Introducción a Python

Python

- Lenguaje interpretado
- Sintaxis sencilla
- Uso de sangrado para delimitar bloques
- Multiparadigma: programación orientada a objetos, estructurada, funcional, etc.
- *Tipado* dínamico y fuerte

```
def fib(n):
    """Imprime la serie de Fibonacci hasta n."""
    a, b = 0, 1
    while a < n:
        print(a, end=' ')
        a, b = b, a+b
    print()                 # línea vacía en salida estándar


fib(2000)
```

Variables

- Comienzan por una letra o guion bajo
- Sólo contienen caracteres alfanuméricos o guiones bajos
- Sensibles a mayúsculas y minúsculas
- Son referencias a un objeto
- Tipos dinámicos

```
a = "cadena"
type(a)      # <class 'str'>
a =  5
type(a)      # <class 'int'>

b = "10 fichas"
print(a + b)     # error no existe + entre 'int' y 'str'

# identificadores apuntan al mismo objeto entero 1
a = b = c = 1
# comprueba si apuntan al mismo objeto
a is b
# identificadores apuntan a objetos distintos
a, b, c = 1, 2, "juan"
```

Tipos de datos

- **Cadena de caracteres**: str
- **Numéricos**: int, float, complex
- **Secuencias**: list, tuple, range
- **Asociativos**: dict
- **Conjuntos**: set, frozenset
- **Booleanos**: bool
- **Binarios**: bytes, bytearray, memoryview

Listas

- Habitualmente contienen elementos homogéneos
- Permiten duplicados
- Mutables
- Utilizan corchetes []

```
lista = list()
lista.append(4)
lista.append(9)
elemento = lista[0]
elemento = lista[-1]

otra = [1 ,3 ,8 ,9 ,20 ,22]
print(otra[2:4])          # imprime 8 y 9
print(otra[0:5:2])        # imprime 1, 8, 20
len(otra)                 # tamaño
```

Tuplas

- Habitualmente contienen elementos heterogéneos
- Permiten duplicados
- Inmutables
- Utilizan paréntesis ()

```
t = (12345, 54321, 'hello!')

x, y, z = t

t[0] = 234    # error
```

Diccionarios

- A partir de la versión 3.7 son estructuras ordenadas
- Pares clave : valor
- Sin claves duplicadas
- Mutables

```
diccionario = {'manzana': 4098, 'platano': 4139}
diccionario['naranja'] = 4127

diccionario.keys()
diccionario.values()

'naranja' in diccionario
'naranja' not in diccionario
```

Estructuras de control

```
a = 200

b = 33


if b > a:

  print("b es mayor que a")

else:

  print("b es menor que a")
```

```
for numero in range(2, 6):
  print(numero)

frutas = ["fresa", "manzana", "platano"]
for fruta in frutas:
  print(fruta)
for indice in range(0, len(frutas)):
  print(frutas[indice])

diccionario = {'manzana': 4098, 'platano': 4139}
for clave in diccionario:
  print(clave, ":", diccionario[clave])
```

Funciones

- Palabra def introduce la definición de una función
- Siempre devuelven un valor. Si no hay sentencia return, devuelven None

```
def fib2(n):
    """Devuelve lista con serie de Fibonacci hasta n."""
    resultado = []
    a, b = 0, 1
    while a < n:
        resultado.append(a)
        a, b = b, a+b
    return resultado

f100 = fib2(100)    # llamada
print(f100)         # imprime el resultado
```

Clases

- Palabra reservada class
- __init__ constructor de la clase
- self referencia a la instancia actual de la clase
- La palabra self es una convención
- self tiene que ser el primer parámetro de toda función de la clase

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

  def saludar(self):
    print("Hola, me llamo:", self.nombre, self.apellido)

sujeto = Persona("Juan", "Fernandez")
sujeto.saludar()
```

Clases

- Soporta herencia
- super() referencia a la clase padre sin nombrarla directamente

```
class Estudiante(Persona):
  universidad = “Granada”
  def __init__(self, nombre, apellido, año):
    super().__init__(nombre, apellido)
    self.graduadoEn = año

est = Estudiante("Miguel", "Hernández", 2021)
print(est.graduadoEn)
```

Módulos

```
from fibo import fib, fib2

fib(500)
```

```
# fibo.py
def fib(n):
    a, b = 0, 1
    while a < n:
        print(a, end=' ')
        a, b = b, a+b
    print()

def fib2(n):
    result = []
    a, b = 0, 1
    while a < n:
        result.append(a)
        a, b = b, a+b
    return result
```