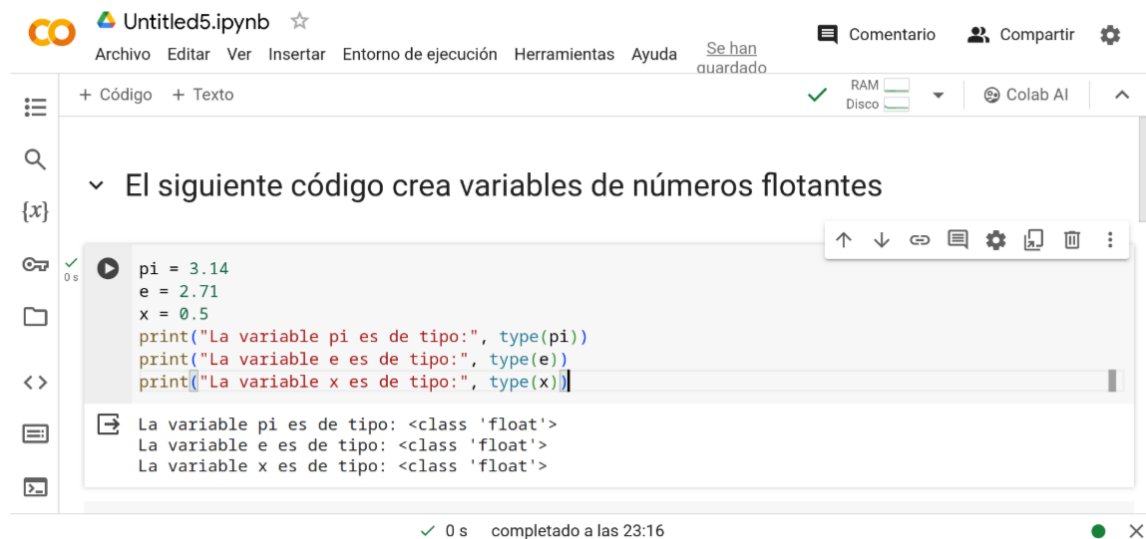
La variable x es de tipo: <class 'int'>
La variable y es de tipo: <class 'int'>
La variable z es de tipo: <class 'int'>

✓ 0 s completado a las 23:01

Flotantes 'float'

Los flotantes (float) en Python son un tipo de datos que representa números con una parte decimal. Por ejemplo: -2.5, 0.0, 3.14, 2.71828.

Ejemplo:



Untitled5.ipynb

Archivo Editar Ver Insertar Entorno de ejecución Herramientas Ayuda [Se han guardado](#)

+ Código + Texto

✓ RAM
Disco

Colab AI

El siguiente código crea variables de números flotantes

```
pi = 3.14
e = 2.71
x = 0.5
print("La variable pi es de tipo:", type(pi))
print("La variable e es de tipo:", type(e))
print("La variable x es de tipo:", type(x))
```

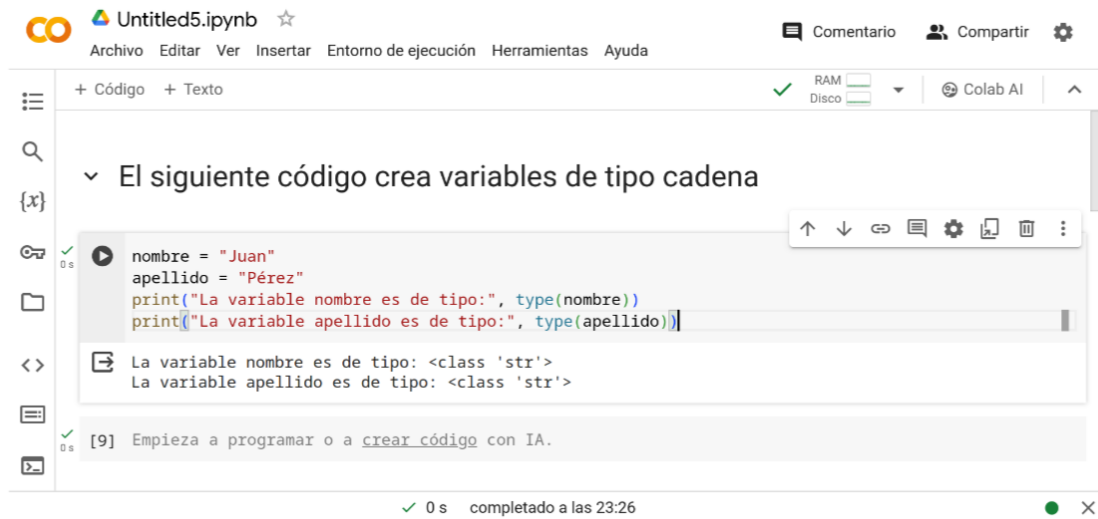
La variable pi es de tipo: <class 'float'>
La variable e es de tipo: <class 'float'>
La variable x es de tipo: <class 'float'>

✓ 0 s completado a las 23:16

Cadenas 'str'

Las cadenas de caracteres (str) en Python son un tipo de datos que se utiliza para representar texto. Las cadenas son secuencias inmutables de caracteres y pueden ser creadas utilizando comillas simples (' '), comillas dobles (" "), o incluso comillas triples (''' ' o "" "" "" "") para cadenas de varias líneas. Por ejemplo: "Juan Perez", "Hello World".

Ejemplo:



Untitled5.ipynb

Archivo Editar Ver Insertar Entorno de ejecución Herramientas Ayuda

+ Código + Texto

✓ RAM
Disco

Colab AI

✓ El siguiente código crea variables de tipo cadena

```
nombre = "Juan"
apellido = "Pérez"
print("La variable nombre es de tipo:", type(nombre))
print("La variable apellido es de tipo:", type(apellido))
```

La variable nombre es de tipo: <class 'str'>
La variable apellido es de tipo: <class 'str'>

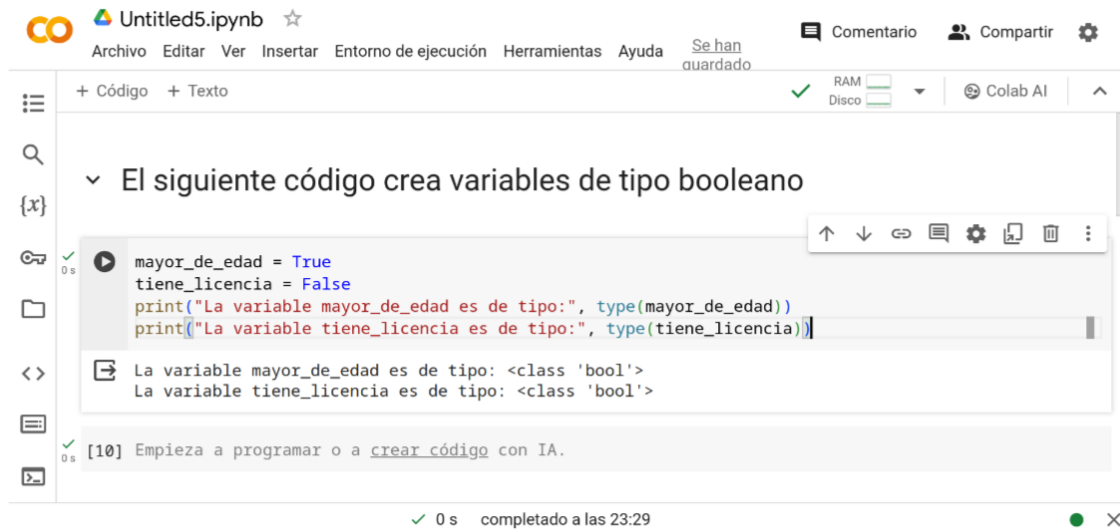
✓ [9] Empieza a programar o a [crear código](#) con IA.

✓ 0 s completado a las 23:26

Booleanos 'bool'

Los booleanos (bool) en Python son un tipo de datos que representa un valor de verdad, que puede ser Verdadero (True) o Falso (False). Los booleanos son fundamentales en programación para la toma de decisiones y la lógica condicional.

Ejemplo:



The screenshot shows a Jupyter Notebook titled 'Untitled5.ipynb'. The code cell contains the following Python code:

```
mayor_de_edad = True
tiene_licencia = False
print("La variable mayor_de_edad es de tipo:", type(mayor_de_edad))
print("La variable tiene_licencia es de tipo:", type(tiene_licencia))
```

The output of the code is:

```
La variable mayor_de_edad es de tipo: <class 'bool'>
La variable tiene_licencia es de tipo: <class 'bool'>
```

Below the output, there is a prompt: [10] Empieza a programar o a crear código con IA.

Nota: Recuerda que en Python el tipado es dinámico, lo que significa que no es necesario especificar el tipo de variable, ya que se infiere automáticamente. Comprender estos tipos de datos es crucial para manipular la información de manera eficiente.

2.6 Operaciones Aritméticas

En Python, las operaciones aritméticas son un conjunto de operaciones básicas que se utilizan para realizar cálculos matemáticos con números.

Nota: Desde este punto del documento, se anexa el código para que, como parte de la práctica, pueda ser copiado y replicado por los estudiantes en sus Notebooks.

- **Suma: ("+"): Se utiliza para sumar dos números.**

Ejemplo:

El siguiente código es un ejemplo para la operación suma:

```
# Ejemplo de la operación suma en Python

# Definimos los números a sumar
numero1 = 10
numero2 = 3
numero3 = 2.5

# Realizamos las sumas
suma1 = numero1 + numero2 # suma1 será 13
suma2 = numero2 + numero3 # suma2 será 5.5
```

```
suma3 = numero1 + numero3 # suma3 será 12.5

# Imprimir los resultados
print("La suma de numero1 y numero2 es:", suma1)
print("La suma de numero2 y numero3 es:", suma2)
print("La suma de numero1 y numero3 es:", suma3)
```

- **Resta: ("-"):** Se utiliza para restar un número de otro.

Ejemplo:

El siguiente código es un ejemplo para la operación resta:

```
# Ejemplo de la operación resta en Python

# Definimos los números a restar
numero1 = 10
numero2 = 3
numero3 = 2.5

# Realizamos las restas
resta1 = numero1 - numero2 # resta1 será 7
resta2 = numero2 - numero3 # resta2 será 0.5
resta3 = numero1 - numero3 # resta3 será 7.5

# Imprimir los resultados
print("La resta de numero1 y numero2 es:", resta1)
print("La resta de numero2 y numero3 es:", resta2)
print("La resta de numero1 y numero3 es:", resta3)
```

- **Producto: ("*"):** Se utiliza para multiplicar dos números.

Ejemplo:

El siguiente código es un ejemplo para la operación producto:

```
# Ejemplo de la operación producto en Python

# Definimos los números a multiplicar
numero1 = 10
numero2 = 3
numero3 = 2.5
```

```
# Realizamos los productos
producto1 = numero1 * numero2 # producto1 será 30
producto2 = numero2 * numero3 # producto2 será 7.5
producto3 = numero1 * numero3 # producto3 será 25.0

# Imprimir los resultados
print("El producto de numero1 y numero2 es:", producto1)
print("El producto de numero2 y numero3 es:", producto2)
print("El producto de numero1 y numero3 es:", producto3)
```

- **Cociente: ("/"):** Se utiliza para dividir un número por otro. El resultado siempre es un número de punto flotante.

Ejemplo:

El siguiente código es un ejemplo para la operación cociente:

```
# Ejemplo de la operación cociente en Python

# Definimos los números a dividir
numero1 = 10
numero2 = 3
numero3 = 2.5

# Realizamos los cocientes
cociente1 = numero1 / numero2 # cociente1 será aproximadamente 3.3333
cociente2 = numero2 / numero3 # cociente2 será 1.2
cociente3 = numero1 / numero3 # cociente3 será 4.0

# Imprimir los resultados
print("El cociente de numero1 y numero2 es:", cociente1)
print("El cociente de numero2 y numero3 es:", cociente2)
print("El cociente de numero1 y numero3 es:", cociente3)
```

- **Cociente entero: ("//"):** Se utiliza para dividir un número por otro y obtener el cociente como un número entero (se descarta la parte decimal).

Ejemplo:

El siguiente código es un ejemplo para la operación cociente:

Ejemplo de la operación cociente entero en Python

Definimos los números a dividir

```
numero1 = 10  
numero2 = 3  
numero3 = 2.5
```

Realizamos los cocientes enteros

```
cociente_entero1 = numero1 // numero2 # cociente_entero1 será  
3  
cociente_entero2 = numero2 // numero3 # cociente_entero2 será  
1.0  
cociente_entero3 = numero1 // numero3 # cociente_entero3 será  
4.0
```

Imprimir los resultados

```
print("El cociente entero de numero1 y numero2 es:",  
cociente_entero1)  
print("El cociente entero de numero2 y numero3 es:",  
cociente_entero2)  
print("El cociente entero de numero1 y numero3 es:",  
cociente_entero3)
```

- **Potencia: ("**"):** Se utiliza para elevar un número a una potencia.

Ejemplo:

El siguiente código es un ejemplo para la operación potencia:

Ejemplo de la operación potencia en Python

Definimos los números base y exponentes

```
numero1 = 10  
numero2 = 3  
numero3 = 2.5
```

Realizamos las operaciones de potencia

```
potencia1 = numero1 ** numero2 # potencia1 será 1000  
potencia2 = numero2 ** numero3 # potencia2 será  
aproximadamente 15.588
```

```
potencia3 = numero1 ** numero3 # potencia3 será  
aproximadamente 316.227  
  
# Imprimir los resultados  
print("La potencia de numero1 elevado a numero2 es:",  
potencia1)  
print("La potencia de numero2 elevado a numero3 es:",  
potencia2)  
print("La potencia de numero1 elevado a numero3 es:",  
potencia3)
```

2.7 Estructuras de datos en Python

Las estructuras de datos son un conjunto de elementos que permiten almacenar y organizar datos de manera efectiva para su posterior manipulación. Python ofrece diversas estructuras de datos integradas que facilitan la representación y el manejo de información.

- **Listas "list":** Una lista es una colección ordenada de elementos que pueden ser de diferentes tipos, como enteros, cadenas, booleanos, etc. Las listas son mutables, lo que significa que se pueden modificar después de su creación.

Ejemplo:

El siguiente código es un ejemplo del uso de listas:

```
# Ejemplo de uso de listas en Python  
  
# Crear una lista de frutas  
frutas = ["manzana", "banana", "cereza", "durazno", "mango"]  
  
# Imprimir un mensaje antes de listar las frutas  
print("Lista de frutas:")  
  
# Imprimir cada elemento de la lista individualmente  
accediendo a cada índice  
print(frutas[0]) # Imprime el primer elemento de la lista:  
"manzana"  
print(frutas[1]) # Imprime el segundo elemento de la lista:
```



```
"banana"
print(frutas[2]) # Imprime el tercer elemento de la lista:
"cereza"
print(frutas[3]) # Imprime el cuarto elemento de la lista:
"durazno"
print(frutas[4]) # Imprime el quinto elemento de la lista:
"mango"
```

- **Tuplas "tuple":** Una tupla es una estructura de datos similar a una lista, pero con la diferencia principal de que las tuplas son inmutables, es decir, una vez creada una tupla, no se pueden modificar sus elementos.

Ejemplo:

El siguiente código es un ejemplo del uso de tuplas:

```
# Ejemplo de uso de tuplas en Python

# Crear una tupla de frutas
frutas = ("manzana", "banana", "cereza", "durazno", "mango")

# Imprimir un mensaje antes de listar las frutas
print("Tupla de frutas:")

# Imprimir cada elemento de la tupla individualmente
accediendo a cada índice
print(frutas[0]) # Imprime el primer elemento de la tupla:
"manzana"
print(frutas[1]) # Imprime el segundo elemento de la tupla:
"banana"
print(frutas[2]) # Imprime el tercer elemento de la tupla:
"cereza"
print(frutas[3]) # Imprime el cuarto elemento de la tupla:
"durazno"
print(frutas[4]) # Imprime el quinto elemento de la tupla:
"mango"
```

- **Diccionarios "dict":** Un diccionario es una estructura de datos que permite almacenar pares de clave-valor. A diferencia de las listas y las tuplas, que utilizan índices numéricos para acceder a

sus elementos, los diccionarios utilizan claves para acceder a los valores asociados.

Los diccionarios en Python son estructuras de datos que permiten almacenar información de manera organizada utilizando pares de clave-valor. Puedes pensar en un diccionario como una especie de agenda telefónica, donde cada nombre (clave) está asociado con un número de teléfono (valor). Lo especial de los diccionarios es que puedes buscar la información rápidamente utilizando las claves, en lugar de tener que recorrer toda la estructura. En Python, los diccionarios se crean utilizando llaves { } y los pares clave-valor se separan por comas.

Ejemplo:

Un diccionario de contactos, podríamos tener nombres como "Juan", "María" y "Carlos" como claves, y los números de teléfono correspondientes como valores asociados a esas claves. Cuando necesitamos encontrar el número de teléfono de alguien, simplemente buscamos su nombre en el diccionario y obtenemos el número asociado.

Ejemplo:

El siguiente código es un ejemplo del uso de diccionarios:

```
# Ejemplo de uso de diccionarios en Python con valores de keys
como listas

# Crear un diccionario de información de contactos con valores
de keys como listas
info_contactos = {
    "Juan": ["123456789", "juan@email.com"],
    "María": ["987654321", "maria@email.com"],
    "Carlos": ["456789123", "carlos@email.com"]
}

# Imprimir un mensaje antes de listar la información de los
contactos
print("Información de contactos:")

# Imprimir la información de cada contacto del diccionario
for nombre, info in info_contactos.items():
```

```
print(f"{nombre}:")  
print(f" Teléfono: {info[0]}")  
print(f" Email: {info[1]}")
```

- **Conjuntos "set":** Un conjunto es una colección desordenada de elementos únicos.

Ejemplo:

El siguiente código es un ejemplo del uso de conjuntos:

```
# Ejemplo de uso de conjuntos en Python  
  
# Crear un conjunto de frutas  
frutas = {"manzana", "banana", "cereza", "durazno", "mango"}  
  
# Imprimir un mensaje antes de listar las frutas  
print("Conjunto de frutas:")  
  
# Imprimir cada elemento del conjunto utilizando un bucle for  
for fruta in frutas:  
    print(fruta)
```

2.8 Autoevaluación

Pregunta 1:

¿Cuál de las siguientes características NO es una característica clave de Python?

Opción 1: Lenguaje de programación interpretado

Opción 2: Tipado estático

Opción 3: Sintaxis clara y legible

Opción 4: Amplia biblioteca estándar

Pregunta 2:

¿Cuál de las siguientes expresiones en Python calcula el valor de la potencia 3^4 ?

Opción 1: $3*4$

Opción 2: 3^4

Opción 3: $3^{**}4$

Opción 4: $4^{**}3$

Pregunta 3:

¿Cuál de las siguientes operaciones es correcta para agregar un nuevo elemento "z" al diccionario `diccionario = {"a": 1, "b": 2, "c": 3}` en Python?

Opción 1: `diccionario.add("z", 4)`

Opción 2: `diccionario.append({"z": 4})`

Opción 3: `diccionario["z"] = 4`

Opción 4: `diccionario.extend({"z": 4})`

Pregunta 4:

¿Cuál de las siguientes opciones es una forma válida de acceder a todas las claves del diccionario `diccionario = {"nombre": "Juan", "edad": 30}` en Python?

Opción 1: `diccionario.keys()`

Opción 2: `diccionario.values()`

Opción 3: `diccionario.items()`

Opción 4: `diccionario.get("nombre")`

3. Bibliografía

- ✓ Álvarez, C. A. (2020). Introducción al Jupyter Notebook y aplicaciones básicas. [Objeto_virtual_de_Informacion_OVI]. Repositorio Institucional UNAD.
<https://repository.unad.edu.co/handle/10596/35681>
- ✓ Hinojosa Gutiérrez, Á. (2015). Python paso a paso: (ed.). RA-MA Editorial, (pp. 21-24). <https://elibro-net.bibliotecavirtual.unad.edu.co/es/ereader/unad/107213?page=1>
- ✓ Hinojosa Gutiérrez, Á. (2015). Python paso a paso: (ed.). RA-MA Editorial, (pp. 39). <https://elibro-net.bibliotecavirtual.unad.edu.co/es/ereader/unad/107213?page=1>

- ✓ Hinojosa Gutiérrez, Á. (2015). Python paso a paso: (ed.). RA-MA Editorial, (pp. 49-88). <https://elibro-net.bibliotecavirtual.unad.edu.co/es/ereader/unad/107213?page=1>
- ✓ Hinojosa Gutiérrez, Á. (2015). Python paso a paso: (ed.). RA-MA Editorial, (pp. 89-104). <https://elibro-net.bibliotecavirtual.unad.edu.co/es/ereader/unad/107213?page=1>
- ✓ Hinojosa Gutiérrez, Á. (2015). Python paso a paso: (ed.). RA-MA Editorial, (pp. 105-122). <https://elibro-net.bibliotecavirtual.unad.edu.co/es/ereader/unad/107213?page=1>
- ✓ Cuevas Álvarez, A. (2016). Python 3: curso práctico. RA-MA Editorial, (pp. 145-182). <https://elibro-net.bibliotecavirtual.unad.edu.co/es/ereader/unad/106404?page=1>
- ✓ Código Marzal Varó, A. García Sevilla, P. & Gracia Luengo, I. (2016). Introducción a la programación con Python 3. D - Universitat Jaume I. Servei de Comunicació i Publicacions, (pp. 228-336). <https://elibro-net.bibliotecavirtual.unad.edu.co/es/ereader/unad/51760?page=1>
- ✓ Noguera, A. d. (2021). Lenguaje de programación Python. [Objeto_virtual_de_Informacion_OVI]. Repositorio Institucional UNAD. <https://repository.unad.edu.co/handle/10596/39346>
- ✓ Aldana, J. M. (2023). *Estructuras de control cíclicas*. [Objeto_virtual_de_aprendizaje_OVA]. Repositorio Institucional UNAD. <https://repository.unad.edu.co/handle/10596/58883>
- ✓ Amórtegui, M. P. (2021). Algoritmos y Programación. [Objeto_virtual_de_Informacion_OVI]. Repositorio Institucional UNAD. <https://repository.unad.edu.co/handle/10596/44535>