MÓDULO: PROGRAMACIÓN PYTHON
PROFESOR: JAVIER GONZÁLEZ SANTURDE

Lección 2: Tipo de comentarios

En Python existen 2 tipos de comentarios: # y “””

1.- Comentario de 1 sola línea

```
#Imprimir por pantalla
print("------------ Usar print e input: entrada-salida datos --------")
```

2.- Comentario después de una instrucción de código

```
print(“Hola”)   #Esto es comentario
```

3.- Comentario de varias líneas

```
"""Declaración y asignación de valores a variables
Siempre el nombre de una variable debe comenzar por _ o una letra, no puede comenzar con un número"""“
Numero = 5
```

Lección 4: Función PRINT

Función PRINT – 3 formas para insertar variables:

1.- .format

```
nombre="Antonio"
edad=18
print("Buenos dias {}, feliz {} cumpleaños".format(nombre, edad))
print("Buenos dias {r1}, feliz {r2} cumpleaños".format(r2=edad,r1=nombre))

resultado = 10/3
print("resultado 10/3 = {}".format(resultado))
print("Resultado, formateado a 1 entero y 2 decimales, 10/3 = {r:1.2f}".format(r=resultado))
```

2.- f-string

```
nombre="Antonio"
edad=18
print(f"Buenos dias {nombre}, feliz {edad} cumpleaños")
```

CARACTERS ESPECIALES DENTRO DE LAS CADENAS DE TEXTO – USO del carácter "\"

```
print("Cadena con saltos: \n\t Primera cadena \n\t Segunda cadena")
```

3.- Operador % - Tipos de operadores: %c (carácter), %s (cadena texto), %d (entero), %f (número float), %o (número octal), %x (número hexadecimal)

```
numero1 = 5
numero2= 2.5
Cadena = "Jose"
print ("Hola %s, el resultado de dividir %d entre %f es 2" % (cadena, numero1, numero2)
```

Ejercicio 3: Cadena de Texto

Problema:

1. Crear una variable “numero1” que contiene el valor 5
2. Crear una variable “numero2” que contiene el valor 8
3. Crear una variable “suma” que contiene la suma de las 2 variables anteriores
4. Imprimir por pantalla el resultado utilizando la función “print” con .format y f-string

Solución:

```
numero1=5
numero2=8
suma = numero1 + numero2
Print(f”Resultado de {numero1} + {numero2} = {suma}”)
Print(“Resultado de {r1} + {r2} = {r3}”.format(r1=numero1,r2=numero2,r3=suma)
```

Lección 5: Entradas por teclado

Función INPUT:

```
#Solicitar datos por teclado
entrada = input ("Dime tu nombre:")
print("Tu nombre es:",entrada)

print(“Introduce tu nombre”)
entrada = input()
print(“Tu nombre es “ + entrada)

print(“Introduce tu nombre”)
print(“Tu nombre es “ + input())
```

Lección 3: Variables, números y conversiones de datos

Tipos de números en Python:

- **Enteros o int**: 0,1… 2exp31 y -1,-2,…-2exp31 procesadores de 32bits o 2exp64 en 64bits
- **Decimales o float**: el carácter separador es un "." punto. Ejemplo: 34.56

Tipos de operaciones aritméticas:

Suma: +
Resta: -
Multiplicación: *
División: /
División entera de 2 números: //
Módulo de la división o resto: %
Exponencial: **
Asigna el valor negativo a un número: -5

Ejemplos:

```
entero = 5
entero1 = -5
entero2 = 5**2
entero3 = 5%4
print("entero es de tipo:",type(entero))

float1 = 10.12
float2 = 10.11
print("float1 + float2 = ",float1,"+",float2,"=",float1+float2)
```

Ejercicio 5: Operadores aritméticos

Problema:

1. Crear una variable “nota1” que tenga el valor 6
2. Crear una variable “nota2” que tenga el valor 4
3. Crear una variable “nota3” que tenga el valor 7
4. Crear una variable “nota_media” que tenga el valor medio de las 3 notas anteriores
5. Crear una variable ”nota_final” que tenga el valor “aprobado” si la nota_media es igual o mayor a 5

Solución:

```
nota1=6
nota2=4
nota3=7
nota_media = (nota1 + nota2 + nota3)/3

if (nota_media >= 5):
   nota_final ="aprobado"

print(f"nota_media = {nota_media}.nota_final = {nota_final}")
```

Resultado:

```
nota_media = 5.666666666666667.nota_final = aprobado
```

Lección 6: Operadores de asignación

Operadores asignación: += -= *= /= **3

```
numero = 5
numero +=4          #Número valdrá 5+4 = 9

numero = 5
numero -=4          #Número valdrá 5-4 = 1

numero = 5
numero *=4          #Número valdrá 5*4 = 20

numero = 10
numero /=2          #Número valdrá 10/2 = 5

numero = 2
numero **=3         #Número valdrá 2 exp 3 = 8
```

Lección 7: Operadores de comparación

Operadores comparación: == != > < >= <=

```
cadena = “hola”
cadena == “hola”      # Esta expresión da el valor TRUE

cadena = “Hola”
cadena != “Hola”      # Esta expresión da el valor FALSE

Numero = 5
Numero > 2            # Esta expresión da el valor TRUE

Numero = 5
Numero < 2            # Esta expresión da el valor FALSE

Numero = 5
Numero >= 5           # Esta expresión da el valor TRUE, pues numero es mayor o igual que 5

Numero = 5
Numero >= 5           # Esta expresión da el valor FALSE, pues numero es menor o igual que 5
```

Lección 8: Operadores lógicos

Operadores lógicos: and or not

numero1 = 5
numero2 = 6
numero3 = 7
numero4 = 8

(numero1 < numero2) **and** (numero3 < numero4) #Esta expresión da el valor TRUE

(numero1 < numero2) **and** (numero3 > numero4) #Esta expresión da el valor FALSE

(numero1 < numero2) **or** (numero3 > numero4) #Esta expresión da el valor TRUE

not (numero1 < numero2) #Esta expresión da el valor FALSE

Lección 9: Operadores identidad y de pertenencia

Operadores identidad: is is not

Frutas1 = [“manzanas”,”peras”]
Frutas2 = frutas1
Frutas1 is frutas2 # Esta expresión es TRUE
Frutas1 is not Frutas2 # Esta expresión es FALSE

Operadores de pertenencia: in not in

Frutas1 = [“manzanas”,”peras”]
Frutas2 = “manzanas”

Frutas2 **in** Frutas1 #Esta expresión es TRUE pues “manzanas” está dentro de Frutas1

Frutas3 = “melocotón”
Frutas3 **not in** Frutas1 #Esta expresión es TRUE pues “melocotón” NO está dentro de Frutas1

Ejercicio 6: Operadores de comparación

```
1. Crea una variable "minimo" con el valor 20
2. Crea una variable "maximo" con el valor 500
3. Recoge un valor del teclado y almacénalo en la variable "dato"
4. Convierte la variable "dato" en un número y almacénalo en la variable "numero"
5. Si el "numero" es menor que el valor de "minimo", mostrar el texto "Valor bajo"
6. Si el "numero" es mayor que el valor de "maximo", mostrar el texto "Valor alto"
7. Si el "numero" está entre el valor de "minimo" y "maximo", mostrar "Valor medio"
```

Solución:

```
Minimo = 20
Maximo= 500
Dato = input ()
Numero = int(dato)
if (Numero < Minimo):
        print("Valor bajo")
elif (Numero > Maximo):
        print("Valor alto")
else:
        print("Valor medio")
```

Ejercicio 7: Operadores de pertenencia

```
1. Crear una variable "numeros" con la lista de los números del uno al 10 (ambos incluidos)
2. Mostrar el valor de la variable "numeros"
2. Recoger un dato del teclado y almacenarlo en la variable "dato"
3. Convertir "dato" en numérico y almacenarlo en la variable "numero".
4. Si el valor de "numero" está en la lista de números, mostrar el mensaje "Si"
5. Si el numero introducido no está en la lista de números, mostrar el mensaje "No"
```

Solución:

```
numeros = [1,2,3,4,5,6,7,8,9,10]
print(numeros)

dato = input("Dime un número:")
numero = int(dato)

if (numero in numeros):
   print (f"El número introducido {numero} pertenece a {numeros}")
else:
   print (f"El número introducido {numero} no pertenece a {numeros}")
```

Control de Flujo: Instrucción if – elif - else

#Ejemplos (Poner especial cuidado con la indentación del código pues en Python tiene significado)

```
A=1
B=2
C=3
D=4
if (A<B) & (C>D):
        print("La condición es correcta")
else:
        print("La condición no es correcta")

A=1
B=2
if A<B:
        print("A es menor que B")
elif (A==B):
        print("A es igual que B")
else:
        print("A es menor que B")
```

BUCLES: Instrucción for

Nota: en un bucle for la iteracción se puede realizar sobre los elementos de una lista, tupla, diccionarios y cadenas de texto

#Ejemplos (Poner especial cuidado con la indentación del código pues en Python tiene significado)

```
colores = [“rojo”,”azul”]
for item in colores:          #Bucle for iterando sobre los elementos de una lista
        print(item)

cadena = “Hola Mundo”
for caracter in cadena:   #Bucle for iterando sobre los elementos de una cadena de texto
        print(caracter)

For item1 in range(5):      #Bucles for anidados iterando a través de número enteros, empieza en 0 y termina en 4
                            #range(3,5),significa que la secuencia empieza en 3 y termina en 4 (5-1)
                            #range(3,8,2) significa que la secuencia empieza en 3, termina en 7 y va saltando de 2 en 2
        for item2 in range (3):
                            print(“Iteración: %d,%d”%(item1,item2))

for item in range(10):
        if item == 5:
                            break          #Con break lo que hacemos es salir automáticamente del bucle FOR
                            continue       #Si en lugar de break usamos continue entonces lo que hacemos es finalizar esa
        iteración y continuar al principio del FOR con la siguiente iteración.
        print(item)
```

BUCLES: Instrucción while

#Ejemplos (Poner especial cuidado con la indentación del código pues en Python tiene significado)

```
valor = 1
while valor < 10:                    #Bucle con interaciones desde 1 hasta 9
        print (valor)
        valor += 1

#Uso de sentencia break en bucles
valor = 1
while valor < 10:                    #Bucle con interaciones desde 1 hasta 9
        print (valor)
        valor += 1
        if valor == 5:
                break        #Finalizo el bucle cuando valor vale 5.

    #Uso de sentencia continue en bucles
    print("Bucle while con sentencia continue")
    valor = 1
    while valor < 10:   #Bucle con interaciones desde 1 hasta 9
      valor += 1
      if valor == 6:
        print("En iteracción 6 saltamos al inicio del bucle")
        continue         #Finalizo la iteración y saltamos al inicio del bucle.
      print (valor)
```

Ejercicio 12: Bucles While

Problema: definir una variable de control “pedirnumero” e inicializarla a true. A continuación ejecutar un bucle para solicitar por teclado un número. Mientras este número no sea menor que 10 se seguirá solicitando números por teclado.

Solución:

```
pedirnumero = True
while pedirnumero == True:
        numero = int(input(Introduce un número entero inferior a 10: “))
        if numero < 10:
                pedirnumero = False

print(“FIN del bucle, el número es inferior a 10”)
```

Ejecución:

```
Introduce un número entero inferior a 10: 12
Introduce un número entero inferior a 10: 15
Introduce un número entero inferior a 10: 10
Introduce un número entero inferior a 10: 4
FIN del bucle, el número es inferior a 10
```

Ejercicio 13: Bucles for

```
Crea un diccionario con los siguientes pares de valores
    manzana, apple
    naranja, orange
    platano, banana
    limon, lemon

Muestra la traducción para la palabra "naranja"
Añade un elemento nuevo con "piña" y "pineapple"
Haz un bucle para mostrar todos los elementos del diccionario
```

Solución:

```
diccionario = {"manzana":"apple","naranja":"orange","platano":"banana","limon":"lenon"}
print(diccionario)
print("La traducción de manzana es: %s"%(diccionario["manzana"]))
print("Añadimos un nuevo elemento al diccionario")
diccionario["piña"]="pineapple"
print(diccionario)
for clave,valor in diccionario.items():
   print(clave,"-",valor)
```

Ejecución:

```
{'manzana': 'apple', 'naranja': 'orange', 'platano': 'banana', 'limon': 'lenon'}
La traducción de manzana es: apple
Añadimos un nuevo elemento al diccionario
{'manzana': 'apple', 'naranja': 'orange', 'platano': 'banana', 'limon': 'lenon', 'piña': 'pineapple'}
manzana - apple
naranja - orange
platano - banana
limon - lenon
piña - pineapple
```

Ejercicio 14: Control de flujo if

```
Creamos una variable "nota" que tenga el valor 4.5
Creamos una variable "trabajo_realizado" que tenga el valor "si"
Calcular el valor de la variable "nota_final", teniendo en cuenta que, si la nota_final es mayor o igual a 4, y el
valor de la variable "trabajo_realizado" es igual a "si", entonces "nota_final" sera igual a "aprobado", en caso
contrario sera igual a "suspenso"
```

Solución:

```
nota = 4.5
trabajo_realizado = "si"
if (nota >= 4)&(trabajo_realizado == "si"):
                    nota_final="aprobado"
else:
                     nota_final="suspenso"

print("La nota final es: "+nota_final)
```

Ejecución:

```
La nota final es: aprobado
```

Ejercicio 15: Bucle while

```
1. Crear una variable "inicio" con el valor 1
2. Crear una variable "fin" con el valor 6
3. Hacer un bucle while que muestre tantas filas como valores haya entre "inicio" y "fin"
4. En cada iteracción del bucle mostrar el texto "Esta es la fila " + número de fila en la que está.
```

Solución:

```
inicio = 1
fin = 6
contador = inicio
while contador < fin:
   print("Esta es la fila: %d"%(contador))
   contador +=1
```

Ejecución:

```
Esta es la fila: 1
Esta es la fila: 2
Esta es la fila: 3
Esta es la fila: 4
Esta es la fila: 5
```

Listas: colección ordenada de elementos que pueden ser de cualquier tipo de datos

```
colores = [“rojo”,”amarillo”,”verde”]
print(colores[0])
colores[2]=“azul”        #Modificamos el elemento 2 que es “verde” por “azul”
print(colores)
print(“Numero de elementos de la lista: %d “ %(len(colores)))   #Longitud o nº elementos de la lista
colores.append(“naranja”) #Añade un nuevo elemento al final de la lista
print(colores)
colores.remove(“rojo”)    #Borra el elemento “rojo” de la lista

for color in colores:         #Recorremos todos los elementos de la lista y los mostramos
                print (color)

colores.clear()             #Borramos todos los elementos de la lista
print(colores)

#Uso del operador + con listas
colores1= [“azul”,”negro”]
colores_concatenados = colores + colores1
print(colores)
colores1 = colores1 + [“naranja”]  #Añadimos un elemento al final de la lista
print(colores1)
colores1.sort()  #Ordenamos los elementos de la lista
print(colores1)
del colores1[2]  #Eliminamos el elemento de la posición 2 de la lista
print("Hemos eliminado el elemento 2 de la lista: ",colores1)
```

Listas: colección ordenada de elementos que pueden ser de cualquier tipo de datos

```
#Uso del operador * con listas (concatenar una lista con ella misma un nº X de veces)
colores1= [“azul”,”negro”]
print(“Lista colores1: “,colores1)
colores1 = colores1 * 2   #Duplica 2 veces el contenido de la lista
print(“Lista duplicada: “,colores1)

#Creación de listas dentro de los elementos de una lista a modo de Matrices
lista_matriz=[“A”,[“a”,”b”,”c”],”B”,”C”]   #En la posición 1 hemos insertado como elemento otra lista
print(“Contenido de lista_matriz: “,lista_matriz)
print(“Elemento 0 de lista_matriz: “,lista_matriz[0])
print(“Elemento 1 de lista_matriz: “,lista_matriz[1])
print(“Elemento 1-0 de lista_matriz: “,lista_matriz[1][0])
print(“Elemento 1-1 de lista_matriz: “,lista_matriz[1][1])
print(“Elemento 1-2 de lista_matriz: “,lista_matriz[1][2])
print(“Acceso a los elementos 2 y 3 de la lista: “,lista_matriz[2:4])  #Acceso a una porción de la lista
print(“Acceso a los elementos 0 y 1 de la lista: “,lista_matriz[:2])
print(“Acceso a los elementos 1, 2 y 3 de la lista: “,lista_matriz[1:])
```

Listas: colección ordenada de elementos que pueden ser de cualquier tipo de datos

```
#Métodos y funciones de las listas
colores.append(“naranja”) #Añade un nuevo elemento al final de la lista
colores.insert(1,“naranja”) #Inserta un elemento en la posición 1 de la lista
colores.remove(“rojo”)     #Borra el elemento “rojo” de la lista
colores.reverse()                    #Invierte el orden de los elementos de la lista
colores.sort()                       #Ordenamos los elementos de la lista. Parámetro opcional
                                     reverse=true para indicar que la ordenación se realiza
descendente
colores.pop()                        #Elimina el último elemento de la lista.
colores.count(“rojo”)                          #Cuenta el número de veces que aparece dicho
elemento en la lista
colores.index(”rojo”,inicio,fin)     #Devuelve la posición de la primera ocurrencia del elemento
“rojo” en                            la lista. Inicio y fin, son parámetros opcionales que indica la
posición                             inicial y final de búsqueda.
colores.clear()                      #Elimina todos los elementos de la lista
```

Tuplas: colección ordenada de elementos que no se pueden modificar, añadir o eliminar elementos.

```
tupla_colores = (“rojo”,”verde”,”amarillo”)
print (tupla_colores)
print (tupla_colores[0])
print(“Número de elementos de la tupla: %d“ % (len(tupla_colores)))
tupla_colores[2] = “azul”  #Da un error pues las tuplas no se pueden modificar
print(“Acceso a una porción de la tupla, posición 2 y 3: “,tupla_colores[1:3])
print(“Acceso a una porción de la tupla, posición 0 y 1: “,tupla_colores[:2])

tupla_colores = tupla_colores * 3        #Triplica 3 veces el contenido de la tupla
print("tupla_colores triplicada: ",tupla_colores)

#Métodos y funciones de las tuplas  (son menos funciones que en las Listas)
tupla_colores.count(“rojo”)      #Cuenta el número de veces que aparece dicho elemento en la tupla
Tupla_colores.index(”rojo”,inicio,fin)      #Devuelve la posición de la primera ocurrencia del elemento “rojo” en la lista. Inicio y fin, son parámetros opcionales que indica la posición inicial y final de búsqueda.
```

Conjuntos: colección desordenada de elementos que pueden ser de cualquier tipo de datos

```
conjunto_colores = {“rojo”,”amarillo”,”verde”}
print(conjunto_colores)
for color in conjunto_colores:          #Recorremos todos los elementos del conjunto y los mostramos
        print (color)

print (conjunto_colores[0])             #Da un error ya que la colección está desordenada.

print(“Numero de elementos del conjunto: %d “ %(len(conjunto_colores)))

conjunto_colores.add(“negro”)           #Añadimos un nuevo elemento, pero no lo añade en ningún posición
                                        determinada ya que se trata de una colección desordenada.
print(conjunto_colores)


conjunto_colores.remove(“verde”)                #Borramos un elemento del conjunto
print(conjunto_colores)
```

Diccionarios: colección ordenada de elementos cuyos índices no son numéricos como en listas y tuplas sino identificadores. Al igual que listas y tuplas los elementos pueden ser de cualquier tipo. En definitiva los diccionarios son colecciones de elementos **compuestos de pares: clave – valor**. Las claves son únicas y no pueden repetirse.

Nota: las claves pueden ser de los siguientes tipos: números, cadenas, booleanos, bytes y tuplas.

```
diassemanaingles = {"Lunes":"Monday","Martes":"Tuesday","Miercoles":"Wednesday","Jueves":"Thursday"}
print("Muestro contenido del diccionario:", diassemanaingles)
print("Muestra el valor de la clava Lunes: ",diassemanaingles["Lunes"])
diassemanaingles["Viernes"]="Friday“       #Añado un nuevo elemento al diccionario
print("Añado un nuevo valor al diccionario:", diassemanaingles)
diassemanaingles["Viernes"]="Friday-modificado“  #Modifico un valor del diccionario
print("Modifico un valor al diccionario:", diassemanaingles)
del diassemanaingles["Viernes"]                #Elimino el valor viernes del diccionario
print("Elimino un valor al diccionario:", diassemanaingles)
print("Numero de elementos del diccionario: ",len(diassemanaingles))
print("Elemento mayor del diccionario: ",max(diassemanaingles))            #Orden alfabético
print("Elemento menor del diccionario: ",min(diassemanaingles))            #Orden alfabético
diassemanaingles.pop(“Lunes”)                          #Eliminamos el valor lunes del diccionario
print("Muestro contenido del diccionario:", diassemanaingles)
for clave in diassemanaingles:     #Recorremos todos los elementos del diccionario y solo mostramos las claves
        print (clave)
for clave,valor in diassemanaingles.items():    #Recorremos todos los elementos del diccionario y los mostramos con un FOR
    print (clave,valor)
```

Diccionarios: colección ordenada de elementos cuyos índices no son numéricos.

#Métodos y funciones del diccionario

diccionario=diassemanasingles.copy() #Realiza una copia exacta del diccionario en uno nuevo.

diccionario.pop(clave) #Devuelve el valor de la clave y además elimina dicho elemento.

diccionario.popitem() #Devuelve un elemento aleatorio del diccionario y lo elimina.

diccionaro.get(clave,defecto) #Devuelve el valor de la clave y en caso de no existir devuelve el valor indicado en el segundo parámetro. Este segundo parámetro es opcional.

diccionario.update(otro_diccionario) #Actualiza un diccionario por otro pasado por parámetro.

diccionario.clear() #Elimina todos los elementos del diccionario

diccionario.ítems() #Devuelve un objeto iterable que puede utilizarse en bucles. Devuelve todos los elementos (claves y valores)

diccionario.keys() #Devuelve un objeto iterable que puede utilizarse en bucles. Devuelve todos las claves.

diccionario.values() #Devuelve un objeto iterable que puede utilizarse en bucles. Devuelve todos los valores.

Ejercicio 8: Operaciones con colecciones - Listas

```
Dada la siguiente lista = [1,2,5,25,33,56,75,21,56,89,43,13,62,24]
Mostrar mediante el metodo "print" y el operador "in", si el numero 21 está en la lista
```

Solución:

```
lista=[1,2,5,25,33,56,75,21]
print(lista)
numero=int(input("Introduce un numero: "))

if numero in lista:                    #Comprueba si el número pertenece a la lista
   print("El valor %d está en la lista"%(numero))
else:
   print("El valor %d no está en la lista"%(numero))

posicion = lista.count(numero)         #Obtengo el nº de veces que número se encuentra en la lista
if posicion > 0:
   print("El valor %d está en la lista"%(numero))
else:
   print("El valor %d no está en la lista"%(numero))
```

Ejercicio 9: Operaciones con colecciones - Tuplas

```
1. Crear una variable "tupla" que sea una tupla de los siguientes nombres: Antonio,Pedro y Maria
2. Mostrar el valor de la variable "tupla"
3. Recoger un dato por teclado y almacenarlo en la variable "dato"
4. Si el valor de "dato" está dentro de los valores de la variable "tupla", mostrar "Si"
5. Si el valor de "dato" no está dentro de los valores de la variable "tupla", mostrar "No"
```

Solución:

```
tupla=("Antonio","Pedro","Maria")
print(tupla)
dato=input("Introduce un nombre: ")
if (dato in tupla):
   print("El nombre %s SI pertenece a la tupla"%(dato))
else:
   print("El nombre %s NO pertenece a la tupla"%(dato))
```

Ejecución:

```
('Antonio', 'Pedro', 'Maria')
Introduce un nombre: Antonio
El nombre Antonio SI pertenece a la tupla
```

Ejercicio 10: Operaciones con colecciones - Conjuntos

```
1. Crear un variable "conjunto" que sea un conjunto de los valores 1,2,3,4 y 5
2. Mostrar el valor de la variable "conjunto"
3. Añadir los números 6,7,8 y 9 a la variable "conjunto"
4. Mostrar ahora el valor de la variable "conjunto"
5. Eliminar el numero 9 de la variable "conjunto"
6. Mostrar ahora el valor de la variable "conjunto"
7. Verificar que tipo de dato es la variable "conjunto" mediante type()
```

Solución:

```
conjunto={1,2,3,4,5}
print(conjunto)
conjunto.add(6)
conjunto.add(7)
conjunto.add(8)
conjunto.add(9)
print(conjunto)
conjunto.remove(9)
print(conjunto)
print(type(conjunto))
```

Ejecución:

```
{1, 2, 3, 4, 5}
{1, 2, 3, 4, 5, 6, 7, 8, 9}
{1, 2, 3, 4, 5, 6, 7, 8}
<class 'set'>
```

Ejercicio 11: Operaciones con colecciones - Diccionarios

```
1. Crear una variable "diccionario" con los pares de valores siguientes
    clave=uno   valor=one
    clave=dos   valor=two
    clave=tres  valor=three
2. Mostrar por pantalla el valor de la variable "diccionario"
3. Añadir un nuevo elemento al diccionario
    clave=cuatro  valor=four
4. Mostrar ahora el valor del diccionario
5. Recoger un valor introducido por teclado y almacenarlo en "dato"
6. Utilizar "dato" como clave del diccionario para recuperar su valor
```

Solución:

```
diccionario={"uno":"one","dos":"two","tres":"three"}
print(diccionario)
#Añadimos un nuevo elemento
diccionario["cuatro"]="four"
print(diccionario)
dato=input("Introduce un valor:")
print("El valor de la clave %s es %s"%(dato,diccionario[dato]))
```

Ejecución:

```
{'uno': 'one', 'dos': 'two', 'tres': 'three'}
{'uno': 'one', 'dos': 'two', 'tres': 'three', 'cuatro': 'four'}
Introduce un valor:tres
El valor de la clave tres es three
```

Lección 4: Cadenas de Texto

Tipos de operaciones con cadenas:

```
cadena = “Hola Mundo”
cadena[0]              #Muestra el primer carácter de la cadena (H)
cadena[9]              #Muestra la posición 9 de la cadena (o)
cadena[-1]                    #Muestra la última posición (o)
cadena[-2]                    #Muestra la penúltima posición (d)
cadena[2:7]            #Muestra desde la posición 2 a la 6
cadena[2:]                    #Muestra desde la posición 2 hasta el final
cadena[:2]                    #Muestra desde la posición inicial hasta la posición 2
```

Concatenar cadenas y operador * (multiplicador) y operador “in”:

```
cadena1 = “hola”
cadena2 = “Mundo”
cadena3 = cadena1 + cadena2
len(cadena1)           #Muestra la longitud de cadena (4)
cadena1.upper()        #Muestra el valor de cadena en Mayúscula pero no cambia el valor
cadena1.lower()        #Muestra el valor de cadena en Minúscula
cadenamultiplicacion = cadena1 * 5 #se multiplica 5 veces la cadena
print(“¿Está la cadena1 incluida en cadena3? “,cadena1 in cadena3)
```

Cadena.split

Descomponer el contenido por elementos en una lista de valores

```
cadena=“uva,pera,manzana”
cadena.split(‘,’)          #Devuelve una lista:  [‘uva’, ‘pera’, ‘manzana’]
```

Lección 4: Cadenas de Texto: funciones para manipular cadenas de texto

```
cadena = “hola mundo”
cadena_final = cadena.capitalize()  #Pone la primera letra en mayúscula

cadena_final = cadena.title()                #Pone la primera letra de cada palabra en mayúscula

Cadena_final = cadena.upper()                #Pone en mayúscula todo el texto
Cadena_final = cadena.lower()                #Pone en minúscula todo el texto

Longitud = len(cadena)              #Devuelve la longitud en nº de caracteres de la cadena

Cadena.isalnum()                    #Permite saber si todos los caracteres son alfanuméricos, devuelve booleano.
Cadena.isalpha()                    #Permite saber si todos los caracteres son alfabéticos, , devuelve booleano.
Cadena.isdigit()                    #Permite saber si todos los caracteres son dígitos, , devuelve booleano.
Cadena.isnumeric()                  #Permite saber si todos los caracteres son numéricos, , devuelve booleano.
Cadena.islower()                    #Permite saber si todos los caracteres son minúsculas.
Cadena.isupper()                    #Permite saber si todos los caracteres son mayúsculas.
Cadena.istitle()                    #Permite saber si el primer caracteres de todas las palabras está en mayúscula.
Cadena.isspace()                    #Permite saber si la cadena de texto está compuesta únicamente por espacios.

Cadena_final = cadena.lstrip()      #Elimina todos los caracteres en blanco al comienzo de la cadena.
Cadena_final = cadena.rstrip()      #Elimina todos los caracteres en blanco al final de la cadena.
Cadena_final = cadena.strip()       #Elimina todos los caracteres en blanco al comienzo y final de la cadena.

Cadena_final = cadena.max()         #Permite conocer el carácter alfabético mayor de la cadena.
Cadena_final = cadena.min()         #Permite conocer el carácter alfabético menor de la cadena.
cadena.startswith(cadena_inicio)    #Permite conocer si una cadena empieza por un texto concreto.
cadena.endswith(cadena_final)       #Permite conocer si una cadena termina por un texto concreto
```

Lección 4: Cadenas de Texto: funciones para manipular cadenas de texto

cadena = “hola mundo”

Cadena_final= cadena.replace(cadena_buscada, cadena_reemplaza)
#Permite sustituir todas las ocurrencias de una cadena localizadas por otra cadena.

Lista = cadena.split(“,”)
#Permite separar en una lista todos los elementos de la cadena separados por el parámetro indicado. Si no se indica separador se entiende por defecto el espacio en blanco.

Lista = cadena.splitlines()
#Permite convertir una cadena de texto en una lista de elementos que se encuentran separados en la cadena de texto por saltos de línea.

Posición = cadena.find (cadena_buscada)
#Permite localizar una cadena dentro de otra buscando de izquierda a derecha. Devuelve la posición en la que comienza la cadena buscada. En caso de no encontrarse se devuelve -1.

Posición = cadena.rfind (cadena_buscada)
#Permite localizar una cadena dentro de otra buscando de derecha a izquierda. Devuelve la posición en la que comienza la cadena buscada. En caso de no encontrarse se devuelve -1.

Cadena_final = cadena.center(longitud, relleno)
#Permite centrar la cadena en otra cadena final de longitud determinada. Los huecos a la derecha e izquierda se rellenarán con el carácter “relleno” especificado como parámetro. Si no se indica “relleno” se utilizará espacios en blanco.
Cadena_final = cadena.ljust(longitud, relleno) #Igual que el anterior pero alineando a la izquierda.
Cadena_final = cadena.rjust(longitud, relleno) #Igual que el anterior pero alineando a la derecha.

Ejercicio 1: Cadena de Texto

Problema:
Crear una variable “cadena” que contiene el texto “Esto es un texto de ejemplo”
Según la posición de cada letra en la cadena, calcular que valores (x,y) hay que poner para seleccionar solo la palabra “texto”
Cadena[x:y] = “texto”

Solución:
```
cadena=“Esto es un texto de ejemplo”
print(cadena[11:16])
```

Ejercicio 2: Cadena de Texto

Problema:
1. Crear una variable “cadena” que contiene el texto “Esto es un texto de ejemplo”
2. Crear una variable “longitud” que contiene la longitud de la variable “cadena”
3.- Crear una variable “strlongitud” que tenga el valor de “longitud” pero convertida a cadena de caracteres
4.- Crear una variable “mayúsculas” que contenga la variable “cadena” en mayúsculas
5.- Crear una variable “resultado” que contenga “mayúsculas” con el texto “tiene longitud de “ y “strlongitud”

Solución:

```
cadena=“Esto es un texto de ejemplo”
cadena
longitud = len(cadena)
longitud
strlongitud = str(longitud)
strlongitud
mayusculas = cadena.upper()
mayusculas
resultado =mayúsculas + “: tiene longitud de ” + strlongitud
resultad
```

Ejercicio 3: Cadena de Texto

Problema:

1. Crear una variable “numero1” que contiene el valor 5
2. Crear una variable “numero2” que contiene el valor 8
3. Crear una variable “suma” que contiene la suma de las 2 variables anteriores
4. Imprimir por pantalla el resultado utilizando la función “print” con .format y f-string

Solución:

```
numero1=5
numero2=8
suma = numero1 + numero2
Print(f”Resultado de {numero1} + {numero2} = {suma}”)
Print(“Resultado de {r1} + {r2} = {r3}”.format(r1=numero1,r2=numero2,r3=suma)
```

Ejercicio 4: Cadena de Texto

Problema:

1. Imprime por pantalla el texto “Introduce el primer número”
2. Crear la variable “dato1” con el primer valor introducido en el paso anterior
3. Imprime por pantalla el texto “Introduce el segundo número”
4. Crear la variable “dato2” con el primer valor introducido en el paso anterior
5. Convertir la variable “dato1” en una variable numérica denominada “numero1”
6. Convertir la variable “dato2” en una variable numérica denominada “numero2”
7. Crear la variable “suma” con la suma de “numero1” y “numero2”
8. Convertir la variable “suma” en una variable de texto denominada “strSuma”
9. Crear la variable “resultado” con la concatenación de la “La suma es “ y “strSuma”
10. Imprimir el valor de “resultado”

Solución:

```
dato1 = input(“Introduce el primer numero:”)
dato2 = input(“Introduce el segundo numero:”)
numero1 = int(dato1)
numero2 = int(dato2)
suma = numero1 + numero2
strSuma = str(suma)
resultado = “La suma es “ + strSuma
print (“La suma de “+str(numero1)+” +“+str(numero2)+” = “+strSuma)
print (“Resultado = “+resultado)
```

Módulos

MODULOS: son ficheros de código en Python que podemos incorporar a nuestro programa mediante la instrucción “import” y que nos permite utilizar toda la funcionalidad que dicho módulo tiene.

Nota: los módulos importados en Python los intentará localizar en el directorio local donde se encuentre el programa en Python que estemos ejecutando y si no los buscará en las librerías lib y lib/site-packages.

Módulos

modulo1.py

```
#Hemos creado un fichero modulo1.py con esta función saludar
def saludar(nombre):
        print ("Hola, soy "+nombre)
def despedirse(nombre):
        print("Adios "+nombre)
```

Miprograma1.py

```
#Creamos nuestro programa en Python en el que importamos el módulo modulo1.py
import modulo1   #importamos todo el modulo1.py
modulo1.saludar("Jose")#Hacemos uso de la función saludar que está dentro del  módulo1
```

Módulos

Miprograma2.py

#Creamos nuestro programa en Python en el que importamos solo la función despedirse del módulo1

from modulo1 import despedirse

#importamos solo la función despedirse del modulo1.py

despedirse(“Jose”) #Hacemos uso de la función despedirse que está dentro del módulo1

saludar(“Jose”) #Daría un error, pues la función saludar no la hemos importado

Módulos

Miprograma3.py

#Creamos nuestro programa en Python en el que importamos solo la función despedirse

del módulo1

from modulo1 import despedirse as adios

#importamos la función despedirse, pero la nombramos como adiós

adios(“Jose”) #Hacemos uso de la función despedirse que está dentro del módulo1

Import os

Crear archivos y directorios, administrar archivos...

os.name: devuelve el nombre del sistema operativo.

os.getcwd(): devuelve el directorio de trabajo actual.

os.chdir(path): cambia el directorio de trabajo actual al especificado por path.

os.listdir(path): devuelve una lista de archivos y directorios en el directorio especificado por path.

os.mkdir(path): crea un directorio en la ruta especificada por path.

os.makedirs(path): crea directorios en la ruta especificada por path y crea cualquier directorio faltante en la ruta.

os.remove(path): elimina el archivo especificado por path.

os.rmdir(path): elimina el directorio especificado por path.

os.removedirs(path): elimina el directorio especificado por path y elimina cualquier directorio vacío en la ruta.

os.rename(src, dst): cambia el nombre del archivo o directorio especificado por src a dst.

os.path.exists(path): devuelve True si la ruta especificada por path existe, False en caso contrario.

os.path.isfile(path): devuelve True si la ruta especificada por path es un archivo, False en caso contrario.

os.path.isdir(path): devuelve True si la ruta especificada por path es un directorio, False en caso contrario.

username = os.environ['USERPROFILE'] -> **C:/users/nombre_usuario/**

Import os

os.path.join(*paths): une varios componentes de una ruta en un solo camino. Por ejemplo, os.path.join('/usr', 'local', 'bin') devuelve '/usr/local/bin'.

os.path.split(path): divide una ruta en dos partes: el directorio y el archivo. Devuelve una tupla que contiene ambos valores. Por ejemplo, os.path.split('/home/user/file.txt') devuelve ('/home/user', 'file.txt').

os.path.splitext(path): divide una ruta en dos partes: el nombre del archivo y la extensión. Devuelve una tupla que contiene ambos valores. Por ejemplo, os.path.splitext('/home/user/file.txt') devuelve ('/home/user/file', '.txt').

os.path.basename(path): devuelve el nombre del archivo de la ruta especificada por path. Por ejemplo, os.path.basename('/home/user/file.txt') devuelve 'file.txt'.

os.path.dirname(path): devuelve el nombre del directorio que contiene el archivo especificado por path. Por ejemplo, os.path.dirname('/home/user/file.txt') devuelve '/home/user'.

os.path.isabs(path): devuelve True si la ruta especificada por path es una ruta absoluta, False si es una ruta relativa.

os.path.expanduser(path): expande el carácter ~ en la ruta del directorio del usuario. Por ejemplo, os.path.expanduser('~') devuelve la ruta del directorio del usuario actual.

os.path.getsize(path): devuelve el tamaño en bytes del archivo especificado por path.

os.path.getmtime(path): devuelve la marca de tiempo de la última modificación del archivo especificado por path. La marca de tiempo se devuelve en segundos desde la época.

os.path.abspath(path): convierte una ruta relativa en una ruta absoluta.

Import os

```
import os

ruta_actual = os.path.dirname(os.path.abspath(__file__))

print(ruta_actual)

print ("\n"+ os.name)
```

Import random

```
import random

b1 = random.random()

b2=random.randrange(1,7)
```

Crear archivos de texto

```
carpeta = r"C:\Users\javier.gsanturde\OneDrive - Institución Educativa SEK\UCJC\carpeta1"
carpeta_archivo= carpeta + r"\texto.json"
#ruta_todo=os.path.join(carpeta,"nombre.txt") # usando join
if not os.path.exists(carpeta):
#especificar carpeta_archivos de carpeta_archivos en sistemas Windows.
  #os.makedirs(carpeta) #os.mkdir
 os.mkdir(carpeta)

#ESCRIBIR
archivo_salida = open(carpeta_archivo, 'w') #wt
archivo_salida.write("hola clase")
archivo_salida.close()
```

Crear archivos de texto

#LEER

```
archivo_entrada = open(carpeta_archivo, 'r') #rt
print(archivo_entrada.read())
```

#MOVIMIENTOS DEL CURSOR DENTRO DEL ARCHIVO

```
archivo_entrada.seek(10) #salta a la posición 10 e cursor
print(archivo_entrada.read())
archivo_entrada.seek(0) #salta a la posición 0 el cursor. Inicio del documento.
archivo_entrada.read(20)# leemos hasta la posición 20 del documento
archivo_entrada.read()# desde el 20 hasta el final
print(archivo_entrada.read())
print(archivo_entrada.tell())# posición del útlimo caracter
archivo_entrada.close
```

Crear archivos de texto

```
# Leer contenido del archivo de entrada
with open(carpeta_archivo, 'r') as archivo_entrada:
contenido = archivo_entrada.read()

# Escribir en un archivo de salida
with open(carpeta_archivo, 'w') as archivo_salida:
    # Escribir contenido en el archivo de salida
    archivo_salida.write(contenido)
```

Pasar de json a diccionario

```
import json
# Creamos un objeto en Python
persona = {
   "nombre": "Juan",
   "edad": 25,
   "ciudad": "Madrid"
}
# Convertimos el objeto en un json y lo guardamos en un archivo de texto
with open("persona.json", "w") as archivo:
  json.dump(persona, archivo)  #json.dump: para convertir este objeto en un json

# Leemos el archivo
with open("persona.txt", "r") as archivo:
   json_string = archivo.read()
   persona_json = json.loads(json_string)  #json.loads: de json a un objeto
print(persona_json.keys())
for a in (persona_json.keys()):
   print (a)
   print(persona_json[a])
```

FUNCION: Es la definición de un trozo de código que se puede invocar desde cualquier parte del programa.

Nota 1: la definición de las funciones suele hacerse al principio de los programas.

Nota 2: por defecto, las variables que se usan dentro de la función únicamente tienen validez dentro de la función, no se podrá acceder desde fuera de la función a variables definidas dentro.

#Ejemplos

```
#Definición de una función sin parámetros
def saludar():
        print("Buenos días")

saludar()                   #Llamamos a la función saludar()

#Definición de una función con parámetros
def saludar(nombre):
        print ("Buenos días %s" % (nombre))

saludar("Antonio")          #Llamamos a la función saludar()

#Definición de una función con retorno de valor
def sumar (numero1, numero2):
        suma = numero1 + numero2
        return suma

numero1 = 5
numero2 = 3
resultado = sumar (numero1, numero2)
print ("El resultado de sumar %d + %d = %d" % (numero1, numero2, resultado))
```

FUNCION: uso de “return” devolución de 1 o múltiples valores por la función.

#Ejemplos

```
#Definición de una función que devuelve 1 valor
def sumar(a,b):
        suma = a+b
        return suma                #Devuelve 1 valor la función

print (“la suma de 5 + 2 = %d” % (sumar(5,2))


Definición de una función que devuelve 2 valores
def sumaryrestar(a,b):
        suma = a+b
        resta = a-b
        return suma,resta          #Devuelve 2 valores la función

suma,resta = sumaryrestar(5,2)
print (“la suma de 5+2 = %d”% suma)
print (“la resta de 5-2 = %d”% resta)
```

FUNCION: paso de un número variable de parámetros a la función (añadir un * delante del parámetro).

#Ejemplos

```
#Definición de una función que devuelve 1 valor
def sumar(*valores):          #De esta forma indicamos que la función puede recibir varios parámetros
        suma = 0
        for item in valores:
                suma = suma + item
        return suma

sumar(1,2,3,4,5,6)
```

FUNCION: uso de variables globales y locales dentro de una función.

#Ejemplos

```
#Vamos a declarar una variable global, en el programa principal, y una variable local dentro de una función con el mismo nombre.
Veamos su comportamiento.
def Variables():
        variable = 3          #Definimos una variable local con el nombre "variable"
        print ("Valor dentro de la función %d" %variable)

variable = 5                  #Definimos una variable global con el nombre "variable"
Variables ()
print ("Variable en el programa principal: %d" % variable)
```

#Resultado de la ejecución es:

```
Valor dentro de la función 3
Variable en el programa principal: 5
```

FUNCION: uso de variables globales y locales dentro de una función.

#Ejemplos

```
#Vamos a declarar una variable global, en el programa principal, y vamos a hacer uso de dicha variable dentro de una función.
def Variables():
        global variable          #Indicamos que vamos a utilizar la variable global
        print ("Valor dentro de la función %d" %variable)
        variable = 3
        print ("Valor dentro de la función %d" %variable)

variable = 5                     #Definimos una variable global con el nombre "variable"
print ("Variable en el programa principal: %d" % variable)
Variables ()
print ("Variable en el programa principal: %d" % variable)
```

#Resultado de la ejecución es:

```
Variable en el programa principal: 5
Valor dentro de la función 5
Valor dentro de la función 3
Variable en el programa principal: 3
```

FUNCION: paso de parámetros por referencia (para ser modificados por la función)

#Ejemplos

```
#Definición de una función pasándole un parámetro como referencia (para ser modificado)
#Esto sucede cuando los parámetros son estructuras como colecciones, listas, tuplas, etc.
#Sin embargo, si el parámetro que le pasamos a la función es una variable simple (cadena, entero, etc.) entonces
#dicho parámetro no se puede modificar dentro de la función

colores = ["blanco","azul","amarillo"]
color = "negro"

print ("Tenemos una colección de colores",colores)
print ("Vamos a insertar el color: ",color)

def insertar_color (colores,color):
   colores.append(color)

insertar_color (colores, color)        #Llamamos a la función color pasándolo la colección colores por referencia
print ("La colección ahora es:" ,colores)
```

FUNCION: polimorfismo en Python
En Python a diferencia de Java no permite redefinir un método varias veces

#Ejemplos

```
#Esto en Python no es posible realizarlo
def saludar()
def saludar(nombre)

#Ejemplo de valores por defecto en los parámetros de una función
#Esta es la forma que tiene Python de implementar el polimorfismo
def restar(a=5,b=1):      #En este caso no hace falta pasarle parámetros a esta función pues tienen valores por defecto.      return a-b

print("Opcion 1: "+str(restar(5,2))) #Paso los 2 parámetros
print("Opcion 2: "+str(restar()))      #No pasamos ningún parámetro
print("Opcion 3: "+str(restar(6)))     #Pasamos el parámetro a = 6, b tomará el valor 1 por defecto.
print("Opcion 4: "+str(restar(b=0,a=10)))  #Pasamos los 2 parámetros indicando el nombre de cada parámetro.
print("Opcion 5: "+str(restar(b=5)))   #Pasamos el parámetro b=5, a tomará el valor 5 por defecto.
```

IMPORTANTE: todos los parámetros que se definan en una función deben ser pasados al invocar a dicha función. Si el parámetro toma algún valor por defecto entonces no es necesario pasarlo al invocar dicha función.

FUNCION: valores por defecto “None” en los parámetros

#Ejemplos

```
def restar (a=None, b=None):      #Defino valores por defecto None
   if (a==None or b==None):
      print ("Error")
   else:
      return a-b

print("Opcion 1: "+str(restar(5,2)))
print("Opcion 2: "+str(restar()))
print("Opcion 3: "+str(restar(6)))
```

#Resultado

```
Opcion 1: 3
Error
Opcion 2: None
Error
Opcion 3: None
```

Lección 12: Clases-atributos-métodos, objetos y funciones

CLASE: Es la definición del constructor de un objeto

Nota: en una clase se definen atributos (variables) y métodos

#Ejemplos

```
class ClaseSilla:        #Definimos una clase con 2 atributos
    color = “blanco”
    precio = 100

#Creo un objeto a partir de la clase Silla
objetoSilla1 = ClaseSilla()
print(“La silla 1 es de color %s y su precio es de %d”%(objetoSilla1.color,objetoSilla1.precio))

class Persona:        #Definimos una clase con 1 constructor y un método
     nombre = “sin-nombre”
     edad = 0

    def __init__(self,nombre,edad):        #Constructor de la clase   ¡¡ojo, escribir doble _
        self.nombre = nombre
        self.edad = edad

    def saludar(self):                              #Método saludar
         print(“Hola, me llamo %s y tengo %d años”%(self.nombre,self.edad))

#Creo un objeto de la clase Persona
persona1 = Persona("Jose",40)
print("Los datos de persona 1 son %s y %d años"%(persona1.nombre,persona1.edad))
persona1.saludar()
```

IMPORTANTE: en los métodos de una clase el parámetro 1º debe ser “self”.

Self.Atributo es la forma de acceder a un atributo de la clase

CLASE: asignación de objetos

Nota: asignando 2 objetos se asignan los valores de sus atributos.

#Ejemplo

```
class Persona:
   def __init__ (self, nombre, apellidos, edad):
      self.Nombre = nombre
      self.Apellidos = apellidos
      self.Edad = edad
   def MostrarPersona(self):
      print("Nombre: " + self.Nombre)
      print("Apellidos: " + self.Apellidos)
      print("Edad: " + str(self.Edad))

print("OBJETOS ORIGINALES")
p1 = Persona("Alfredo","Moreno Muñoz", 35) #Se crea 1 objeto de la clase Persona
p1.MostrarPersona()
p2 = Persona("Valeria","Moreno", 1)          #Se crea 1 objeto de la clase Persona
p2.MostrarPersona()
p1=p2                                #ASIGNAMOS 2 OBJETOS
print("OBJETOS TRAS ASIGNACIÓN")
p1.MostrarPersona()
p2.MostrarPersona()
```

#RESULTADO EJECUCIÓN

```
OBJETOS ORIGINALES
Nombre: Alfredo
Apellidos: Moreno Muñoz
Edad: 35
Nombre: Valeria
Apellidos: Moreno
Edad: 1
OBJETOS TRAS ASIGNACIÓN
Nombre: Valeria
Apellidos: Moreno
Edad: 1
Nombre: Valeria
Apellidos: Moreno
Edad: 1
```

CLASE: encapsulación de métodos y atributos (públicos y privados)

Nota: públicos (son accesibles desde el exterior de la clase directamente), privados (solo pueden ser accesibles desde dentro de la propia clase).

Nota: los métodos y atributos privados se definen con doble "_" delante de su nombre. Ej.- __alias

#Ejemplo

```
class Persona:
   alias = ""              #Atributo público
   __nombre = ""       #Atributo privado

   def __init__ (self,alias,nombre):
      self.alias = alias
      self.__nombre = nombre

   def getNombre(self):
      return self.__nombre

   def __mostrarNombrePrivado(self):     #Método privado
         print("Método privado, alias: %s, nombre: %s" %(self.alias,self.__nombre))
   def mostrarNombrePublico(self):       #Método público
         print("Método público, alias: %s, nombre: %s" %(self.alias,self.__nombre))

p1 = Persona("sr","Jose Manuel")
#p1.__nombre        #Da un error pues se trata de un atributo privado
print ("Alias:"+p1.alias)
print ("Nombre:"+p1.getNombre())
#p1.__mostrarNombrePrivado()   #Da un error pues se trata de un método privado
p1.mostrarNombrePublico()
```

#RESULTADO EJECUCIÓN

```
Alias:sr
Nombre:Jose Manuel
Método público, alias: sr, nombre: Jose Manuel
```

CLASE: herencia entre clases y subclases

Nota: para la herencia se utiliza la siguiente sintáxis class clasehija (Clasepadre1, Clasepadre2,..):

#Ejemplo

```
class Persona:
   def __init__ (self):
      self.__Nombre = ""
      self.__Apellidos = ""
      self.__Edad = 0
   def GetNombre(self):
      return self.__Nombre
   def GetApellidos(self):
      return self.__Apellidos
   def GetEdad(self):
      return self.__Edad

class Alumno(Persona):   #Clase Alumno que hereda de clase Persona
   def __init__ (self):
      self.__Curso = ""
      self.__Asignaturas = ""
   def GetCurso(self):
      return self.__Curso
   def GetAsignaturas(self):
      return self.__Asignaturas
   def MostrarAlumno(self):
      print("Alumno:")
      print("\tNombre:",self.GetNombre())
      print("\tApellidos:",self.GetApellidos())
      print("\tEdad:",self.GetEdad())
      print("\tCurso:",self.__Curso)
      print("\tMatrículas:",self.__Asignaturas) alumno = Alumno()

alumno.SetNombre("Alfredo")
alumno.SetApellidos("Moreno Muñoz")
alumno.SetEdad(35)
alumno.SetCurso("Bachillerato")
alumno.SetAsignaturas(["Matemáticas","Tecnología","Inglés"])
alumno.MostrarAlumno()
```

CLASE: herencia-múltiple entre clases y subclases

Nota: para la herencia se utiliza la siguiente sintáxis class clasehija (Clasepadre1, Clasepadre2,..):

#Ejemplo de herencia múltiple

```
class Persona:
   Nombre=""
   Edad=0

class Profesor:
   Telefono=""

class Investigador(Persona,Profesor):  #Heredo de 2 clases
   Especialidad=""
   def mostrarDatos(self):
      print("Nombre:"+self.Nombre)
      print(“Edad:"+str(self.Edad))
      print(“Telefono:"+self.Telefono)
      print(“Especialidad:"+self.Especialidad)

p1 = Investigador()
p1.Nombre = "Juan"
p1.Edad = 50
p1.Telefono = "91543"
p1.Especialidad = "Ingenieria"
p1.mostrarDatos()
```

#RESULTADOS DE EJECUCIÓN

```
NombreJuan
Edad 50
Telefono:91543
Especialidad:Ingenieria
```

Ejercicio 16: Clases y objetos

```
Crear una clase "Coche" que tenga estos atributos:
marca, color, combustible y cilindrada

Crear la función "__init__" que asigna los parametros de la clase a los atributos de la clase

Crear otra función "mostrar_caracteristicas" que mediante print muestre por pantalla
las características (o atributos) que tiene el coche

Crear un objeto "coche1" de la clase "Coche" con los atributos "Opel","rojo","gasolina","1.6"

Ejecutar la función "mostrar_caracteristicas" del objeto "coche1"
```

Solución:

```
#Defino la clase
class Coche:
  marca=""
  color=""
  combustible=""
  cilindrada=""
  def __init__ (self,marca,color,combustible,cilindrada):
    self.marca = marca
    self.color = color
    self.combustible = combustible
    self.cilindrada = cilindrada
  def mostrarCaracteristicas(self):
    print("Características, marca=%s, color=%s, combustible=%s,
cilindrada=%s"%(self.marca,self.color,self.combustible,self.cilindrada))

#Creo un objeto coche
coche1 = Coche("opel","rojo", "gasolina","1.6")
coche1.mostrarCaracteristicas()
coche1.marca
```

RESULTADO DE EJECUCIÓN

```
Características, marca=opel, color=rojo, combustible=gasolina, cilindrada=1.6
'opel'
```

Lección 13: Módulos (ficheros de Python para Importar)

MODULOS: son ficheros de código en Python que podemos incorporar a nuestro programa mediante la instrucción "import" y que nos permite utilizar toda la funcionalidad que dicho módulo tiene.
Nota: los módulos importados en Python los intentará localizar en el directorio local donde se encuentre el programa en Python que estemos ejecutando y si no los buscará en las librerías lib y lib/site-packages.

#Ejemplo

modulo1.py

```
#Hemos creado un fichero modulo1.py con esta función saludar
def saludar(nombre):
        print ("Hola, soy "+nombre)
def despedirse(nombre):
        print("Adios "+nombre)
```

Miprograma1.py

```
#Creamos nuestro programa en Python en el que importamos el módulo modulo1.py
import modulo1          #importamos todo el modulo1.py
modulo1.saludar("Jose")      #Hacemos uso de la función saludar que está dentro del  módulo1
```

Miprograma2.py

```
#Creamos nuestro programa en Python en el que importamos solo la función despedirse del módulo1
from modulo1 import despedirse        #importamos solo la función despedirse del modulo1.py
despedirse("Jose")      #Hacemos uso de la función despedirse que está dentro del  módulo1
saludar("Jose")                       #Daría un error, pues la función saludar no la hemos importado
```

Miprograma3.py

```
#Creamos nuestro programa en Python en el que importamos solo la función despedirse del módulo1
from modulo1 import despedirse  as adios        #importamos la función despedirse pero la nombramos como adios
adios("Jose")      #Hacemos uso de la función despedirse que está dentro del  módulo1
```


67

Lección 13: Módulos (ficheros de Python para Importar)

PIP: es un gestor de paquetes y de módulos que viene instalado con Python.

#Uso de PIP

Paso 1: Abrimos la consola de Windows (cmd)
Paso 2: Vamos a la carpeta donde se encuentra la instalación de Python
Paso 3: ejecutamos el comando $ pip –V , para ver la versión del gestor

```
c:\Users\JoseManuelQuesada\AppData\Local\Microsoft\WindowsApps\PythonSoftwareFoundation.Python.3.9_qbz5n2kfra8p0>pip -V
pip 22.0.3 from C:\Users\JoseManuelQuesada\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.9_qbz5n2kfra8p0\Loca
lCache\local-packages\Python39\site-packages\pip (python 3.9)
```

Paso 4: $pip –h , para ver las opciones de este comando
Paso 5: $pip install camelcase , para instalar este paquete desde internet
Paso 6: $pip uninstall camelcase, para desinstalar este paquete
Paso 7: $pip list , muestra un listado con todos los paquetes y módulos instalados y sus versiones

```
c:\Users\JoseManuelQuesada\AppData\Local\Microsoft\WindowsApps\PythonSoftwareFoundation.Python.3.9_qbz5n2kfra8p0>pip lis
t
Package    Version
---------- -------
camelcase  0.2
pip        22.0.3
```

Ejercicio 17: Uso de módulos

```
Crear un modulo "modulo1.py"
Añadir la clase "Coche" creada en un ejercicio anterior al modulo "modulo1"
Añadir la función lambda "media" creada en un ejercicio anterior al modulo "modulo1"

Crear un programa en Python "programa1.py"
Importar el modulo "modulo1" antes creado
Crear un objeto "coche1" al instanciar la clase "Coche"
Mediante print mostrar las características del coche
Calcular la media de 3 notas y mostrar el resultado con print

Ejecutar el programa "programa.py" y ver el resultado
```

Solución:

```
#defino modulo1 que lo he llamado modulococheb.py
class Coche:
    marca=""
    color=""
    combustible=""
    cilindrada=""
    def __init__ (self,marca,color,combustible,cilindrada):
                self.marca = marca
                self.color = color
                self.combustible = combustible
                self.cilindrada = cilindrada
    def mostrarCaracteristicas(self):
                print("Características, marca=%s, color=%s, combustible=%s,
cilindrada=%s"%(self.marca,self.color,self.combustible,self.cilindrada))

media3 = lambda a,b,c : (a+b+c)/3
# defino programa1.py que lo he llamado micoche.py
import modulococheb                    #importamos todo el modulococheb.py

print("Creamos un coche y lo inicializamos")
coche1 = modulococheb.Coche("Seat","Azul","Gasoil","1.600")
coche1.mostrarCaracteristicas()
print("Calculamos la media de 3 notas: 2, 4 y 6 =",modulococheb.media3(2,4,6))
```


69