```
import random, string

#Funsion de suma
def suma(n1,n2):
    return n1 + n2

#variables
n1 = 134
n2 = 456
#imprimir en consola
print("la suma de " + str(n1) + " mas " + str(n2) + " es " +
str(suma(n1,n2)))
```

```
la suma de 134 mas 456 es 590
```

Arreglo

```
#Arreglo vacio
array = []
#for de n a n
for i in range (1, 12):
    #agregar al arreglo
    array.append(str(i) + " -> " + str(i*2))

print(array)
```

```
['1 -> 2', '2 -> 4', '3 -> 6', '4 -> 8', '5 -> 10', '6 -> 12', '7 ->
14', '8 -> 16', '9 -> 18', '10 -> 20', '11 -> 22']
```

VARIABLES

- Entero
- Decimal
- Booleano

```
#Variables
entero = 24
decimal = 24.5
booleano = True
cadena = "texto"

#condicional
if booleano:
    print("Soy un entero " + str(entero) + " -> " + str(type(entero)))
    print("Soy un decimal " + str(decimal) + " -> " +
str(type(decimal)))
    print("Soy un booleano " + str(booleano)  + " -> " +
str(type(booleano)))
    print("Soy una cadena " + str(cadena)  + " -> " +
str(type(cadena)))
```

```
Soy un entero 24 -> <class 'int'>
Soy un decimal 24.5 -> <class 'float'>
Soy un booleano True -> <class 'bool'>
Soy una cadena texto -> <class 'str'>
```

Operaciones

- Suma
- Resta
- Multiplicacion
- Potencia
- Division
- Division entera
- Resto

```
#Operaciones
n1 = 92
n2 = 3
#suma -> +
print("la suma de " + str(n1) + " mas " + str(n2) + " es " + str(n1 +
n2))
#resta -> +
print("la resta de " + str(n1) + " menos " + str(n2) + " es " + str(n1
- n2))
#multiplicacion -> *
print("la multiplicacion de " + str(n1) + " por " + str(n2) + " es " +
str(n1 * n2))
#potenciacion -> *
print("la potenciacion de " + str(n1) + " elebado a " + str(n2) + " es
" + str(n1 ** n2))
#Division normal
print("la division de " + str(n1) + " entre " + str(n2) + " es " +
str(n1 / n2))
#Division que devuelve el cociente entero
print("El cosiente entero de la division de " + str(n1) + " entre " +
str(n2) + " es " + str(n1 // n2))
#Division que devuelve el resto
print("El resto de la division de " + str(n1) + " entre " + str(n2) +
" es " + str(n1 % n2))
```

```
la suma de 92 mas 3 es 95
la resta de 92 menos 3 es 89
la multiplicacion de 92 por 3 es 276
la potenciacion de 92 elebado a 3 es 778688
la division de 92 entre 3 es 30.666666666666668
El cosiente entero de la division de 92 entre 3 es 30
El resto de la division de 92 entre 3 es 2
```

Algunos metodos para strings

format

```
#Datos
nombre = "Edain"
edad = 24
trabajo = "programador"
salario = 15.000
casado = False

if(casado):
    casado = "si"
else:
    casado = "no"

#Mensaje
Mensaje = '''
Hola mi nombre es {}
tengo {} años
trabajo de {} y gano {}
y {} estoy casado
'''

print(Mensaje.format(nombre, edad, trabajo, salario, casado))
```

```
Hola mi nombre es Edain
tengo 24 años
trabajo de programador y gano 15.0
y no estoy casado
```

Convercion de textos a mayusculas o minusculas

```
texto = "hola mundo soy edain"
print(texto)
textoMa = texto.upper()
print(textoMa)
```

```
hola mundo soy edain
HOLA MUNDO SOY EDAIN
```

```
texto = "HOLA MUNDO SOY EDAIN"
print(texto)
textoMi = texto.lower()
print(textoMi)
```

```
HOLA MUNDO SOY EDAIN
hola mundo soy edain
```

```
texto = "HOLA MUNDO SOY EDAIN"
print(texto)
```

```
textoMi = texto.capitalize()
print(textoMi)
```

```
HOLA MUNDO SOY EDAIN
Hola mundo soy edain
```

Separar cadenas y ponerlas en una lista

```
texto = "Hola_amigos"
print(texto.split("_"))
```

```
['Hola', 'amigos']
```

```
texto = "Hola_amigos"
[text1, text2] = texto.split("_")
print(text1)
print(text2)
```

```
Hola
amigos
```

Remplazar cadenas por otras

```
texto = "Hola mi nombre es Edain"
print(texto)
print(texto.replace("Edain", "Jose"))
```

```
Hola mi nombre es Edain
Hola mi nombre es Jose
```

ESTRUCTURAS DE DATOS

Listas

Creacion y control de elementos

```
#Lista
lista = ["Pera", "Mango", "Manzana", "Platano", "Sandia"]
print(lista)
```

```
['Pera', 'Mango', 'Manzana', 'Platano', 'Sandia']
```

```
print("la lista " + str(lista) + " tiene " +  str(len(lista)) + "
elementos")
```

```
la lista ['Pera', 'Mango', 'Manzana', 'Platano', 'Sandia'] tiene 5
elementos
```

saber elemento de una lista

```
print(lista)
print("El primer elemento de la lista es " + str(lista[0]))
print("El quinto elemento de la lista es " + str(lista[4]))
```

```
['Pera', 'Mango', 'Manzana', 'Platano', 'Sandia']
El primer elemento de la lista es Pera
El quinto elemento de la lista es Sandia
```

optener rangos de la lista (desde / asta)

```
print(lista)
print("los datos que siguen despues del segundo elemento son " +
str(lista[2:]))
print("los datos que estam antes del penultimo elemento son " +
str(lista[:3]))
print("los datos que de la lista desde el segundo al penultimo
elemento son " + str(lista[1:4]))
```

```
['Pera', 'Mango', 'Manzana', 'Platano', 'Sandia']
los datos que siguen despues del segundo elemento son ['Manzana',
'Platano', 'Sandia']
los datos que estam antes del penultimo elemento son ['Pera', 'Mango',
'Manzana']
los datos que de la lista desde el segundo al penultimo elemento son
['Mango', 'Manzana', 'Platano']
```

ultimos elementos

```
print(lista)
print("el ultimo elemento de la lista es " + str(lista[-1]))
print("el penultimo elemento de la lista es " + str(lista[-2]))
```

```
['Pera', 'Mango', 'Manzana', 'Platano', 'Sandia']
el ultimo elemento de la lista es Sandia
el penultimo elemento de la lista es Platano
```

Buscar elementos

saber si un elemento esta en la lista

```
lista = ["Edain", "Maria", "Jose", "Pedro", "Manuel"]
print(lista)

#nombre a buscar
nombre = "Jose"

#Si n esta en la lista
resultado = nombre in lista
print("¿"+ nombre + " esta en la lista?: " + str(resultado))
```

```
['Edain', 'Maria', 'Jose', 'Pedro', 'Manuel']
¿Jose esta en la lista?: True
```

saber el indice de el elemento buscado (index())

```
print(lista)
#condicional
if resultado:
    i = lista.index(nombre)
    print(nombre + " esta en el indice {}".format(i))
else:
    print(nombre + " NO esta en la lista")
```

```
['Edain', 'Maria', 'Jose', 'Pedro', 'Manuel']
Jose esta en el indice 2
```

Eliminar elementos

utilizando el metodo pop()

```
lista = ["Rojo", "Verde", "Azul"]
print(lista)

#Optenemos el indice del ultimo elemento
ultimo = lista.index(lista[-1])
#Eliminamos el ultimo elemento
lista.pop(ultimo)
#Nueva lista
print(lista)
```

```
['Rojo', 'Verde', 'Azul']
['Rojo', 'Verde']
```

Inserta elemento en n indice

utilizando insert(i, obj)

```
lista = ["Mexico", "Chile"]
print(lista)
#nuevo elemento
ne = "Argentina"
#insertamos en la segunda posicion
lista.insert(1,ne)
print(lista)
```

```
['Mexico', 'Chile']
['Mexico', 'Argentina', 'Chile']
```

Agregar elemento al final de la lista

usando append(obj)

```
lista = ["Perro", "Gato", "Gallina"]
print(lista)
#nuevo elemento
newElement = "Elefante"
#Agregamos al final de la lista
```

```
lista.append(newElement)
print(lista)
```

```
['Perro', 'Gato', 'Gallina']
['Perro', 'Gato', 'Gallina', 'Elefante']
```

Ordenado elementos de una lista

Normal usando --> sort()

- De la A a la Z
- Del menor al mayor

```
#Listas
lista = ["Zorro", "Pantera", "Aguila", "Gorila", "Leon"]
listaN = [2, 4, 7, 0, 12, 35, 24, 3, 2, 1, 7]
print(lista)
print(listaN)
#ordenamos
lista.sort()
listaN.sort()
print(lista)
print(listaN)
```

```
['Zorro', 'Pantera', 'Aguila', 'Gorila', 'Leon']
[2, 4, 7, 0, 12, 35, 24, 3, 2, 1, 7]
['Aguila', 'Gorila', 'Leon', 'Pantera', 'Zorro']
[0, 1, 2, 2, 3, 4, 7, 7, 12, 24, 35]
```

A la inverza usando --> sort(reverse=True)

- De la Z a la A
- Del mayor al menor

```
lista = ["Vaca", "Zorra", "Burro", "Pantera", "Aguila", "Gorila",
"Leon"]
listaN = [2, 4, 7, 0, 12, 35, 24, 3, 2, 1, 7]
print(lista)
print(listaN)
#ordenamos
lista.sort(reverse=True)
listaN.sort(reverse=True)
print(lista)
print(listaN)
```

```
['Vaca', 'Zorra', 'Burro', 'Pantera', 'Aguila', 'Gorila', 'Leon']
[2, 4, 7, 0, 12, 35, 24, 3, 2, 1, 7]
['Zorra', 'Vaca', 'Pantera', 'Leon', 'Gorila', 'Burro', 'Aguila']
[35, 24, 12, 7, 7, 4, 3, 2, 2, 1, 0]
```

Voltear una lista

voltear una lista donde el ultimo elemento sera el primero y asi sucesivamente

```
lista = [1,2,3,4,"A","B","C","D"]
print(lista)

#Volteamos la lista
lista.reverse()
print(lista)
```

```
[1, 2, 3, 4, 'A', 'B', 'C', 'D']
['D', 'C', 'B', 'A', 4, 3, 2, 1]
```

Tuplas

Son muy similares a las listas pero se definen con parentesis () y no se pueden modificar o alterar

Creacion

```
frutas = ("Mango", "Pera", "Manzana", "Melon")
print(frutas)
```

```
('Mango', 'Pera', 'Manzana', 'Melon')
```

Diccionarios

Los diccionarios en Python nos permiten almacenar una serie de mapeos entre dos conjuntos de elementos, llamados keys and values (Claves y Valores). Todos los elementos en el diccionario se encuentran encerrados en un par de corchetes {}

Creacion

```
receta = {"Agua":2, "Aceite":4, "Azucar":7, "Harina":3}
print(receta)
```

```
{'Agua': 2, 'Aceite': 4, 'Azucar': 7, 'Harina': 3}
```

Saber el indice de un elemento

```
i = receta["Agua"]
print("El indice de Agua es {}".format(i))
```

```
El indice de Agua es 2
```

Optener datos

Obtener las claves de un diccionario

Utilizando keys() y guardadolo en una lista

```
lista = list(receta.keys())
print(receta.keys())
print(lista)
```

```
dict_keys(['Agua', 'Aceite', 'Azucar', 'Harina'])
['Agua', 'Aceite', 'Azucar', 'Harina']
```

Obtener los valores de un diccionario

Utilizando values() y guardadolo en una lista

```
lista = list(receta.values())
print(receta.values())
print(lista)
```

```
dict_values([2, 4, 7, 3])
[2, 4, 7, 3]
```

Saver si un elemento esta en el diccionario

utilizando in

```
res = "Agua" in receta
t = ""
if(res):
    t = "SI"
else:
    t = "NO"

print("Agua {} esta en el diccionario".format(t))
```

```
Agua SI esta en el diccionario
```

ESTRUCTURAS CONDICIONALES

usos de if, elif y else

```
#valores
nombre = "Edain"
edad = 60
#Condicional
if edad >= 0 and edad < 17:
    print("{} es un niño".format(nombre))
elif edad >= 17 and edad < 31:
    print("{} es un joven".format(nombre))
elif edad >= 31 and edad < 60:
    print("{} es un adulto".format(nombre))
elif edad >= 60 and edad < 135:
    print("{} es un anciano".format(nombre))
else:
    print("ERROR, Edad invalida")
```

```
Edain es un anciano
```

Uso de and (Y / &&)

```
#Edades de 3 personas
edadPedro = random.randint(0, 25);
```

```
edadMaria = 18;
edadJose = 11;

print("Edad de Maria " + str(edadPedro))
print("Edad de Maria " + str(edadMaria))
print("Edad de Jose " + str(edadJose))


#si pedro es mayor que Maria Y mayor que Jose
if edadPedro > edadMaria and edadPedro > edadJose:
    print("Pedro es el hermano mayor")
#si pedro es mayor que Maria Y menor que Jose O si pedro es menor que
Maria Y mayor que Jose
elif edadPedro > edadMaria and edadPedro < edadJose or edadPedro <
edadMaria and edadPedro > edadJose:
    print("Pedro es el hermano de enmedio")
else:
    print("Pedro es el hermano menor")


Edad de Maria 12
Edad de Maria 18
Edad de Jose 11
Pedro es el hermano de enmedio
```

Uso de or (O - ||)

```
#dias de la semana
dias = ["lunes", "martes","miercoles","jueves",
"viernes","sabado","domingo"]
#Escojemos un dia de la lista
dia = random.choice(dias)

#Si el dia es sabado o domingo
if dia == "sabado" or dia == "domingo":
    print("{} es fin de semana".format(dia))
elif dia == "lunes" or dia == "martes" or dia == "miercoles" or dia ==
"jueves" or dia == "viernes":
    print("{} es entre semana".format(dia))


miercoles es entre semana
```

Uso de !=

```
#USO DE !=
#Escojemos 2 numero
n1 = random.randint(1, 3)
n2 = random.randint(1,3)

#n1 es diferente de n2
if n1 != n2:
```

```
    print("{} NO es igual a {}".format(n1,n2))
else:
    print("{} y {} son IGUALES".format(n1,n2))
```

```
1 NO es igual a 2
```

Uso de not

- En un booleano

```
a = True
#a es falso
if not a:
    print('a es falso.')
else:
    print("a es verdadero")
```

```
a es verdadero
```

- En una cadena

```
a = ""

if not a:
    print('La cadena esta vacía.')
else:
    print(a)
```

```
La cadena esta vacía.
```

- En una lista

```
a = []
if not a:
    print('La lista esta vacía.')
else:
    print(a)
```

```
La lista esta vacía.
```

- En un diccionario

```
a = dict({})
if not a:
    print('El diccionario esta vacío.')
else:
    print(a)
```

```
El diccionario esta vacío.
```

- En una tupla

```
a = tuple("hola")
if not a:
    print('La tupla esta vacía.')
else:
    print(a)
```

```
('h', 'o', 'l', 'a')
```

ESTRUCTURAS REPETITIVAS

While

Ejemplo 1

```
i = 0
j = 0
cadena = ""
noAleatorio = random.randint(3, 6)
#Estructura 1
while i < noAleatorio:
    j = 0;
    #Estructura 2
    while j < noAleatorio*2:
        cadena += str(random.randint(1,9))
        j +=1
    cadena += "\n"
    i+=1

print(cadena)
```

```
3257964274
5848484635
2359914333
4546565851
7379845755
```

Ejemplo 2

```
i = 0;
j = 0;
n = random.randint(5, 10)
dic = dict({})

while i < n:
    j = 0
    cad = ""
    while j < random.randint(3, 10):
        cad+=random.choice(string.ascii_letters)
        j+=1
    dic[ random.randint(1000, 9999)] = cad
    i+=1
print(dic)
```

```
{3937: 'aQLLw', 3965: 'xOhJ', 1757: 'SDhjUQi', 1868: 'POBMW', 8417:
'QfZQvw', 5927: 'cTzWVdLA', 2560: 'VAAKai', 5530: 'KmcTth', 2300:
'EaPGREU'}
```

Ejemplo 3 (Uso de while ... else)

Python ofrece una estructura adicional (else) en el bucle whilw

```
alumnos = ["Jose", "Pedro", "Maria", "Pedro", "Karen"]
alumnoBuscar = "Edain"
i = 0
while i < len(alumnos):
    if alumnos[i] == alumnoBuscar:
        print("Es(La) alumno(a) {} tiene es el numero {} de la
lista".format(alumnoBuscar, numeros[i]))
        break
    i += 1
else:
    print('No se encontró a el alumno {}'.format(alumnoBuscar))
```

```
No se encontró a el alumno Edain
```

Uso de for

Ejemplo 1 (Recooriendo una lista por valor)

```
nums = [4, 78, 9, 84]
for n in nums:
    print(n, end=" ")
```

```
4 78 9 84
```

Ejemplo 2 (Recooriendo diccionarios)

```
#Recorrer las claves del diccionario.
valores = {'A': 4, 'E': 3, 'I': 1, 'O': 0}
for k in valores:
    print(k, end=" ")
print()
#Iterar sobre los valores del diccionario
valores = {'A': 4, 'E': 3, 'I': 1, 'O': 0}
for v in valores.values():
    print(v, end=" ")
print()
#Iterar a la vez sobre la clave y el valor de cada uno de los
elementos del diccionario.
valores = {'A': 4, 'E': 3, 'I': 1, 'O': 0}
for k, v in valores.items():
    print('k=', k, ', v=', v, end=" ")
```

```
A E I O
4 3 1 0
k= A , v= 4 k= E , v= 3 k= I , v= 1 k= O , v= 0
```

Ejemplo 3 (for y la clase range)

- range(max): Un iterable de números enteros consecutivos que empieza en 0 y acaba en max - 1

```
#Mostrar datos de una lista
lista = list(["Perro", "Gato", "Vaca"])
for i in range (len(lista)):
    print("El elemento {} de la lista es {}".format(i, lista[i]))
```

```
El elemento 0 de la lista es Perro
El elemento 1 de la lista es Gato
El elemento 2 de la lista es Vaca
```

- range(min, max): Un iterable de números enteros consecutivos que empieza en min y acaba en max - 1

```
#Mostrar numeros de n a n-1/5 al 10
for i in range(5, 11):
    print(i, end=" ")
```

```
5 6 7 8 9 10
```

- range(min, max, step): Un iterable de números enteros consecutivos que empieza en min acaba en max - 1 y los valores se van incrementando de step en step. Este último caso simula el bucle for con variable de control.

```
#Por ejemplo, para mostrar por pantalla los números pares del 1 al 10
#podríamos usar la función range del siguiente modo:
for num in range(0, 11, 2):
    print(num, end=" ")
```

```
0 2 4 6 8 10
```

Ejemplo 4 (Uso de break. Encontrar un elemento en una colección)

```
coleccion = [2, 4, 5, 7, 8, 9, 3, 4]
for e in coleccion:
    if e == 7:
        print(e)
        break
```

```
7
```

Ejemplo 5 (Uso de continue. Imprimir solo los números pares de una colección)

```
coleccion = [2, 4, 5, 7, 8, 9, 3, 4]
for e in coleccion:
    if e % 2 != 0:
        continue
    print(e, end=" ")
```

```
2 4 8 4
```

Ejemplo 6 (Uso de for ... else)

Python ofrece una estructura adicional (else) en el bucle for

```
numeros = [1, 2, 4, 6, 5, 8, 6, 4]
for i in numeros:
    if i == 3:
```

```
        print("Es numero {}, esta en el indice {}".format(i,
numeros[i]))
        break
else:
    print('No se encontró el número 3')
```

```
No se encontró el número 3
```

CONVERSION DE DATOS

A enteros -> int(x) Convierte x en un entero

```
st = "3455"
print("cadena: {}".format(st))
n = int(st)
print("entero: {}".format(n))
```

```
cadena: 3455
entero: 3455
```

A n largos -> int(x) Convierte x en un entero largo

```
st = "3434567883555665"
print("cadena: {}".format(st))
n = int(st)
print("numero largo: {}".format(n))
```

```
cadena: 3434567883555665
numero largo: 3434567883555665
```

A flotantes -> float(x) Convierte x en un número de punto flotante

```
st = "245.3466"
print("cadena: {}".format(st))
n = float(st)
print("numero decimal: {}".format(n))
```

```
cadena: 245.3466
numero decimal: 245.3466
```

A cadenas -> str(x) Convierte x a una cadena. x puede ser del tipo float. entero o largo

```
n = 123.2
print("numero: {}".format(n))
st = str(n)
print("cadena: {}".format(st))
```

```
numero: 123.2
cadena: 123.2
```

A hexadecimal -> hex(x) Convierte x entero en una cadena hexadecimal

```
n = 46546
print("entero: {}".format(n))
```

```
hexs = hex(n)
print("cadena hexadecimal: {}".format(hexs))
```

```
entero: 46546
cadena hexadecimal: 0xb5d2
```

A caracter -> chr(x) Convierte x entero a un caracter

```
n = 234
print("entero: {}".format(n))
c = chr(n)
print("caracter: {}".format(c))
```

```
entero: 234
caracter: ê
```

A Enter C -> ord (x) Convierte el carácter x en un entero

```
c = "ê"
print("caracter: {}".format(c))
n = ord(c)
print("entero: {}".format(n))
```

```
caracter: ê
entero: 234
```

LECTURA DE DATOS

Uso de input

Ejemplo 1

```
print("¡Hola! Aqui podras realizar sumas")

#Pedir datos
n1 = input("Por favor ingrese el primer valor: ")
n2 = input("Por favor ingrese el segundo valor: ")

#numero1 será entero, así que usamos int()
n1 = int(n1)

#numero2 será un real, así que usamos float()
n2 = int(n2)

# Mostramos el resultado de la suma
print(n1, "+", n2, "=", n1 + n2)
```

```
¡Hola! Aqui podras realizar sumas
Por favor ingrese el primer valor: 455
Por favor ingrese el segundo valor: 566
455 + 566 = 1021
```

Ejemplo 2

```
print("Adivina el numero aleatorio")

#Numero aleatorio del 1 al 5
nA = random.randint(1, 5)
#No de intentos
intentos = 3
for i in range (intentos):
    n = input("Ingresa el intento {}: ".format(i+1))
    if(int(n) == nA):
        print("GANASTE, El numero aleatorio es {}".format(nA))
        break
else:
    print("PERDISTE, El numero aleatorio era {}".format(nA))
```

```
Adivina el numero aleatorio
Ingresa el intento 1: 3
GANASTE, El numero aleatorio es 3
```

FUNCIONES

Se utiliza la pabra recervada def

Ejemplo 1 (Accion)

```
def saludo(st):
    print("Hola {}".format(st))

persona = "Pedro"
saludo(persona)
```

```
Hola Pedro
```

Ejemplo 2 (Retornado un valor)

```
def sumarDatosDeUnaLista(lis):
    return sum(lis)

lista = list([1,3,6,2,7,4,9,8,3,6])
print(lista)
print("La suma total de una lista es
{}".format(sumarDatosDeUnaLista(lista)))
```

```
[1, 3, 6, 2, 7, 4, 9, 8, 3, 6]
La suma total de una lista es 49
```

*Ejemplo 3 (Usando *args)*

*args señala que llegaran n parametros a la funcion sin necesidad de ponerlos y se convertiran en una tupla

```
#Calcular promedio
def calcularPromedio(*args):
    print("la tupla que se creo es {}".format(args))
    pro = 0
    #Recorremos los n valores que nos llegaron
    for ar in args:
        pro += ar
    #Retornamos promedio
    return pro/len(args)

calificaciones = [7, 9, 10,  6, 9, 6, 7, 9, 10, 3]
prom = calcularPromedio(7, 9, 10,  6, 9, 6, 7, 9, 10, 3)
print("El promedio de las calificaciones {} es de
{}".format(calificaciones, prom))
```

```
la tupla que se creo es (7, 9, 10, 6, 9, 6, 7, 9, 10, 3)
El promedio de las calificaciones [7, 9, 10, 6, 9, 6, 7, 9, 10, 3] es
de 7.6
```

*Ejemplo 3 (Usando **kwargs)*

*kwargs (Clave/Valor) señala que llegaran n parametros a la funcion sin necesidad de ponerlos y se convertiran en un diccionario

```
def crearDiccionario(**kwargs):
    print("Diccionario que se creo: {}".format(kwargs))
    for kw in kwargs:
        print("clave: {} valor: {}".format(kw, kwargs[kw]))
crearDiccionario(n=345, nombre="Jose", lista=[3,5,7,4]);
```

```
Diccionario que se creo: {'n': 345, 'nombre': 'Jose', 'lista': [3, 5,
7, 4]}
clave: n valor: 345
clave: nombre valor: Jose
clave: lista valor: [3, 5, 7, 4]
```

EXCEPCIONES

Las principales excepciones definidas en Python son:

- TypeError : Ocurre cuando se aplica una operación o función a un dato del tipo inapropiado.
- ZeroDivisionError : Ocurre cuando se itenta dividir por cero.
- OverflowError : Ocurre cuando un cálculo excede el límite para un tipo de dato numérico.
- IndexError : Ocurre cuando se intenta acceder a una secuencia con un índice que no existe.
- KeyError : Ocurre cuando se intenta acceder a un diccionario con una clave que no existe.

- FileNotFoundError : Ocurre cuando se intenta acceder a un fichero que no existe en la ruta indicada.
- ImportError : Ocurre cuando falla la importación de un módulo.

Control de excepciones

Divicion entre 0

```
def division(a, b):

    res = ""

    try:
        result = a / b
    except ZeroDivisionError:
        res = 'ERROR, ¡No se puede dividir por cero!'
    else:
        res = 'El resultado de {} entre {} es {}'.format(a, b, (a/b))
    finally:
        print(res)


division(1, 0)
division(1, 2)
```

```
ERROR, ¡No se puede dividir por cero!
El resultado de 1 entre 2 es 0.5
```

Error en valor

```
try:
    n = int(input("Escribe un numero"))
    print("tu valor es {}".format(n))
except ValueError:
    print("ERROR, dato invalido")
```

```
Escribe un numerodf
ERROR, dato invalido
```

Sin espesificar (NO RECOMENDABLE)

```
try:
    n = int(input("Escribe un numero"))
    print("tu valor es {}".format(n))
except:
    print("ERROR, dato invalido")
```

```
Escribe un numerodf
ERROR, dato invalido
```

POO

Clases

- self es como un this
- pass es un marcador de posición que evita que el programa emita un mensaje de error. La sentencia conserva la sintaxis

```
#Clase
class Animal:
    #Contructor
    def __init__(self, especie, edad):
        self.especie = especie
        self.edad = edad

    # Método genérico pero con implementación particular
    def vacio(self):
        # Método vacío
        pass

    # Moverce
    def moverse(self, n):
        print("Soy un(a) {} y me movi {} metros".format(self.especie,
n))

    # Método genérico con la misma implementación
    def describeme(self):
        print("Soy un Animal del tipo", type(self).__name__)

leon = Animal("Leon",12)
aguila = Animal("Aguila",3)
print(type(Animal))
print(type(leon))
print(leon.moverse(2))
print(type(aguila))
print(aguila.moverse(32))
```

```
<class 'type'>
<class '__main__.Animal'>
Soy un(a) Leon y me movi 2 metros
None
<class '__main__.Animal'>
Soy un(a) Aguila y me movi 32 metros
None
```

Herencia

```
#Clase heredada
class Perro(Animal):
    def hablar(self):
        print("Guau!")
#Clase heredada
```

```
class Vaca(Animal):
    def hablar(self):
        print("Mooo!")

#Objetos
mi_perro = Perro('mamífero', 10)
mi_vaca = Vaca('mamífero', 23)
#Acciones
mi_perro.describeme()
mi_perro.moverse(33)
mi_perro.hablar()
mi_vaca.describeme()
mi_vaca.moverse(2)
mi_vaca.hablar()
```

```
Soy un Animal del tipo Perro
Soy un(a) mamífero y me movi 33 metros
Guau!
Soy un Animal del tipo Vaca
Soy un(a) mamífero y me movi 2 metros
Mooo!
```

Palabra super

```
#Clase heredada
class Oveja(Animal):

    #Constructor
    def __init__(self, especie, edad, sexo):
        # Palabra super
        super().__init__(especie, edad)
        self.sexo = sexo

    def hablar(self):
        print("Beee!")

    def miSexo(self):
        print("Soy del sexo {}".format(self.sexo))

mi_oveja = Oveja('mamifero', 4, "Hembra")

mi_oveja.describeme()
mi_oveja.moverse(33)
mi_oveja.miSexo()
mi_oveja.hablar()
```

```
Soy un Animal del tipo Oveja
Soy un(a) mamifero y me movi 33 metros
Soy del sexo Hembra
Beee!
```

Herencia multiple

En Python es posible realizar herencia múltiple. En otros posts hemos visto como se podía crear una clase padre que heredaba de una clase hija, pudiendo hacer uso de sus métodos y atributos. La herencia múltiple es similar, pero una clase hereda de varias clases padre en vez de una sola.

Veamos un ejemplo. Por un lado tenemos dos clases Clase1 y Clase2, y por otro tenemos la Clase3 que hereda de las dos anteriores. Por lo tanto, heredará todos los métodos y atributos de ambas.

```
#Clase 1
class Clase1:
    #Contructor 1
    def __init__(self, valor1, valor2):
        self.valor1 = valor1
        self.valor2 = valor2

    def dialogo1(self):
        print("Soy la clase 1 y tengo {} y {} como
valores".format(self.valor1,self.valor2))

#Clase 2
class Clase2:
    #Contructor 2
    def __init__(self, valor3, valor4):
        self.valor3 = valor3
        self.valor4 = valor4

    def dialogo2(self):
        print("Soy la clase 2 y tengo {} y {} como
valores".format(self.valor3,self.valor4))

#Clase con herencia multiple
class Clase3(Clase1, Clase2):
    #Contructor multiple
    def __init__(self, valor1, valor2, valor3, valor4, valor5,
valor6):
        #Mandamos argumentos a los padres
        Clase1.__init__(self, valor1, valor2)
        Clase2.__init__(self, valor3, valor4)
        self.valor5 = valor5
        self.valor6 = valor6

    def dialogo3(self):
        print("Soy la clase 3 y tengo {} y {} como
valores".format(self.valor5,self.valor6))
```

```
obj1 = Clase3(1,2,3,4,5,6)
obj1.dialogo1()
obj1.dialogo2()
obj1.dialogo3()
```

```
Soy la clase 1 y tengo 1 y 2 como valores
Soy la clase 2 y tengo 3 y 4 como valores
Soy la clase 3 y tengo 5 y 6 como valores
```

Es posible también que una clase herede de otra clase y a su vez otra clase herede de la anterior.

```
class Clase4:
    pass
class Clase5(Clase4):
    pass
class Clase6(Clase5):
    pass
```

INTERFACES

En la programación orientada a objetos, un interfaz define al conjunto de métodos que tiene que tener un objeto para que pueda cumplir una determinada función en nuestro sistema. Dicho de otra manera, un interfaz define como se comporta un objeto y lo que se puede hacer con el.

Ejemplo 1 - Interfaces formales

Los interfaces formales pueden ser definidas en Python utilizando el módulo por defecto llamado ABC (Abstract Base Classes). Los abc fueron añadidos a Python en la PEP3119. Simplemente definen una forma de crear interfaces (a través de metaclases) en los que se definen unos métodos (pero no se implementan) y donde se fuerza a las clases que usan ese interfaz a implementar los métodos.

El interfaz más sencillo que podemos crear es de la siguiente manera, heredando de abc.ABC.

```
from abc import ABC
class Mando(ABC):
    pass
```

La siguiente sintaxis es también válida, y aunque se sale del contenido de este capítulo, es importante que asocies el módulo abc con las metaclases.

```
from abc import ABCMeta
class Mando(metaclass=ABCMeta):
    pass
```

En este ejemplo podemos observar como se usa el decorador @abstractmethod.

Un método abstracto es un método que no tiene una implementación, es decir, que no lleva código. Un método definido con este decorador, forzará a las clases que implementen dicho interfaz a codificarlo.

Veamos como queda nuestro interfaz formal Mando.

```
#Importaciones
from abc import abstractmethod
from abc import ABCMeta

#Interface
class Mando(metaclass=ABCMeta):
    #Metodo abstracto
    @abstractmethod
    def siguiente_canal(self):
        pass

    @abstractmethod
    def canal_anterior(self):
        pass

    @abstractmethod
    def subir_volumen(self):
        pass

    @abstractmethod
    def bajar_volumen(self):
        pass

class MandoSamsung(Mando):
    def siguiente_canal(self):
        print("Samsung->Siguiente")
    def canal_anterior(self):
        print("Samsung->Anterior")
    def subir_volumen(self):
        print("Samsung->Subir")
    def bajar_volumen(self):
        print("Samsung->Bajar")

#No se puede crear un obj mando
try:
    mando = Mando()
except TypeError:
    print("Error, No se puede crear un Objeto mando con metodos
astraptos", end="\n\n")
#Obj MandoSamsung
mandoSng = MandoSamsung()
mandoSng.siguiente_canal()
mandoSng.canal_anterior()
```

```
mandoSng.subir_volumen()
mandoSng.bajar_volumen()

Error, No se puede crear un Objeto mando con metodos astraptos

Samsung->Siguiente
Samsung->Anterior
Samsung->Subir
Samsung->Bajar
```

HILOS EN PYTHON

Un hilo es un proceso del sistema operativo con características distintas de las de un proceso normal:

- Los hilos existen como subconjuntos de los procesos.
- Los hilos comparten memoria y recursos.
- Los hilos ocupan una dirección diferente en la memoria

En Python 2.X se pueden crear hilos utilizando el módulo threads y en Python 3.X se pueden crear utilizando el módulo _threads. El módulo threading será utilizado para interactuar con el módulo _threads.

¿Cuando implementar hilos? Cuando se quiera ejecutar una función al mismo tiempo que se ejecuta un programa. Cuando se crea software para servidores se quiere que el servidor no reciba solo una sino múltiples conexiones. En pocas palabras, los hilos permiten completar varias tareas al mismo tiempo.

Ejemplo 1 con hilos (se iniciaran 10. Cada uno imprimirá su ID)

```
#Importacion
import threading
# Clase hilo
class MiHilo(threading.Thread):

    #Constructor
    def __init__(self,x):
        self.__x = x
        threading.Thread.__init__(self)

     # run() se utiliza para definir el comportamiento del hilo
    def run (self):
        print(str(self.__x))


# Iniciamos 3 hilos.
for i in range(3):
    #Iniciamos
    MiHilo(i).start()
```

```
    #Pausamos
    sleep(1)
```

```
0
1
2
```

Hilos Cronometrados en Python:

En python, la clase Timer (temporizador) es una subclase de la clase Threads (hilos). Esto significa que se comporta de manera similar. La clase Timer puede ser utilizada para crear hilos temporizados. Los temporizadores son iniciados con el método .start() al igual que los hilos normales. El siguiente programa crea un hilo que se inicia luego de 3 segundos:

```
from threading import *
def hola():
    #Pausamos
    sleep(1)
    print("Hola mundo!, soy el hilo {}".format(1))
# Creacion del hilo
t = Timer(3,hola)      # Timer() recibe como primer parámetro el
intervalo de tiempo que será usado
                       # y como segundo parámetro la función a
ejecutar. También recibe como 3er parámetro
# Ejecución del Hilo   # argumentos (args) y como 4to parametro
argumentos de palabras clave (kwargs)
t.start()              # Por defecto, ambos tienen como valor 'None'
```

```
Hola mundo!, soy el hilo 1
```

FUNCIONES CON TEMPORIZADOR

Uso de la función time.time() en Python

Esta función devuelve el tiempo en segundos.

Son los segundos que pasan después de la época: el 1 de enero de 1970, 00:00:00 (UTC). Esta función utiliza el tiempo establecido del sistema informático para devolver la salida, es decir, el número de segundos.

```
#Importacion
import time

#Optenemos el tiempo actual
start = time.time()

#Bucle de 1 a 20,0000
for r in range(1,200000):
    pass
```

```
#Optenemos el tiempo actual
end = time.time()

#A el final le restamos el inicio
print(format(end-start))
```

```
0.022666454315185547
```

Entre start y end, aparece el cuerpo principal del código. Aquí, se toma como ejemplo un bucle for. Producción:

Uso de la función time.Process_time() en Python

Esta función devuelve el tiempo en fracciones de segundo.

La referencia de tiempo de todo el proceso también se registra en la función y no solo el tiempo transcurrido durante el proceso.

```
#Importacione
from time import process_time, sleep

#Tiempo inicial
start = process_time()

#Estructura repetitiva
for r in range(20):
    print(r, end=" ")

#Tiempo final
end = process_time()

#Salida
print(end, start)
print(end-start)
```

```
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 29.609375 29.609375
0.0
```

El tiempo empleado en time.sleep() no se mide con esta función, lo que significa que solo mide la diferencia de tiempo entre dos referencias de tiempo consecutivas.

Uso de la función time.Perf_counter en Python

Esta función solo debe aplicarse a procesos pequeños ya que es muy precisa.

También conocida como Contador de rendimiento, esta función ayuda a obtener el conteo de tiempo entre dos referencias de una manera más precisa. Esta función solo debe aplicarse a procesos pequeños ya que es muy precisa.

También podemos usar time.sleep() entre esta función. Con esta función, la ejecución del código puede suspenderse durante varios segundos. La función sleep() toma un valor flotante como argumento.

```
#Importaciones
from time import perf_counter, sleep

#Segundos a tardar
segundos = 3

#Inicio
start = perf_counter()

#Bucle
for r in range(n):
    #Segundo actual
    print("Segundo {}".format(r+1))
    #Pausa en segundos
    sleep(1)

#Final
end = perf_counter()

#Calcular tiempo trascurrido
print(end-start)
```

```
Segundo 1
Segundo 2
Segundo 3
3.012806399958208
```