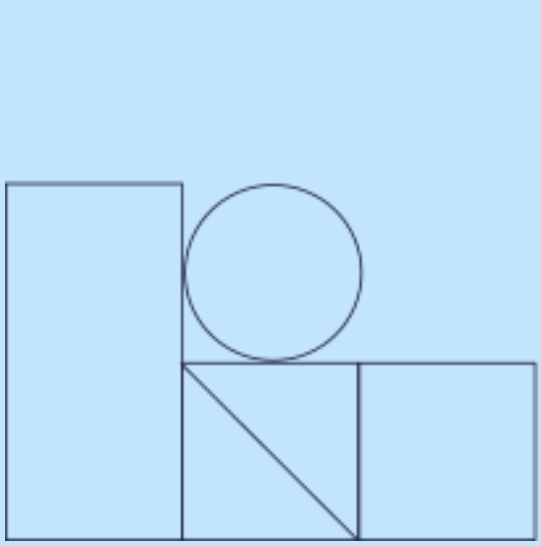
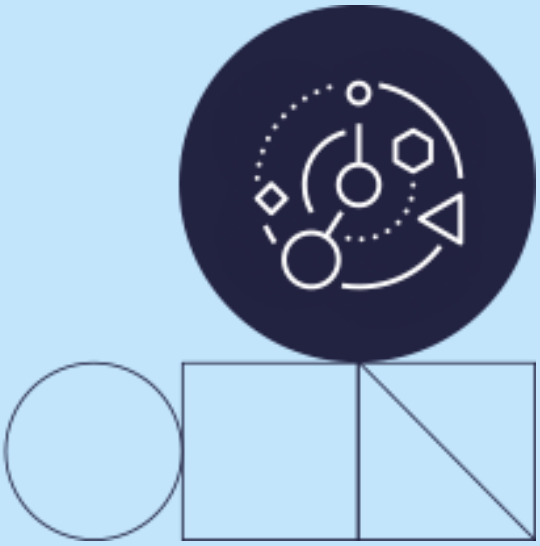
IBM SkillsBuild

IBM

Noções básicas e sintaxe de Python

Exemplos de estruturas de controle





Índice

Introdução

O código a seguir contém uma lista de declarações e operações comuns com estruturas de controle de fluxo de execução do programa Python, objetos iteráveis e funções. O formato é o de um arquivo **.py** totalmente funcional e executável, para que o aprendiz possa copiar o código e colá-lo em um ou mais arquivos **.py** e, portanto, tê-los como arquivos de referência.

Condicional if....elif...else

```
#   Programa que pide una nota por consola y valora si el alumno ha aprobado o no.

notaIn=int(input("Introduzca nota:"))

if notaIn<5:
    calificacion="Suspenso"
else: calificacion="Aprobado"

print(calificacion)

# IF no sólo evalúa un boleano, también si una variable contiene información

variable = 19

if variable:
    print("Contiene información")
else:
    print("No contiene información")


#En este ejemplo sí evalúa un boleano

variable = 19

if variable == True:
    print("Contiene información")
else:
    print("No contiene información")

#   Programa que pide una edad por consola y valora si el usuario es mayor de edad o no.

edad=int(input("Introduce edad: "))

if edad<18:
    print("No puedes pasar")
elif edad>100:
    print("Edad incorrecta")
else:
    print("Adelante")
```

```
#   Programa que pide una nota por consola y valora las posibles calificaciones del alumno.

nota=int(input("Introduce tu nota: "))

if nota<5:
    print("Suspenso")
elif nota<7:
    print("Aprobado")
elif nota<9:
    print("Notable")
else:
    print("Sobresaliente")


#   IF abreviado
n_num1 = 5
n_num2 = 10
if n_num1 > n_num2: print(n_num1 , "es mayor que" , n_num2)

#   IF...ELSE abreviado
a = 2
b = 330
print("A") if a > b else print("B")

#   Se pueden concatenar operadores de comparación:

edad=117
if 0<edad<100:  # Sería como poner: if edad>0 and edad<100
    print("Edad correcta")
else:
    print("Edad incorrecta")

 #      Otro ejemplo de operadores de comparación concatenados

salarioPresidente = int(input("Introduce salario presidente: "))
print("El salario del presidente es de" , salarioPresidente)

salarioDirector = int(input("Introduce salario Director: "))
```

```
print("El salario del director es de" , salarioDirector)

salarioJefe = int(input("Introduce salario jefe: "))
print("El salario del jefe es de" , salarioJefe)

salarioOperario = int(input("Introduce salario operario: "))
print("El salario del operario es de" , salarioOperario)

if salarioOperario<salarioJefe<salarioDirector<salarioPresidente:
    print("Todo ok")
else:
    print("Algo no va bien")

#       Operadores AND y OR
distancia = int(input("Introduce distancia: "))
numHermanos = int(input("Introduce número de hermanos en el centro: "))
notaMedia = int(input("Introduce notaMedia: "))

if distancia>20 or numHermanos<2 or notaMedia<=5:
    print("NO eres candidato a la beca")
else:
    print("Sí eres candidato a la beca")

#   Operador IN

opcion = input("ELige opcion: opcion1, opcion2, opcion3, opcion4: ")
pasoMinusculas = opcion.lower()
if pasoMinusculas in("opcion1", "opcion2", "opcion3", "opcion4"):
    print("Opción válida: " + pasoMinusculas)
else:
    print("Opción inválida: " + pasoMinusculas)
```

```
 #  If anidados. Queremos comprar un coche. Necesitamos ser mayores de edad y tener 20000€

n_edad = int(input("Introduzca su edad: "))
n_dinero = int(input("Introduzca presupuesto: "))

if n_edad < 18:
      print("No tienes la edad suficiente para conducir.")
else:
  if n_dinero < 20000:
    print("Tienes la edad pero no el dinero para comprar el coche.")
  else:
    print("Puedes comprar el coche.")
```

Operador ternário

```
#   Operador ternario

num = 12

var = "par" if (num % 2 == 0) else "impar"

print(var)

#   Sería como escribir

num = 12

if num % 2 == 0:
    print="Par"
else: print="Impar"
```

O laço for

```
# El bucle for

#   Ejecuta el print dos veces

for i in [1,10]:
  print("Hola")


#   Imprime el contenido del diccionario

for i in ["primavera", "verano", "otoño",
"invierno"]:
  print(i)

#   Repite el print tantas veces como
caracteres hay en el string

#   Evaluamos si un mail contiene el
caracter @

for i in "frase":
  print("Hola", end=" ")

miEmail=input("Introduce email")
email=False
for i in miEmail:
    if i=="@":
        email=True
if email==True:        #Se puede
simplificar    if email:
    print("El email es correcto")
else:
    print("EL mail no es correcto")

#   Podemos unir valores de texto con
valores de variable a la hora de
imprimir:

for i in range(5):
    print(f"Valor de la variable {i}")

for i in range(5,10):
    print(f"Valor de la variable {i}")
```

```
#   Podemos poner un tercer argumento con
el que especificamos de cuanto en cuanto
va el conteo:

for i in range(5,10,2):
    print(f"Valor de la variable {i}")


# validar un mail en función de si tiene
@ simplemente recorriendo la logitud del
string:

valido=False
email=input("Introduce tu email: ")

for i in range(len(email)):
    if email[i]=="@":
        valido=True

if valido:
    print("Email correcto")
else:
    print("Email incorrecto")

#   Las cadenas son objetos iterables,
contienen una secuencia de caracteres:

for x in "banana":
      print(x)

#   Con la instrucción break podemos
detener el ciclo antes de que haya pasado
por todos los elementos:
#   Salga del bucle cuando x es "banana"

frutas = ["manzana", "banana", "cereza"]
for x in frutas:
  print(x)
  if x == "banana":
    break

#   Salga del bucle cuando x es "banana",
pero esta vez el corte se produce antes
de la impresión:

frutas = ["manzana", "banana", "cereza"]
for x in frutas:
  if x == "banana":
    break
```

```
  print(x)

#   Con la instrucción continue podemos
detener la iteración actual del ciclo y
continuar con la siguiente:
#   En este caso no me imprimiría
"banana"

frutas = ["manzana", "banana", "cereza"]
for x in frutas:
  if x == "banana":
    continue
  print(x)


#   Para recorrer un conjunto de código
un número específico de veces, podemos
usar la función range ()

for x in range(6):
      print(x)

#   Función range con parámetro de inicio
incrementado por defecto en 1.

for x in range(2, 6):
      print(x)

#   Función range con parámetro de inicio
incrementado en 3.

for x in range(2, 30, 3):
      print(x)

#   Bucle for anidado
#   Imprime cada color para cada fruta:

color = ["verde", "amarilla", "roja"]
frutas = ["manzana", "banana", "cereza"]

for x in frutas:
  for y in color:
    print(x, y)


#   Los bucles for no pueden estar
vacíos.
```

```
#   Si por alguna razón tenemos un bucle
for sin contenido, usaremos la
instrucción pass para evitar un error.

for x in [0, 1, 2]:
  pass
```

O laço while

```

#   WHILE

#   Imprime edad cuando el contador
llegue a 18

edad = 0
while edad < 18:
    edad=edad+1
print("Tienes "+str(edad))

#   Pregunta la edad mientras sea
negativa

edad=int(input("Introduce edad: "))

while edad<0:
    print("Edad incorrecta")
    edad=int(input("Introduce edad: "))

print("tu edad es: "+str(edad))

#   Calcula la raiz cuadrada de un
número. Tenemos tres intentos y el número
no puede ser negativo.

import math;
intentos=0;
num = int(input("Introduce numero: "))

while num<0:
    intentos=intentos+1
    print("Incorrecto")
    num=int(input("Introduce numero: "))

    if intentos==2:
```

```
        print("Demasiados intentos")
        break;

if intentos<2:
    intentos=intentos+1
    solucion=math.sqrt(num)
    print("la raiz cuadrada de
"+str(num)+ " es: "+str(solucion))

#   Bucle while con un if anidado y un
break
#   Salga del bucle cuando num sea 3:

num = 1
while num < 6:
  print(num)
  if num == 3:
    break
  num += 1
```

```
miLista = [22, True, "una cadena", [1,
2]]
print(miLista[0])

miLista = [[1,2] , [3,4] , [5,6]]
miVar = miLista[1,1]
print(miVar)

#   Función con una lista como parámetro

def miFunccion(listaFrutas):
      for x in listaFrutas:
        print(x)

frutas = ["Manzana", "banana", "cereza"]

miFunccion(frutas)
```

Listas

```
#   LISTAS

"""
La lista es un tipo de colección ordenada
y modificable.
Es decir, una secuencia de valores de
cualquier tipo, ordenados y de tamaño
variable.
Se escriben entre corchetes. []
"""

miLista=["Angel", 43, 667767250]
miLista2 = [22, True, "una lista", [1,
2]]

#   MÉTODOS DE LAS LISTAS

#   Hacer una lista de una cadena
miLista = list("PYTHON")
print(miLista)

#   Acceder a los elementos de una lista
```

Tuplas

```
#   TUPLAS

"""Una tupla es una colección ordenada e
inmutable.
En Python, las tuplas se escriben entre
paréntesis.
"""

#       Declaración de una tupla

miTupla = ("manzana", "banana", "cereza")
print(miTupla[1])

#   Otra forma de declararla

miTupla = tuple(("manzana", "banana",
"cereza"))
print(miTupla)


#   Indexación Negativa
```

```
miTupla = ("manzana", "banana", "cereza")
print(miTupla[-1])

#   Rango de índices
#   Devuelve el tercer, cuarto y quinto
elemento:

miTupla = ("manzana", "banana", "cereza",
"naranja", "kiwi", "melon", "mango")
print(miTupla[2:5])

#   Convierta la tupla en una lista para
poder cambiarla:

miTupla = ("manzana", "banana", "cereza")
miLista = list(miTupla)
miLista[1] = "kiwi"
miTupla = tuple(miLista)

print(miTupla)

#   Recorrer una tupla

miTupla = ("manzana", "banana", "cereza")
for x in miTupla:
  print(x)

#   Comprobar si existe un elemento
#   Compruebe si "manzana" está presente
en la tupla:

miTupla = ("manzana", "banana", "cereza")
if "manzana" in miTupla:
  print("Sí, 'manzana' está en la
tupla.")

#   Otra forma, simplemente con un
boolean

print("manzana" in miTupla)

#   Longitud de la tupla

miTupla = ("manzana", "banana", "cereza")
print(len(miTupla))


# Unir dos tuplas
```

```
tupla1 = ("a", "b" , "c")
tupla2 = (1, 2, 3)

tupla3 = tupla1 + tupla2
print(tupla3)

#   Cuantas veces se encuentra el
elemento 4 en miTupla?

miTupla = ("manzana", "banana", "cereza"
, "manzana")
print(miTupla.count("manzana"))

#   Desempaquetdo de tupla

miTupla=("Angel", 4, 5.345, True, 4)
nombre, num1, num2, valor1, num3=miTupla

print(nombre)
print(num1)
print(num2)
print(valor1)
print(num3)
```

Dicionários

```
#   DICCIONARIOS

"""Los diccionarios, también llamados
matrices asociativas, deben su nombre a
que son
colecciones que relacionan una clave y un
valor.
Un diccionario es una colección
desordenada, modificable e indexada.
En Python, los diccionarios se escriben
entre llaves y tienen claves y valores.
"""

#   Declaración de un diccionario

miDiccionario = {
  "brand": "Ford",
  "model": "Mustang",
```

```
  "year": 1964
}
print(miDiccionario)


#   A los valores almacenados en un
diccionario se accede por su clave

peliculas = {"Love Actually": "Richard
Curtis",
"Kill Bill": "Tarantino",
"Amélie": "Jean-Pierre Jeunet"}

peliculas["Love Actually"]

# Reasignar valores a un diccionario

peliculas["Kill Bill"] = "Quentin
Tarantino"
print(peliculas)

#   Usar una tupla dentro de un
diccionario:

miDiccionario3={"nombre":"Jordan",
"Equipo":"Bulls", "Anillos":[1991, 1992,
1993, 1996, 1997, 1998]}
print(miDiccionario3["Anillos"])

#   Quitar valores de un diccionario

peliculas = {"Love Actually": "Richard
Curtis",
"Kill Bill": "Tarantino",
"Amélie": "Jean-Pierre Jeunet"}

peliculas.pop("Love Actually")

print(peliculas)

#   Crear un diccionario a partir de dos
listas:

lista_claves=["nombre", "edad"]
lista_valores=["Angel", 43]
diccionario = dict(zip(lista_claves ,
lista_valores))
print(diccionario)
```

```
#   Para comprobar si una clave está en
el diccionario:

"nombre" in diccionario   #Devuelve true
o false


#   Imprima las claves del diccionario:

peliculas = {"Love Actually": "Richard
Curtis",
"Kill Bill": "Tarantino",
"Amélie": "Jean-Pierre Jeunet"}
for x in peliculas:
      print(peliculas[x])

#   Longitud de un diccionario:

peliculas = {"Love Actually": "Richard
Curtis",
"Kill Bill": "Tarantino",
"Amélie": "Jean-Pierre Jeunet"}

print(len(peliculas))

#   Agregar elementos a un diccionario:

miDiccionario = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
miDiccionario["color"] = "red"
print(miDiccionario)

#   Eliminar el último elemento
insertado:

miDiccionario = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
miDiccionario.popitem()
print(miDiccionario)
```

```
#   La palabra clave del elimina el
elemento con el nombre de clave
especificado:

miDiccionario = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
del miDiccionario["model"]
print(miDiccionario)


#   La palabra clave del también puede
eliminar completamente el diccionario:

miDiccionario = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
del miDiccionario
print(miDiccionario)

#   La palabra clave clear() vacía el
diccionario:

miDiccionario = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
miDiccionario.clear()
print(miDiccionario)



#   Copiar un diccionario

miDiccionario = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
midict = miDiccionario.copy()
print(midict)


#   Otra forma de copiar un diccionario

miDiccionario = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
midict = dict(miDiccionario)
print(midict)

#   Diccionarios anidados

child1 = {
  "name" : "Emil",
  "year" : 2004
}
child2 = {
  "name" : "Tobias",
  "year" : 2007
}
child3 = {
  "name" : "Linus",
  "year" : 2011
}

myfamily = {
  "child1" : child1,
  "child2" : child2,
  "child3" : child3
}

print(myfamily["child1"])
```

Sets

```
#   SETs, conjuntos

"""
Un conjunto es una colección de elementos
únicos que no está ordenada ni indexada,
por lo que no puede estar seguro de en
qué orden aparecerán los elementos.
En Python, los conjuntos se escriben
entre llaves.
```

```
Una vez que se crea un conjunto, no puede cambiar sus elementos, pero puede agregar nuevos elementos.
"""

#   Declaración:

mi_conjunto  = {"Angel", "Sara", "Pilar"}
mi_conjunto2 = {"Angel", "Manolo", "Juan"}

#   Otra forma de declararlo

mi_conjunto3 = set(("Angel", "Sara", "Pilar"))
print(mi_conjunto3)


#   Para añadir un nuevo elemento:

mi_conjunto.add("Antonio")

#   Para añadir varios elementos:

mi_conjunto.update({"Fran", "María"})

#   Unión de colecciones. Si hay algún valor repetido sólo aparecerá una vez.

union= mi_conjunto | mi_conjunto2

#   Intersección de conjuntos:

interseccion= mi_conjunto & mi_conjunto2

#   Nos crea otro conjunto con los valores que estaban duplicados en ambos conjuntos.
#   En nuestro caso sólo Angel.

#   Diferencia de conjuntos. Nos crea otro conjunto con los elementos que no están duplicados.

diferencia= mi_conjunto - mi_conjunto2

#   Para comprobar si un elemento está en un conjunto:

"Angel" in mi_conjunto    #   Devuelve true o false

#   Recorra el conjunto e imprima los valores:

miSet = {"manzana", "banana", "cereza"}

for x in miSet:
  print(x)

"""
No puede acceder a los elementos de un conjunto haciendo referencia a un índice, ya que los conjuntos no están ordenados, los elementos no tienen índice.
"""

#   Obtenga la cantidad de elementos en un conjunto:

miSet = {"manzana", "banana", "cereza"}

print(len(miSet))

#   Elimine "banana" utilizando el método remove() :

miSet = {"manzana", "banana", "cereza"}

miSet.remove("banana")

print(miSet)

#   Elimine "banana" utilizando el método discard() :

miSet = {"manzana", "banana", "cereza"}

miSet.discard("banana")

print(miSet)

"""
Si el elemento a eliminar no existe, remove() generará un error.
Si el elemento a eliminar no existe, discard() NO generará un error.
"""
```

```
#   Elimine el último elemento utilizando el método pop() :
"""También puede usar el método pop()
para eliminar un elemento,
pero este método eliminará el último elemento.
Recuerde que los conjuntos no están ordenados,
por lo que no sabrá qué elemento se elimina.
"""

miSet = {"manzana", "banana", "cereza"}

x = miSet.pop()

print(x)

print(miSet)

#   El método clear() vacía el conjunto:

miSet = {"manzana", "banana", "cereza"}

miSet.clear()

print(miSet)

#   La palabra clave del borrará completamente el conjunto:

miSet = {"manzana", "banana", "cereza"}

del miSet

print(miSet)

#   Unión de conjuntos
#   El método union() devuelve un nuevo conjunto con todos los elementos de ambos conjuntos:

set1 = {"a", "b" , "c"}
set2 = {1, 2, 3}

set3 = set1.union(set2)
print(set3)
```

```
#   El método update() inserta los elementos en set2 en set1:

set1 = {"a", "b" , "c"}
set2 = {1, 2, 3}

set1.update(set2)
print(set1)

#   Tanto union() como update() excluirán cualquier elemento duplicado.
```

Funções

```
#   -----------------------------------
#   FUNCIONES
#   -----------------------------------

#   Definición de una función. Importante la identación:
def my_funcion():
      print("Estamos ejecutando la función.")

#   Llamada a la función. En otra parte de mi código, llamamos a la función para que se ejecute:

my_funcion()

#   -----------------------------------


def suma():
    num1 = 3
    num2 = 5
    print("suma =", num1+num2)

suma()

# Otra opción:
def suma():
      num1 = 3
      num2 = 5
```

```
      resultado = num1 + num2
      return resultado



print(suma())

#   ------------------------------------
#El bloque de código que ejecutará la
función incluye todas las declaraciones
con indentación
# dentro de la función.

def miFunción():
    print('this will print')
    print('so will this')

x = 7
# la asignación de x no es parte de la
función ya que no está indentada

#--------------------------------------
#Las variables definidas dentro de una
función solo existen dentro del ámbito de
esa función.


def duplica(num):
    x = num * 2
    return x

print(x)  # error - x no está definida
print(duplica(4))  # muestra 8

#--------------------------------------

#Si la definición de una función incluye
parámetros, debe proporcionar el mismo
número
# de parámetros cuando llame a la
función.
def multiplica(arg1, arg2):
      return arg1 * arg2

#print(multiplica(3))  # TypeError:
multiplica() utiliza exactamente 2
argumentos (0 proporcionados)
```

```
print(multiplica('a', 5))  # 'aaaaa'
mostrado en la consola

#print(multiplica('a', 'b'))  #
TypeError: Python no puede multiplicar
dos strings

#--------------------------------------
#Puedes pasar los parámetros en el orden
que desees, utilizando el nombre del
parámetro.
def suma(a, b):
    return a + b

result = suma(b=2, a=3)
print(result)
# result = 4

#--------------------------------------

#También podríamos pasar varios valores
que retornar a return.
def f(x, y):
      return x * 2, y * 2

a, b = f(1, 2)

""" Sin embargo, esto no quiere decir que
las funciones Python puedan de-volver
varios valores,
 lo que ocurre en realidad es que Python
crea una tupla al vuelo cuyos elementos
 son los valores a retornar, y esta única
variable es la que se devuelve."""




#   ------------------------------------
# función con un parámetro
#   ------------------------------------

#   Declaración de una función

def miFuncion(num1, num2):
    return(num1+num2)


#   Llamada a la función
```

```
print(miFuncion(2, 3))


#   ------------------------------------
def holaConNombre(name):
  print("Hola " + name + "!")

holaConNombre("Angel")  # llamada a la
función, 'Hola Angel!' se muestra en la
consola
#   ------------------------------------


#-------------------------------------
#   Funcion con parametro por defecto
#-------------------------------------

def imprimir(precio, iva = 1.21):
    print(precio * iva)

imprimir(300, 1.08)

#   Funciones con argumentos variables
#   Me crea una tupla de nombre "otros"

def varios(param1, param2, *otros):
    for val in otros:
        print (val)

varios(1, 2)
varios(1, 2, 3)
varios(1, 2, 3, 4)


"""
También se puede preceder el nombre del
último parámetro con **,
en cuyo caso en lugar de una tupla se
utilizaría un diccionario.
Las claves de este diccionario serían los
nombres de los parámetros
indicados al llamar a la función y los
valores del diccionario,
los valores asociados a estos parámetros.
"""

def varios(param1, param2, **otros):
    for i in otros.items():
        print (i)

varios(1, 2, tercero = 3)

# ---------------------------
#-------------------------------------
#ARGUMENTOS VARIABLES EN FUNCIONES. EL
ARGUMENTO CON * SERÁ UNA TUPLA
# PYTHON NO TIENE SOBRECARGA DE FUNCIONES
#-------------------------------------
def listarNombres(*nombres):
    for nombre in nombres:
        print(nombre)

listarNombres('Juan', 'Karla', 'María',
'Ernesto')
listarNombres('Laura', 'Carlos')

#-------------------------------------
# hACER LO MISMO PERO PASANDO
DICCIONARIOS COMO ARGUMENTOS. KWARGS

def listarTerminos(**KWARGS):
    for clave, valor in KWARGS.items():
        print(f'{clave}: {valor}')

listarTerminos(IDE='Integrated
Developement Environment', PK='Primary
Key')
listarTerminos(DBMS='Database Management
System')


#-------------------------------------

def mi_funcion(nombre, apellido):
    print('saludos desde mi función')
    print(f'Nombre: {nombre}, Apellido:
{apellido}')
    # Sería como imprimir así:
    print('Nombre:', nombre, 'Apellido:',
apellido)

mi_funcion('Juan', 'Perez')
mi_funcion('Karla','Lara')

#-------------------------------------
# RETURN
#-------------------------------------
```

```
#function definiciones de function no
pueden estar vacías, pero si por alguna
razón tiene
# una definición de function sin
contenido, ingrese la instrucción pass
para evitar un error.


def myfunction():
  pass

#--------------------------------------

def sumar(a, b):
    return a + b

resultado = sumar(5, 3)
print(f'Resultado sumar: {resultado}')
# print(f'Resultado sumar: {sumar(5,3}')

#También podíamos haber llamado a la
función dentro de nuestro método print
print(f'Resultado sumar: {sumar(5,3)}')

# --------------------------
# función con múltiples parámetros con
una sentencia de retorno
def multiplica(val1, val2):
  return val1 * val2

multiplica(3, 5)  # muestra 15 en la
consola

#--------------------------------------

# esta es una función básica de suma
def suma(a, b):
  return a + b

result = suma(1, 2)
# result = 3


#--------------------------------------
# VALORES POR DEFECTO
#--------------------------------------
# esta es una función básica de suma con
balores predeterminados
```

```
def suma(a, b=3):
  return a + b

result = suma(1)
# result = 4


#--------------------------------------
# Indicio de qué tipo de dato vamos a
manejar:
#--------------------------------------

def sumar(a:int = 0, b:int = 0) -> int:
#def sumar(a = 0, b = 0):
    return a + b

    resultado = sumar()
#print(f'Resultado sumar: {resultado}')
print(f'Resultado sumar: {sumar(45,
654)}')
#uanque le hemos dicho el tipo de los
parámetros no estamos obligados a
cumplirlo.
print(f'Resultado sumar: {sumar("aNGEL",
"Garcia")}')



#--------------------------------------
"""
Crear una función para sumar los valores
recibidos de tipo numérico,
utilizando argumentos variables *args
como parámetro de la función
y regresar como resultado la suma de
todos los valores pasados como
argumentos.
"""

# Definimos nuestra funcion para sumar
valores
def sumar_valores(*args):
    resultado = 0
    # Iteramos cada elemento
    for valor in args:
        # resultado = resultado + valor
        resultado += valor
    return resultado
```

```
# Llamada a la funcion
print(sumar_valores(3, 5, 9, 4, 6, 45,
444))

#--------------------------------------
# Distintos tipos de datos como
argumentos en Python

def desplegarNombres(nombres):
    for nombre in nombres:
        print(nombre)

#nombres = ['Juan', 'Karla', 'Guillermo']
#desplegarNombres(nombres)
#desplegarNombres('Carlos')
#desplegarNombres((8, 9))
desplegarNombres([10, 11])

#--------------------------------------
# FUNCIONES RECURSIVAS

# 5! = 5 * 4 * 3 * 2 * 1
# 5! = 5 * 4 * 3 * 2
# 5! = 5 * 4 * 6
# 5! = 5 * 24
# 5! = 120

def factorial(numero):
    if numero == 1:
        return 1
    else:
        return numero * factorial(numero-
1)

numero = 6
resultado = factorial(numero)
print(f'El factorial de {numero} es
{resultado}')

#--------------------------------------
"""
Imprimir numeros de 5 a 1 de manera
descendente usando funciones recursivas.
Puede ser cualquier valor positivo,
ejemplo, si pasamos el valor de 5, debe
imprimir:
5
4
3
2
1

En caso de pasar el valor de 3, debe
imprimir:
3
2
1

Si se pasan valores negativos no imprime
nada
"""

def imprimir_numero_recursivo(numero):
    if numero >= 1:
        print(numero)
        imprimir_numero_recursivo(numero
- 1)
    elif numero == 0:
        return
    elif numero < 0:
        print('Valor incorrecto...')

imprimir_numero_recursivo(5000000)

#--------------------------------------

def cuenta_regresiva(numero):
      numero -= 1
      if numero > 0:
            print (numero)
            cuenta_regresiva(numero)
      else:
            print ("Booooooooom!")
            print ("Fin de la función"),
numero

cuenta_regresiva(5)

#--------------------------------------
"""
Ejercicio: Calculadora de Impuestos
Crear una función para calcular el total
de un pago incluyendo un impuesto
aplicado.
# Formula: pago_total = pago_sin_impuesto
+ pago_sin_impuesto * (impuesto/100)
"""
```

```
# Funcion que calcula el total de un pago incluyendo el impuesto
def
calcular_total_pago(pago_sin_impuesto, impuesto):
    pago_total = pago_sin_impuesto + pago_sin_impuesto * (impuesto/100)
    return pago_total

# Ejecutamos la funcion
pago_sin_impuesto =
float(input('Proporcione el pago sin impuestos: '))
impuesto = float(input('Proporcione el monto del impuesto:'))
pago_con_impuesto =
calcular_total_pago(pago_sin_impuesto, impuesto)
print(f'Pago con impuesto: {pago_con_impuesto}')

#---------------------------------------
#---------------------------------------
"""
Ejercicio: Convertidor de Temperatura
Realizar dos funciones para convertir de grados celsius a fahrenheit y viceversa.
"""

# Funcion que convierte de celsius a fahrenheit
def celsius_fahrenheit(celsius):
    return celsius * 9 / 5 + 32

# Funcion que convierte de fahrenheit a celsius
def fahrenheit_celsius(fahrenheit):
    return (fahrenheit - 32) * 5 / 9

# Realizamos algunas pruebas de conversion
celsius = float(input('Proporcione su valor en celsius: '))
resultado = celsius_fahrenheit(celsius)
# Imprimimos el resultado
print(f'{celsius} C a F: {resultado:.2f}')
```

```
# Realizamos la prueba de grados fahrenheit a celsius
fahrenheit = float(input('Proporcione su valor en fahrenheit: '))
resultado =
fahrenheit_celsius(fahrenheit)
# Imprimimos el resultado
print(f'{fahrenheit} F a C: {resultado:0.2f}')
```