Guía de estudio N.º 1 — Conociendo Python (Parte I)

 ¡Hola! Te damos la bienvenida a esta nueva guía de estudio.

¿En qué consiste esta guía?

Esta guía te enseñará los conceptos básicos de Python para que puedas empezar a programar en este lenguaje. Aprenderás cómo usar algunas herramientas esenciales y podrás poner en práctica lo que aprendas con ejemplos y ejercicios.

¡Prepárate para explorar el mundo de la programación en Python!

¡Vamos con todo!

Tabla de contenidos

¡Comencemos!

¿Qué es Python?

Python es un lenguaje de programación de alto nivel y de propósito general, utilizado en una amplia variedad de aplicaciones, desde desarrollo web hasta análisis de datos y aprendizaje automático. Se destaca por su simplicidad, facilidad de lectura y escritura de código, y una gran cantidad de bibliotecas y herramientas de código abierto disponibles. Algunas de las empresas importantes que usan Python son:

Fuente: FlatIcon

- **YouTube**: la plataforma de videos más grande del mundo usa Python para la gestión de datos y el procesamiento de videos.
- **Instagram**: la popular red social utiliza Python en su backend para el procesamiento de imágenes y la gestión de datos.
- **Spotify**: el servicio de streaming de música utiliza Python en su backend para la gestión de datos y la personalización de recomendaciones.

La instalación de Python en una máquina depende del sistema operativo utilizado, pero se puede descargar e instalar desde el sitio web oficial de Python.

Puedes verificar si tienes Python instalado en tu sistema abriendo la terminal o línea de comandos y escribiendo:

python --version

Si Python está instalado, mostrará la versión actual. Si no está instalado, devolverá un error.

¡Felicidades, ya puedes continuar aprendiendo!

Interactuando con el usuario

Input y print

La interacción con el usuario es una forma importante de hacer que un script de Python sea más dinámico y útil. Al interactuar con el usuario, el script puede adaptarse y responder a sus necesidades en tiempo real, lo que puede hacer que la experiencia de uso sea más satisfactoria y eficiente. Si el script puede guiar al usuario a través del proceso de ingreso de datos o instrucciones, es más probable que los usuarios no técnicos puedan utilizarlo de manera efectiva.

Para interactuar con el usuario contaremos esencialmente con dos funciones:

- **print():** Esta función permite imprimir texto en la pantalla. Puede tomar uno o más argumentos, separados por comas, y los imprimirá en orden, separados por un espacio por defecto.

  Si hemos guardado algún valor en una variable (por ejemplo, **edad**), podemos insertarlo dentro de un string que se imprime en pantalla utilizando:

```
print(f"Tienes {edad} años")
```

- **input():** Esta función permite al usuario ingresar datos desde el teclado. En su forma más simple, input() no toma ningún argumento y solo espera a que el usuario ingrese algún texto y presione Enter. El texto que ingrese el usuario se devolverá como un objeto de tipo str (cadena).

```
valor=input()
```

  Se puede agregar la opción de incluir un texto tipo “instrucción”, que se imprimirá en pantalla.

```
valor=input("Ingrese su nombre y presione enter")
```

Un programa comúnmente recibe inputs y produce outputs, los cuales serán objetos de Python. En ambos casos, cualquier información ingresada por el usuario mediante el teclado también se considerará un objeto de Python.

Para garantizar que el código funcione correctamente y sin errores, es importante realizar la conversión de tipo apropiada en la entrada recibida. Ya veremos esto.

Comentarios

El símbolo # nos permite agregar comentarios en nuestro código, es decir, textos que no se considerarán como parte del programa. Más que una interacción con el usuario, pueden considerarse una suerte de interacción con otra persona que acceda a nuestro código, ya que puede permitirle comprender de mejor forma la lógica o estructura que hayamos utilizado en nuestro programa

Tipos de datos

En Python, los tipos de datos se refieren a las distintas categorías de valores que se pueden utilizar en un programa. Algunos de los tipos de datos más comunes son:

- **Cadenas de caracteres (strings)**: secuencia de caracteres entre comillas simples o dobles. Se pueden concatenar, cortar y buscar subcadenas. Ejemplos: "Hola mundo", "123", "True", etc.

- **Valores booleanos (boolean)**: valores que indican verdadero o falso. En Python, los valores booleanos son True y False.

- **Numéricos:** distinguiremos aquí, esencialmente, dos tipos de datos numéricos:
    - **Números enteros (integers)**: números enteros positivos o negativos sin parte decimal. Ejemplos: 1, 2, 10, -5, etc.

    - **Números con punto flotante (floats)**: números con parte decimal. Ejemplos: 3.14, 2.5, -0.5, etc.

- **Listas (lists)**: Secuencia de elementos ordenados, que pueden ser de cualquier tipo de datos. Las listas se crean usando corchetes [] y los elementos se separan por comas. Ejemplos: [1, 2, 3], ["manzana", "pera", "naranja"], [1, "hola", True], etc.

- **Diccionarios (dictionaries)**: colección de pares clave-valor, donde cada clave se asocia con un valor. Los diccionarios se crean usando llaves {} y cada par clave-valor se separa por comas. Ejemplos: {"nombre": "Juan", "edad": 25, "ciudad": "Santiago"}, {"1": "uno", "2": "dos", "3": "tres"}, etc.

En Python, para acceder a los elementos de una lista se usa el **índice** de cada elemento. Debemos considerar que el índice **siempre comienza en cero**, es decir, el primer elemento es el de índice 0. Para obtener un elemento de una lista simplemente escribimos el nombre de la lista seguido del **índice entre paréntesis cuadrados**. Por ejemplo:

```python
# Accediendo a los elementos de una lista.

mi_lista = ["manzana", "banana", "naranja", "piña"]
print(mi_lista[0])   # salida: "manzana"
print(mi_lista[1])   # salida: "banana"
print(mi_lista[2])   # salida: "naranja"
print(mi_lista[3])   # salida: "piña"
```

```
manzana
banana
naranja
piña
```

Para acceder al valor de un elemento en un diccionario, se utiliza la clave correspondiente en lugar del índice. Por ejemplo, si tenemos un diccionario llamado mi_diccionario que contiene la clave "nombre" y su valor es "Juan", podemos acceder al valor de la siguiente manera: mi_diccionario["nombre"]. Esto nos devolverá el valor "Juan".

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

# Acceder al valor de una clave:
print(mi_diccionario["nombre"])    # salida: "Juan"
print(mi_diccionario["edad"])      # salida: 25
print(mi_diccionario["ciudad"])    # salida: "Santiago"

# Agregar un nuevo par clave-valor:
mi_diccionario["profesión"] = "Ingeniero"

# Acceder al valor asociado a la nueva llave:
print(mi_diccionario["profesión"])  # salida: "Ingeniero"
```

```
Juan
25
Santiago
Ingeniero
```

Operaciones

Operaciones con Strings

Los strings (cadenas de texto) son uno de los tipos de datos más utilizados en Python y en programación en general. Los strings son objetos que contienen una secuencia de caracteres, como palabras, frases o incluso párrafos completos.

Pueden ser definidos utilizando comillas simples (' ') o dobles (" "). Algunas operaciones que se pueden realizar con strings son la concatenación (+), la multiplicación (*), el acceso a

caracteres individuales (mediante el operador []), la longitud de un string (mediante la función **len()**), y la búsqueda de subcadenas (mediante la función **find()**).

Algunos métodos para strings:

- **lower()**: Convierte todos los caracteres de una cadena a minúsculas.
- **upper()**: Convierte todos los caracteres de una cadena a mayúsculas.
- **replace(sub, new_sub)**: Reemplaza todas las ocurrencias de sub con new_sub en una cadena.
- **split(sep)**: Divide una cadena en una lista de subcadenas utilizando el separador sep.
- **strip()**: Elimina los espacios en blanco al inicio y al final de una cadena.

```
texto = "Hola Mundo"
texto = texto.lower()
print(texto)
```

```
hola mundo
```

```
texto = "Hola Mundo"
texto = texto.upper()
print(texto)
```

```
HOLA MUNDO
```

```
texto = "Hola Mundo"
texto = texto.replace("Mundo", "Python")
print(texto)
```

```
Hola Python
```

```
texto = "Hola,Mundo"
lista = texto.split(",")
print(lista)
```

```
['Hola', 'Mundo']
```

```
texto = "   Hola Mundo    "
texto = texto.strip()
print(texto)
```

```
Hola Mundo
```

¡Prueba estos métodos en Python escribiendo tu propio código!
¡Desafíate y sigue aprendiendo!

Operaciones aritméticas

En Python, los tipos de datos numéricos como int o float, entre otros, no tienen métodos propios. Sin embargo, se pueden utilizar operadores y funciones para realizar operaciones aritméticas y matemáticas con estos tipos de datos. Por ejemplo, los operadores se utilizan para realizar sumas, restas, multiplicaciones y divisiones con números. Los operadores aritméticos en Python son:

- Suma (+)
- Resta (-)
- Multiplicación (*)
- División (/)
- División entera (//)
- Módulo o residuo (%)

- Potencia (**)

Vamos a probar dos funciones útiles en Python: **abs(x)** y **round(x, n)**.

- **La función abs(x):** devuelve el valor absoluto de un número, es decir, su valor numérico sin signo. Por ejemplo, abs(-5) devuelve 5.

- **La función round(x, n):** redondea un número x a n dígitos de precisión. Por ejemplo, round(3.14159, 2) devuelve 3.14.

¡Anímense a probar estas funciones en su propio código y descubrir todas las formas en que pueden ser útiles en su trabajo de programación!

Operaciones con Listas

Concatenación

Las listas son uno de los tipos de datos más útiles en Python. Las listas son objetos que contienen una colección ordenada de elementos, que pueden ser de cualquier tipo: strings, números, otros objetos, incluso otras listas. En Python, las listas se representan utilizando corchetes y los elementos se separan por comas.

Se pueden agregar nuevos elementos a una lista utilizando el **operador de concatenación** "**+**"

```
mi_lista = [1, 2, 3]
mi_lista = mi_lista + [4]
print(mi_lista)
```

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

o utilizando el método **append()**:

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

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

La función **append()** se utiliza para agregar un elemento al final de una lista existente, modificando el objeto original. Por otro lado, el operador "**+**" se utiliza para concatenar dos listas y crear una nueva lista con los elementos de ambas listas.

Acceso a elementos

```
numeros = [1, 2, 3, 4, 5]
primer_elemento = numeros[0]
print(primer_elemento)
```

```
1
```

Modificación de elementos

```
numeros = [1, 2, 3, 4, 5]
numeros[0] = 6
print(numeros)
```

```
[6, 2, 3, 4, 5]
```

Longitud

```
numeros = [1, 2, 3, 4, 5]
longitud = len(numeros)
print(longitud)
```

```
5
```

Las listas son una herramienta fundamental para cualquier programador, y conocer las operaciones que se pueden realizar con ellas te permitirá realizar tareas más complejas y resolver problemas más avanzados en tus programas de Python.

¡Sigue explorando y aprendiendo!

Operaciones con Diccionarios

Existen dos formas principales de trabajar con diccionarios en Python: usando corchetes y usando el método .get(). Usando corchetes, puedes acceder a los valores del diccionario a través de sus claves. Por ejemplo, si tienes un diccionario llamado **mi_diccionario** y quieres acceder al valor correspondiente a la clave '**nombre**', puedes hacerlo así:

```
mi_diccionario = {'nombre': 'Juan', 'edad': 25, 'ciudad': 'Santiago'}
print(mi_diccionario['nombre'])
```

```
Juan
```

En Python, el método **.get()** se utiliza para acceder a un valor de un diccionario, pero en lugar de arrojar un error si la clave no existe, devuelve un valor predeterminado en su lugar.

```
mi_diccionario = {'nombre': 'Juan', 'edad': 25, 'ciudad': 'Santiago'}
print(mi_diccionario.get('telefono', 'No disponible'))
```

```
No disponible
```

Para agregar elementos a un diccionario, simplemente debes usar la sintaxis de corchetes y asignar un valor a una nueva clave. Por ejemplo, si quieres agregar una nueva clave '**telefono**' en el diccionario **mi_diccionario**, puedes hacer lo siguiente:

```
mi_diccionario = {'nombre': 'Juan', 'edad': 25, 'ciudad': 'Santiago'}
mi_diccionario['telefono'] = "+569 123456789"

print(mi_diccionario.get('telefono', 'No disponible'))
```

```
+569 123456789
```

Cada línea de código que escribas te acerca más a tus objetivos.

Importación de librerías

Las librerías son conjuntos de módulos y funciones predefinidas que puedes utilizar para ampliar las capacidades de Python. Para utilizar una librería en tu código, primero debes importarla como se muestra.

```
import random

# Generar un número entero aleatorio entre 0 y 9
num_aleatorio = random.randint(0, 9)
print("El número aleatorio es:", num_aleatorio)

# Generar un número decimal aleatorio entre 0 y 1
num_decimal_aleatorio = random.random()
print("El número decimal aleatorio es:", num_decimal_aleatorio)
```

```
El número aleatorio es: 6
El número decimal aleatorio es: 0.837069227410973
```

- **randint(a, b)**: Esta función genera un número entero aleatorio entre a y b, ambos inclusive. En el ejemplo, hemos generado un número entero aleatorio entre 0 y 9 utilizando esta función y lo hemos guardado en la variable **num_aleatorio**.

- **random()**: Esta función genera un número decimal aleatorio entre 0 y 1. En el ejemplo, hemos generado un número decimal aleatorio utilizando esta función y lo hemos guardado en la variable **num_decimal_aleatorio**.

Finalmente, hemos utilizado la función print para mostrar los números aleatorios generados en la pantalla.

Jupyter Notebook

Jupyter Notebook es una herramienta interactiva que permite la creación y el uso de notebooks, es decir, documentos que combinan código, texto enriquecido y elementos multimedia, en los que es posible ejecutar fragmentos de código en tiempo real y visualizar la salida de los mismos.

Cuando trabajamos en **Jupyter Notebook**, cada celda se ejecuta de forma independiente. Esto significa que podemos definir variables y objetos en una celda y luego usarlos en

celdas posteriores. Esto significa que podemos acceder y utilizar los objetos que hemos creado en cualquier momento mientras la sesión de Jupyter Notebook esté activa.

Google Colab

Google Colab es una plataforma gratuita de Google que permite el uso de Notebooks de Jupyter para la ejecución de código en Python. Al igual que **Jupyter Notebook**, es una herramienta interactiva que permite la combinación de código, texto y visualizaciones en un solo documento.

Sin embargo, a diferencia de **Jupyter Notebook**, que se ejecuta localmente en la computadora del usuario, **Google Colab** se ejecuta en la nube, lo que significa que no se necesita una instalación local de Python o Jupyter. Además, **Google Colab** ofrece recursos computacionales en línea, lo que significa que el usuario puede ejecutar código en hardware más potente que su propia computadora.

Google Colab también permite la colaboración en tiempo real, lo que significa que varias personas pueden trabajar juntas en un mismo documento, lo que es ideal para el trabajo en equipo y la enseñanza de programación.

¡Manos a la obra! - Crea tu notebook

Revisa el documento llamado **Tutorial MarkDown** ubicado en el LMS y practica el uso de marcas en Jupyter Notebook o Colab.

Reflexiona:

- ¿Por qué es importante comprender el concepto de entrada y salida de datos en Python?
- ¿Para qué sirven las variables en Python?
- ¿Cuál es la diferencia entre un notebook de Jupyter y un archivo de Python normal?