

```
In [ ]: import warnings
warnings.filterwarnings('ignore')
```

PREGUNTAS GENERALES

1. ¿ POR QUÉ SE LLAMA Python ?

https://es.wikipedia.org/wiki/Historia_de_Python

Por Monty Python

2. en qué año se creo Python?

<https://es.wikipedia.org/wiki/Python>

Publicado en 1991, aunque desde 1989 hay artículos sobre ello

La historia del lenguaje de programación Python se remonta hacia finales de los 80s

y principio de los 90s, su implementación comenzó en diciembre de 1989

cuando en Navidad Guido Van Rossum que trabajaba en el (CWI)

(un centro de investigación holandés de carácter oficial que, entre otras cosas,

3. quién lo creó?

https://es.wikipedia.org/wiki/Guido_van_Rossum

4. por qué a PyPI lo llaman la quesería

<https://stackoverflow.com/questions/5393986/why-was-pypi-called-the-cheese-shop>

al principio no había muchas librerías disponibles en PyPI al igual que

en un sketch de la comedia Monty Python

The insert() method

insert: ejemplo

```
In [1]: # The insert() method
# inserts an element to the list at the specified index

# crea una lista de vocales
vocal = ['a', 'e', 'i', 'u']

# 'o' is inserted at index 3 (4th position)
vocal.insert(3, 'o')

print('Listado:', vocal)

# Output: List: ['a', 'e', 'i', 'o', 'u']
```

Listado: ['a', 'e', 'i', 'o', 'u']

suma de elementos de una lista

```
In [2]: L = [10,20,30]
sum(L)
```

Out[2]: 60

```
In [3]: # L.sum()

# AttributeError: 'list' object has no attribute 'sum'
```

OTRA FORMA DE SUMAR LOS ELEMENTOS DE UNA LISTA

```
In [4]: suma=0
for numero in L:
    suma = suma + numero
# 10  <=  0  + 10
# 30  <= 10  + 20
# 40  <= 30  + 30
print('la suma de los elementos de la lista es:', suma)
```

la suma de los elementos de la lista es: 60

borrar elementos de una lista en un index concreto

```
In [5]: L
```

Out[5]: [10, 20, 30]

```
In [6]: # SI QUIERO BORRAR EL ELEMENTO DE INDEX 1
del L[1]

L
```

Out[6]: [10, 30]

insert + del + sum(lista) : pregunta del examen

```
In [7]: # cuál es el resultado del siguiente fragmento de código?
# A) 4
# B) 3
# C) NINGUNA DE LAS ANTERIORES
# D) EL código es erróneo

x = [0, 1, 2]
x.insert(0, 1)
del x[1]
print(sum(x))
```

4

```
In [8]: # resuelto paso a paso
```

```
In [9]: x = [0, 1, 2]      # 0 1 2
print('x inicial:', x)

x.insert(0, 1)      # 1 0 1 2      (insertas en indice 0 un "1")
print('x insertando en index 0 un "1":', x)

del x[1]            # 1  1 2  =====> 1 1 2
print("x borrando el elemento en index [1]:", x)

print('la suma de todos los valores que componen x: (sum(x))')
print(sum(x))      # 4              (1+1+2)
```

```
x inicial: [0, 1, 2]
x insertando en index 0 un "1": [1, 0, 1, 2]
x borrando el elemento en index [1]: [1, 1, 2]
la suma de todos los valores que componen x: (sum(x))
4
```

Funciones y argumentos: ejemplo 1

```
In [10]: # cuál es la salida del siguiente fragmento de código?
# A) 3 3
# B) 3 2
# C) ninguna de las anteriores
# D) el código es erróneo

def test(x=1, y=2):
    x = x + y
    y += 1
    print(x, y)
```

```
test(2, 1)
```

3 2

```
In [11]: # y+=1  equivalente a: y = y + 1
```

```
In [12]: def test(x=1, y=2):  
    x = x + y      # ==> x = 2 + 1 = 3, x = 3, y = 1  
    y += 1         # ==> y = y + 1 ==> y = 2  
    print(x, y)  
  
test(2, 1)
```

3 2

el ejemplo explicado paso a paso

para comprobar que una vez modificamos los valores de los argumentos en la propia llamada

éstos son modificados

```
In [13]: def test(x=1, y=2):  
    print('argumentos de la función: x=1, y=2')  
    print('valores en la llamada:    x=2, y=1')  
    print("valores reales en este instante:")  
    print('x =', x, 'y =', y)  
    print("\n")  
    x = x + y      # x = x + y  -> 3 <= 2 + 1  
    y += 1         # y = y + 1  -> 2 <= 1 + 1  
    print(x, y)    # 3, 2
```

```
In [14]: test(2, 1)
```

argumentos de la función: x=1, y=2
valores en la llamada: x=2, y=1
valores reales en este instante:
x = 2 y = 1

3 2

```
In [15]: test(4, 3)
```

argumentos de la función: x=1, y=2
valores en la llamada: x=2, y=1
valores reales en este instante:
x = 4 y = 3

7 4

Funciones y argumentos: ejemplo 2

```
In [16]: # ejercicio parecido
```

```
# en la llamada a la función al decir test(2,1)
```

```
# sabe que se refiere a x=2, y=1 ?

# RESPUESTA: SI
# porque va hacia la llamada y se encuentra en ese orden

# def test(x,y):
# que en este caso se encuentra asi:
# def test(x=1, y=2)
```

```
In [17]: # en base a ello..

# calcula el resultado del siguiente trozo de código

# piensa si funciona el código y, si es que si, qué valor devuelve la lla

# cuál es la salida del siguiente fragmento de código?
# A) 3 3
# B) 3 2
# C) ninguna de las anteriores
# D) el código es erróneo
```

```
In [18]: def test(x=1, y=2):
          x = x + y      # ==> x = 3 - y = 2
          y += 1         # ==> x = 3 - y = 3
          print(x, y)

          test(y=2, x=1)
```

3 3

```
In [19]: test(y=3, x=4)

# x = x + y ==> x = 4 + 3 = 7 ==> x = 7 - y = 3
# y = y + 1 =====> x = 7 - y = 4
# print(x,y) =====> 7 4
```

7 4

```
In [20]: test(3, 4)    # ==> x = 3, y = 4

# x = x + y ==> x = 3 + 4 = 7 ==> x = 7 - y = 4
# y = y + 1 =====> x = 7 - y = 5
# print(x,y) =====> 7 5
```

7 5

Funciones y argumentos: ejemplo 3

```
In [21]: # otro ejemplo parecido,

# pero en la llamada no indicamos qué valores tienen "x" e "y"
```

```
In [22]: def test(x=1, y=2):
          x = x + y      # x = 1 + 2 =====> x = 3
          y += 1         # y = y + 1 =====> y = 3
          print(x, y)    # print(x,y) =====> 3 3
```

```
test()
```

3 3

Funciones y argumentos: ejemplo 4

```
In [23]: # Ojo, que no es lo mismo que este otro ejemplo porque aquí NO SABE,
# lo que valen 'x' e 'y'
```

```
In [24]: """
def test(x, y):
    x = x + y          # x = 1 + 2 =====> x = 3
    y += 1             # y = y + 1 =====> y = 3
    print(x, y)        # print(x,y) =====> 3 3

test()"""
```

```
Out[24]: '\ndef test(x, y):\n    x = x + y          # x = 1 + 2 =====> x = 3\n    \n    y += 1             # y = y + 1 =====> y = 3\n    print(x, y)\n    # print(x,y) =====> 3 3\n    \n\ntest()'
```

Funciones y argumentos: ejemplo 5

omitiendo algún valor en la llamada

```
In [25]: def test(x=1, y=2):
    x = x + y          # x =
    y += 1             # y = y + 1
    print(x, y)        #

test(2)
```

4 3

Funciones y argumentos: ejemplo 6

```
In [26]: def test(x=1, y=2):
    x = x + y          # x = 1 + 4 =====> x = 5 , y = 4
    y += 1             # y = y + 1 =====> x = 5 , y = 5
    print(x, y)        # print(x,y) =====> 5 5

test(y=4)
```

5 5

Funciones y argumentos: ejemplo 7

mismo ejemplo sin decir y=4, simplemente 4

```
In [27]: def test(x=1, y=2):
    x = x + y          # x = 4 + 2 =====> x = 6, y = 2
    y += 1             # y = y + 1 =====> x = 6, y = 3
    print(x, y)        # print(x,y) =====> 6 3
```

```
test(4)
```

6 3

Funciones y argumentos: ejemplo 8

otro ejemplo más donde vemos que una coma la admite en la llamada

sin necesidad de establecer el valor de "y"

```
In [28]: def test(x=1, y=2):  
        x = x + y      # x = 4 + 2 =====> x = 6, y = 2  
        y += 1         # y = y + 1 =====> x = 6, y = 3  
        print(x, y)    # print(x,y) =====> 6 3  
  
test(4,)
```

6 3

Funciones y argumentos: ejemplo 9

ejemplo donde vemos que no permite poner varias comas

```
In [29]: def test(x=1, y=2, z=2):  
        x = x + y      # x = 4 + 2 =====> x = 6, y = 2  
        y += 1         # y = y + 1 =====> x = 6, y = 3  
        z = z + 1  
        print(x, y, z) # print(x,y) =====> 6 3  
  
test(4,,2)  
  
# SyntaxError: invalid syntax
```

Cell In[29], line 7

```
test(4,,2)  
      ^
```

SyntaxError: invalid syntax

Funciones y argumentos: ejemplo 10

```
In [30]: def test(x=1, y=2):  
        x = x + y      #  
        y += 1         # y = y + 1 =====>  
        print(x, y)    # print(x,y) =====>  
  
test(,4)  
  
# SyntaxError: invalid syntax
```

Cell In[30], line 6

```
test(,4)  
    ^
```

SyntaxError: invalid syntax

NO hacer copias de listas

y lo que ello implica..

copias y asignación de listas: ejemplo 1

CUANDO QUIERO UNA COPIA DE UNA LISTA SE DEBE HACER ASI:

```
In [31]: listado = [1,2,3]

        copia_listado = listado.copy()
        copia_listado
```

```
Out[31]: [1, 2, 3]
```

```
In [32]: id(listado)
```

```
Out[32]: 140250412561664
```

```
In [33]: id(copia_listado)
```

```
Out[33]: 140250286128512
```

```
In [34]: del listado[1]
        listado
```

```
Out[34]: [1, 3]
```

```
In [35]: copia_listado
```

```
Out[35]: [1, 2, 3]
```

hemos visto que aunque hemos modificado la lista inicial "listado"

HEMOS MANTENIDO LOS VALORES INICIALES EN "COPIA_LISTADO"

copias y asignación de listas: ejemplo 2

creo una lista (lista inicial) asigno otra lista y le asigno esa lista inicial modifico un elemento de la lista inicial

COMPRUEBO que TAMBIÉN modifica la segunda lista (lista 2)

```
In [36]: list1 = [1, 3]
        list2 = list1 # list2 = [1, 3]

        list1[0] = 4 # list1 = [4, 3]

        print(id(list1))
        print(id(list2))
```



```
print('list2:', list2)
print('list1:', list1)
```

```
140250285853120
140250285853120
list2: [4, 3]
list1: [4, 3]
```

copias y asignación de listas: ejemplo 3

ahora modifico la lista 2 y compruebo..

```
In [37]: list1 = [1, 3]
list2 = list1 # list2 = [1, 3]

list2[0] = 4 # list2 = [4, 3]

print('list2:', list2)
print('list1:', list1)
```

```
list2: [4, 3]
list1: [4, 3]
```

VEMOS que en el momento que asignas una a otra,

la otra también se modifica

```
In [38]: list1==list2
```

```
Out[38]: True
```

```
In [39]: if list1==list2:
print('son ambas [4,3]')
```

```
son ambas [4,3]
```

Aquí vemos que:

en el momento que no hago una copia y uso el operador = (operador de asignación)

lo que nos encontramos es que cuando modificas una lista (la lista original)

estamos también modificando la otra

copias y asignación de listas: ejemplo 4

asignación de valores (varios valores),

nota, recordar Numpy, porque era similar

```
In [40]: numeros = [1, 2, 3]
numeros
```

```
Out[40]: [1, 2, 3]
```

```
In [41]: numeros[0:2] # 0 hasta 2 - 1 = 0 a 1
```

```
Out[41]: [1, 2]
```

```
In [42]: numeros[:2] # 0 hasta 1
```

```
Out[42]: [1, 2]
```

```
In [43]: numeros[1]
```

```
Out[43]: 2
```

```
In [44]: numeros[1:2] # 1 hasta 1
```

```
Out[44]: [2]
```

```
In [45]: type(numeros[1:2])
```

```
Out[45]: list
```

```
In [46]: type(numeros[1])
```

```
Out[46]: int
```

```
In [47]: numbers = [1,2,3]
numbers
```

```
Out[47]: [1, 2, 3]
```

```
In [48]: del numbers[1:2] # 1 hasta 2 -1 --> de 1 al 1 --> eliminar el valor de po
numbers
```

```
Out[48]: [1, 3]
```

copias y asignación de listas: ejemplo 5

el ejercicio de examen

```
In [49]: nums = [1, 2, 3]
vals = nums # vals = [1, 2, 3]
del vals[1:2]
# añadido:
print('vals:', vals) # borro 2, [1,3]
print('nums:', nums) # = que vals

# conclusión:
# SI NO HACES UNA COPIA DE LA LISTA,
# TE GUARDA LOS CAMBIOS
```

```
vals: [1, 3]
```

```
nums: [1, 3]
```

copias y asignación de listas: ejemplo 6

lo mismo que ejemplo 5 pero modificando la otra lista

```
In [50]: nums = [1, 2, 3]
vals = nums          # vals = [1, 2, 3]
del nums[1:2]
# añadido:
print('vals:', vals)  # borro 2, [1,3]
print('nums:', nums)  # = que vals

# conclusión:
# SI NO HACES UNA COPIA DE LA LISTA,
# TE GUARDA LOS CAMBIOS
```

```
vals: [1, 3]
nums: [1, 3]
```

copias y asignación de listas: ejemplo 7

otro ejercicio similar, pero con copias de listas

```
In [51]: listado = [1,2,3]

copia_listado = listado.copy()
copia_listado
```

```
Out[51]: [1, 2, 3]
```

```
In [52]: listado[0] = 1000
print('listado:', listado)
print('copia_listado:', copia_listado)
```

```
# posibles soluciones

# A)
# listado = [1,2,3]
# copia_listado = [1,2,3]

# B)
# listado = [1000,2,3]
# copia_listado = [1,2,3]

# C)
# listado = [1000,2,3]
# copia_listado = [1000,2,3]

# D)
# ninguna de las anteriores
```

```
listado: [1000, 2, 3]
copia_listado: [1, 2, 3]
```

copias y asignación de listas: ejemplo 8

```
In [53]: list1 = [1, 3] # [1, 3]
list2 = list1 # [1, 3]
list1[0] = 4 # [4, 3]
print('list1 antes de modificar el 10 de index 1', list1)
```

```

print('lista2 antes de modificar el 10 de index 1', list2)
list2[1] = 10 # [4, 10]

print('list1:', list1)
print('list2:', list2)

# A. [4,10] en ambas listas

'''
list1 = [1, 3]
list1 = [4, 3]
list1 = [4, 10]

list2 = [1, 3]
list2 = [4, 3]
list2 = [4, 10]
'''

```

lista1 antes de modificar el 10 de index 1 [4, 3]
 lista2 antes de modificar el 10 de index 1 [4, 3]
 list1: [4, 10]
 list2: [4, 10]

Out[53]: '\nlist1 = [1, 3]\nlist1 = [4, 3]\nlist1 = [4, 10]\n\n\nlist2 = [1, 3]\nlist2 = [4, 3]\nlist2 = [4, 10]\n'

copias y asignación de listas: ejemplo 9

In [54]:

```

list1 = [1, 3]
list2 = list1
list1[0] = 4
list2[1] = 10 # [4, 10]
del list1[0] # [10]

print('list1:', list1)
print('list2:', list2)

# A.
# lista 1 => [10]
# lista 2 => [10]

```

list1: [10]
 list2: [10]

copias y asignación de listas: ejemplo 10

In [55]:

```

list1 = [1, 3]
list2 = list1
list1[0] = 4
list2[1] = 10
del list1[0] # [10]
del list2[0] # []

print('list1:', list1)
print('list2:', list2)

# []

```

```
list1: []  
list2: []
```

copias y asignación de listas: ejemplo 11

```
In [56]: # cuál es la salida del siguiente código  
# A. lista vacía  
# B. 10  
# C. el código es erróneo  
# D. ninguna de las anteriores  
  
'''  
list1 = [1, 3]  
list2 = list1  
list1[0] = 4  
list2[1] = 10  
del list1[0] # [10]  
del list2[1]  
  
print('list1:', list1)  
print('list2:', list2)  
'''  
  
# ERROR QUE ME DEVUELVE:  
# IndexError: list assignment index out of range  
  
# EXPLICACIÓN  
# cuando tienes en la penúltima línea: [10]  
# tal y como vimos en el ejemplo 10. TENEMOS 1 ÚNICO ELEMENTO.  
# SI TIENES 1 ELEMENTO, ==> NO Puedes eliminar el de index 1, solo index 0  
# (como en el anterior ejemplo)  
  
# POR ESO DEVUELVE UN ERROR
```

```
Out[56]: "\nlist1 = [1, 3]\nlist2 = list1 \nlist1[0] = 4 \nlist2[1] = 10 \nde  
l list1[0] # [10]\ndel list2[1] \n\nprint('list1:', list1) \nprint('lis  
t2:', list2)\n"
```

Funciones: variables globales y locales

(el ejercicio de examen es el primero)

variables globales Vs variables locales en las funciones

Variables globales y locales: ejemplo 1

```
In [57]: num = 1  
  
def func(num):  
    num = num + 3  
    print(num)
```

```
func(1)

print(num)
```

4
1

Variables globales y locales: ejemplo 2

```
In [58]: num = 1

def func(num):
    num = num + 3
    print(num)

func(6)

print(num)
```

9
1

Variables globales y locales: ejemplo 3

```
In [59]: num = 1

def func():
    global num
    num = num + 3
    print(num)

func()
print(num)
```

4
4

Variables globales y locales: ejemplo 4

```
In [60]: num = 1

def func():
    global num
    num = num + 3
    print(num)

func()
num = 2
num = 20
print(num)
```

4
20

Variables globales y locales: ejemplo 5

```
In [61]: '''
num = 1

def func():
    num = num + 3
    print(num)

func()

print(num)
'''

# UnboundLocalError: local variable 'num' referenced before assignment

# 'num' ahora DESCONOCEMOS SU VALOR EN "num = num + 3"
# PORQUE ACTÚA DE MANERA LOCAL
```

```
Out[61]: '\nnum = 1\n \ndef func():\n     num = num + 3   \n     print(num)       \n\nfunc()               \n \nprint(num)   \n'
```

Variables globales y locales: ejemplo 6

el mismo ejemplo PERO AHORA SI LE DIGO UN POSIBLE VALOR PARA NUM

pudiera ser también de examen

```
In [62]: num = 1

def func():
    num = 10
    num = num + 3
    print(num)

print('la llamada a la función: ')
func()
print('\n')

print('el valor de num original...:', num)
```

la llamada a la función:
13

el valor de num original...: 1

Variables globales y locales: ejemplo 7

ojo, que, aunque la variable sea global ESTAMOS REASIGNANDO VALORES A LA VARIABLE

```
In [63]: num = 1

def func():
    global num
    num = 10
    num = num + 3
```

La llamada a la función:
13

```
el valor de num original...: 13
```

Variables globales y locales: ejemplo 8

LA PALABRA GLOBAL DESPUÉS DE "num = 10"

In [64]:

```
"""
num = 1

def func():
    num = 10
    global num
    num = num + 3
    print(num)

print('la llamada a la función: ')
func()
print('\n')

print('el valor de num original...:', num)
"""
```

```
Out[64]:  """\nnum = 1\n\ndef func():\n    num = 10\n    global num\n    num = num + 3\n    print(num)\n\n\nprint('la llamada a la función: ')\nfunc()\nprint('\n')\nprint('el valor de num original...', num)\n"""
```

Variables globales y locales: ejemplo 9

ojo, que, aunque la variable sea global ESTAMOS REASIGNANDO VALORES A LA VARIABLE

AQUI, PARA VERLO DEFINITIVAMENTE, HEMOS REASIGNADO VARIAS VECES

In [65]:

```
num = 1

def func():
    global num
    num = 10
    num = 20
    num = 30
    num = num + 3
```



```

    print(num)

    print('la llamada a la función: ')
    func()
    print('\n')

    print('el valor de num original...:', num)

```

la llamada a la función:
33

el valor de num original...: 33

Variables globales y locales: ejemplo 10

un ejemplo diferente (el cual si que funciona)

es una posible corrección al código del examen

```

In [66]: num = 1

def func():
    global num
    num = num + 3
    print(num)

func()

print(num)

```

4
4

Ejemplo con try except

(el primero es el de examen)

```

In [67]: """
try:
    print(5/0)
    break
except:
    print("Sorry, something went wrong...")
except (ValueError, ZeroDivisionError):
    print("Too bad...")
"""

# SyntaxError: 'break' outside loop

```

```

Out[67]: '\ntry:\n    print(5/0)\n    break\nexcept:\n    print("Sorry, something
went wrong...")\nexcept (ValueError, ZeroDivisionError):\n    print("Too
bad...")\n'

```

```
In [68]: # ejemplo 2 de este tipo (este si funciona)

# se ha intentado que ejecute la parte de ZeroDivisionError
```

```
In [69]: try:
          print(5/0)
        except (ValueError, ZeroDivisionError):
          print("Too bad...")
        except:
          print("Sorry, something went wrong...")
```

Too bad...

```
In [70]: try:
          print(5/0)
        except (ValueError):
          print("Too bad...")
        except:
          print("Sorry, something went wrong...")
```

Sorry, something went wrong...

```
In [71]: try:
          print(5/0)
        except ValueError:
          print("Too bad...")
        except:
          print("Sorry, something went wrong...")
```

Sorry, something went wrong...

UNA POSIBILIDAD

```
In [72]: try:
          print(5/0)
        except Exception as e:
          print(type(e))
          print(str(e))
```

<class 'ZeroDivisionError'>
division by zero

```
In [73]: """try:
          print(5/0)
        except:
          print("Sorry, something went wrong...")
        except (ValueError, ZeroDivisionError):
          print("Too bad...")"""

# SyntaxError: 'break' outside loop
```

```
Out[73]: 'try:\n    print(5/0)\nexcept:\n    print("Sorry, something went wrong...")\nexcept (ValueError, ZeroDivisionError):\n    print("Too bad...")'
```

Ejemplo con while

While: ejemplo 1

```
In [74]: i = 0
while i <= 3:      # -----> 0-1-2-3 (posibilidades)
    print("antes", i)
    i += 2         # i = i + 2 -----> 0-2      (4 ya no valdría)
    print("despues", i)
    print('*')     # ejecuta 2 veces --> **
```

```
antes 0
despues 2
*
antes 2
despues 4
*
```

while: ejemplo 2:

EL MISMO CÓDIGO PERO: print('*') YA NO SE ENCUENTRA INDENTADO.

ESTÁ **FUERA** DEL WHILE (DESPUÉS DEL WHILE)

```
In [75]: i = 0
while i <= 3:      # -----> 0-1-2-3 (posibilidades)
    i += 2         # i = i + 2 -----> 0-2      (4 ya no valdría)
    print('YA HA SUMADO 2')
    print('estamos DENTRO del WHILE')
    print('en el siguiente valor de i:', i)
    print('\n')

    print('\n')
    print('Termina el bucle while')
    print('\n')

    print('AHORA IMPRIMOS UN SOLO ASTERISCO')
    print('*')
```

```
YA HA SUMADO 2
estamos DENTRO del WHILE
en el siguiente valor de i: 2
```

```
YA HA SUMADO 2
estamos DENTRO del WHILE
en el siguiente valor de i: 4
```

```
Termina el bucle while
```

```
AHORA IMPRIMOS UN SOLO ASTERISCO
*
```

while: ejemplo 3

suma 1 en vez de 2

```
In [76]: i = 0
while i <= 3:    # -----> 0-1-2-3 (posibilidades)
    i += 1       # i = i + 1 -----> 1-2-3-4
    print('*')   # ejecuta 2 veces --> * * * *

# i = 0 ==> i = 0 + 1 ==> i = 1

# ...

# 3<=3 ==> si
# 4 <= 3 + 1

# 4<=3 ==> NO

*
*
*
*
```

Ejercicio prototipo: cuál es el resultado de..

repaso: diferencia entre * y esto: **

```
In [77]: 2*3    # multiplicación
```

```
Out[77]: 6
```

```
In [78]: 2**3   # 2 elevado a 3 (2*2*2)
```

```
Out[78]: 8
```

ejercicio de examen:

se puede encontrar en PCEP y en PCAP

```
In [79]: 2**3**2
# izquierda a derecha: 2**3 = 8 --> 8**2 = 64    ==> No correcto
# derecha a izquierda: 3**2 = 9 --> 2**9 = 512    ==> SI es correcto
```

```
Out[79]: 512
```

```
In [80]: # 2**(3**2) ==> 2**9
```

```
In [81]: 2**1  
  
# 2==>4==>8==>16==>32==>64==>128==>256==>512
```

```
Out[81]: 2
```

```
In [82]: 2**9
```

```
Out[82]: 512
```

Ejercicio con funciones y bucle while

funciones y while: ejemplo 1

```
In [83]: """  
def func(text, num):  
    while num > 0:  
        print(text)  
        num = num - 1  
  
func('Hello', 3)  
"""
```

```
# bucle infinito
```

```
Out[83]: "\ndef func(text, num):\n    while num > 0:\n        print(text)\n        num = num - 1\n\nfunc('Hello', 3)\n"
```

funciones y while: ejemplo 2

```
In [84]: # Aqui si sería 3 Hello la solución
```

```
def func(text, num):  
    while num > 0:  
        print(text)  
        num = num - 1  
  
func('Hello', 3)
```

```
Hello  
Hello  
Hello
```

Ejercicio con and, or, not y condicionales

Algunos links con info

<https://realpython.com/lessons/operator-precedence/#:~:text=For%20Boolean%20operations%2C%20all%20and,the%20or%20op>

<https://stackoverflow.com/questions/12494568/boolean-operators-precedence>

<https://www.programiz.com/python-programming/precedence-associativity>

<https://www.scaler.com/topics/operator-precedence-in-python/>

<https://stackoverflow.com/questions/16679272/priority-of-the-logical-operators-order-of-operations-for-not-and-or-in-pyth>

<https://blog.finxter.com/how-does-and-precedence-work-in-a-python-boolean-expression/>



Ejercicios iniciales

```
In [85]: x = True  
x
```

```
Out[85]: True
```

```
In [86]: not x
```

```
Out[86]: False
```

```
In [87]: True and False
```

```
Out[87]: False
```

```
In [88]: True and True
```

```
Out[88]: True
```

```
In [89]: True or True
```

```
Out[89]: True
```

```
In [90]: True or False
```

```
Out[90]: True
```

```
In [91]: False or False
```

```
Out[91]: False
```

```
In [92]: True or False or False
```

```
Out[92]: True
```

```
In [93]: # ejercicios con varios and y or
```

condición1 and condición2 or condición3 and condición4

(condición1 and condición2) or (condición3 and condición4)

resultado12 or resultado34

```
In [94]: True and False or True and True    # F or T ==> T
```

Out[94]: True

```
In [95]: False and False or True and True   # F or T ==> T
```

Out[95]: True

```
In [96]: True and False or False and True   # F or F ==> F
```

Out[96]: False

Ejercicio 1: ejercicio de examen

```
In [97]: x = True
y = False
z = False

if not x or y:    # False or False        =====> FALSE
    print(1)
elif not x or not y and z: # False or (True and False)    =====> FALSE
    print(2)
elif not x or y or not y and x: # False or False or (True and True) =====
    print(3) # esto imprime
else:
    print(4)
```

3

Ejercicio 2

```
In [98]: x = True
y = False
z = False

if not x or y:    # False or False    ==> FALSE
    print(1)
elif x and not y or z: # (True and True) or False    =====> TRUE
    print(2) # esto imprime
elif not x or y or not y and x: # False or False or (True and True) =====
    print(3)
else:
    print(4)
```

2

Ejercicio 3

```
In [99]: x = True
y = False
z = False
```

```

if not x or y:      # False or False
    print(1)
elif x and y or not z: # (True and False) or True
    print(2) # esto imprime
elif not x or y or not y and x: # False or False or (True and True)
    print(3)
else:
    print(4)

```

2

Ejercicio 4

```

In [100...] x = True
            y = False
            z = False

if not x or y:      # False or False      ==> FALSE
    print(1)
elif x and y or z: # (True and False) or False      ==> FALSE
    print(2)
elif x and not y or not y and x: # (True and True) or (True and True)
    print(3) # esto imprime
else:
    print(4)

```

3

Ejercicio 5

```

In [101...] x = True
            y = False
            z = False

if not x or y:      # False or False      ==> FALSE
    print(1)
elif x and y or z: # (True and False) or False      ==> FALSE
    print(2)
elif x and y or y and x: # (True and False) or (False and True)
    print(3)
else:
    print(4) # esto imprime

```

4

Operaciones básicas

repaso inicial de la división

```

In [102...] 3/2 # división con decimales (EL RESULTADO SIEMPRE ES DECIMAL)

```

```

Out[102...] 1.5

```

```

In [103...] 3 // 2 # cociente entero ( EL RESULTADO SIEMPRE ES ENTERO)

```


Out[103...] 1

```
In [104...] 3 % 2  # resto de la división
```

Out[104...] 1

```
In [105...] 10/3
```

Out[105...] 3.3333333333333335

```
In [106...] 10//3
```

Out[106...] 3

```
In [107...] 10%3
```

Out[107...] 1

```
In [108...] 10/2  # ojo, el resultado es decimal, porque SIEMPRE es decimal
```

Out[108...] 5.0

```
In [109...] 4**2  # 4 ELEVADO AL CUADRADO
```

Out[109...] 16

```
In [110...] 4.1 ** 2
```

Out[110...] 16.81

ejercicio de examen resuelto paso a paso

```
In [111...] x = 1 / 2 + 3 // 3 + 4 ** 2
#      0.5  +   1      +  16      # 17.5
print(x)
```

17.5

ejemplo similar y de examen también (prototipo)

```
In [112...] y = 2 / 2 + 3 // 3 + 4 ** 2
print(y)

# en el anterior fragmento de código el resultado es..
# A. 18
# B. 18.0
# C. 17
# D. ninguna de las anteriores
```

18.0

```
In [113...] # el ejercicio paso a paso
y = 2 / 2 + 3 // 3 + 4 ** 2
y
```

```
# y = (2 / 2) + (3 // 3) + (4 ** 2)
# y = 1.0 + 1 + 16
# y = 18.0

# CUANDO 1 DE ELLOS ES DECIMAL, EN UNA OPERACIÓN (+ EN ESTE CASO)
# CONVIERTE EN DECIMAL A TODO EL RESULTADO
```

Out[113... 18.0

OTROS EJEMPLOS..

In [114... 1/2, 3/2, 3//3, 4**2, 1//2

Out[114... (0.5, 1.5, 1, 16, 0)

In [115... print('división:', 5/3, ', cociente: ', 5//3, ', resto:', 5%3)

división: 1.6666666666666667 , cociente: 1 , resto: 2

In [116... # paso a paso

```
x = 1 / 2 + 3 // 3 + 4 ** 2
x
# x = (1 / 2) + (3 // 3) + (4 ** 2)
# x = 0.5 + 1 + 16 = 17.5
```

Out[116... 17.5

Asignación múltiple de variables

In [117... ## ejemplo inicial

In [118... x = 10
y = 20
print(x)
print(y)

10
20

In [119... x,y = 15,25
print(x)
print(y)

15
25

In [120... x, y, z, t, w = 0, 15, 25, -4, 100
print(x)
print(y)
print(z)
print(t)
print(w)

0
15
25
-4
100

ejercicio de examen

```
In [121... t,u = 10,-20  
t,u
```

```
Out[121... (10, -20)
```

```
In [122... # cuál es el resultado?  
# A) 1 1 1  
# B) 1 1 2  
# C) Ninguna de las anteriores  
# D) El código es erróneo  
  
x = 1  
y = 2  
  
x, y, z = x, x, y  
# x,y,z = 1, 1, 2  
# esto lo que realmente quiere decir es que..  
# x = 1 =====> x = 1  
# y = 1  
# z = 2  
# es decir se lo hemos asignado en una sola línea  
  
z, y, z = x, y, z  
# z,y,z = 1, 1, 2  
# de igual manera que justo arriba, esto significa que..  
# z = 1  
# y = 1 =====> y = 1  
# z = 2 =====> z = 2  
  
print(x, y, z)      # 1 1 2
```

```
1 1 2
```

argumentos en diferentes posiciones

```
In [123... def fun(x, y, z):  
    # x = 0, y = 3, z = 1  
    return x + 2 * y + 3 * z  
    # 0 + (2 * 3) + (3 * 1)  
    # 0 + 6 + 3  
    # 9  
  
print(fun(0, z=1, y=3))    # ==> 9  
  
# duda? es 9 o no funciona ==> La conclusión es que sí que funciona  
  
# CONCEPTO:
```

```
# PODEMOS (AUNQUE NO DEBEMOS) modificar el orden de las variables en la l  
# con respecto a la definición de la propia función  
# (para ello, habría que indicar qué variable tiene qué valor)
```

9

```
In [124... def fun(x, y, z):  
    print('DENTRO DE LA FUNCIÓN: x, y, z, son: ', x,y,z) # 0,3,1  
    # aunque esté en diferentes posiciones lo coge igualmente  
    return x + 2 * y + 3 * z # 0 + 2*3 + 3*1 = 0+6+3 = 9  
  
print(fun(0, z=1, y=3))
```

DENTRO DE LA FUNCIÓN: x, y, z, son: 0 3 1

9

print y los separadores

```
In [125... z = y = x = 1  
print(x, y, z)
```

1 1 1

ejemplo 1

```
In [126... # ¿ cuál es el resultado del siguiente fragmento de código?  
# A) 1 1 1  
# B) 111  
# C) ninguna de las anteriores  
# D) el código es erróneo  
  
z = y = x = 1  
print(x, y, z, sep='')
```

111

ejemplo 2

```
In [127... z = y = x = 1  
print(x, y, z, sep=' ')
```

1 1 1

ejemplo 3

```
In [128... # cuál es la salida aqui?  
# A) 1*1*1  
# B) *1*1*1*  
# C) ninguna de las anteriores  
# D) el código es erróneo  
  
z = y = x = 1  
print('x:', x)  
print('y:', y)  
print('z:', z)
```

```
print("\n")
print(x, y, z, sep='*')
```

```
x: 1
y: 1
z: 1
```

```
1*1*1
```

pop en una lista

```
In [129... nums = [3, 4, 5, 20, 5, 25, 1, 3]
nums.pop(2) # borra el elemento en una posición co
print(nums)
```

```
# [3, 4, 5, 20, 5, 25, 1, 3]
# [3, 4, , 20, 5, 25, 1, 3]
```

```
[3, 4, 20, 5, 25, 1, 3]
```

División en Python

```
In [130... x = 5
y = 3

print('división:', x / y) # 5/3 = 1.66
print('cociente:', x // y) # 3 * (1) = 3
print('resto:', x % y) # 3 + (2) = 5
```

```
división: 1.6666666666666667
cociente: 1
resto: 2
```

```
In [131... type(x / y), type(x // y), type(x % y)
```

```
Out[131... (float, int, int)
```

```
In [132... type(6 / 2), 6 / 2
```

```
Out[132... (float, 3.0)
```

Ejemplo con diccionarios

diccionarios: repaso previo

```
In [133... L = []  
len(L)
```

Out[133... 0

```
In [134... dct = {}  
print('dct:', dct)  
print('len(dct):', len(dct))
```

dct: {}
len(dct): 0

```
In [135... # {'key1': value1, 'key2': value2}  
  
diccionario1 = {'clave1': [10,20,30], 'clave2': [15,25,35]}  
diccionario1
```

Out[135... {'clave1': [10, 20, 30], 'clave2': [15, 25, 35]}

```
In [136... import pandas as pd  
df = pd.DataFrame({'clave1': (10,20,30), 'clave2': (15,25,35)})  
df
```

Out[136...

	clave1	clave2
0	10	15
1	20	25
2	30	35

```
In [137... df['clave1']
```

Out[137...

0	10
1	20
2	30

Name: clave1, dtype: int64

```
In [138... df['clave3'] = [12,22,32]  
df
```

Out[138...

	clave1	clave2	clave3
0	10	15	12
1	20	25	22
2	30	35	32

```
In [139... # vuelvo a imprimir para verlo  
diccionario1
```

Out[139... {'clave1': [10, 20, 30], 'clave2': [15, 25, 35]}

```
In [140... diccionario1['clave1']
```

Out[140... [10, 20, 30]

```
In [141... diccionario1['clave2']
```

```
Out[141...] [15, 25, 35]
```

```
In [142...] diccionario1['clave1'][0]
```

```
Out[142...] 10
```

```
In [143...] diccionario1['clave1'][0] = 1000  
diccionario1
```

```
Out[143...] {'clave1': [1000, 20, 30], 'clave2': [15, 25, 35]}
```

```
In [144...] # otro pequeño ejercicio con explicaciones, previo a los ejemplos
```

```
In [145...] dct = {}  
dct['key1'] = (1, 2)  
dct['key2'] = (2, 1)  
dct
```

```
Out[145...] {'key1': (1, 2), 'key2': (2, 1)}
```

diccionarios: ejemplo 1

```
In [146...] dct = {}  
print('el diccionario venía vacío')  
dct['clave1'] = (1, 2)  
dct['clave2'] = (2, 1)  
  
print('el diccionario ahora tiene elementos, y son:', dct)  
# {'1': (1,2),  
#   '2': (2,1)}  
  
print('ahora imprimo: primero valor 1 y valor 2')  
print('y dentro de cada uno SEÑALO index 0 de esa tupla e index 1')  
print('dct y clave1 nos dan el valor 1:', dct['clave1'])  
print('index 0: ', dct['clave1'][0])  
print('index 1: ', dct['clave1'][1])  
  
print('dct y clave 2 nos dan el valor 2:', dct['clave2'])  
print('index 0: ', dct['clave2'][0])  
print('index 1: ', dct['clave2'][1])  
  
print(dct.keys())
```

el diccionario venía vacío

el diccionario ahora tiene elementos, y son: {'clave1': (1, 2), 'clave2': (2, 1)}

ahora imprimo: primero valor 1 y valor 2

y dentro de cada uno SEÑALO index 0 de esa tupla e index 1

dct y clave1 nos dan el valor 1: (1, 2)

index 0: 1

index 1: 2

dct y clave 2 nos dan el valor 2: (2, 1)

index 0: 2

index 1: 1

dict_keys(['clave1', 'clave2'])

diccionarios: ejemplo 2 (el ejercicio del examen)

```
In [147... dct = {}
dct['1'] = (1, 2)
dct['2'] = (2, 1)
print(dct)

# {'1': (1,2),
#  '2': (2,1)}

print(dct.keys())

for x in dct.keys():
    print(dct[x][1], end='')
    # dct["1"][1] --> 2
    # dct['2'][1] --> 1
    # 21

{'1': (1, 2), '2': (2, 1)}
dict_keys(['1', '2'])
21
```

diccionarios: ejemplo 3

```
In [148... # otro ejemplo, donde podemos ver que primero imprime el 2, después el 1

dct = {}
dct['1'] = (1, 2)
dct['2'] = (2, 1)

# {'1': (1,2),
#  '2': (2,1)}

for x in dct.keys():
    print(dct[x][1])

2
1
```

string a lista

```
In [149... print(list('hello'))

['h', 'e', 'l', 'l', 'o']
```

Ejemplo con funciones

```
In [150... def func(p1, p2):
    print('p1 y p2 iniciales:', p1, p2)
    p1 = 1
```



```

    p2[0] = 42
    print('p1 y p2 finales:', p1, p2)

# -----
x = 3
y = [1, 2, 3]

func(x, y)      # func(3, [1,2,3])
# -----

print("\n")

print('x e y después de la función:', x, y)

print('ahora x e y[0]:', x, y[0])

# ----
# las listas se pueden modificar.

```

p1 y p2 iniciales: 3 [1, 2, 3]
 p1 y p2 finales: 1 [42, 2, 3]

x e y después de la función: 3 [42, 2, 3]
 ahora x e y[0]: 3 42

float de un string

In [151]: `print(float('1.3'))`

1.3

try except algunos ejemplos

excepciones: ejemplo 1

```

In [1]: try:
    first_prompt = input("Enter the first value: ")      # kangaroo
    a = len(first_prompt)                                # len(kangaroo) =
    second_prompt = input("Enter the second value: ")    # 0
    b = len(second_prompt) * 2                           # 2*len(0) = 2*1
    print('a:', a)
    print('b:', b)
    print(a/b)                                            # 8/2 = 4
except ZeroDivisionError:
    print("Do not divide by zero!")
except ValueError:
    print("Wrong value.")
except:
    print("Error.Error.Error.")

```

```
a: 8
b: 2
4.0
```

excepciones: ejemplo 2

```
In [7]: try:
        first_prompt = input("Enter the first value: ")    # kangaroo
        a = first_prompt
        second_prompt = input("Enter the second value: ")  # 0
        b = second_prompt
        print(a/b)
    except ZeroDivisionError:
        print("Do not divide by zero!")
    except ValueError:
        print("Wrong value.")
    except:
        print("Error.Error.Error.")
```

Error.Error.Error.

excepciones: ejemplo 3

```
In [6]: try:
        first_prompt = input("Enter the first value: ")    # 10
        a = first_prompt
        second_prompt = input("Enter the second value: ")  # 0
        b = second_prompt
        print('a:', a)
        print('b:', b)
        print(a/b)
    except ZeroDivisionError:
        print("Do not divide by zero!")
    except ValueError:
        print("Wrong value.")
    except:
        print("Error.Error.Error.")
```

a: 10

b: 0

Error.Error.Error.

Suma de strings vs Suma de números

```
In [8]: # enteros

x = int(input()) # 2
y = int(input()) # 4
print(x + y)
```

6

```
In [11]: # DECIMALES
```

```
x = float(input()) # 2
y = float(input()) # 4
print(x)
print(y)
print(x + y)
```

2.0
4.0
6.0

In [12]: *# strings*

EL EJERCICIO DE EXAMEN

```
x = input() # 2
y = input() # 4
print(x)
print(type(x))
print(y)
print(type(y))
print('\n')
print(x + y)
```

'2' + '4' ==> '24'

2
<class 'str'>
4
<class 'str'>

24

In []: *# recordamos suma de strings con "hola" "mundo"*

In [13]: *# strings*

EL EJERCICIO DE EXAMEN

```
x = input() # hola
y = input() # mundo
print(x)
print(type(x))
print(y)
print(type(y))
print('\n')
print(x + y)
```

hola
<class 'str'>
mundo
<class 'str'>

holamundo

Desplazar los Bits

Desplazar un valor un bit a la **izquierda** corresponde a multiplicarlo por dos, respectivamente desplazar un bit a la **derecha** es como dividir entre dos.

Los operadores de cambio en Python son un par de dígrafos '<<' y '>>', sugiriendo claramente en que dirección actuará el cambio.

value << bits value >> bits

Ejemplo:

17 >> 1 --> 17//2 (17 dividido entre 2 a la potencia de 1) --> es 8 (desplazarse hacia la derecha en un bit equivale a la división entera de dos)

17 << 2 --> 17 * 4 (17 dividido entre 2 a la potencia de 2) --> es 68 (desplazarse hacia la izquierda en dos bits equivale a la multiplicación entera por cuatro)

```
In [14]: 17 >> 1
```

```
Out[14]: 8
```

```
In [15]: 17 << 2
```

```
Out[15]: 68
```

Bitwise right shift

```
In [ ]: # The bitwise right shift operator (>>) is analogous to the left one,
# but instead of moving bits to the left,
# it pushes them to the right by the specified number of places.
```

```
In [16]: a = 10
# 0000 1010 (binario)
a >> 1
# 0000 0101 (binario) ==> 5 decimal
```

```
Out[16]: 5
```

Bitwise left shift

```
In [17]: a = 5
print('a:', a)
# 0000 0101 (binario)
b = a << 1 # a * 2^n --> 5 * 2^1=10
print('b:', b)
# 0000 1010 (binario) ==> 10
c = a << 2 # 5 * 2^2 = 20
print('c:', c)
# 0001 0100 (binario) ==> 20
```

```
d = a << 3 # 5 * 23 = 40
print('c:', d)
```

a: 5
b: 10
c: 20
c: 40

string y count

```
In [ ]: # string.count(value, start, end)
```

```
In [18]: # https://www.w3schools.com/python/ref\_string\_count.asp

txt = "I love apples, apple are my favorite fruit"

x = txt.count("apple")

print(x)
```

2

```
In [19]: string = "Python is awesome, isn't it?"
substring = "i"

len(string)
```

Out[19]: 28

```
In [ ]: # string = "Python is awesome, isn't it?"
#           6 + 2       7       5       2       + los espacios + la coma + el ?
```

```
In [20]: # count after first 'i' and before the last 'i'
count = string.count(substring, 0, 5)

# print count
print("The count is:", count)
```

The count is: 0

```
In [21]: # count after first 'i' and before the last 'i'
count = string.count(substring, 0, 8)

# print count
print("The count is:", count)
```

The count is: 1

```
In [22]: # count after first 'i' and before the last 'i'
count = string.count(substring, 0, 7)

# print count
print("The count is:", count)
```

The count is: 0

```
In [23]: # count after first 'i' and before the last 'i'
count = string.count(substring, 8, 25)

# print count
print("The count is:", count)
```

The count is: 1

```
In [24]: # count after first 'i' and before the last 'i'
count = string.count(substring, 8, 28)

# print count
print("The count is:", count)
```

The count is: 2

```
In [25]: # count after first 'i' and before the last 'i'
count = string.count(substring, 0, 10)

# print count
print("The count is:", count)
```

The count is: 1

```
In [ ]: # otro ejemplo
```

```
In [26]: data = 'abbabadaadbbaccabc'
print(data.count('ab', 1))
```

2

```
In [27]: data = 'abbabadaadbbaccabc'
print(data.count('ab', 1, len(data)))
```

2

```
In [30]: data = 'abbabadaadbbaccabc'
print(data.count('ab', 0, 1))
```

0

Un ejemplo con Try Except

```
In [31]: try:
    value = input("Enter a value: ")
    print(type(value))
    print(value/value)
except ValueError:
    print("Bad input...")
except ZeroDivisionError:
    print("Very bad input...")
except TypeError:
    print("Very very bad input...")
except:
    print("Booo!")

# TypeError..
```

```
<class 'str'>  
Very very bad input...
```

Función con diccionario

funciones y diccionarios: ejemplo 1

```
In [32]: data = {}  
  
def func(d, key, value):  
    d[key] = value  
  
print(func(data, '1', 'Peter'))  
  
# la función no tiene retorno
```

None

funciones y diccionarios: ejemplo 2

```
In [33]: data = {}  
  
def func(d, key, value):  
    d[key] = value  
    return d  
  
print(func(data, '1', 'Peter'))  
  
{'1': 'Peter'}
```

len("\\")

```
In [40]: x = '\\'   
print(len(x))  
  
# SyntaxError: EOL while scanning string literal
```

```
Cell In[40], line 1  
    x = '\\'   
        ^  
SyntaxError: EOL while scanning string literal
```

```
In [35]: x = '\\\\'  
print(len(x))
```

1

```
In [41]: x = '\\\\\\'  
print(len(x))
```

2

```
In [42]: x = '\\\\'
print(len(x))

# SyntaxError: EOL while scanning string literal
```

```
Cell In[42], line 1
    x = '\\\\'
      ^
SyntaxError: EOL while scanning string literal
```

len(salto de linea)

```
In [ ]: # comentario
```

```
In [ ]: # comentario de linea 1
        # comentario de linea 2
```

```
In [ ]: """
comentario de linea 1
comentario de linea 2
comentario de linea 3
"""
```

```
In [43]: x = ""
print(len(x))
```

0

```
In [44]: x = ""
        ""
print(len(x))
```

1

```
In [45]: x = ""
        ""
print(len(x))
```

2

```
In [46]: # hay un espacio en blanco en la línea donde comienza x, antes del salto
x = ""
        ""
print(len(x))
```

3

```
In [47]: x = ""Hola
        ""
print(len(x))
```

5

```
In [48]: x = ""Hola mundo
        ""
```



```
print(len(x))

# 4 por "hola"
# 1 por el espacio en blanco
# 5 por "mundo"
# 1 por el salto de línea
```

11

Ejercicio prototipo de Funciones

ejemplo 1 de funciones

```
In [49]: def func(x, y):
        if x == y:
            return x
        else:
            return func(x, y-1)

print(func(0, 3))

# func(0, 3) => return func(x, y-1) ==> return func(0, 2)
# func(0, 2) => return func(0, 1)
# func(0, 1) => return func(0, 0)
# func(0, 0) => return x => 0
```

0

ejemplo 2 de funciones: ejemplo 1 modificado

```
In [50]: def func(x, y):
        if x == y:
            return 'Hemos llegado a x=y=0'
        else:
            return func(x, y-1)

print(func(0, 3))

# func(0, 3) => return func(x, y-1)
# func(0, 2) => return func(0, 1)
# func(0, 1) => return func(0, 0)
# func(0, 0) => return 'Hemos llegado a x=y=0'
```

Hemos llegado a x=y=0

ejemplo 3 de funciones: otra ligera modificación del ejercicio 1 (paso a paso)

```
In [51]: def func(x, y):
        if x == y:
```

```

        return x
    else:
        print('x es:', x, '.. y es:', y, '..por lo de (y-1)')
        return func(x, y-1)

print(func(0, 3))

# func(0, 3) => return func(x, y-1)
# func(0, 2) => return func(0, 1)
# func(0, 1) => return func(0, 0)
# func(0, 0) => return 'Hemos llegado a x=y=0'

```

```

x es: 0 .. y es: 3 ..por lo de (y-1)
x es: 0 .. y es: 2 ..por lo de (y-1)
x es: 0 .. y es: 1 ..por lo de (y-1)
0

```

ejemplo 4 de funciones

otro ejemplo en el cual, no llega a ningún sitio

```

In [52]: def func(x, y):
        if x == y:
            return x
        else:
            return func(x, y-1)

print(func(0, -3))

# RecursionError: maximum recursion depth exceeded in comparison

```

```

-----
-
RecursionError                                Traceback (most recent call las
t)
Cell In[52], line 8
      4     else:
      5         return func(x, y-1)
----> 8 print(func(0, -3))
      10 # RecursionError: maximum recursion depth exceeded in comparison

Cell In[52], line 5, in func(x, y)
      3     return x
      4 else:
----> 5     return func(x, y-1)

Cell In[52], line 5, in func(x, y)
      3     return x
      4 else:
----> 5     return func(x, y-1)

[... skipping similar frames: func at line 5 (2969 times)]

Cell In[52], line 5, in func(x, y)
      3     return x
      4 else:
----> 5     return func(x, y-1)

Cell In[52], line 2, in func(x, y)
      1 def func(x, y):
----> 2     if x == y:
      3         return x
      4     else:

RecursionError: maximum recursion depth exceeded in comparison

```

pycache


```
In [ ]: # https://towardsdatascience.com/pycache-python-991424aabad8
```

LIFO / FIFO

```
In [ ]: # Last In First Out (LIFO)
        # First In First Out (FIFO)
```

ASCII / UNICODE

```
In [ ]: # https://elcodigoascii.com.ar/  
# https://dinahosting.com/blog/que-es-utf-8/
```

 No description has been provided for this image

```
In [53]: ord('0')
```

```
Out[53]: 48
```

```
In [54]: chr(48)
```

```
Out[54]: '0'
```

```
In [55]: type(ord('0')) # 48
```

```
Out[55]: int
```

```
In [56]: type(chr(48)) # '0'
```

```
Out[56]: str
```

```
In [57]: ord('9'), chr(57)
```

```
Out[57]: (57, '9')
```

```
In [58]: # tiene sentido la siguiente línea de código?  
  
# chr(B) ==> no tiene sentido, le faltan las comillas simples  
# chr('B') ==> no tiene sentido porque queremos el número  
  
ord('B')
```

```
Out[58]: 66
```

```
In [59]: # dime el código necesario para imprimir la 'f'  
  
# chr('102') ==> no tiene sentido, porque 102 es número, no string  
  
chr(102)
```

```
Out[59]: 'f'
```

```
In [ ]: # escriba las líneas de código necesarias para conseguir  
# los números de 'a' y de 'A'
```

```
In [60]: # 'a'  
  
ord('a')
```

```
Out[60]: 97
```

```
In [61]: # 'A'  
  
ord('A')
```

Out[61]: 65

```
In [62]: ord('a') - ord('A')
# 97    -    65
#      32
```

Out[62]: 32

```
In [63]: ord('c') - ord('a')
# 99    -    97
#      2
```

Out[63]: 2

```
In [64]: chr(32)
```

Out[64]: ' '

```
In [65]: chr(39)
```

Out[65]: "'"

```
In [66]: ord("'")
```

Out[66]: 39

```
In [ ]: # ejemplo
```

```
In [67]: ord('b')
```

Out[67]: 98

```
In [68]: print(chr(ord('p') + 3)) # s

# en la 'p' el ord es el 112
#      chr( 112 + 3)
#      chr( 115)
# en el 115 se encuentra la 's'
#      s
```

s

```
In [69]: chr(ord('p') + 3)
```

Out[69]: 's'

```
In [ ]: # OTRO EJEMPLO (EJERCICIO PROTOTIPO)
```

```
In [70]: 'mike' > 'Mike'
```

Out[70]: True

```
In [ ]: # The expression:

# 'mike' > 'Mike'

# is
```

```
# A. erroneous  
# B. False  
# C. True
```

```
In [71]: ord('m'), ord('M'), ord('m') > ord('M')
```

```
Out[71]: (109, 77, True)
```

```
In [72]: 'mike' > 'Mike'
```

```
Out[72]: True
```

```
In [ ]: # Solución  
# C
```

operadores booleanos

```
In [ ]: ## not
```

```
In [73]: x = True  
x
```

```
Out[73]: True
```

```
In [74]: not x
```

```
Out[74]: False
```

```
In [ ]: # qué pasa si sumo True con números?
```

```
In [75]: True + 2 # True es 1
```

```
Out[75]: 3
```

```
In [76]: False + 2 # False es 0
```

```
Out[76]: 2
```

```
In [77]: True + 3.5
```

```
Out[77]: 4.5
```

```
In [78]: False + 3.5
```

```
Out[78]: 3.5
```

and

```
In [79]: print('and:', 0&0, 0&1, 1&0, 1&1)
```

and: 0 0 0 1

or

```
In [80]: print('or:', 0|0, 0|1, 1|0, 1|1)
```

or: 0 1 1 1

XOR (PUERTA LÓGICA)

```
In [81]: print('XOR:', 0^0, 0^1, 1^0, 1^1)
```

```
# 2^3 (NO ES 2 AL CUBO)
# 2**3
```

XOR: 0 1 1 0

```
In [82]: 2^3 # NO ES 2 AL CUBO!!!
```

Out[82]: 1

ejercicio de examen

```
In [83]: x = 0
y = 1
x = x ^ y # x = 0 ^ 1 => x = 1
y = x ^ y # y = 1 ^ 1 => y = 0
y = x ^ y # y = 1 ^ 0 => y = 1
print(x, y) # x = 1 y = 1
```

1 1

and, or, xor

```
In [84]: a = 1 # => a = 1
b = 0 # => b = 0
c = a & b # 1 AND 0 => c = 0 # & # and
d = a | b # 1 OR 0 => d = 1
e = a ^ b # 1 XOR 0 => e = 1

print(c + d + e) # 0 + 1 + 1 = 2
```

2

```
In [ ]: # explicación de algunas cosas, repaso
```

```
In [85]: # and, & => y
# or, | = ó
a = True
b = not a
b
```

Out[85]: False

True en las sumas

```
In [86]: w = 7
x = 3
y = 4
z = True          # 1 en las sumas
a = w + x * y      # 7 + 3 * 4 => a = 19
b = w + x / z      # 7 + 3 / 1 => b = 10.0

# b = 7 + 3 / True ==> b = 7 + (3/1) ==> b = 10

print('a:', a)
print('b:', b)

if a > b:
    print('TRUE')
else:
    print('FALSE')
```

a: 19
b: 10.0
TRUE

STRINGS

```
In [87]: # EJEMPLO 1

'a'+'b'
```

Out[87]: 'ab'

```
In [88]: # EJEMPLO 2

'1'+'4'
```

Out[88]: '14'

```
In [89]: # EJEMPLO 3

str(1) + str(4)    # str(14)
```

Out[89]: '14'

```
In [ ]: ## Suma de String + un número: EJEMPLO 4

# '1' + 4

# TypeError: can only concatenate str (not "int") to str
```



```
In [90]: ## Multiplicación de Strings: EJEMPLO 5
```

```
4 * 'a'
```

```
Out[90]: 'aaaa'
```

VARIABLE GLOBAL y variable local

```
In [1]: v = 1
```

```
def fun():  
    global v  
    v = 2  
    return v  
  
print(v)
```

1

```
In [2]: # explicación 1
```

```
v = 1  
  
def fun():  
    global v  
    v = 2  
    return v  
  
print('antes de llamar a la función v vale:', v, '(print del test)')  
print("\n")  
print('aquí llamo a la función:', fun(), '(antes solo definida)')  
print('v después de llamar a la función vale:', v, '(v es global)')
```

antes de llamar a la función v vale: 1 (print del test)

aquí llamo a la función: 2 (antes solo definida)

v después de llamar a la función vale: 2 (v es global)

```
In [3]: # explicación 2
```

```
v = 1  
  
def fun():  
    v = 2  
    return v  
  
print('antes de llamar a la función v vale:', v, '(print del test)')  
print("\n")  
print('aquí llamo a la función:', fun(), '(antes solo definida)')  
print('v después de llamar a la función vale:', v, '(v es local)')
```

antes de llamar a la función v vale: 1 (print del test)

aquí llamo a la función: 2 (antes solo definida)

v después de llamar a la función vale: 1 (v es local)

LENGUAJES INTERPRETADOS

```
In [ ]: # https://www.freecodecamp.org/espanol/news/lenguajes-compilados-vs-inter  
# https://aulab.es/noticia/18/diferencia-entre-lenguajes-de-programacion-
```

ejemplo if elif elif .. else

```
In [ ]: # conversor de notas  
  
# 1: 90 through 100 -> A  
# 2: 80 through 89 -> B  
# 3: 70 through 79 -> C  
# 4: 65 through 69 -> D  
# 5: 0 through 64 -> F
```

forma 1

```
In [4]: # Letter Grade Converter  
grade = int(input('Enter a numeric grade:'))  
if grade >= 90:  
    letter_grade = 'A'  
elif grade >= 80:  
    letter_grade = 'B'  
elif grade >= 70:  
    letter_grade = 'C'  
elif grade >= 65:  
    letter_grade = 'D'  
else:  
    letter_grade = 'F'  
print('Your letter grade is:', letter_grade)
```

Your letter grade is: C

forma 2: en pocas líneas

```
In [5]: # Letter Grade Converter  
grade = int(input('Enter a numeric grade:'))  
if grade >= 90: letter_grade = 'A'  
elif grade >= 80: letter_grade = 'B'  
elif grade >= 70: letter_grade = 'C'  
elif grade >= 65: letter_grade = 'D'  
else: letter_grade = 'F'  
print('Your letter grade is:', letter_grade)
```

Your letter grade is: C

ejemplo con continue

In []: *# Which one of the lines should you put in the snippet below to match the*

Expected output:

1245

Code:

```
# c = 0
# while c < 5:
#     c = c + 1
#     if c == 3:
#         # enter code here
#     print(c, end="")
```

A. exit

B. continue

C. print()

D. break

In [6]:

```
c = 0
while c < 5:
    c = c + 1
    if c == 3:
        continue
    print(c, end="")
```

1245

Bucles en una linea

In [8]:

```
x = 15

if x > 10:
    print(True)
else:
    print(False)
```

True

In [7]:

```
x = 15
mayor_10 = True if x > 10 else False

print(mayor_10)
```

True

Caso del if elif

solo ejecuta la primera sentencia correcta que encuentra

```
In [9]: def funcion_if(x):  
        if x > 9: print('numero:', x, ', evaluamos: x > 9')  
        elif x > 7: print('numero:', x, ', evaluamos: x > 7')  
        elif x > 5: print('numero:', x, ', evaluamos: x > 5')  
        else: print('numero:', x, ', evaluamos: else')
```

```
In [10]: funcion_if(10)  
numero: 10 , evaluamos: x > 9
```

```
In [11]: funcion_if(8)  
numero: 8 , evaluamos: x > 7
```

```
In [12]: funcion_if(6)  
numero: 6 , evaluamos: x > 5
```

```
In [13]: funcion_if(4)  
numero: 4 , evaluamos: else
```

While...Else, For...Else

Podemos encontrarnos que después de un bucle **for** aparece asociado un **else**, también ocurre en el caso de **while**, en ambos casos lo que haya en el else se ejecuta siempre al final **for** o **while**. Veamos algunos ejemplos:

Ejemplo 1: for...else

```
In [15]: # ¿ Cuál será el valor de x?  
  
x = 0  
  
for j in range(2): # posibilidades de 0, 1  
    for i in range(2): # posibilidades de 0, 1  
        if i == j:  
            x += 1  
        else:  
            x += 1  
  
# Primera vuelta se compara el 0 de j con las opciones de i que son  
# j=0, i=0 --> i == j -- x = 0 + 1 --> x = 1  
# j=0, i=1 --> i != j --> x = 1  
# Acaba la primera vuelta y entra en el else: x = 1 + 1 --> x = 2
```

```

# Segunda vuelta compara el 1 de j con las opciones de i que son 0, 1
# j=1, i=0 --> i != j --> x = 2
# j=1, i=1 --> i == j --> x = 2 + 1 --> x = 3
# Acaba la Segunda vuelta y entra en el else: x = 3 + 1 --> x = 4

# resultado: 4
print(x)

# A. 2
# B. 1
# C. 4
# D. 5

# Solución: C

```

4

Ejemplo 2: for...else

In [22]: # ¿Cuál será el valor de x en el siguiente código?

```

x = 0

for i in range(4): # 0, 1, 2, 3
    if 2 * i < 4:
        x += 1
        # i = 0 --> 2 * 0 < 4 --> x = 0 + 1 --> x = 1
        # i = 1 --> 2 * 1 < 4 --> x = 1 + 1 --> x = 2
        # i = 2 --> 2 * 2 < 4 --> No se cumple se para!!
    else:
        x += 1
        # x = 2 --> x = 2 + 1 --> x = 3

# Resultado: 3
print(x)

# A. 2
# B. 3
# C. Error en el código
# D. 5

# Solución: B

```

3

Ejemplo 3: While...else

In []: # ¿Cuántos '#' mostrará el siguiente código?

```

x = 0
while x < 30: # 0 < 30
    x *= 2 # x = 0 * 2 = 0 --> Siempre será 0!!!!
    if x > 10:
        continue
    print("#") # Imprime #
else:
    print("#")

```

```
# A. 2
# B. Bucle infinito
# C. 4
# D. Error de código

# Solución: B
```

Ejemplo 4: while...else

```
In [1]: # ¿Cuántos '#' mostrará el siguiente código?

x = 0

while x != 0: # 0 != 0 --> no se cumple NO ENTRA!!!!
    x -= 1
    print("#", end='')
else:
    print("#") # Sólo imprime 1!!!

# A. 1
# B. Bucle infinito
# C. 4
# D. Error de código

# Solución: A
```

#

Ejemplo 5: while...con break ...else

```
In [10]: i = 4

while i > 0:
    i -= 2      # i = i - 2
    print('*') # *
    if i == 2: # cumple la condición
        break  # se sale
else:
    print('*')
```

*

En este caso el bucle while se para y el else no se ejecutaría!!!

Colocar el código en orden correcto

Ejemplo 1

```
In [ ]: # Coloca el código correcto:
        # opciones: when, try, except

# # -----:
```

```
# print(x)
# # -----:
# print('Error de código')
```

```
In [ ]: # Solución
# try
# except
```

Ejemplo 2

```
In [ ]: # Pon el código en orden correcto para recoger un decimal de la variable
# Nota: No debes usar todas las opciones

# opciones: float, int, (, input("Inserta el valor de masa"), print(), ), =
# mass = -----
```

```
In [ ]: # Solución
# float(input("Insertar el valor de masa"))
```

Ejemplo 3

```
In [ ]: # Pon el código en orden correcto para recoger un string de la variable t
# Nota: No debes usar todas las opciones

# opciones: test, input, (, "Inserta el valor de test", print(), ), =
# -----
```

```
In [ ]: # Solución
# test = input("Inserta el valor de test")
```

Ejemplo 4

```
In [11]: # Como obtenemos un único * si y = 7*2 - 7:

# Opciones: >, >=, <, ==

# if y ---- 40:
#     print("")
# elif y ---- 40:
#     print("***")
# else:
#     print("****")
```

```
In [12]: 42 >= 40
```

```
Out[12]: True
```

```
In [13]: 40 >= 40
```

```
Cell In[13], line 1
    40 => 40
      ^
SyntaxError: invalid syntax
```

```
In [14]: y = 7**2 - 7

if y > 40:
    print("*")
elif y < 40:
    print("**")
else:
    print("***")
```

*

```
In [15]: # solución
# >
# <
```

Gracias por la atención

Isabel Maniega