Notas para la Sesión 1: Fundamentos de programación en Python

1. Introducción a Python

Historia y características

Python es un lenguaje de programación de alto nivel creado por Guido van Rossum en 1989, con un lanzamiento oficial en 1991. Se caracteriza por su sintaxis clara y legible, lo que permite que el código sea fácil de leer y mantener. Es un lenguaje interpretado y orientado a objetos, con una amplia biblioteca estándar y una gran cantidad de módulos externos que lo hacen muy versátil. Python se utiliza en diversos campos, como desarrollo web, análisis de datos, inteligencia artificial y más.

Instalación y configuración del entorno de trabajo

Para instalar Python, se debe descargar el instalador desde la página oficial https://www.python.org/downloads/ y seguir las instrucciones. Una vez instalado, se puede verificar la instalación ejecutando `python --version` o `python3 --version` en la línea de comandos. Para trabajar con Python, se recomienda instalar un entorno de desarrollo integrado (IDE) como Visual Studio Code o PyCharm, o utilizar Jupyter Notebook para análisis de datos.

Uso del intérprete y ejecución de scripts

El intérprete de Python se puede ejecutar en la línea de comandos escribiendo `python` o `python3`. Para ejecutar un script de Python, se debe crear un archivo con la extensión `.py` (por ejemplo, `hello.py`) y escribir el código deseado. Para ejecutar el script, se debe ingresar en la línea de comandos `python hello.py` o `python3 hello.py`.

2. Variables y tipos de datos básicos

Nombres de variables y asignación

En Python, las variables se asignan utilizando el operador `=`, y se pueden nombrar utilizando letras, números y guiones bajos, siempre y cuando no comiencen con un número. Por ejemplo:

```
x = 5
name = "John"
is_happy = True
```

Tipos de datos: int, float, str y bool

Python admite varios tipos de datos básicos, como enteros (`int`), números de punto flotante (`float`), cadenas de texto (`str`) y valores booleanos (`bool`):

```python
integer = 42
floating_point = 3.14
string = "Hello, World!"
boolean = True
```

Conversión entre tipos de datos

Se pueden convertir variables de un tipo de dato a otro utilizando funciones como `int()`, `float()` y `str()`:

```python
int_to_float = float(42)
float_to_int = int(3.14)
int_to_str = str(42)
```

3. Operaciones básicas y operadores

Aritméticos

Python admite operaciones aritméticas básicas como suma, resta, multiplicación, división, división entera, módulo (resto) y exponente:

```python
a = 10
b = 3

addition = a + b
subtraction = a - b
multiplication = a * b
division = a / b
integer_division = a // b
modulo = a % b
exponentiation = a ** b
```

Relacionales

Los operadores relacionales permiten comparar valores y devuelven un valor booleano (`True` o `False`). Estos operadores incluyen igualdad, desigualdad, mayor que, menor que, mayor o igual que y menor o igual que:

```python
a = 10
b = 3

equals = a == b          # Verifica si a es igual a b
not_equals = a != b      # Verifica si a es distinto de b
greater_than = a > b     # Verifica si a es mayor que b
less_than = a < b        # Verifica si a es menor que b
greater_equals = a >= b  # Verifica si a es mayor o igual que b
less_equals = a <= b     # Verifica si a es menor o igual que b
```

Lógicos

Los operadores lógicos permiten combinar expresiones booleanas y devuelven un valor booleano (`True` o `False`). Los operadores lógicos básicos son AND, OR y NOT:

```
a = True
b = False

and_operator = a and b  # Devuelve True si ambos a y b son True, de lo contrario, devuelve False
or_operator = a or b    # Devuelve True si al menos uno de a o b es True, de lo contrario, devuelve False
not_operator = not a    # Devuelve True si a es False, y viceversa
```

Estos operadores son útiles para combinar condiciones en estructuras de control como `if`, `elif` y `else`. Por ejemplo, se pueden utilizar para verificar si un número es divisible por dos números diferentes:

```
x = 30

if x % 2 == 0 and x % 3 == 0:
    print("x es divisible por 2 y 3")
else:
    print("x no es divisible por 2 y 3")
```

4. Funciones y manejo de cadenas (strings)

Definición y llamado de funciones

Las funciones son bloques de código reutilizables que se definen con la palabra clave `def`, seguida del nombre de la función, paréntesis y dos puntos. Dentro de los paréntesis, se pueden especificar argumentos que la función recibirá al ser llamada. Para llamar a una función, se utiliza su nombre seguido de paréntesis y los argumentos correspondientes.

```
def greet(name):
    print("Hello, " + name)

greet("John")
```

En el ejemplo anterior, se define una función llamada `greet` que recibe un argumento `name`. La función imprime un saludo utilizando el valor del argumento.

Argumentos y valores de retorno

Las funciones pueden recibir múltiples argumentos y devolver valores utilizando la palabra clave `return`.

```python
def add(a, b):
    return a + b

result = add(5, 3)
print(result)
```

En este ejemplo, la función `add` recibe dos argumentos, `a` y `b`, y devuelve su suma. Al llamar a la función y asignar su valor de retorno a la variable `result`, se puede imprimir el resultado.

Strings: métodos y manipulación

Los strings son secuencias de caracteres y son uno de los tipos de datos más comunes en Python. Existen varias operaciones y métodos para trabajar con strings.

```python
text = "Hello, World!"

length = len(text)  # Obtiene la longitud del string
uppercase = text.upper()  # Convierte el string a mayúsculas
lowercase = text.lower()  # Convierte el string a minúsculas
capitalized = text.capitalize()  # Convierte la primera letra del string a mayúscula
replaced = text.replace("World", "Python")  # Reemplaza una subcadena por otra
substring = text[0:5]  # Obtiene una subcadena a partir de un rango de índices
```

Formateo de strings

El formateo de strings permite insertar valores de variables dentro de un string. Existen varias formas de hacerlo en Python.

```python
name = "John"
age = 30

formatted_string = f"My name is {name} and I am {age} years old."
print(formatted_string)

formatted_string2 = "My name is {} and I am {} years old.".format(name, age)
print(formatted_string2)
```

En el primer ejemplo, se utiliza una f-string, que permite incluir expresiones entre llaves directamente en el string. En el segundo ejemplo, se utiliza el método `format()` para insertar los valores de las variables en las llaves del string.

Concatenación de strings

La concatenación de strings es el proceso de unir varios strings en uno solo. En Python, se puede utilizar el operador `+` para concatenar strings.

```python
greeting = "Hello"
name = "John"

message = greeting + ", " + name + "!"
print(message)
```

Métodos útiles para trabajar con strings

```python
text = "  Hello, World!  "

stripped = text.strip()  # Elimina espacios al inicio y al final
print(stripped)

is_alpha = text.isalpha()  # Verifica si todos los caracteres son alfabéticos
print(is_alpha)

is_digit = text.isdigit()  # Verifica si todos los caracteres son dígitos
print(is_digit)

split_text = text.split(",")  # Separa el string en una lista basado en un delimitador
print(split_text)

joined_text = ", ".join(split_text)  # Une una lista de strings en un solo string usando un delimitador
print(joined_text)
```

En el bloque anterior, se presentan varios métodos útiles para trabajar con strings:

- `strip()`: elimina los espacios en blanco al inicio y al final de un string. Es útil para limpiar datos de entrada o texto obtenido de archivos o bases de datos.
- `isalpha()`: verifica si todos los caracteres en el string son alfabéticos. Puede ser útil para validar datos ingresados por el usuario.
- `isdigit()`: verifica si todos los caracteres en el string son dígitos. Al igual que `isalpha()`, puede ser útil para validar datos ingresados por el usuario.
- `split(delimiter)`: separa un string en una lista de strings, utilizando el delimitador especificado. Es útil para procesar datos separados por comas (CSV) o cualquier otro formato de texto estructurado.
- `join(delimiter)`: une una lista de strings en un solo string, utilizando el delimitador especificado. Es útil para crear cadenas de texto a partir de listas o para exportar datos en un formato específico.

Listas y estructuras de datos básicas

Las listas son una estructura de datos fundamental en Python. Permiten almacenar y manipular colecciones de elementos. Se pueden crear listas utilizando corchetes y separando los elementos con comas.

```
numbers = [1, 2, 3, 4, 5]
names = ["Alice", "Bob", "Charlie"]
mixed_list = [42, "Hello", 3.14, True]
```

Acceso y modificación de elementos en listas

Para acceder a los elementos de una lista, se utiliza el índice del elemento entre corchetes. Los índices en Python comienzan en 0.

```
names = ["Alice", "Bob", "Charlie"]

first_name = names[0]  # Alice
second_name = names[1]  # Bob
last_name = names[-1]  # Charlie

names[1] = "Bobby"  # Modifica el segundo elemento de la lista
```

Métodos y operaciones útiles para trabajar con listas

Las listas en Python tienen varios métodos y operaciones que facilitan su manejo:

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

length = len(numbers)  # Obtiene la longitud de la lista
sum_numbers = sum(numbers)  # Calcula la suma de todos los elementos de la lista

numbers.append(6)  # Agrega un elemento al final de la lista
numbers.extend([7, 8, 9])  # Agrega múltiples elementos al final de la lista
numbers.insert(0, 0)  # Inserta un elemento en una posición específica

numbers.pop()  # Elimina y devuelve el último elemento de la lista
numbers.pop(0)  # Elimina y devuelve el elemento en la posición especificada

numbers.sort()  # Ordena la lista de menor a mayor
numbers.reverse()  # Invierte el orden de la lista
```

Ejercicios propuestos

1. Crear una función que reciba dos números y devuelva su multiplicación.
2. Crear una función que reciba un string y devuelva el string invertido.
3. Crear una función que reciba una lista de números y devuelva la suma de todos los elementos.