Anotaciones

Introducción

Lenguaje de alto nivel destacado por su legibilidad y se caracteriza por

- Multiparadigma: soporta la programación imperativa, programación orientada a objetos y funcional
- Multiplataforma: se puede encontrar un intérprete de Python para los principales sistemas operativos
- Dinámica y fuertemente tipado: el tipo de las variables se deciden en tiempo de ejecución. Pero no se puede usar una variable en un contecto fuera de su tipo sin efectuar una conversión
- Interpretado: el código no se complica a lenguaje máquina, sino que se ejecutan las intrucciones a medida que se las lee

---

Comentarios en línea, en bloque y docstrings

Python permite comentarios en linea con # en bloquecon triples comillas simples. También existen los docstring que son usados para generar la documentación de un programa en forma automática con alguna herramienta externa

```
´´´Comentario en bloque
´´´
# Comentario en línea #
```

Tipos de datos en Python

Hay una gran variedad de datos en Python, entre ellos algunos que son considerados tipos de datos básicos son

- int
- float
- string
- bool

Familia	Tipos
Texto	str
Numéricos	int, float, complex
Colecciones	list, tuple, range
Mapeos	dict
Conjuntos	set, frozenset
Booleanos	bool
Binarios	bytes, bytearray, memoryview

Cadena de caracteres

Está compuesta por cero o más caracteres. Las cadenas pueden delimitarse con comillas simples o dobles

```
# Definición de cadenas usando comillas dobles
dia1 = "Lunes"
x = "" # x es un string de longitud cero
# Definición de cadenas usando comillas simples
dia2 = 'Martes'
z = "121" # z contiene dígitos, pero es un string
```

len()

Retorna la cantidad de caracteres que contiene un string

```
nombre='Codo a Codo'
print(len(nombre)) #se imprime 11
```

.upper()

Devuelve la cadena co todos sus caracteres en mayúsculas

.lower()

Devuelve la cadena con todos sus caracteres en minúsculas

.capitalize()

Devuelve la cadena con su primer mayúscula y todos los demás en minúsculas

Operadores

Un operador es un carácter o conjunto de caracteres que actúa sobre una, dos o más variables para llevar a cabo una operación con un resultado determinado. Los operadores son de asignación, aritméticos, de pertenencia, relacionales, lógicos

- Operadores aritméticos

Operador	Descripción
+	**Suma:** Suma dos operandos.
-	**Resta:** Resta al operando de la izquierda el valor del operando de la derecha. Utilizado sobre un único operando, le cambia el signo.
*	**Multiplicación:** Producto de dos operandos.
/	**División:** Divide el operando de la izquierda por el de la derecha (el resultado siempre es un float).
%	**Operador módulo:** Obtiene el resto de dividir el operando de la izquierda por el de la derecha.
//	**División entera:** Obtiene el cociente entero de dividir el operando de la izquierda por el de la derecha.
**	**Potencia:** El resultado es el operando de la izquierda elevado a la potencia del operando de la derecha.

- Operadores de pertenencia: se usan para comprobar si un caracter o cadena se encuentra dentro de otra

OPERADOR	DESCRIPCION
in	Devuelve **True** si el valor se encuentra en una secuencia; **False** en caso contrario.
not in	Devuelve **True** si el valor no se encuentra en una secuencia; **False** en caso contrario.

```
cadena = "Codo a Codo"
print("C" in cadena) # True
print("n" in cadena) # False
print("Codo" in cadena) # True
print("A" not in cadena) # True
print("o" not in cadena) # False
```

- Operadores relaciones: los operadores relaciones comparan dos o más valores, su resultado será true o false

Operador	Descripción
>	Mayor que. `True` si el operando de la izquierda es estrictamente mayor que el de la derecha; `False` en caso contrario.
>=	Mayor o igual que. `True` si el operando de la izquierda es mayor o igual que el de la derecha; `False` en caso contrario.
<	Menor que. `True` si el operando de la izquierda es estrictamente menor que el de la derecha; `False` en caso contrario.
<=	Menor o igual que. `True` si el operando de la izquierda es menor o igual que el de la derecha; `False` en caso contrario.
==	Igual. `True` si el operando de la izquierda es igual que el de la derecha; `False` en caso contrario.
!=	Distinto. `True` si los operandos son distintos; `False` en caso contrario.

- Operadores lógicos

OPERADOR	DESCRIPCIÓN	USO
and	Devuelve True si **ambos** operandos son True	a and b
or	Devuelve True si **alguno** de los operandos es True	a or b
not	Devuelve True si el operandos False, y viceversa	not a

Expresiones

Una expresión es una unidad de código que devuelve un valor y está formada por una combinación de operandos y operadores

Una sentencia define una acción. Puede contener alguna expresión. Son las instrucciones que componen el código de un programa y determinan su comportamiento

print()

Permite mostrar datos en la terminal. Recibe como parámetros variable y/o literales, separados por comas

```
print(¨Hola mundo!¨)
```

input()

Proporciona un mecanismo para que el usario introduzca datos en nuestro programa

```
nombre= input(¨Ingrese su nombre:¨)
```

Estructuras de control

- if

```
nota = float(input("Ingrese la calificación: "))
if nota >= 7:
print("Aprobado.")

# Otro ejemplo #
edad = float(input("Ingrese su edad: "))
if edad >= 18:
print("Puedes pasar.")
else:
print("No admitido.")
```

- while

```
cont= 1
suma= 0
while cont <= 5:
num= int(input("Ingrese un número: "))
suma = suma + num # Acumulamos, es equivalente suma += num
cont = cont + 1 # Incrementamos, es equivalente cont += 1
print("La suma es:", suma)
print("El promedio es:", suma/cont)
```

- for

```
suma= 0
for cont in range(5):
num= int(input("Ingrese un número: "))
suma = suma + num # Acumulamos, es equivalente suma += num
print("La suma es:", suma)
print("El promedio es:", suma/(cont+1))
```

Listas

Secuencia ordenada de elementos. Pueden tener elementos del mismo tipo o combinar distintos tipos de datos.

Las listas en Python son un tipo contenedor, compuesto y se usan para almacenar conjuntos de elementos relacionados

Las listas se crean asignando a una variable una secuencia de elementos encerrados entre corchetes [] y separados por comas

```
numeros = [1,2,3,4,5] #Lista de números
dias = ["Lunes", "Martes", "Miércoles"] #Lista de strings
elementos = [] #Lista vacía
sublistas = [ [1,2,3], [4,5,6] ] # lista de listas
```

Las listas se pueden imprimir directamente y el acceso a sus elementos se hace mediante subíndices

Recorrido de listas con for y while

```
lista = [2,3,4,5,6]
suma = 0
for i in range(len(lista)):
suma = suma + lista[
```

```
lista = [2,3,4,5,6]
suma = 0
i = 0
while i < len(lista):
suma = suma + lista[i]
i = i + 1
print(suma) # 20
```

Métodos

- max(): devuelve el mayor elemento de una lista
- min(): devuelve el menor elemento de una lista
- sum(): devuelve la suma de los elementos de una lista
- list(): convierte cualquier secuencia a una lista
- append(): agrega un elemento al final de la lista
- insert(pos,elemento) inserta un elemento en una posición determinada
- pop(posición) elimina un elemento de una posición determinada
- remove(valor) elimina un elemento en la lista
- index(valor): busca un valor y devuelve su posición
- count() devuelve la cantidad de repeticiones de un elemento
- reverse() invierte el orden de los elementos de una lista
- sort() ordena los elementos de lista, de menor a mayor
- clear() elimina todos los elementos de la lista

Diccionarios

Conjunto no ordenado de pares clave:valor. Estas claves son únicas, y si se intenta guardar un valor a una clave ya existente se pierde dicho valor

Las claves pueden ser cualquier elemento de tipo inmutable, se representan como una lista de pares separados por comas y encerrados entre llaves

```
diccionario = {'Juan': 56, 'Ana': 15}
# Creación: Por compresión
diccionario = {x: x ** 2 for x in (2, 4, 6)}
```

Accesos

- Usando keys() para acceder a las claves
- items() a las clave-valor

Tuplas

Conjunto de elementos separados por comas y encerrados entre paréntesis. Las tuplas son inmutables y en general contienen una secuencia heterogénea de elementos

---

Funciones