0_Programacion Python

July 6, 2025

Contenido creado por:

Isabel Maniega

[1]:
```
import warnings
warnings.filterwarnings('ignore')
```

---

1 Tipos de almacenamiento de datos en Python

- Listas–> []
- tuplas –> ()
- diccionarios –> {clave : valor}
- conjuntos (set) –> {,}
- dataframe –> tabla
- matrices –> [[],[]]

Formatos: - entero –> int - decimales –> float - objetos –> obj - cadena de texto –> str - boleanos –> True/ False

1.1 Listas

[2]:
```
lista = [1, 5, 6, 8, None, 6, None]

lista
```

[2]:
```
[1, 5, 6, 8, None, 6, None]
```

[3]:
```
lista[0], lista[1], lista[2]
```

[3]:
```
(1, 5, 6)
```

[4]:
```
lista[-1], lista[-2]
```

[4]:
```
(None, 6)
```

[5]:
```
lista[-3] = 3
lista
```

[5]:
```
[1, 5, 6, 8, 3, 6, None]
```

[6]:
```
lista.append(10)
lista
```

[6]:
```
[1, 5, 6, 8, 3, 6, None, 10]
```

1.2 Tuplas

[7]:
```
valor = (1, 5, 10, 60,)
valor
```

[7]:
```
(1, 5, 10, 60)
```

[8]:
```
type(valor)
```

[8]:
```
tuple
```

[9]:
```
valor[1]
```

[9]:
```
5
```

[10]:
```
valor[1] = 10
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[10], line 1
----> 1 valor[1] = 10

TypeError: 'tuple' object does not support item assignment
```

[11]:
```
valor.append(10)
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
Cell In[11], line 1
----> 1 valor.append(10)

AttributeError: 'tuple' object has no attribute 'append'
```

[12]:
```
valor2 = (5)
type(valor2)
```

[12]:
```
int
```

[13]:
```
valor2 = (5, )
type(valor2)
```

[13]:
```
tuple
```

[14]:
```
valor_final = valor + valor2
valor_final
```

[14]:
```
(1, 5, 10, 60, 5)
```

1.3 Matriz

[15]:
```
import numpy as np

matriz = np.array([1, 2, 50, 60])
matriz
```

[15]:
```
array([ 1,  2, 50, 60])
```

[16]:
```
matriz[2]
```

[16]:
```
np.int64(50)
```

[17]:
```
matriz = np.append(matriz, 20)
matriz
```

[17]:
```
array([ 1,  2, 50, 60, 20])
```

[18]:
```
matriz[2] = 100
matriz
```

[18]:
```
array([  1,   2, 100,  60,  20])
```

[19]:
```
matriz = np.array([[1, 2], [3, 4]])
matriz
```

[19]:
```
array([[1, 2],
       [3, 4]])
```

[20]:
```
matriz[0][1]
```

[20]:
```
np.int64(2)
```

1.4 Diccionarios

[21]:
```
# diccionario: clave-valor
# {'clave': 'valor'}
# {'clave1': 'valor1', 'clave2': 'valor2'}
```

```
diccionario = {'1': 25, '2': 30}
diccionario
```

[21]:
```
{'1': 25, '2': 30}
```

[22]:
```
diccionario['2']
```

[22]:
```
30
```

[23]:
```
diccionario['3'] = 400
diccionario
```

[23]:
```
{'1': 25, '2': 30, '3': 400}
```

[24]:
```
diccionario['1'] = 500
diccionario
```

[24]:
```
{'1': 500, '2': 30, '3': 400}
```

[25]:
```
diccionario2 = dict(Estudiantes=lista)
diccionario2
```

[25]:
```
{'Estudiantes': [1, 5, 6, 8, 3, 6, None, 10]}
```

[26]:
```
# claves que conforman el diccionario:
diccionario.keys()
```

[26]:
```
dict_keys(['1', '2', '3'])
```

[27]:
```
# valores que conforman el diccionario:
diccionario.values()
```

[27]:
```
dict_values([500, 30, 400])
```

[28]:
```
# clave-valor que conforman el diccionario:
diccionario.items()
```

[28]:
```
dict_items([('1', 500), ('2', 30), ('3', 400)])
```

1.5 Dataframes

[29]:
```
import pandas as pd

data = {'Nombre': lista}
df = pd.DataFrame(data)
df
```

[29]:
```
   Nombre
0     1.0
1     5.0
2     6.0
3     8.0
4     3.0
5     6.0
6     NaN
7    10.0
```

1.6 Formatos

[30]:
```
# enteros:

x = 10
print(type(x),':', x)
```

```
<class 'int'> : 10
```

[31]:
```
# float siempre se escribe con punto (.)
# sino sería una tupla:

y = 1,2
print(type(y))

y = 1.2
print(type(y), ':', y)

# abreviamos mediante el .X, significa un cero delante:
y = .5
print(y)
```

```
<class 'tuple'>
<class 'float'> : 1.2
0.5
```

[32]:
```
# string

texto = 'Hola Mundo'
print(texto)
texto = "Hola Mundo"
print(texto)

# Alternar comillas dobles con simples o viceversa:
texto = "Hola 'Mundo'"
print(texto)

# Se una el simbolo de escape para usar comillas simples fuera y dentro:
```

```
texto = 'Hola \'Mundo\''
print(texto)

# concatenacion:
texto = "Hola" + " " + "Mundo"
print(texto)

# Extraer los caracteres:
texto[2]
```

```
Hola Mundo
Hola Mundo
Hola 'Mundo'
Hola 'Mundo'
Hola Mundo
```

[32]:
```
'l'
```

[33]:
```
x = 'abcd'
x = '-'.join(x)
x
```

[33]:
```
'a-b-c-d'
```

[34]:
```
# Boleanos

x = True
print(type(x), ':', x)
y = False
print(type(y), ':', y)
```

```
<class 'bool'> : True
<class 'bool'> : False
```

---

2 The insert() method

2.1 insert: ejemplo

[35]:
```
# 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']
```

2.2 suma de elementos de una lista

[36]:
```
L = [10,20,30]
sum(L)
```

[36]:
```
60
```

[37]:
```
# L.sum()

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

2.2.1 OTRA FORMA DE SUMAR LOS ELEMENTOS DE UNA LISTA

[38]:
```
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
```

2.3 borrar elementos de una lista en un index concreto

[39]:
```
L
```

[39]:
```
[10, 20, 30]
```

[40]:
```
# SI QUIERO BORRAR EL ELEMENTO DE INDEX 1
del L[1]

L
```

[40]:
```
[10, 30]
```

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

[41]:
```
# 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
```

[42]:
```
# resuelto paso a paso
```

[43]:
```
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
```

2.5 Funciones y argumentos: ejemplo 1

[44]:
```
# 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
```

[45]:
```
# y+=1  equivalente a: y = y + 1
```

[46]:
```
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

[47]:
```
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
```

[48]:
```
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
```

[49]:
```
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
```

2.6 Funciones y argumentos: ejemplo 2

[50]:
```
# 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)
```

[51]:
```
# 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 llamada a␣
 ↪la función?


# 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
```

[52]:
```
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
```

[53]:
```
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
```

[54]:
```
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
```

2.7 Funciones y argumentos: ejemplo 3

[55]:
```
# otro ejemplo parecido,

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

[56]:
```
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
```

2.8 Funciones y argumentos: ejemplo 4

[57]:
```
# Ojo, que no es lo mismo que este otro ejemplo porque aquí NO SABE,
# lo que valen 'x' e 'y'
```

[58]:
```
"""
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()"""
```

[58]:
```
'\ndef test(x, y):\n    x = x + y          # x = 1 + 2 ======> x = 3        \n
y += 1             # y = y + 1 ======> y = 3\n    print(x, y)        # print(x,y)
=====> 3 3    \n\ntest()'
```

2.9 Funciones y argumentos: ejemplo 5

omitiendo algún valor en la llamada

[59]:
```
def test(x=1, y=2):
    x = x + y          # x =
    y += 1             # y = y + 1
    print(x, y)        #

test(2)
```

```
4 3
```

2.10 Funciones y argumentos: ejemplo 6

[60]:
```
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
```

2.11 Funciones y argumentos: ejemplo 7

mismo ejemplo sin decir y=4, simplemente 4

[61]:
```
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
```

2.12 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"

[62]:
```
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
```

2.13 Funciones y argumentos: ejemplo 9

ejemplo donde vemos que no permite poner varias comas

[63]:
```
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[63], line 7
    test(4,,2)
           ^
SyntaxError: invalid syntax
```

2.14 Funciones y argumentos: ejemplo 10

[64]:
```
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[64], line 6
    test(,4)
         ^
SyntaxError: invalid syntax
```

---

3 NO hacer copias de listas

y lo que ello implica..

3.1 copias y asignación de listas: ejemplo 1

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

[65]:
```
listado = [1,2,3]

copia_listado = listado.copy()
copia_listado
```

[65]: [1, 2, 3]

[66]:
```
id(listado)
```

[66]: 126575855291008

[67]:
```
id(copia_listado)
```

[67]: 126575855174784

[68]:
```
del listado[1]
listado
```

[68]: [1, 3]

[69]:
```
copia_listado
```

[69]: [1, 2, 3]

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

HEMOS MANTENIDO LOS VALORES INICIALES EN "COPIA_LISTADO"

3.2 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)

[70]:
```
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)
```

```
126575855374976
126575855374976
list2: [4, 3]
list1: [4, 3]
```

3.3 copias y asignación de listas: ejemplo 3

ahora modifico la lista 2 y compruebo..

[71]:
```
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

[72]:
```
list1 == list2
```

[72]:
```
True
```

[73]:
```
list1 is list2
```

[73]:
```
True
```

[74]:
```
if list1 == list2:
    print('son ambas [4,3]')
```

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

Aqui 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

3.4 copias y asignación de listas: ejemplo 4

asignación de valores (varios valores),

nota, recordar Numpy, porque era similar

[75]:
```
numeros = [1, 2, 3]
numeros
```

[75]:
```
[1, 2, 3]
```

[76]:
```
numeros[0:2] # 0 hasta 2 - 1 = 0 a 1
```

[76]:
```
[1, 2]
```

[77]:
```
numeros[:2] # 0 hasta 1
```

[77]:
```
[1, 2]
```

[78]:
```
numeros[1]
```

[78]: 2

[79]:
```
numeros[1:2] # 1 hasta 1
```

[79]: [2]

[80]:
```
type(numeros[1:2])
```

[80]: list

[81]:
```
type(numeros[1])
```

[81]: int

[82]:
```
numbers = [1,2,3]
numbers
```

[82]: [1, 2, 3]

[83]:
```
del numbers[1:2] # 1 hasta 2 -1 --> de 1 al 1 --> eliminar el valor de posicion␣
↪1 --> 2
numbers
```

[83]: [1, 3]

[84]:
```
datos = [1, 2, 5, 6, 8, 10]
datos
```

[84]: [1, 2, 5, 6, 8, 10]

[85]:
```
datos [::2] # recoger datos de dos en dos, saltar dos posiciones
```

[85]: [1, 5, 8]

3.5 copias y asignación de listas: ejemplo 5

el ejercicio de examen

[86]:
```
nums = [1, 2, 3]
vals = nums              # vals = [1, 2, 3]
del vals[1:2]
# añado:
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]
```

3.6 copias y asignación de listas: ejemplo 6

lo mismo que ejemplo 5 pero modificando la otra lista

[87]:
```
nums = [1, 2, 3]
vals = nums              # vals = [1, 2, 3]
del nums[1:2]
# añado:
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]
```

3.7 copias y asignación de listas: ejemplo 7

otro ejercicio similar, pero con copias de listas

[88]:
```
listado = [1,2,3]

copia_listado = listado.copy()
copia_listado
```

[88]:
```
[1, 2, 3]
```

[89]:
```
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]
```

3.8 copias y asignación de listas: ejemplo 8

[90]:
```
list1 = [1, 3] # [1, 3]
list2 = list1  # [1, 3]
list1[0] = 4   # [4, 3]
print('lista1 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]
```

[90]:
```
'\nlist1 = [1, 3]\nlist1 = [4, 3]\nlist1 = [4, 10]\n\n\nlist2 = [1, 3]\nlist2 =
 [4, 3]\nlist2 = [4, 10]\n'
```

3.9 copias y asignación de listas: ejemplo 9

[91]:
```
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]
```

3.10 copias y asignación de listas: ejemplo 10

[92]:
```
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: []
```

3.11 copias y asignación de listas: ejemplo 11

[93]:
```
# 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
```

[93]:
```
"\nlist1 = [1, 3]\nlist2 = list1  \nlist1[0] = 4   \nlist2[1] = 10  \ndel
list1[0] # [10]\ndel list2[1] \n\nprint('list1:', list1)  \nprint('list2:',
list2)\n"
```

---

4 Funciones: variables globales y locales

(el ejercicio de examen es el primero)

variables globales Vs variables locales en las funciones

4.1 Variables globales y locales: ejemplo 1

[94]:
```
num = 1

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

func(1)

print(num)
```

```
4
1
```

4.2 Variables globales y locales: ejemplo 2

[95]:
```
num = 1

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

func(6)
```

```
print(num)
```

```
9
1
```

4.3 Variables globales y locales: ejemplo 3

[96]:
```
num = 1

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

func()
print(num)
```

```
4
4
```

4.4 Variables globales y locales: ejemplo 4

[97]:
```
num = 1

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

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

```
4
20
```

4.5 Variables globales y locales: ejemplo 5

[98]:
```
'''
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
```

[98]:
```
'\nnum = 1\n\ndef func():\n    num = num + 3   \n    print(num)      \n\nfunc()
\n\nprint(num)  \n'
```

4.6 Variables globales y locales: ejemplo 6

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

pudiera ser también de examen

[99]:
```
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
```

4.7 Variables globales y locales: ejemplo 7

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

[100]:
```
num = 1

def func():
    global num
    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…: 13
```

4.8 Variables globales y locales: ejemplo 8

LA PALABRA GLOBAL DESPUÉS DE "num = 10"

[101]:
```
"""
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)
"""

# SyntaxError: name 'num' is assigned to before global declaration
```

[101]:
```
"\nnum = 1\n\ndef func():\n    num = 10\n    global num\n    num = num + 3    \n
print(num)       \n\nprint('la llamada a la función: ')\nfunc()
\nprint('\n')\n\nprint('el valor de num original…:', num)  \n"
```

4.9 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

[102]:
```
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
```

4.10 Variables globales y locales: ejemplo 10

un ejemplo diferente (el cual si que funciona)

es una posible corrección al código del examen

[103]:
```
num = 1

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

func()

print(num)
```

```
4
4
```

---

5 Ejemplo con try except

(el primero es el de examen)

[104]:
```
"""
try:
    print(5/0)
    break
except:
    print("Sorry, something went wrong...")
```

```
except (ValueError, ZeroDivisionError):
    print("Too bad...")
"""

# SyntaxError: 'break' outside loop
```

[104]:
```
'\ntry:\n    print(5/0)\n    break\nexcept:\n    print("Sorry, something went wrong…")\nexcept (ValueError, ZeroDivisionError):\n    print("Too bad…")\n'
```

[105]:
```
# ejemplo 2 de este tipo (este si funciona)

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

[106]:
```
try:
    print(5/0)
except (ValueError, ZeroDivisionError):
    print("Too bad...")
except:
    print("Sorry, something went wrong...")
```

```
Too bad…
```

[107]:
```
try:
    print(5/0)
except (ValueError):
    print("Too bad...")
except:
    print("Sorry, something went wrong...")
```

```
Sorry, something went wrong…
```

[108]:
```
try:
    print(5/0)
except ValueError:
    print("Too bad...")
except:
    print("Sorry, something went wrong...")
```

```
Sorry, something went wrong…
```

UNA POSIBILIDAD

[109]:
```
try:
    print(5/0)
except Exception as e:
    print(type(e))
    print(str(e))
```

```
<class 'ZeroDivisionError'>
division by zero
```

[110]:
```
try:
    print(5/0)
except:
    print("Sorry, something went wrong...")
except (ValueError, ZeroDivisionError):
    print("Too bad...")

# SyntaxError: 'break' outside loop
```

```
  Cell In[110], line 3
    except:
    ^
SyntaxError: default 'except:' must be last
```

---

6 Ejemplo con while

EJERCICIO MUY PROTOTIPO EN PCEP

6.1 While: ejemplo 1

[111]:
```
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
*
```

6.2 while: ejemplo 2:

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

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

[112]:
```
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
*
```

6.3 while: ejemplo 3

suma 1 en vez de 2

[113]:
```
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
```

*
*
*
*

---

7 Ejercicio prototipo: cuál es el resultado de..

7.1 repaso: diferencia entre * y esto: **

[114]:
```
2*3   # multiplicación
```

[114]: 6

[115]:
```
2**3   # 2 elevado a 3 (2*2*2)
```

[115]: 8

7.2 ejercico de examen:

se puede encontrar en PCEP y en PCAP

[116]:
```
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
```

[116]: 512

[117]:
```
# 2**(3**2) ==> 2**9
```

[118]:
```
2**1

# 2==>4==>8==>16==>32==>64==>128==>256==>512
```

[118]: 2

[119]:
```
2**9
```

[119]: 512

---

8 Ejercicio con funciones y bucle while

8.1 funciones y while: ejemplo 1

[120]:
```
"""
def func(text, num):
    while num > 0:
        print(text)
    num = num - 1

func('Hello', 3)
"""


# bucle infinito
```

[120]:
```
"\ndef func(text, num):\n    while num > 0:\n        print(text)\n    num = num
- 1\n\nfunc('Hello', 3)\n"
```

8.2 funciones y while: ejemplo 2

[121]:
```
# 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
```

---

9 Ejercicio con and, or, not y condicionales

9.1 Algunos links con info

https://realpython.com/lessons/operator-precedence/#:~:text=For%20Boolean%20operations%2C%20all%20and

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/

9.2 Ejercicios iniciales

[122]:
```
x = True
x
```

[122]: True

[123]:
```
not x
```

[123]: False

[124]:
```
True and False
```

[124]: False

[125]:
```
True and True
```

[125]: True

[126]:
```
True or True
```

[126]: True

[127]:
```
True or False
```

[127]: True

[128]:
```
False or False
```

[128]: False

[129]:
```
True or False or False
```

[129]: True

[130]:
```
# ejercicios con varios and y or
```

condición1 and condición2 or condición3 and condición4

(condición1 and condición2) or (condición3 and condición4)

resultado12 or resultado34

[131]:
```
True and False or True and True   # F or T  ===> T
```

[131]: True

[132]:
```
False and False or True and True  # F or T ===> T
```

[132]: True

[133]:
```
True and False or False and True  # F or F ===> F
```

[133]: False

9.3 Ejercicio 1: ejercicio de examen

[134]:
```
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)  ====> TRUE
    print(3) # esto imprime
else:
    print(4)
```

3

9.4 Ejercicio 2

[135]:
```
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)  ===> TRUE
    print(3)
else:
    print(4)
```

2

9.5 Ejercicio 3

[136]:
```
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

9.6 Ejercicio 4

[137]:
```
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

9.7 Ejercicio 5

[138]:
```
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

10 Operaciones básicas

10.1 repaso inicial de la división

[139]:
```
3/2  # división con decimales (EL RESULTADO SIEMPRE ES DECIMAL)
```

[139]: 1.5

[140]:
```
3 // 2  # cociente entero ( EL RESULTADO SIEMPRE ES ENTERO)
```

[140]: 1

[141]:
```
3 % 2   # resto de la división
```

[141]: 1

[142]:
```
10/3
```

[142]: 3.3333333333333335

[143]:
```
10//3
```

[143]: 3

[144]:
```
10%3
```

[144]: 1

[145]:
```
10/2   # ojo, el resultado es decimal, porque SIEMPRE es decimal
```

[145]: 5.0

[146]:
```
4**2  # 4 ELEVADO AL CUADRADO
```

[146]: 16

[147]:
```
4.1 ** 2
```

[147]: 16.81

10.2 ejercicio de examen resuelto paso a paso

[148]:
```
x = 1 / 2 + 3 // 3 + 4 ** 2

#    0.5  +   1     +  16     # 17.5
print(x)
```

17.5

10.3 ejemplo similar y de examen también (prototipo)

[149]:
```
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
```

[150]:
```
# 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
```

[150]:
```
18.0
```

10.4 OTROS EJEMPLOS..

[151]:
```
1/2, 3/2, 3//3, 4**2, 1//2
```

[151]:
```
(0.5, 1.5, 1, 16, 0)
```

[152]:
```
print('división:', 5/3, ', cociente: ', 5//3, ', resto:', 5%3)
```

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

[153]:
```
# 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
```

[153]:
```
17.5
```

---

11 Asignación múltiple de variables

[154]:
```
## ejemplo inicial
```

[155]:
```
x = 10
y = 20
print(x)
print(y)
```

```
10
20
```

[156]:
```
x,y = 15,25
print(x)
print(y)
```

```
15
25
```

[157]:
```
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
```

11.1 ejercicio de examen

[158]:
```
t,u = 10,-20
t,u
```

[158]:
```
(10, -20)
```

[159]:
```
# 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
```

---

12 argumentos en diferentes posiciones

[160]:
```
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 llamada
# con respecto a la definición de la propia función
# (para ello, habría que indicar qué variable tiene qué valor)
```

```
9
```

[161]:
```
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
```

13 print y los separadores

[162]:
```
z = y = x = 1
print(x, y, z)
```

```
1 1 1
```

13.1 ejemplo 1

[163]:
```
# ¿ 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
```

13.2 ejemplo 2

[164]:
```
z = y = x = 1
print(x, y, z, sep=' ')
```

```
1 1 1
```

13.3 ejemplo 3

[165]:
```
# 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
```

---

14 pop en una lista

[166]:
```
nums = [3, 4, 5, 20, 5, 25, 1, 3]
nums.pop(2)                          # borra el elemento en una posición concreta
print(nums)

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

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

---

15 División en Python

[167]:
```
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
```

[168]:
```
type(x / y), type(x // y), type(x % y)
```

[168]:
```
(float, int, int)
```

[169]:
```
type(6 / 2), 6 / 2
```

[169]:
```
(float, 3.0)
```

---

16 Ejemplo con diccionarios

16.1 diciconarios: repaso previo

[170]:
```
L = []
len(L)
```

[170]:
```
0
```

[171]:
```
dct = {}
print('dct:', dct)
print('len(dct):', len(dct))
```

```
dct: {}
len(dct): 0
```

[172]:
```
# {'key1': value1, 'key2': value2}

diccionario1 = {'clave1': [10,20,30], 'clave2': [15,25,35]}
diccionario1
```

[172]:
```
{'clave1': [10, 20, 30], 'clave2': [15, 25, 35]}
```

[173]:
```
import pandas as pd

df = pd.DataFrame({'clave1': (10,20,30), 'clave2': (15,25,35)})
df
```

[173]:

	clave1	clave2
0	10	15
1	20	25
2	30	35

[174]:
```
df['clave1']
```

[174]:
```
0    10
1    20
2    30
Name: clave1, dtype: int64
```

[175]:
```
df['clave3'] = [12,22,32]
df
```

[175]:

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

[176]:
```
# vuelvo a imprimir para verlo
diccionario1
```

[176]:
```
{'clave1': [10, 20, 30], 'clave2': [15, 25, 35]}
```

[177]:
```
diccionario1['clave1']
```

[177]:
```
[10, 20, 30]
```

[178]:
```
diccionario1['clave2']
```

[178]:
```
[15, 25, 35]
```

[179]:
```
diccionario1['clave1'][0]
```

[179]:
```
10
```

[180]:
```
diccionario1['clave1'][0] = 1000
diccionario1
```

[180]:
```
{'clave1': [1000, 20, 30], 'clave2': [15, 25, 35]}
```

[181]:
```
# otro pequeño ejercicio con explicaciones, previo a los ejemplos
```

[182]:
```
dct = {}
dct['key1'] = (1, 2)
dct['key2'] = (2, 1)
dct
```

[182]:
```
{'key1': (1, 2), 'key2': (2, 1)}
```

16.2 diccionarios: ejemplo 1

[183]:
```
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'])
```

16.3 diccionarios: ejemplo 2 (el ejercicio del examen)

[184]:
```
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
```

16.4 diccionarios: ejemplo 3

[185]:
```
# 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
```

---

17 string a lista

[186]:
```
print(list('hello'))
```

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

---

18 Ejemplo con funciones

[187]:
```
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
```

---

19 float de un string

[188]:
```
print(float('1.3'))
```

```
1.3
```

---

20 try except algunos ejemplos

20.1 excepciones: ejemplo 1

[189]:
```
try:
    first_prompt = input("Enter the first value: ")     # kangaroo
    a = len(first_prompt)                               # len(kangaroo) = 8
    second_prompt = input("Enter the second value: ")   # 0
    b = len(second_prompt) * 2                          # 2*len(0) = 2*1 = 2
    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.")
```

```
Enter the first value:  kangaroo
Enter the second value:  0
```

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

20.2 excepciones: ejemplo 2

[190]:
```
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.")
```

```
Enter the first value:  kangaroo
Enter the second value:  0
```

```
Error.Error.Error.
```

20.3 excepciones: ejemplo 3

[191]:
```
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.")
```

```
Enter the first value:  10
Enter the second value:  0
```

```
a: 10
b: 0
Error.Error.Error.
```

---

21 Suma de strings vs Suma de números

[192]:
```
# enteros

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

```
 2
 4
```

```
6
```

[193]:
```
# DECIMALES

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

```
 2
 4
```

```
2.0
4.0
6.0
```

[194]:
```
# 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
 4
```

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


24
```

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

[195]:
```
# 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
 mundo

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


holamundo
```

---

22 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)

[196]:
```
17 >> 1
# a = 17 --> n = 1 --> 17 / 2¹ = 8
```

[196]: 8

[197]:
```
17 << 2
# a = 17 --> n = 2 --> 17 * 2² = 68
```

[197]: 68

22.1 Bitwise right shift

[198]:
```
# 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.
```

[199]:
```
a = 10
#   0000 1010 (binario)
a >> 1
#   0000 0101 (binario) ====> 5 decimal

# 10 / 2¹ = 10/2 = 5
```

[199]: 5

22.2 Bitwise left shift

[200]:
```
a = 5
print('a:', a)
#  0000 0101  (binario)
b = a << 1 # a * 2^n --> 5 * 2¹=10
print('b:', b)
#  0000 1010  (binario)  ===> 10
c = a << 2 # 5 * 2² = 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
```

---

23 string y count

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

[201]:
```
# 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

[202]:
```
string = "Python is awesome, isn't it?"
substring = "i"

len(string)
```

[202]: 28

[203]:
```
# string = "Python is awesome, isn't it?"
#             6  + 2     7       5    2   + los espacios + la coma + el ? del␣
 ↪final
```

[204]:
```
# 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
```

[205]:
```
# 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
```

[206]:
```
# 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
```

[207]:
```
# 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
```

[208]:
```
# 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
```

[209]:
```
# 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
```

[210]:
```
# otro ejemplo
```

[211]:
```
data = 'abbabadaadbbaccabc'
print(data.count('ab', 1))
```

```
2
```

[212]:
```
data = 'abbabadaadbbaccabc'
print(data.count('ab', 1, len(data)))
```

```
2
```

[213]:
```
data = 'abbabadaadbbaccabc'
print(data.count('ab', 0, 1))
```

```
0
```

---

24 Un ejemplo con Try Except

[214]:
```
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..
```

```
Enter a value:  5
<class 'str'>
Very very bad input…
```

25 Función con diccionario

25.1 funciones y diccionarios: ejemplo 1

[215]:
```
data = {}

def func(d, key, value):
    d[key] = value

print(func(data, '1', 'Peter'))

# la función no tiene retorno
```

```
None
```

25.2 funciones y diccionarios: ejemplo 2

[216]:
```
data = {}

def func(d, key, value):
    d[key] = value
    return d

print(func(data, '1', 'Peter'))
```

```
{'1': 'Peter'}
```

26 len("\")

[217]:
```
x = '\'
print(len(x))

# SyntaxError: EOL while scanning string literal
```

```
  Cell In[217], line 1
    x = '\'
        ^
SyntaxError: unterminated string literal (detected at line 1)
```

[218]:
```
x = '\\'
print(len(x))
```

```
1
```

[219]:
```
x = '\\\\'
print(len(x))
```

```
2
```

[220]:
```
x = '\\\'
print(len(x))

# SyntaxError: EOL while scanning string literal
```

```
  Cell In[220], line 1
    x = '\\\'
        ^
SyntaxError: unterminated string literal (detected at line 1)
```

---

27 len(salto de linea)

[221]:
```
# comentario
```

[222]:
```
# comentario de linea 1
# comentario de linea 2
```

[223]:
```
"""
comentario de linea 1
comentario de linea 2
comentario de linea 3
"""
```

[223]:
```
'\ncomentario de linea 1\ncomentario de linea 2\ncomentario de linea 3\n'
```

[224]:
```
x = """"""
print(len(x))
```

```
0
```

[225]:
```
x = """
"""
print(len(x))
```

```
1
```

[226]:
```
x = """
```

```
"""
print(len(x))
```

2

[227]:
```
# hay un espacio en blanco en la línea donde comienza x, antes del salto de␣
 ↪línea
x = """

"""
print(len(x))
```

3

[228]:
```
x = """Hola
"""
print(len(x))
```

5

[229]:
```
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

---

28 Ejercicio prototipo de Funciones

28.1 ejemplo 1 de funciones

[230]:
```
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
```

28.2 ejemplo 2 de funciones: ejemplo 1 modificado

[231]:
```
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
```

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

[232]:
```
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
```

28.4 ejemplo 4 de funciones

otro ejemplo en el cual, no llega a ningún sitio

[233]:
```
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 last)
Cell In[233], 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[233], line 5, in func(x, y)
      3     return x
      4 else:
----> 5     return func(x, y-1)

Cell In[233], line 5, in func(x, y)
      3     return x
      4 else:
----> 5     return func(x, y-1)

    [… skipping similar frames: func at line 5 (2974 times)]

Cell In[233], line 5, in func(x, y)
      3     return x
      4 else:
----> 5     return func(x, y-1)

RecursionError: maximum recursion depth exceeded
```

---

29 pycache

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

---

30 LIFO / FIFO

[234]:
```
# Last In First Out (LIFO)
# First In First Out (FIFO)
```

---

31 ASCII / UNICODE

[ ]:
```
# https://elcodigoascii.com.ar/

# https://dinahosting.com/blog/que-es-utf-8/
```

[235]:
```
ord('0')
```

[235]:
```
48
```

[236]:
```
chr(48)
```

[236]:
```
'0'
```

[237]:
```
type(ord('0'))  # 48
```

[237]:
```
int
```

[238]:
```
type(chr(48)) # '0'
```

[238]:
```
str
```

[239]:
```
ord('9'), chr(57)
```

[239]:
```
(57, '9')
```

[240]:
```
# 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')
```

[240]:
```
66
```

[241]:
```
# dime el código necesario para imprimir la 'f'

# chr('102')   ==> no tiene sentido, porque 102 es número, no string

chr(102)
```

[241]:
```
'f'
```

[242]:
```
# escriba las lineas de código necesarias para conseguirÇ
# los números de 'a' y de 'A'
```

[243]:
```
# 'a'

ord('a')
```

[243]:
```
97
```

[244]:
```
# 'A'

ord('A')
```

[244]:
```
65
```

[245]:
```
ord('a') - ord('A')
# 97     -   65
#      32
```

[245]:
```
32
```

[246]:
```
ord('c') - ord('a')
# 99     -   97
#        2
```

[246]:
```
2
```

[247]:
```
chr(32)
```

[247]:
```
' '
```

[248]:
```
chr(39)
```

[248]:
```
"'"
```

[249]:
```
ord("'")
```

[249]:
```
39
```

[250]:
```
# ejemplo
```

[251]:
```
ord('b')
```

[251]:
```
98
```

[252]:
```
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
```

[253]:
```
chr(ord('p') + 3)
```

[253]:
```
's'
```

[254]:
```
# OTRO EJEMPLO (EJERCICIO PROTOTIPO)
```

[255]:
```
'mike' > 'Mike'
```

[255]:
```
True
```

[256]:
```
# The expression:

# 'mike' > 'Mike'

# is

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

[257]:
```
ord('m'), ord('M'), ord('m') > ord('M')
```

[257]:
```
(109, 77, True)
```

[258]:
```
'mike' > 'Mike'
```

[258]:
```
True
```

[259]:
```
# Solución
# C
```

---

32 operadores booleanos

[260]:
```
## not
```

[261]:
```
x = True
x
```

[261]: True

[262]:
```
not x
```

[262]: False

[263]:
```
# qué pasa si sumo True con números?
```

[264]:
```
True + 2  # True es 1
```

[264]: 3

[265]:
```
False + 2  # False es 0
```

[265]: 2

[266]:
```
True + 3.5
```

[266]: 4.5

[267]:
```
False + 3.5
```

[267]: 3.5

33 and

[268]:
```
print('and:', 0&0, 0&1, 1&0, 1&1)
```

```
and: 0 0 0 1
```

34 or

[269]:
```
print('or:', 0|0, 0|1, 1|0, 1|1)
```

```
or: 0 1 1 1
```

35 XOR (PUERTA LÓGICA)

[270]:
```
print('XOR:', 0^0, 0^1, 1^0, 1^1)

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

```
XOR: 0 1 1 0
```

[271]:
```
2^3   # NO ES 2 AL CUBO!!!
```

[271]:
```
1
```

35.1 ejercicio de examen

[272]:
```
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
```

36 and, or, xor

[273]:
```
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
```

[274]:
```
# explicación de algunas cosas, repaso
```

[275]:
```
# and, & => y
# or, | = ó
a = True
b = not a
b
```

[275]:
```
False
```

---

37 True en las sumas

[276]:
```
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
```

---

38 STRINGS

[277]:
```
# EJEMPLO 1

'a'+'b'
```

[277]:
```
'ab'
```

[278]:
```
# EJEMPLO 2

'1'+'4'
```

[278]:
```
'14'
```

[279]:
```
# EJEMPLO 3

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

[279]:
```
'14'
```

[280]:
```
## Suma de String + un número: EJEMPLO 4

# '1' + 4

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

[281]:
```
## Multiplicación de Strings: EJEMPLO 5

4 * 'a'
```

[281]: 'aaaa'

---

39 VARIABLE GLOBAL y variable local

[282]:
```
v = 1

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

print(v)
```

1

[283]:
```
# 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('aqui 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)


aqui llamo a la función: 2 (antes solo definida)
v después de llamar a la función vale: 2 (v es global)
```

[284]:
```
# 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('aqui 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)


aqui llamo a la función: 2 (antes solo definida)
v después de llamar a la función vale: 1 (v es local)
```

---

40 LENGUAJES INTERPRETADOS

[285]:
```
# https://www.freecodecamp.org/espanol/news/
 ↪lenguajes-compilados-vs-interpretados/

# https://aulab.es/noticia/18/
 ↪diferencia-entre-lenguajes-de-programacion-compilados-e-interpretados
```

---

41 ejemplo if elif elif .. else

[286]:
```
# 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

[287]:
```
# 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)
```

```
Enter a numeric grade: 75

Your letter grade is: C
```

forma 2: en pocas líneas

[288]:
```
# 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)
```

```
Enter a numeric grade: 75
```

```
Your letter grade is: C
```

---

42 ejemplo con continue

[289]:
```
# Which one of the lines should you put in the snippet below to match the␣
 ↪expected output?


# 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
```

[290]:
```
c = 0
while c < 5:
    c = c + 1
    if c == 3:
        continue
    print(c, end="")
```

```
1245
```

---

43 Bucles en una linea

[291]:
```
x =15

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

```
True
```

[292]:
```
x = 15
mayor_10 = True if x > 10 else False

print(mayor_10)
```

```
True
```

---

44 Caso del if elif

solo ejecuta la primera sentencia correcta que encuentra

[293]:
```
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')
```

[294]:
```
funcion_if(10)
```

```
numero: 10 , evaluamos: x > 9
```

[295]:
```
funcion_if(8)
```

```
numero: 8 , evaluamos: x > 7
```

[296]:
```
funcion_if(6)
```

```
numero: 6 , evaluamos: x > 5
```

[297]:
```
funcion_if(4)
```

```
numero: 4 , evaluamos: else
```

---

45 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:

45.1 Ejemplo 1: for...else

[298]:
```
# ¿ 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 0, 1
    # 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
```

45.2 Ejemplo 2: for...else

[299]:
```
# ¿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

45.3 Ejemplo 3: While...else

[301]:
```
# ¿Cuántos '#' mostrá 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
```

45.4 Ejemplo 4: while...else

[302]:
```
# ¿Cuántos '#' mostrá 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
```

#

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

[303]:
```
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!!!

---

46 Lambda

[304]:
```
def funcion_1(x, y):
    return x * y

funcion_1(4, 3)
```

[304]: 12

[305]:
```
(lambda x, y: x * y)(4, 3)
```

[305]: 12

[306]:
```
funcion_lambda = lambda x, y: x * y

funcion_lambda(4,3)
```

[306]: 12

---

47 List comprehension

[307]:
```
listado = []
lista_inicial = [10, 41, 45, 75, 87, 5]

for i in lista_inicial:
    listado.append(i)
listado
```

[307]: [10, 41, 45, 75, 87, 5]

[308]:
```
lista_inicial = [10, 41, 45, 75, 87, 5]

[i for i in lista_inicial]
```

[308]: [10, 41, 45, 75, 87, 5]

[309]:
```
listado = []
lista_inicial = [10, 41, 45, 75, 87, 5]

for i in lista_inicial:
    if i >= 45:
        listado.append(i)
    else:
        listado.append(-1)
listado
```

[309]: [-1, -1, 45, 75, 87, -1]

[310]:
```
lista_inicial = [10, 41, 45, 75, 87, 5]

listado = [i if i >= 45 else -1 for i in lista_inicial]
listado
```

[310]: [-1, -1, 45, 75, 87, -1]

---

48 Diccionarios

[311]:
```
diccionario = {"clave1": 4, "clave2": 8, "clave3": 10}
diccionario
```

[311]:
```
{'clave1': 4, 'clave2': 8, 'clave3': 10}
```

[312]:
```
diccionario.keys()
```

[312]:
```
dict_keys(['clave1', 'clave2', 'clave3'])
```

[313]:
```
type(diccionario.keys())
```

[313]:
```
dict_keys
```

[314]:
```
for key in diccionario.keys():
    print(key, end=' ')
```

```
clave1 clave2 clave3
```

[315]:
```
diccionario.values()
```

[315]:
```
dict_values([4, 8, 10])
```

[316]:
```
type(diccionario.values())
```

[316]:
```
dict_values
```

[317]:
```
for value in diccionario.values():
    print(value, end= ' ')
```

```
4 8 10
```

[318]:
```
diccionario.items()
```

[318]:
```
dict_items([('clave1', 4), ('clave2', 8), ('clave3', 10)])
```

[319]:
```
type(diccionario.items())
```

[319]:
```
dict_items
```

[320]:
```
for key, value in diccionario.items():
    print(key, value, end=' ')
```

```
clave1 4 clave2 8 clave3 10
```

[321]:
```
for i in diccionario:
    print(diccionario[i])
```

```
4
8
10
```

[322]:
```
claves = ["clave1", "clave2", "clave3"]
valores = [4, 8, 10]
```

```
diccionario2 = dict(zip(claves, valores))
diccionario2
```

[322]: {'clave1': 4, 'clave2': 8, 'clave3': 10}

[323]:
```
claves = ["clave1", "clave2", "clave3"]
valores = [4, 8, 10]

listado = list(zip(claves, valores))
listado
```

[323]: [('clave1', 4), ('clave2', 8), ('clave3', 10)]

---

49 Prints

[324]:
```
# Jupyter no necesitamos añadir el print:

x = 1
x
```

[324]: 1

[325]:
```
# Jupyter en el caso de tener más de una variable usaremos la función print:

x = 1
y = 2
print(x)
print(y)
```

```
1
2
```

[326]:
```
# VSC si necesitamos usar print:

x = 1
print(x)
```

```
1
```

[327]:
```
z = x + y
print("La suma de " + str(x) + " más " + str(y) + " es " + str(z))
```

```
La suma de 1 más 2 es 3
```

[328]:
```
print(f"La suma de {x} más {y} es {z}")
```

```
La suma de 1 más 2 es 3
```

[329]:
```
print("La suma de %s más %s es %s" %(x, y, z))
```

```
La suma de 1 más 2 es 3
```

[330]:
```
print("La suma de {} más {} es {}".format(x, y, z))
```

```
La suma de 1 más 2 es 3
```

[331]:
```
print("La suma de ", x, " más ", y, " es " , z)
```

```
La suma de  1  más  2  es  3
```

---

50 Set

[332]:
```
numeros = [
            1, 5, 7,
            98, 74, 65,
            14, 1, 4,
            5, 98, 74
]
numeros
```

[332]:
```
[1, 5, 7, 98, 74, 65, 14, 1, 4, 5, 98, 74]
```

[333]:
```
numeros = set(numeros)
numeros
```

[333]:
```
{1, 4, 5, 7, 14, 65, 74, 98}
```

[334]:
```
numeros.append(6)
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
Cell In[334], line 1
----> 1 numeros.append(6)

AttributeError: 'set' object has no attribute 'append'
```

[335]:
```
numeros[0] = 6
numeros
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[335], line 1
----> 1 numeros[0] = 6
      2 numeros
```

```
TypeError: 'set' object does not support item assignment
```

[336]:
```
numeros = list(numeros)
numeros
```

[336]:
```
[1, 98, 65, 4, 5, 7, 74, 14]
```

[337]:
```
numeros = [
            1, 5, 7,
            98, 74, 65,
            14, 1, 4,
            5, 98, 74
]

numeros = list(set(numeros))
numeros.sort()
numeros
```

[337]:
```
[1, 4, 5, 7, 14, 65, 74, 98]
```

---

51 Pandas - DataFrame

Contiene dos tipos de estructuras:

- **Series**: una matriz etiquetada unidimensional que contiene datos de cualquier tipo como números enteros, cadenas, objetos Python, etc.
- **Dataframe**: una estructura de datos bidimensional que contiene datos como una matriz bidimensional o una tabla con filas y columnas.

[338]:
```
# pip install pandas
```

[2]:
```
import pandas as pd
import numpy as np
```

51.1 Creación de un dataframe

pd.DataFrame()

- A partir de Series:

[3]:
```
# Series:

s = pd.Series([1, 3, 5, np.nan, 6, 8])
s
```

[3]:
```
0    1.0
1    3.0
2    5.0
3    NaN
4    6.0
5    8.0
dtype: float64
```

[4]:
```
# date_range(genera un rango de fecha apartir de un valor, marcando el número␣
↪de datos a generar (periods)
dates = pd.date_range("20130101", periods=6)
dates
```

[4]:
```
DatetimeIndex(['2013-01-01', '2013-01-02', '2013-01-03', '2013-01-04',
               '2013-01-05', '2013-01-06'],
              dtype='datetime64[ns]', freq='D')
```

[5]:
```
df = pd.DataFrame(np.random.randn(6, 4), index=dates, columns=list("ABCD"))
df
```

[5]:

	A	B	C	D
2013-01-01	-1.429719	3.164323	-1.344881	-3.089733
2013-01-02	-0.363130	0.167882	-0.572161	-1.074723
2013-01-03	1.585374	0.938935	0.955911	0.452060
2013-01-04	-0.400563	0.997526	0.780708	0.371307
2013-01-05	-0.652954	-0.065344	-0.364601	0.917402
2013-01-06	0.579016	-0.437385	0.306639	0.084640

[6]:
```
# dtypes nos muestra de que tipo son los datos:

df.dtypes
```

[6]:
```
A    float64
B    float64
C    float64
D    float64
dtype: object
```

- A partir de un diccionario:

[7]:
```
notas = [10, 9, 5, 4, 8]

alumnos = ["Paula", "Fermin", "Pedro", "Luis", "Ana"]

evaluacion = {"Alumnos": alumnos, "Notas": notas}

df_eval = pd.DataFrame(evaluacion)
df_eval
```

[7]:
```
  Alumnos  Notas
0   Paula     10
1  Fermin      9
2   Pedro      5
3    Luis      4
4     Ana      8
```

- A partir de listas:

[8]:
```
df_evaluar = pd.DataFrame(alumnos, columns=["Alumnos"])
df_evaluar
```

[8]:
```
  Alumnos
0   Paula
1  Fermin
2   Pedro
3    Luis
4     Ana
```

Otra forma...

[9]:
```
df_evaluar = pd.DataFrame(list(zip(alumnos, notas)), columns=["Alumnos",␣
 ↪"Notas"])
df_evaluar
```

[9]:
```
  Alumnos  Notas
0   Paula     10
1  Fermin      9
2   Pedro      5
3    Luis      4
4     Ana      8
```

51.2 Vista de los datos

[10]:
```
# Muestra las primeras filas del dataframe, por defecto las 5 primeras
df.head()
```

[10]:
```
                   A         B         C         D
2013-01-01 -1.429719  3.164323 -1.344881 -3.089733
2013-01-02 -0.363130  0.167882 -0.572161 -1.074723
2013-01-03  1.585374  0.938935  0.955911  0.452060
2013-01-04 -0.400563  0.997526  0.780708  0.371307
2013-01-05 -0.652954 -0.065344 -0.364601  0.917402
```

[11]:
```
df.head(2)
```

[11]:
```
                   A         B         C         D
2013-01-01 -1.429719  3.164323 -1.344881 -3.089733
2013-01-02 -0.363130  0.167882 -0.572161 -1.074723
```

[12]:
```
# Muestra las últimas filas de un dataframe, por defecto las 5 últimas:

df.tail()
```

[12]:
```
                   A         B         C         D
2013-01-02 -0.363130  0.167882 -0.572161 -1.074723
2013-01-03  1.585374  0.938935  0.955911  0.452060
2013-01-04 -0.400563  0.997526  0.780708  0.371307
2013-01-05 -0.652954 -0.065344 -0.364601  0.917402
2013-01-06  0.579016 -0.437385  0.306639  0.084640
```

[13]:
```
df.tail(2)
```

[13]:
```
                   A         B         C         D
2013-01-05 -0.652954 -0.065344 -0.364601  0.917402
2013-01-06  0.579016 -0.437385  0.306639  0.084640
```

[14]:
```
# Muestra el valor de la primera columna que suele ser un valor único (id), en␣
 ↪este ejemplo una fecha:

df.index
```

[14]:
```
DatetimeIndex(['2013-01-01', '2013-01-02', '2013-01-03', '2013-01-04',
               '2013-01-05', '2013-01-06'],
              dtype='datetime64[ns]', freq='D')
```

[15]:
```
# Muestra el nombre de las columnas:

df.columns
```

[15]:
```
Index(['A', 'B', 'C', 'D'], dtype='object')
```

[16]:
```
# Para obtener los estadísticos más representativos usamos:

df.describe()
```

[16]:
```
              A         B         C         D
count  6.000000  6.000000  6.000000  6.000000
mean  -0.113663  0.794323 -0.039731 -0.389841
std    1.052037  1.290555  0.880549  1.481934
min   -1.429719 -0.437385 -1.344881 -3.089733
25%   -0.589856 -0.007037 -0.520271 -0.784882
50%   -0.381846  0.553409 -0.028981  0.227973
75%    0.343479  0.982878  0.662191  0.431871
max    1.585374  3.164323  0.955911  0.917402
```

[17]:
```
# Colocar los valores según el indice:

```

```
df.sort_index(axis=1, ascending=False)
```

[17]:
```
                   D         C         B         A
2013-01-01 -3.089733 -1.344881  3.164323 -1.429719
2013-01-02 -1.074723 -0.572161  0.167882 -0.363130
2013-01-03  0.452060  0.955911  0.938935  1.585374
2013-01-04  0.371307  0.780708  0.997526 -0.400563
2013-01-05  0.917402 -0.364601 -0.065344 -0.652954
2013-01-06  0.084640  0.306639 -0.437385  0.579016
```

[18]:
```
# Ordenar los datos según una columna:

df.sort_values(by="B")
```

[18]:
```
                   A         B         C         D
2013-01-06  0.579016 -0.437385  0.306639  0.084640
2013-01-05 -0.652954 -0.065344 -0.364601  0.917402
2013-01-02 -0.363130  0.167882 -0.572161 -1.074723
2013-01-03  1.585374  0.938935  0.955911  0.452060
2013-01-04 -0.400563  0.997526  0.780708  0.371307
2013-01-01 -1.429719  3.164323 -1.344881 -3.089733
```

51.3 Gestionar un dataframe

[19]:
```
df_evaluar = pd.DataFrame(alumnos, columns=["Alumnos"])
df_evaluar
```

[19]:
```
  Alumnos
0   Paula
1  Fermin
2   Pedro
3    Luis
4     Ana
```

Apendizar columnas...

[20]:
```
df_evaluar["Fisica"] = notas
df_evaluar
```

[20]:
```
  Alumnos  Fisica
0   Paula      10
1  Fermin       9
2   Pedro       5
3    Luis       4
4     Ana       8
```

[21]:
```
df_evaluar["Mates"] = [5, 8, 9, 4, 6]
df_evaluar
```

[21]:

	Alumnos	Fisica	Mates
0	Paula	10	5
1	Fermin	9	8
2	Pedro	5	9
3	Luis	4	4
4	Ana	8	6

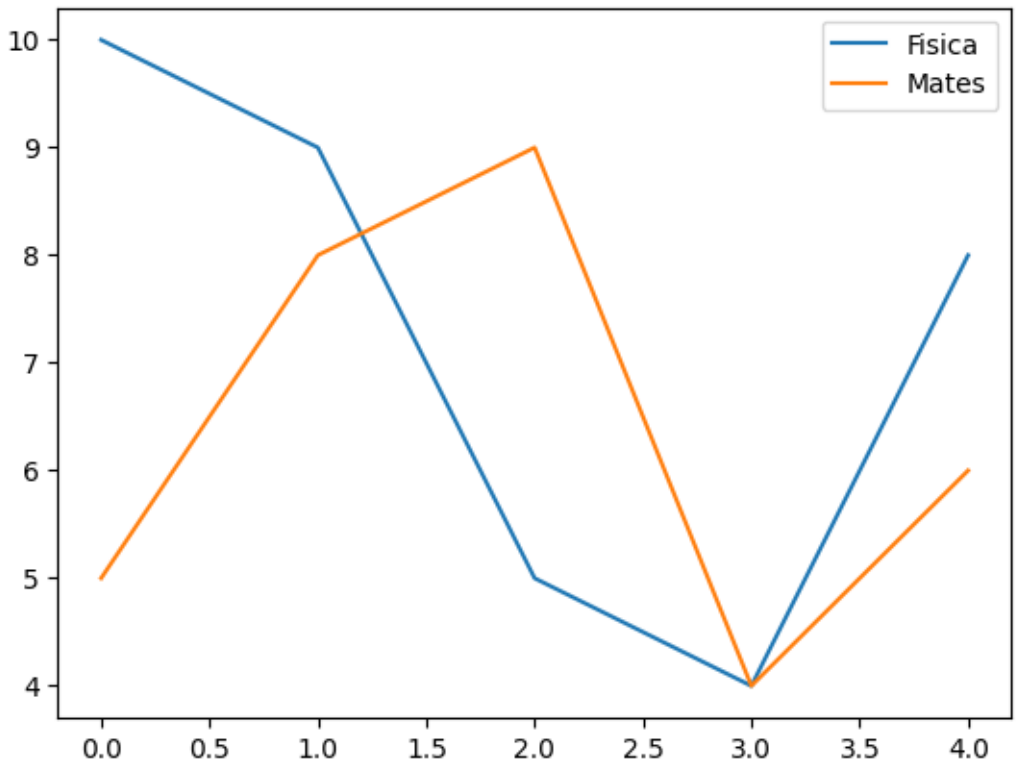
Mostrar información...

[26]:
```
# pip install matplotlib
```

[30]:
```
import matplotlib.pyplot as plt
```

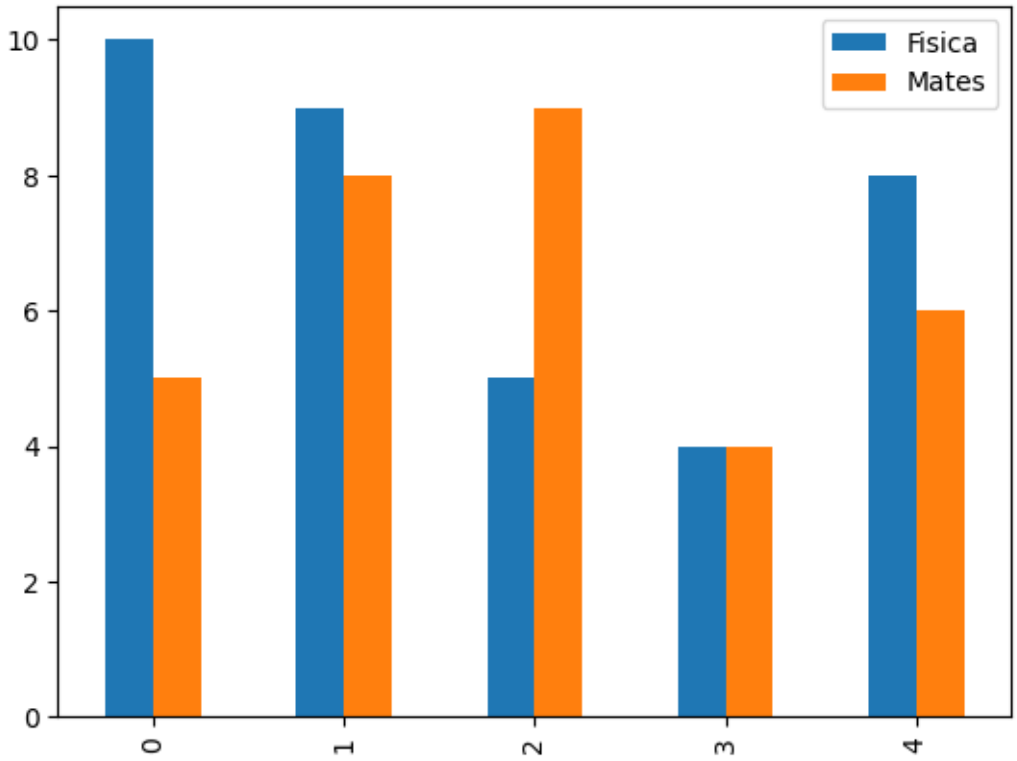
[31]:
```
# Gráfico de lineas:

df_evaluar.plot()
```

[31]: <Axes: >



[33]:
```
df_evaluar.plot(kind="bar") # gráfico de barras
```

[33]: <Axes: >



Filtrar valores...

[22]:
```
# columnas:

print(df_evaluar["Fisica"])
print(df_evaluar.Fisica)
df_evaluar[["Fisica"]]
```

```
0    10
1     9
2     5
3     4
4     8
Name: Fisica, dtype: int64
0    10
1     9
2     5
3     4
4     8
Name: Fisica, dtype: int64
```

[22]:

	Fisica
0	10

```
1       9
2       5
3       4
4       8
```

[360]:
```
# Filas: de la fila 1 a la 3 no incluída

df_evaluar[1:3]
```

[360]:
```
  Alumnos  Fisica  Mates
1  Fermin       9      8
2   Pedro       5      9
```

[361]:
```
# Filas: de la fila 1 a la 3 incluída, todas las columnas

df_evaluar.loc[1:3]
```

[361]:
```
  Alumnos  Fisica  Mates
1  Fermin       9      8
2   Pedro       5      9
3    Luis       4      4
```

[362]:
```
# Filas: de la fila 1 a la 3 incluída, todas las columnas

df_evaluar.loc[1:3, :]
```

[362]:
```
  Alumnos  Fisica  Mates
1  Fermin       9      8
2   Pedro       5      9
3    Luis       4      4
```

[363]:
```
# Filas: de la fila 1 a la 3 incluída, columna de Fisica

df_evaluar.loc[1:3, "Fisica"]
```

[363]:
```
1    9
2    5
3    4
Name: Fisica, dtype: int64
```

[364]:
```
# Filas

df_evaluar.loc[1:3, ["Alumnos", "Fisica"]]
```

[364]:
```
  Alumnos  Fisica
1  Fermin       9
2   Pedro       5
3    Luis       4
```

[365]:
```
# Filas sin etiquetas .iloc(): de la fila 1 a la 4 NO incluída, columnas de␣
↪Alumnos y Física

df_evaluar.iloc[1:4, 0:2]
```

[365]:
```
  Alumnos  Fisica
1  Fermin       9
2   Pedro       5
3    Luis       4
```

Valores Faltantes (Missing values)...

[366]:
```
df_evaluar
```

[366]:
```
  Alumnos  Fisica  Mates
0   Paula      10      5
1  Fermin       9      8
2   Pedro       5      9
3    Luis       4      4
4     Ana       8      6
```

[367]:
```
df_evaluar["Lengua"] = [None, 5, 9, None, 6]
df_evaluar
```

[367]:
```
  Alumnos  Fisica  Mates  Lengua
0   Paula      10      5     NaN
1  Fermin       9      8     5.0
2   Pedro       5      9     9.0
3    Luis       4      4     NaN
4     Ana       8      6     6.0
```

[368]:
```
df_evaluar.isnull()
```

[368]:
```
   Alumnos  Fisica  Mates  Lengua
0    False   False  False    True
1    False   False  False   False
2    False   False  False   False
3    False   False  False    True
4    False   False  False   False
```

[369]:
```
df_evaluar.isnull().sum()
```

[369]:
```
Alumnos    0
Fisica     0
Mates      0
Lengua     2
dtype: int64
```

[370]:
```
df_evaluar.describe()
```

[370]:

	Fisica	Mates	Lengua
count	5.000000	5.000000	3.000000
mean	7.200000	6.400000	6.666667
std	2.588436	2.073644	2.081666
min	4.000000	4.000000	5.000000
25%	5.000000	5.000000	5.500000
50%	8.000000	6.000000	6.000000
75%	9.000000	8.000000	7.500000
max	10.000000	9.000000	9.000000

[371]:
```
df_evaluar = df_evaluar.fillna(df_evaluar["Lengua"].mean())
df_evaluar
```

[371]:

	Alumnos	Fisica	Mates	Lengua
0	Paula	10	5	6.666667
1	Fermin	9	8	5.000000
2	Pedro	5	9	9.000000
3	Luis	4	4	6.666667
4	Ana	8	6	6.000000

[372]:
```
df_evaluar.isin([4])
```

[372]:

	Alumnos	Fisica	Mates	Lengua
0	False	False	False	False
1	False	False	False	False
2	False	False	False	False
3	False	True	True	False
4	False	False	False	False

Gracias por la atención

Isabel Maniega