Curso Python

Ismael Sallami Moreno

Índice general

1. Variables, condicionales y entrada/salida

En python el hecho de declarar variables, condicionales y demás es más simple que en los demás lenguajes de programación. Ejemplo:

```
variable= lo que sea
#nos reconoce el tipo de variable

if variable>5:
    #haz esto....

#manera de almacenar variables externas:
variable = float(input("el texto que queramos...")

# tambien podemos importar otros archivos y coger datos de ellos:

import tabla_perdiodica #(en formato py)
varible=tabla_periodica.Mg # coge el magnesio de la tabla periódica
```

- Como declarar una lista:
Por ejemplo podemos escribir el siguiente código:

```
frutas= ["manzana","fresa"]
```

2. Match

Es igual al "switch"del lenguaje c++, podemos combinarlo con que cada caso deba de cumplir con unas condiciones matemáticas:

```
numero = int(input("Por favor, introduce un numero entero:"))

match numero:
    case 0:
        print("El numero es un cero")
    case numero if numero % 2 == 0:
        print("El numero es par.")
    case numero if numero % 2 != 0:
        print("El numero no es par.")
    case _:
        print("Numero no reconocido")
```

3. Bucles For

En este apartado vamos a ver como funcionan los bucles for en el lenguaje de python. Posee ciertas diferencias en cuanto al lenguaje de c++.
Vamos a analizar un trozo de código:

```
 numeros = [1, 2, 3, 20, 4, 5]

for numero in numeros:
cuadrado = numero ** 2
print(f"El cuadrado de {numero} es {cuadrado}")
```

En este caso podemos ver como para recorrer un bucle for debemos de crear una variable (numero) que va a poseer cada valor de la estructura acumulativa de izquierda a derecha. El hecho de usar **f** en print es para darle el formato adecuado para que pueda imprimir directamente en pantalla las variables a las que se llama entre corchetes.
Ejemplo: Cálculo promedio de una lista de números.

```
numbers=[0,1,2,3,4,5,6,7,8,9]
#we use the function called len(numeros), but we can use the other variable also.
add=0
for number in numbers:
    add+=number
    print(f"we have been able to add the number: {number}")

print("Process finished ...")

print(f"The final result is --> {add/len(numbers)}")
```

4. Bucle while

Ejemplo: realizar el conteo de un número comenzando por 1 hasta llegar a dicho número:

```
numero=int(input("Introduce the number to start the count: "))

contador=1
print("\n Starting the cont... \n")
while contador <= numero:
print(f"El conteo va por --> {contador}")
contador += 1

print("\n count finished ...")
```

5. Cadenas

Es bastante similar a como se trabaja con cadenas en c++.
Ejemplo y análisis:

```
nombre = "marco"
apellido = "MENDOZA"
frace = "Hola Esta Es una Frace"

longitud = len(frace)
print(longitud)

print(apellido[6])

palabras = frace.split()
mayusculas = frace.upper()
print(mayusculas)
texto = apellido.lower()
print(texto)

mensaje = "Hola, Mundo"
print(mensaje)
cambio = mensaje.replace("Hola", "Marco")
print(cambio)

for x in apellido:
print(x)
```

- split: divide una cadena compuesta por varias palabras en cada una de las palabras.
- upper/lower: pone en mayúscula/minúscula la cadena completa.
- replace("1","2"): reemplaza **1** que se encuentra en la cadena donde se aplica el replace por el **2** que

es lo que se quiere poner.

6. Tuplas

Se conoce por tuplas en python a un conjunto de elementos los cuales no se pueden modificar. Pueden ser del cualquier tipo y estos se pueden mezclar en la misma tupla. Se declaran entre paréntesis, y ya dentro de esos paréntesis se introducen las variables separadas por comas.

Ejemplo de código:

```
personas = (("Juan", 25), ("Maria", 16), ("Carlos", 20))
for nombre, edad in personas:
    if edad < 18:
        print(nombre, edad)


numeros = (10, 20, 30, 40, 50, 100, 5000, 100000)

suma = sum(numeros)

print("La suma de los numeros es:", suma)
```

La función sum(), suma los contenidos de la tupla entre sí.

7. Listas

Ejemplo de código:

```python
# Lista de números enteros
numeros = [1, 2, 3, 4, 5]

# Lista de cadenas de texto
frutas = ["manzana", "banana", "cereza"]

# Lista mixta con diferentes tipos de datos
mixta = [1, "hola", 3.14, True]

print(numeros[0])
print(frutas[1])

numeros[2] = 9
print(numeros[2])

numeros.append(8)
print(numeros)

frutas.append("coco")
print(frutas)

del numeros[2]
del frutas[0]
print(numeros)
print(frutas)


for fruta in frutas:
    print(fruta)

suma = sum(numeros)
print(suma)
```

Figura 7.0.1: *Example about of use of list in Python*

La función **append** lo que hace es añadir al final de lista el elemento que se le pasa como parámetro a dicha función.

La función **del** posee la sintaxis que se muestra en el ejemplo y elimina lo que se sitúa a la derecha de dicha función.

8. Diccionarios

Lo diccionarios se pueden definir como estructuras con las que se pueden almacenar diversos tipos de datos clasificados por una clave, podiendo acceder a cada una de las claves.

```
personas = {
    "persona1":{
        "nombre": "Juan",
        "edad": 30,
        "ciudad": "Madrid"
    },
    "persona2":{
        "nombre": "Maria",
        "edad": 28,
        "ciudad": "Barcelona"
    },
    "persona3":{
        "nombre": "Carlos",
        "edad": 35,
        "ciudad": "Valencia"
    }

}

datos = personas["persona1"]
datos2 = personas["persona2"]
datos3 = personas["persona3"]

print(datos["nombre"])
print(datos2["edad"])
print(datos3["ciudad"])



persona1  = {
    "nombre": None,
    "edad": None,
    "direccion": None,
    "telefono": None,
}

persona1["nombre"]= input("Introduce un nombre:")
persona1["edad"]= input("Introduce tu edad:")
persona1["direccion"]= input("Introduce tu direccion:")
persona1["telefono"]= input("Introduce tu telefono:")

print(persona1["nombre"], "tiene", persona1["edad"], "años, vive en", persona1["direccion"], "y su numero de telefono es", persona1["telefono"])
```

Figura 8.0.1: *Example about dictionary in Phyton*

Como podemos ver el diccionario persona almacena a su vez más diccionarios, creando una estructura de datos aún más compleja y funcional. Esta la podemos modificar de la manera que queramos, como se puede ver en el ejemplo.

9. Funciones

Las funciones en python se declaran con **def**. Aqui exponemos un breve ejemplo de como se declaran y su uso:

```python
def suma(a, b):
    resultado = a+b
    return resultado

numero1 = int(input("Introduce un numero:"))
numero2 = int(input("Introduce un segundo numero:"))

resultado = suma(numero1, numero2)
print(resultado)
def espar(numero):
    if numero % 2 == 0:
        return True
    else:
        return False

numero = int(input("Introduce un numero:"))
if espar(numero) == True:
    print(f"{numero} es un numero par")
else:
    print(f"{numero} es un numero inpar")

def listanumeros(lista):
    maximo = max(lista)
    return maximo

numeros = [10, 40, 50, 55, 6]
valor = listanumeros(numeros)
print("El valor maximo almacenado en la  lista es:", valor)
```

Figura 9.0.1: *Function in phyton*

10. Módulos

Se conoce por módulo en python por distintos archivos denominados módulos los cuales poseen ciertas funciones que al importarlas con **import** puedes usar las funciones de dicho modulo en otro archivo .py .
Ejemplo:

```python
#tarea.py

#-------------------------
def es_par(numero):
    if numero % 2 == 0:
        return True
    else:
        return False
#-------------------------

import Tarea
dato = int(input("Ingresa un número: "))

formula = Tarea.es_par(dato)
print(formula)
```

Figura 10.0.1: *Example about import module in python*

11. Clases y objetos

```python
class calculadora1:
    def __init__(self, numero):
        self.resultado = numero

    def sumar(self, numero):
        self.resultado += numero

    def restar(self, numero):
        self.resultado -= numero

    def multiplicar(self, numero):
        self.resultado *= numero

    def dividir(self, numero):
        if numero != 0:
            self.resultado /= numero
        else:
            print("Error: Nose puede dividir por cero.")

    def rusultado(self):
        return self.resultado

calculo = calculadora1(0)

calculo.sumar(5)
calculo.multiplicar(4)
calculo.restar(5)
calculo.dividir(2)

resultado = calculo.resultado
print("Resultado:", resultado)
```

Figura 11.0.1: *First example about class in phyton*

En el caso de phyton, el hecho de crear clases viene definido por como se expone en el ejemplo de la Figura 11.0.1. Podemos ver como se declaran diversos métodos de la clase como sumar, restar, ... El constructor se define con el nombre de **init entre dos barras bajas a los lados**, donde **self** *es el atributo por defecto* por decirlo de alguna manera y la cual podemos definir a nuestro antojo, como en el caso de la imagen que vemos que accede a la parte de **resultado de self** que no se ha declarado previamente.

```python
class calculadora2:
    def suma(self, num1, num2):
        return num1 + num2

    def resta(self, num1, num2):
        return num1 - num2

    def multiplicar(self, num1, num2):
        return num1 * num2

    def divicion(self, num1, num2):
        if num2 == 0:
            return "Error: No se puede dividir por cero."

num1 = float(input("Ingresa el primer numero:"))
num2 = float(input("Ingresa el segundo numero:"))

calc = calculadora2()

resulsuma = calc.suma(num1, num2)
resulresta = calc.resta(num1, num2)
resulmultiplicacion = calc.multiplicar(num1, num2)
resuldivicion = calc.divicion(num1, num2)

print("Suma:", resulsuma)
print("Resta:", resulresta)
print("Multiplicacion:", resulmultiplicacion)
print("Divicion:", resuldivicion)
```

Figura 11.0.2: *Second Example about class in python*

12. Herencia y Polimorfismo

12.1 Herencia

Básicamente el concepto de herencia en python hace referencia al hecho de que una clase hijo puede heredar/usar la información de una clase padre para poder usarla/tratarla.

La clase hijo no tiene por que tener un método constructor/inicializador ya que usa el de la clase padre.

```python
class vehiculo:
    def __init__(self, marca, modelo):
        self.marca = marca
        self.modelo = modelo

    def arrancar(self):
        return f"{self.marca} {self.modelo} esta arrancando"

class coche(vehiculo):
    def acelerar(self):
        return f"{self.marca} {self.modelo} esta acelerando"

class motocicleta(vehiculo):
    def caballito(self):
        return f"{self.marca} {self.modelo} esta haciendo un caballito"

cochee = coche("Toyota", "Camry")
motocicletaa = motocicleta("Harley-Davidson", "Sportster")

print(f"Coche marca y modelo: {cochee.marca}, {cochee.modelo}")
print(f"Motocicleta marca y modelo: {motocicletaa.marca}, {motocicletaa.modelo}")

print(cochee.acelerar())
print(motocicletaa.arrancar())
```

Figura 12.1.1: *example about inheritance in phyton n*

12.2 Polimorfismo

Se trata del mero hecho de que se puedan utilizar objetos que se encuentren en distintas clases y métodos

```
print(cochee.arrancar())
print(motocicletaa.arrancar())
```

Figura 12.2.1: *Example about polymorphism in phyton. We can see how in the picture it use the methods of vehiculo in the class coche*

12.2.1 Métodos abstractos

Para definir un método abstracto en python, es deir, un método que no haga nada, sino que sea implementado por las clases hijos se debe de seguir una implmentación:

```
from abc import ABC, abstractmethod

class Animal(ABC):
    def __init__(self, nombre):
        self.nombre = nombre

    @abstractmethod
    def hablar(self):
        pass

class Perro(Animal):
    def hablar(self):
        return "Guau"

class Gato(Animal):
    def hablar(self):
        return "Miau"

# Crear instancias de las subclases
perro = Perro("Buddy")
gato = Gato("Whiskers")

# Mostrar el nombre y el sonido de cada animal
print(f"{perro.nombre} dice: {perro.hablar()}")
print(f"{gato.nombre} dice: {gato.hablar()}")
```

Figura 12.2.2: *Example about abstract methods*

13. Encapsulamiento

El encapsulamiento en python se trata de la forma en la que declaras las variables, si públicas o privadas. Si son públicas estas pueden ser modificadas por métodos externos a la propia clase, mientras que si es privada no.

```python
#Encapsulamiento

class encap:

    def __init__(self):
        self.__numero = 0

    def operacion(self):
        print(self.__numero + 20)

    def resultado(self):
        return self.__numero

ejemplo = encap()
ejemplo.operacion()
ejemplo.__numero = 100
print(ejemplo.resultado())
```

Figura 13.0.1: *Example about use of private variables. It declares with double low bar at the beginning*