

# Manual de Python



Gonzalo Pascual  
Romero

# Instalación

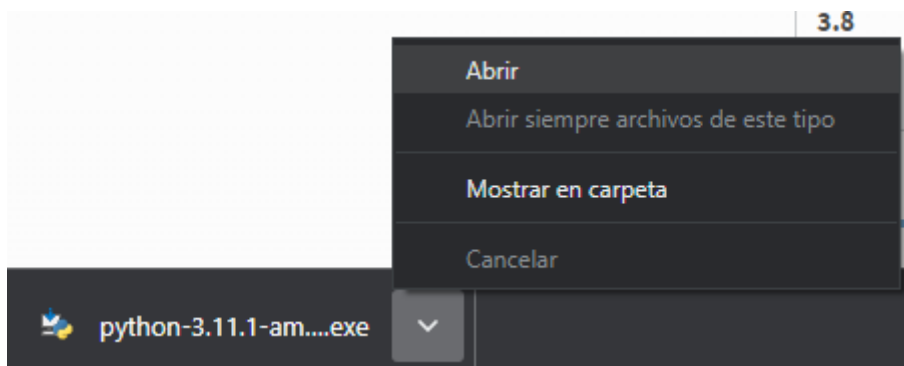
Para instalar Python primero debemos ir al siguiente enlace:

<https://www.python.org/downloads/>

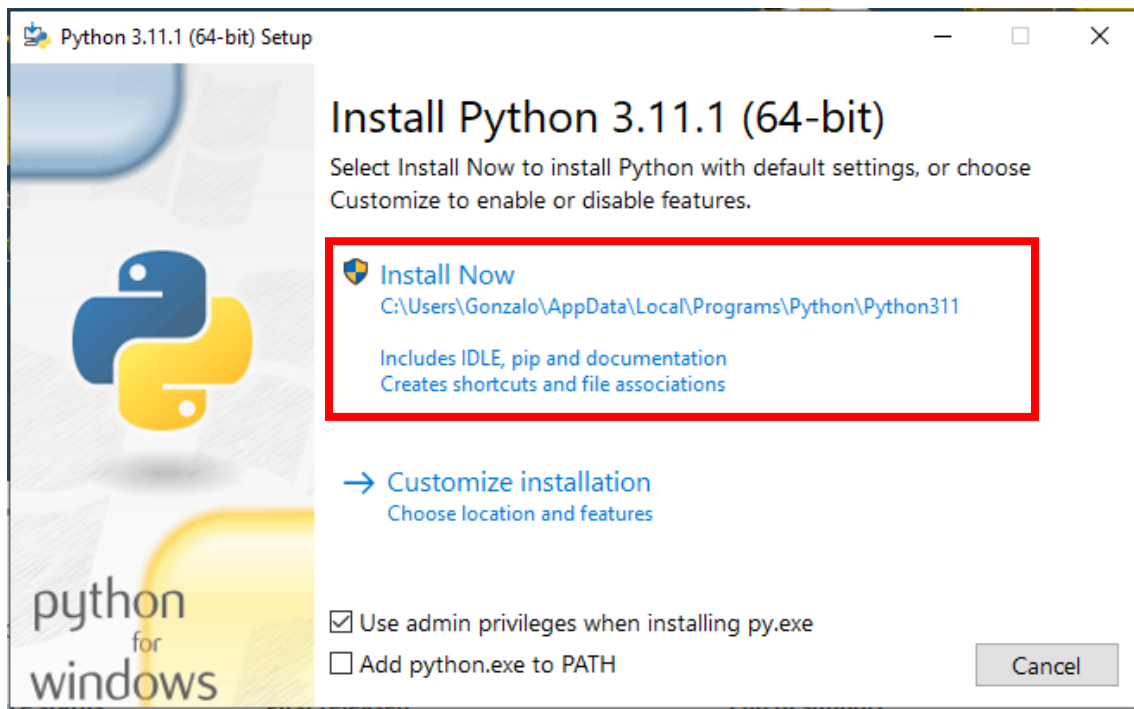
y darle a descargar la versión mas reciente, en este caso descargaremos la versión 3.11.1



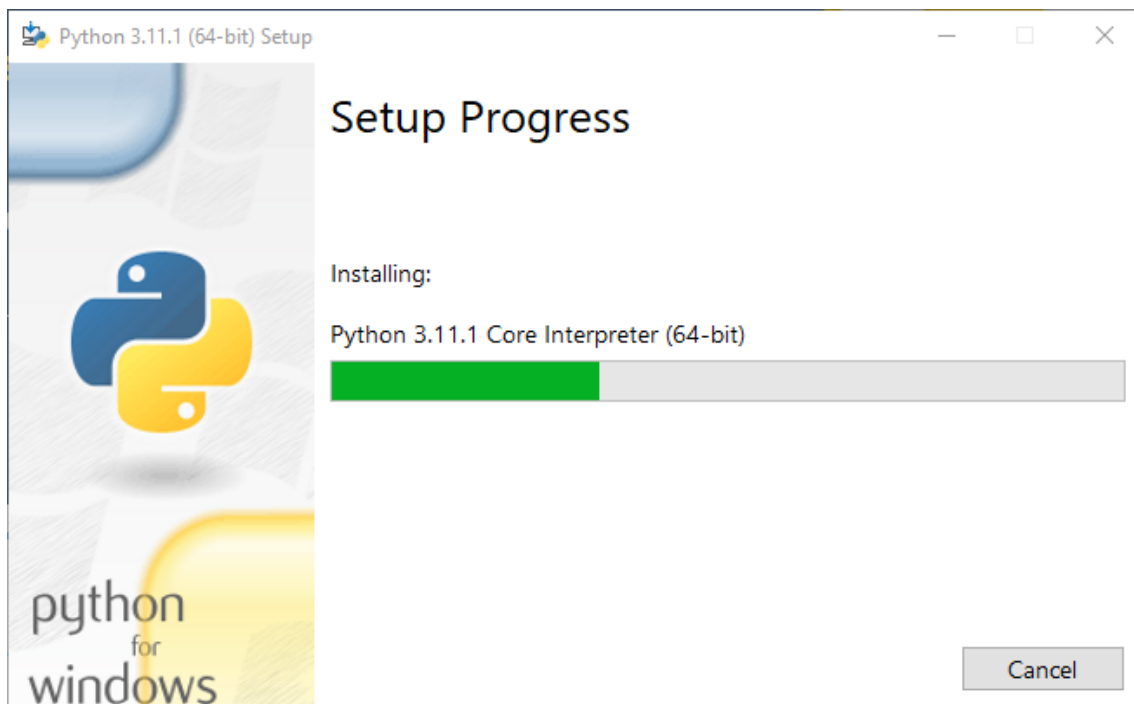
Una vez pulsado nos saldrá en la barra inferior cuanto le falta para descargarse y una vez descargado haremos Click sobre la flecha y pulsaremos abrir



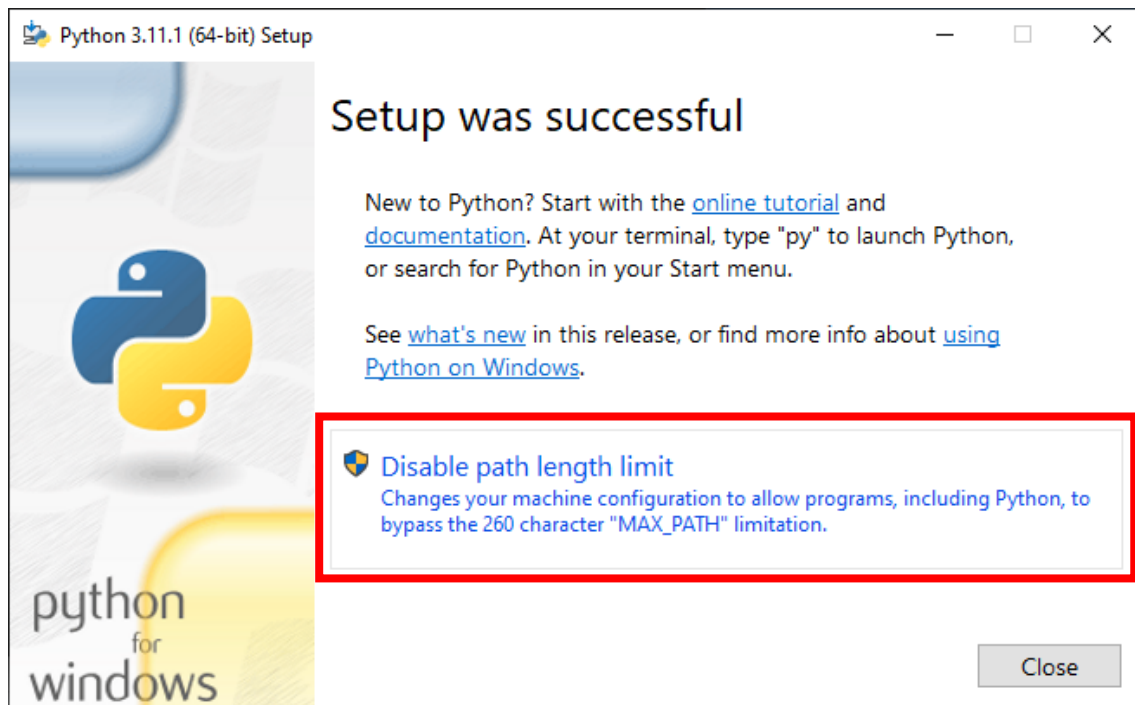
Comenzará la instalación en la que seleccionamos “Install Now” con las características predefinidas



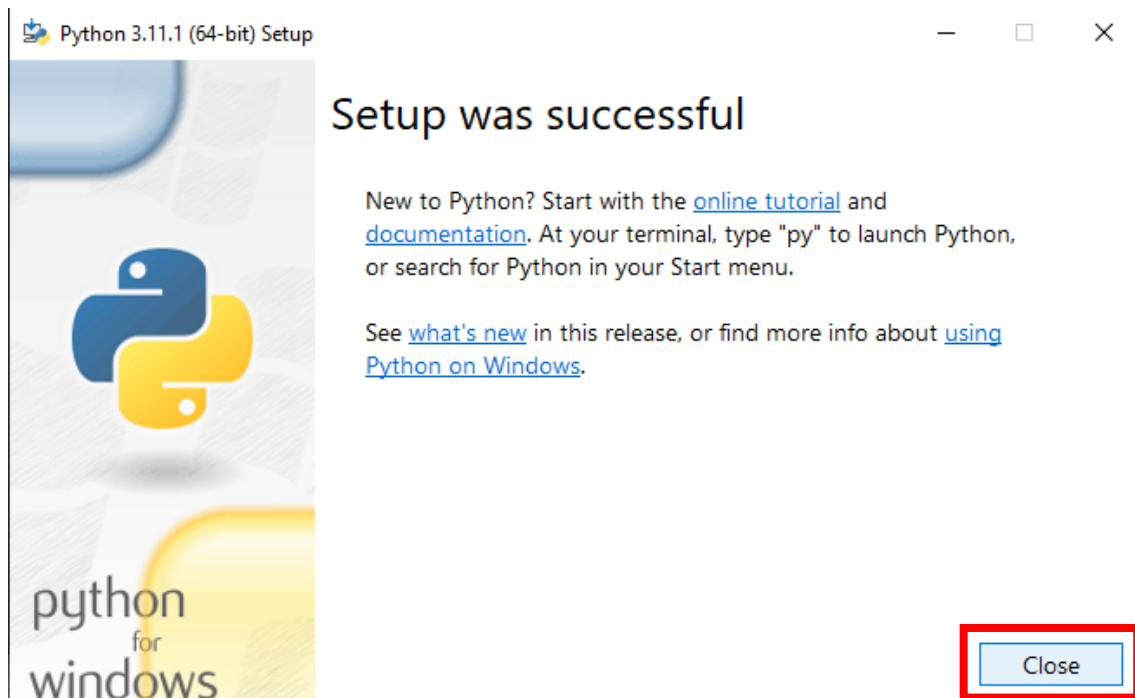
Y se empezara a instalar



Una vez terminado le daremos a “Disable path lenght limit” para que deshabilite la limitación de 260 caracteres en la consola.

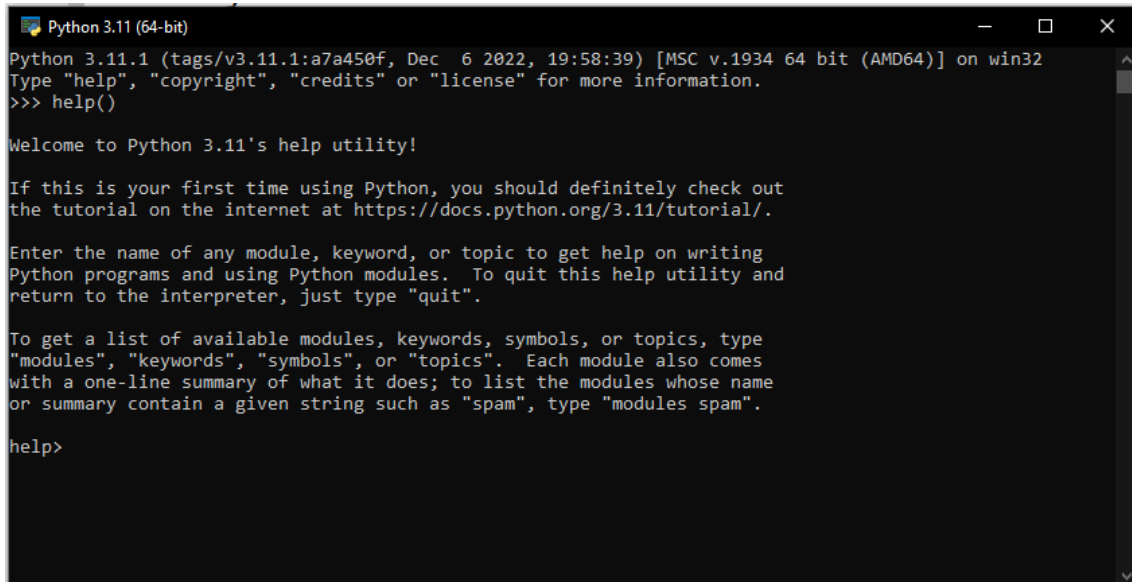


Y por último finalizaremos la instalación dándole a “Close”



Para usarlo se puede abrir desde su terminal o como yo lo voy a hacer a través de visual studio code.

Terminal:



```
Python 3.11 (64-bit)
Python 3.11.1 (tags/v3.11.1:a7a450f, Dec 6 2022, 19:58:39) [MSC v.1934 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> help()

Welcome to Python 3.11's help utility!

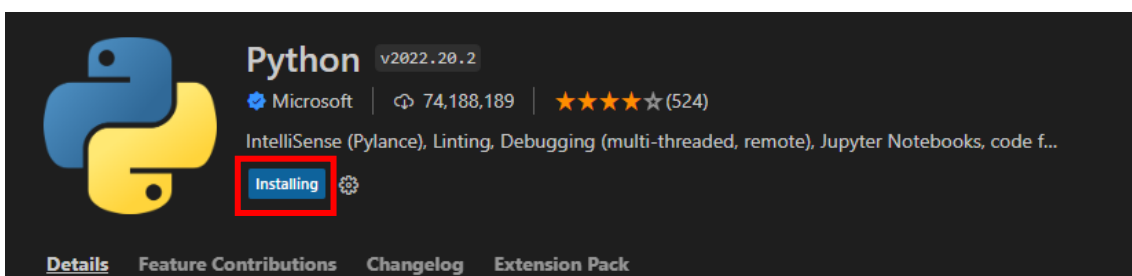
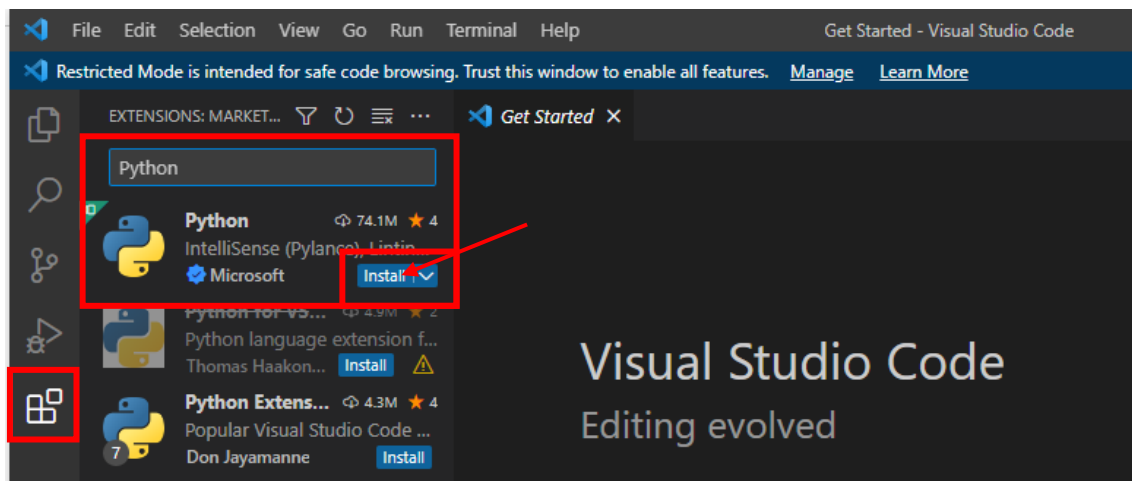
If this is your first time using Python, you should definitely check out
the tutorial on the internet at https://docs.python.org/3.11/tutorial/.

Enter the name of any module, keyword, or topic to get help on writing
Python programs and using Python modules. To quit this help utility and
return to the interpreter, just type "quit".

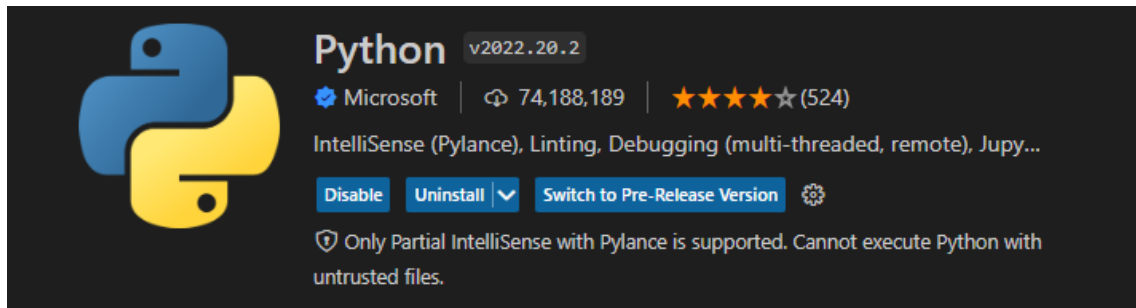
To get a list of available modules, keywords, symbols, or topics, type
"modules", "keywords", "symbols", or "topics". Each module also comes
with a one-line summary of what it does; to list the modules whose name
or summary contain a given string such as "spam", type "modules spam".

help>
```

Para que reconozca Python en Visual Studio Code, tenemos que añadirle la extensión de Python. Por lo que entraremos en la aplicación, iremos a extensiones, buscaremos Python e instalaremos la primera que sea de Microsoft



Y ya la tendríamos instalada.



Creamos un proyecto de Python y ya podríamos programar

```
primero.py X
G: > 2DAM > Sistema de gestión de empresa > primero.py
1 print('hola mundo')
2
3

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
PS C:\Users\Gonzalo> & C:/Users/Gonzalo/AppData/Local/Programs/Python/Python311/Python.exe primero.py
hola mundo
PS C:\Users\Gonzalo>
```

# Operadores

## - Aritméticos

Los operadores aritméticos que tenemos en Python son los siguientes:

+	Suma	$12 + 3 = 15$
-	Resta	$12 - 3 = 9$
*	Multipliación	$12 * 3 = 36$
/	División (con decimales)	$12 / 3 = 4$
%	Modulo (resto operación)	$16 \% 3 = 1$
**	Potencia = $12^3$	$12 ** 3 = 1728$
//	División sin decimales (entero)	$18 // 5 = 3$

```
G:\> 2DAM > Sistema de gesti3n de empresa > primero.py  
1 x = 2+4 #suma  
2 y = 4-5 #resta  
3 z = 4*5 # multiplicaci3n  
4 m = 30/7 #division(con decimales)  
5 v = 23%2 #modulo(resto)  
6 t = 3**2 #potencia  
7 p = 17//5 #divisi3n sin resto  
8 print(x)  
9 print(y)  
10 print(z)  
11 print(m)  
12 print(v)  
13 print(t)  
14 print(p)  
15
```

---

```
PROBLEMS      OUTPUT      DEBUG CONSOLE    TERMINAL  
PS C:\Users\Gonzalo> & C:/Users/Gonzalo/AppData/Local/Microsoft/WindowsApps/python3.10/python.exe C:/Users/Gonzalo/Desktop/Sistema de gesti3n de empresa/primeropy.py  
6  
-1  
20  
4.285714285714286  
1  
9  
3  
PS C:\Users\Gonzalo>
```

## - Relacionales

Los operadores relacionales que tenemos en Python son los siguientes:

>	Mayor que	12 > 3
<	Menor que	12 < 3
==	Igual que	12 == 3
>=	Mayor o igual que	12 >= 3
<=	Menor o igual que	12 <= 3
!=	Diferente a	12 != 3

```
1 x= 12 < 12
2 print (x)
```

PROBLEMS OUTPUT DEBUG

```
PS C:\Users\Gonzalo> & C:\Program Files\Python39\python.exe C:\Users\Gonzalo\AppData\Local\Temp\prueba/primer0.py
False
PS C:\Users\Gonzalo>
```



## - Asignación

Los operadores de asignación que tenemos en Python son los siguientes:

=	x = 5. El valor 5 es asignado a la variable a
+=	x += 5 es equivalente a x = x + 5
-=	x -= 5 es equivalente a x = x - 5
*=	x *= 3 es equivalente a x = x * 3
/=	x /= 3 es equivalente a x = x / 3
%=	x %= 3 es equivalente a x = x % 3
**=	x **= 3 es equivalente a x = x ** 3
//=	x //= 3 es equivalente a x = x // 3
&=	x &= 3 es equivalente a x = x & 3
=	x  = 3 es equivalente a x = x   3
^=	x ^= 3 es equivalente a x = x ^ 3
>>=	x >>= 3 es equivalente a x = x >> 3
<<=	x <<= 3 es equivalente a x = x << 3

```

1  x = 4
2  x+=2
3  print (x)

```

PROBLEMS OUTPUT DEBU

```

PS C:\Users\Gonzalo> & C
presa/primero.py"
6
PS C:\Users\Gonzalo>

```

## - Lógicos

Los operadores lógicos que tenemos en Python son los siguientes:

and	Devuelve True si ambos operandos son True	a and b
or	Devuelve True si alguno de los operandos es True	a or b
not	Devuelve True si alguno de los operandos False	not a

```
G: > 2DAM > Sistema de gestión de e
1  x = 12 == 4
2  y = 23>1
3  print (x and y)

PROBLEMS  OUTPUT  DEBUG CONS

PS C:\Users\Gonzalo> & C:/User
presa/primer.py
False
PS C:\Users\Gonzalo> |
```

## - Pertenencia

Los operadores de pertenencia que tenemos en Python son los siguientes:

In	Devuelve True si se encuentra en la secuencia
Not in	Devuelve True si no se encuentra en la secuencia

```
3. > ZDAM > Sistema de gestion de en
1  a = [1, 2, 3, 4, 5]
2  print ([3] in a)
3  print (6 in a)

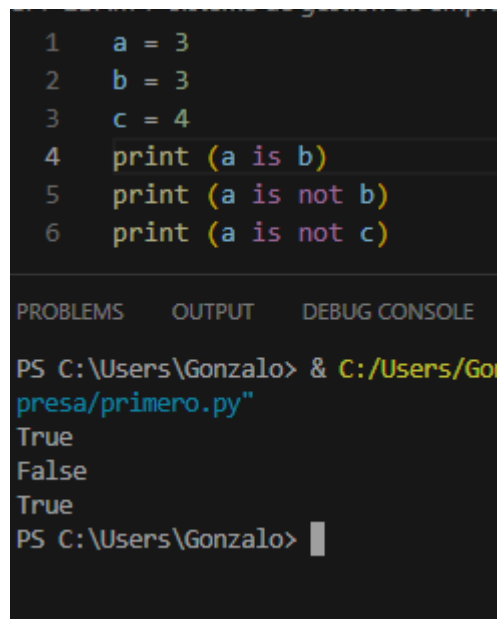
PROBLEMS  OUTPUT  DEBUG CONSOLE

PS C:\Users\Gonzalo> & C:/Users/
presa/primer0.py"
True
False
PS C:\Users\Gonzalo>
```

## - Identidad

Los operadores de asignación que tenemos en Python son los siguientes:

Is	Devuelve True si se encuentra en la secuencia
Is not	Devuelve True si no se encuentra en la secuencia



```

1  a = 3
2  b = 3
3  c = 4
4  print (a is b)
5  print (a is not b)
6  print (a is not c)

```

PROBLEMS OUTPUT DEBUG CONSOLE

```

PS C:\Users\Gonzalo> & C:/Users/Gonzalo/AppData/Local/Programs/Python/Python39-64/Scripts/python.exe C:/Users/Gonzalo/AppData/Local/Programs/Python/Python39-64/Scripts/presalib/presalib/primero.py
True
False
True
PS C:\Users\Gonzalo>

```

## - Sintaxis

(Apartado de sintaxis útil)

El uso de comillas es indiferente, se pueden usar dobles o simples tienen el mismo significado.

Para comentar en una sola línea #

Para comentar más de una línea `""" """` o `''' '''`

Para saltar de línea en el código ponemos `\`

Para limpiar la pantalla: (solo funciona en visual code, en el idle no)

```
import os
os.system('cls')
```

# Tipos de datos

Las variables se definen escribiendo el nombre de la variable = tipo de dato. No hace falta especificarle que tipo de dato es, el mismo programa lo reconoce. También se puede hacer una asignación múltiple que consiste en definir varias variables en la misma línea:

```

1  a, b, c = 'string', 15, True
2  print(a)
3  print(b)
4  print(c)

```

PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL

```

PS C:\Users\Gonzalo> & C:/Users/Gonzalo/
presa/segundo.py"
string
15
True
PS C:\Users\Gonzalo>

```

Int	Entero = 3
Float	Decimal = 3.12
Complejo	Complejo = 3 + 4i
Cadenas	Texto = Hola
Booleanos	Booleano = True/False
Diccionario	Almacén de datos, se pueden añadir todo tipo de datos hasta otros diccionarios, van en pares de clave-valor (ignora los valores repetidos)
Lista y tuplas	Es una secuencia de datos que se almacenan (a diferencia de los arrays pueden ser de distinto tipo) la diferencia entre lista y tupla es que la lista se puede modificar y la tupla no
Conjuntos	Colecciones desordenadas de valores de datos únicos (Pueden ser cualquier tipo de dato salvo que sean mutables y hashables como puede ser una lista, una lista no se puede añadir, pero una tupla sí. Ignora los valores repetidos). Es como el diccionario, pero sin claves. Y los valores son mutables una vez dentro.

```

1  xs = [234, 'hola', True, 7657]
2  xs[2] = 55555555
3
4  dicc = {
5      'lista': xs,
6      'edad': 23,
7      'numero': 42.3,
8      'texto': 'Prueba'
9  }
10 print(dicc)
11

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```

PS C:\Users\Gonzalo> & C:/Users/Gonzalo/AppData/Local/Programs/Python/Python311/python.exe
presa/primero.py"
['lista': [234, 'hola', 55555555, 7657], 'edad': 23, 'numero': 42.3, 'texto': 'Prueba'}
PS C:\Users\Gonzalo>

```

## Entrada y salida

Input(): Para introducir datos por teclado

Print(): Imprimir datos en la consola

Input se puede escribir como `variable = input` o como `variable = input("texto")`, la diferencia es que en la primera hay salto de línea y en la segunda no.

En el Print que lleva variables

```

1  print("¿Cómo se llama?")
2  nombre = input()
3  precio = input("Número: ")
4  print(f"Me alegro de conocerle, {nombre}, {precio}")

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```

PS C:\Users\Gonzalo> & C:/Users/Gonzalo/AppData/Local/Programs/Py
presa/segundo.py"
¿Cómo se llama?
Gonzalo
Número: 23
Me alegro de conocerle, Gonzalo, 23
PS C:\Users\Gonzalo>

```

Para concatenar variables en textos Print ponemos antes una f  
"texto {variable}"

```
x = 3
print(f"Hola número: {x}")
```

De forma predeterminada, la función `input()` convierte la entrada en una cadena, aunque escribamos un número. Si intentamos hacer operaciones, se producirá un error. Para eso tenemos que especificar el tipo aunque si es uno incompatible (Poner decimales en `Int` dará error)

```
1 cantidad = int(input("Dígame una cantidad en pesetas: "))
2 print(f"{cantidad} pesetas son {round(cantidad / 166.386, 2)} euros")
```

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
```

```
PS C:\Users\Gonzalo> & C:/Users/Gonzalo/AppData/Local/Programs/Python/Python311/python.exe -i empresa/segundo.py
Dígame una cantidad en pesetas: 800
800 pesetas son 4.81 euros
```

# Estructuras de control

## - Condicionales(If)

La estructura condicional es el If, se escribe de la siguiente forma (El "Then" de la condición son los dos puntos)

If “condición” :

## Resultado

Elif “condición”:

## Resultado

Else:

## Resultado

```
C:\ZDAW\7 sistema de gestion de empresa > segundo.py
1 compra = int(input("¿Cuanto quiere gastar? "))
2 if compra <= 100:
3     print ("Pago en efectivo")
4 elif compra > 100 and compra < 300:
5     print ("Pago con tarjeta de débito")
6 else:
7     print ("Pago con tarjeta de crédito")

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

PS C:\Users\Gonzalo> & C:/Users/Gonzalo/AppData/Local/Programs/Python/Python38-32/Python.exe C:/Users/Gonzalo/AppData/Local/Programs/Python/Python38-32/Python.exe C:\ZDAW\7 sistema de gestion de empresa/segundo.py
¿Cuanto quiere gastar? 200
Pago con tarjeta de débito
PS C:\Users\Gonzalo>
```

## - Repetitiva(While y For)

**While:** Ejecuta la misma acción mientras que alguna condición se cumpla:

```
1 anio = 2001
2 while anio <= 2012:
3     print (f"Año: {anio}")
4     anio += 1

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

PS C:\Users\Gonzalo> & C:/Users/Gonzalo/AppData/Local/Programs/Python/Python38-32/Python.exe C:/Users/Gonzalo/AppData/Local/Programs/Python/Python38-32/Python.exe C:\ZDAW\7 sistema de gestion de empresa/segundo.py
Año: 2001
Año: 2002
Año: 2003
Año: 2004
Año: 2005
Año: 2006
Año: 2007
Año: 2008
Año: 2009
Año: 2010
Año: 2011
Año: 2012
PS C:\Users\Gonzalo>
```



**For:** El bucle for se utiliza para recorrer los elementos de un objeto iterable (lista, tupla, conjunto, diccionario, ...)

For (elemento) in (iterador):

```
Print(elemento)
```

```
1 colores= ["rojo", "verde", "azul", "amarillo"]
2 for col in colores:
3     print(col)
```

PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL

```
PS C:\Users\Gonzalo> & C:/Users/Gonzalo/AppData/Local/Programs/Python/Python38-32/Python.exe C:/Users/Gonzalo/AppData/Local/Programs/Python/Python38-32/Scripts/python.exe C:/Users/Gonzalo/AppData/Local/Programs/Python/Python38-32/Scripts/presas/segundo.py
rojo
verde
azul
amarillo
PS C:\Users\Gonzalo>
```

**Iterable:** es un objeto que permite recorrer sus elementos uno a uno. Se le puede pasar la función iter().

```

1  nums = [4, 78, 9, 84]
2  it = iter(nums)
3  print(next(it))
4  print(next(it))

```

Los tipos son: List, tupla, dict, set o string

**For en los diccionarios:** En los diccionarios podemos recorrer tanto las claves como el valor, para ello tenemos que especificarlo. Si queremos las claves sería de la primera forma, y si queremos las claves y los valores de la segunda:

```
G:\> 2DAM > Sistema de gestión de empresa > segundo.py > ...  
1 valores = {'A': 4, 'E': 3, 'I': 1, 'O': 0}  
2 for k in valores:  
3     print(k)  
4  
5 for v in valores.items():  
6     print(v)
```

---

PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL

```
PS C:\Users\Gonzalo> & C:/Users/Gonzalo/AppData/Local/Programs/Python/Python39-64/Scripts/python.exe C:/Users/Gonzalo/AppData/Local/Programs/Python/Python39-64/Scripts/presas/segundo.py"  
A  
E  
I  
O  
(('A', 4))  
(('E', 3))  
(('I', 1))  
(('O', 0))
```

**For Range:** Para simular un bucle for basado en una secuencia numérica podemos utilizar el range.

```
1 for i in range(10):
2     print (i)
```

Se pueden poner distintos parámetros:

Range(Max)

Range(Min, Max)

Range(Min, Max, Step)

**Break y continue:** Termina el bucle en caso de que se cumpla una condición

Así parara cuando llegue al 7 imprimiendo solo el 2, 4 y 5

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

Solo imprimirá los números pares

```
coleccion = [1, 2, 3, 4, 5, 6, 7, 8]
for e in coleccion:
    if e % 2 != 0:
        continue
    print(e)
```

For... Else: es una estructura for que se ejecuta siempre y cuando no se haya ejecutado la sentencia break dentro del for. Si encuentra el número 3 entonces terminará en el break y saldrá sin imprimir nada, en el caso de que no lo encuentre, imprimirá que no ha encontrado el número 3.

```
numeros = [1, 2, 4, 5, 8, 6]
for n in numeros:
    if n == 3:
        break
else:
    print('No se encontró el número 3')
```

## Cadenas

Las cadenas en Python son listas de caracteres que pueden almacenarse en la memoria del ordenador.

Se pueden buscar caracteres

```
1 texto = "Hola"
2 print(texto[0])
```

```
PROBLEMS OUTPUT DEBUG CONSOLE
PS C:\Users\Gonzalo> & C:/Users/
presa/segundo.py"
H
PS C:\Users\Gonzalo>
```

```
1 texto = "Hola"
2 print(texto[0:2])
```

```
PROBLEMS OUTPUT DEBUG CONSOLE
PS C:\Users\Gonzalo> & C:/Users/
presa/segundo.py"
Ho
PS C:\Users\Gonzalo>
```

Si queremos buscar de derecha a izquierda lo hacemos con números negativos

```
1 texto = "Hola"
2 print(texto[-2])
```

PROBLEMS OUTPUT DEBUG CONSOLE

PS C:\Users\Gonzalo> & C:/Users/Gonzalo/presa/segundo.py

l

PS C:\Users\Gonzalo>

Para sacar una cadena que no sea continua se pone doble punto ::

```
1 texto = "Buenas tardes"
2 print(texto[0:2])
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

PS C:\Users\Gonzalo> & C:/Users/Gonzalo/presa/segundo.py

Bea ads

PS C:\Users\Gonzalo>

**Método In:**

```
texto = 'buenas tardes'
if 'x' in texto:
    print('True')
else:
    print('False')
```

# Listas

Es una secuencia de datos que pueden ser diferentes que se almacenan, se pueden utilizar para recopilar y organizar información.

Las listas son ordenadas, editables y pueden tener elementos duplicados

```
lista = [23, 5, 6, 12, 67]
print (lista)
```

Se pueden meter bibliotecas en listas

```
serpList = {"Boa": "Serpiente Constrictora",
            "Pitón": "Serpiente Constrictora",
            "Cobra": "Culebra venenosa",
            "Víbora Rayada": "Culebra Inofensiva"}
lista = (serpList)
print (lista)
```

## Funciones:

Función Len para contar el número de elementos

```
lista = [23, 5, 6, 12, 67]
print (len(lista))
```

Función Type para que indique que tipo es

```
1
2 lista = [23, 5, 6, 12, 67]
3 x = "hola"
4 print (type(lista))
5 print (type(x))
6
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
PS C:\Users\Gonzalo> & C:/Users/Gonzalo/AppData/Local/Programs/Python/Python39-64/Scripts/python.exe C:/Users/Gonzalo/AppData/Local/Programs/Python/Python39-64/Scripts/presa/segundo.py
<class 'list'>
<class 'str'>
PS C:\Users\Gonzalo>
```

Símbolo de concatenar es el + o , la coma deja un espacio entre las cadenas y el + no (tener cuidado con los tipos, solo con string los +)

```
serpList = ["Boa", "Pitón", "Víbora Rayada"]  
print(serpList + ["182 centímetros", "121 centímetros", "30 centímetros"])
```

Mostrar rango: [num:num]

```
serpList = ["Boa", "Pitón", "Víbora Rayada"]  
print(serpList[0:2])
```

Repetir valores de la lista: \*3

```
serpList = ["Boa", "Pitón", "Víbora Rayada"]  
print(serpList*3)
```

Añadir elementos: .append

```
serpList = ["Boa", "Pitón", "Víbora Rayada"]  
serpList.append("Cobra")  
print (serpList)
```

Buscar índice de elemento: .index

```
serpList = ["Boa", "Pitón", "Víbora Rayada"]  
serpList.append("Cobra")  
print (serpList.index("Cobra"))
```

Eliminar de la lista: .remove

```
serpList = ["Boa", "Pitón", "Víbora Rayada", "Cobra"]  
serpList.remove("Cobra")  
print (serpList)
```

Ordenar: .sort

```
a = [13,5,7,2,1]
a.sort()
print (a)
```

[1, 2, 5, 7, 13]

.clear(): elimina todos los elementos de la lista.

.copy(): arroja una copia de la lista.

.count(): arroja el número de elementos con el valor indicado.

.extend(): añade los elementos de una lista (o cualquier iterador) al final de la lista actual.

.insert(): añade un elemento en la posición que se indica.

.pop(): elimina el elemento de la posición que se indica.

.reverse(): invierte el orden de la lista.

## Tuplas

Las tuplas son como las listas solo que no dejan editar sus elementos y una vez se ha cargado de contenido, permanece inalterable y no puede ser eliminada. Se pueden hacer con cualquier tipo de dato.

```
tuplas = (12, 23, 56, 23, 78)
print (tuplas)
```

Y se pueden utilizar funciones como len(), range

```
tuplas = (12, 23, 56, 23, 78)
print (len(tuplas))
print (tuplas[1:3])
```

Se puede modificar una tupla creando otra nueva y añadiéndole datos nuevos mediante la concatenación: +

```
algunos_animales = ("perro", "gato", "ratón", "serpiente", "caballo")
todos_los_animales = algunos_animales + ("hámster", "jirafa")
print(todos_los_animales)
```

Pasar una lista a tupla con el método tuple

```
1 lista_colores = ["azul", "rojo", "amarillo", "naranja"]
2 colores = tuple(lista_colores)
3 print(colores)
4 print(type(colores))
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
PS C:\Users\Gonzalo> & C:/Users/Gonzalo/AppData/Local/Programs/Python/Python39-64/pres/segundo.py
('azul', 'rojo', 'amarillo', 'naranja')
<class 'tuple'>
PS C:\Users\Gonzalo>
```

Y para pasar una tupla a lista se haría de la misma forma pero con el método list

```
1 animales = ("perro", "gato", "ratón", "serpiente", "caballo")
2 lista_animales = list(animales)
3 print(type(lista_animales))
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
PS C:\Users\Gonzalo> & C:/Users/Gonzalo/AppData/Local/Programs/Python/Python39-64/pres/segundo.py
<class 'list'>
PS C:\Users\Gonzalo>
```

Se puede eliminar una tupla entera con el método del

```
animales = ("perro", "gato", "ratón", "serpiente", "caballo")
del animales
```



## Función .index y count

La función index te dice en que posición esta el elemento y count cuenta cuantas veces aparece ese elemento en la lista

```

1  animales = ["perro", "gato", "ratón", "serpiente", "caballo", "gato"]
2  print(animales.index("gato"))
3  print(animales.count("gato"))
4

```

PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL

```

PS C:\Users\Gonzalo> & C:/Users/Gonzalo/AppData/Local/Programs/Python/Python311/python.exe -i C:/Users/Gonzalo/AppData/Local/Programs/Python/Python311/pres/segundo.py
1
2
PS C:\Users\Gonzalo>

```

# Conjuntos

Colecciones **desordenadas** de valores de datos únicos (Pueden ser cualquier tipo de dato salvo que sean mutables y hashables como puede ser una lista, una lista no se puede añadir, pero una tupla sí. Ignora los valores repetidos). Es como el diccionario, pero sin claves. Y los valores son mutables una vez dentro.

```

x = {3.14, False, "Hola mundo", 6}

```

También se puede crear un conjunto de esta forma:

```

1
2  x = set(range(10))
3  print(x, type(x))
4

```

PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL

```

PS C:\Users\Gonzalo> & C:/Users/Gonzalo/AppData/Local/Programs/Python/Python311/python.exe -i C:/Users/Gonzalo/AppData/Local/Programs/Python/Python311/pres/segundo.py
{0, 1, 2, 3, 4, 5, 6, 7, 8, 9} <class 'set'>
PS C:\Users\Gonzalo>

```

Al igual que en otros tipos de datos, tenemos funciones como: `.add()` para añadir, `.discard()` para eliminar un elemento, `.clear()` para eliminar todos los elementos del conjunto y `len()` para saber el número de elementos

```

1  x = {3, 87, 6, 0, 23}
2  x.add(5)
3  x.discard(3)
4  print(len(x))
5  print (x)
6  x.clear()
7  print(x)
8

```

PROBLEMS OUTPUT DEBUG CONSOLE

PS C:\Users\Gonzalo> & C:/Users/Gonzalo/Programa/segundo.py  
5  
{0, 23, 5, 6, 87}  
set()

Se pueden hacer uniones de dos conjuntos( `|` )(une los dos conjuntos excluyendo los duplicados), intersecciones( `&` )(devuelve los que estan en ambos conjuntos), y restas( `-` )(elementos de a que no están en b)

```

1  a = {1, 2, 3, 4}
2  b = {3, 4, 5, 6}
3  print(a | b)

```

PROBLEMS OUTPUT DEBUG CONSOLE

PS C:\Users\Gonzalo> & C:/Users/Gonzalo/Programa/segundo.py  
{1, 2, 3, 4, 5, 6}

```

1  a = {1, 2, 3, 4}
2  b = {3, 4, 5, 6}
3  print(a & b)

```

PROBLEMS OUTPUT DEBUG CONSOLE

PS C:\Users\Gonzalo> & C:/Users/Gonzalo/Programa/segundo.py  
{3, 4}

```

1  a = {1, 2, 3, 4}
2  b = {3, 4, 5, 6}
3  print(a - b)

```

PROBLEMS OUTPUT DEBUG CONSOLE

PS C:\Users\Gonzalo> & C:/Users/Gonzalo/Programa/segundo.py  
{1, 2}

Comprobar si B es un subconjunto de A (todos los elementos de B están en A) devuelve true si es verdad

```

1  a = {1, 2, 3, 4}
2  b = {2, 3}
3  print(b.issubset(a))

```

PROBLEMS OUTPUT DEBUG CONSOLE

PS C:\Users\Gonzalo> & C:/Users/Gonzalo/Programa/segundo.py  
True  
PS C:\Users\Gonzalo>



También se pueden poner valores definidos en los parámetros por si no se proporcionan ninguno

```

1  def sumar(a= 2, b= 2):
2      print(a+b)
3  sumar()
4  sumar(3, 3)

```

PROBLEMS OUTPUT DEBUG CONSOLE

```

PS C:\Users\Gonzalo> & C:/Users/Gonzalo/presa/segundo.py
4
6
PS C:\Users\Gonzalo>

```

## Funciones incorporadas en Python:

**Función min:** Devuelve el elemento más pequeño en un iterable o el más pequeño en dos elementos

```

1  x = (2, 45, 5, 6, 9, 1, 3)
2  print(min(x))
3

```

PROBLEMS OUTPUT DEBUG CONSOLE TEST

```

PS C:\Users\Gonzalo> & C:/Users/Gonzalo/presa/segundo.py
1
PS C:\Users\Gonzalo>

```

**Función max:** Igual que la función max, pero devolviendo el valor máximo de la iterable o de dos elementos

```

1  x = (2, 45, 5, 6, 9, 1, 3)
2  print(max(x))
3

```

PROBLEMS OUTPUT DEBUG CONSOLE TEST

```

PS C:\Users\Gonzalo> & C:/Users/Gonzalo/presa/segundo.py
45
PS C:\Users\Gonzalo>

```

**Función divmod:** Devuelve el cociente y el resto al dividir el número a por el número b, el argumento no puede ser un número complejo

(cociente, resto)

```
1  a = 45
2  b = 6
3  print(divmod(a, b))
```

PROBLEMS OUTPUT DEBUG CON

```
PS C:\Users\Gonzalo> & C:/U
2DAM/Sistema de gestión de
(7, 3)
```

**Función Hex(x):** Convierte un número entero en una cadena hexadecimal en minúsculas con el prefijo "0x"

```
1  print(hex(543))
```

PROBLEMS OUTPUT DEBUG

```
PS C:\Users\Gonzalo> &
2DAM/Sistema de gestión
0x21f
PS C:\Users\Gonzalo>
```

**Función len:** devuelve la longitud de tipos de dato como cadenas, conjuntos, listas o tuplas, no existe la función len para números o booleanos

```
print(len("hola buenas tardes"))
```

**Función Chr:** Convierte el número entero que representa en el código Unicode en una cadena que representa un carácter:

```
1 print(chr(49))
2
PROBLEMS OUTPUT DEBUG
PS C:\Users\Gonzalo> & C:\Program Files\Python39\python.exe C:\Users\Gonzalo\AppData\Local\Temp\1\2DAM/Sistema de gestión
1
```

## Funciones Lambda

Son funciones que no están nombradas, también son llamadas anónimas. Se declaran igual con la palabra clave def. Sirven cuando queremos definir una función pequeña de forma concisa.

```
#función lambda
cuadrado = lambda x: x**2
print(cuadrado(3))
#función normal
def cuad(num):
    return num**2
print(cuad(3))
```

# Clases y objetos

## Creación de clases:

Para crear una clase usaremos la palabra clave class, el nombre de la clase y debajo la declaración pass

```
class Galleta:  
    pass
```

## Creación de objetos:

Para crear un objeto de esta clase tenemos que escribir el nombre del objeto y el tipo(la clase). Si lo imprimimos nos dirá de qué clase es

```
galleta = Galleta()
```

## Atributos:

Se pueden hacer atributos de la clase:

```
class Galleta:  
    chocolate = False  
    pass
```

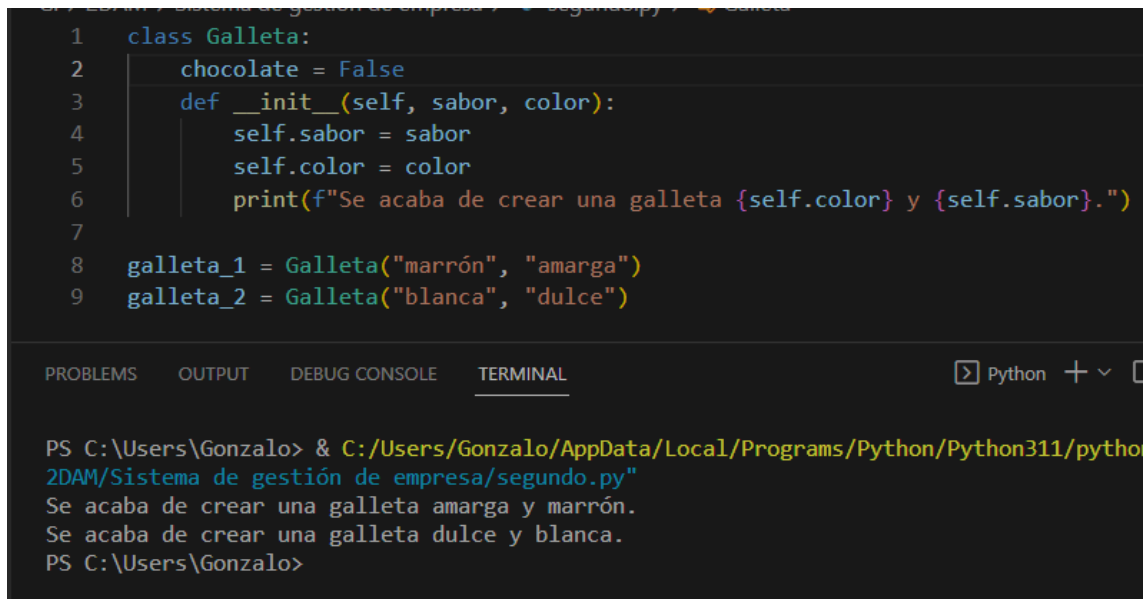
O en el objeto:

```
galleta = Galleta()  
#atributos en el objeto  
galleta.sabor = "salado"  
galleta.color = "marrón"
```





**Constructor:** Es un método que se llama automáticamente al crear un objeto y se define con el nombre `__init__`. Su finalidad es la de construir objetos por eso permite sobre escribir el método que crea los objetos permitiendo enviar datos desde el principio para construirlo



```

1 class Galleta:
2     chocolate = False
3     def __init__(self, sabor, color):
4         self.sabor = sabor
5         self.color = color
6         print(f"Se acaba de crear una galleta {self.color} y {self.sabor}.")
7
8 galleta_1 = Galleta("marrón", "amarga")
9 galleta_2 = Galleta("blanca", "dulce")

```

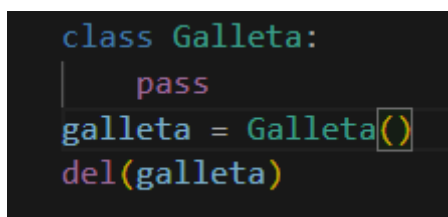
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL Python + -

```

PS C:\Users\Gonzalo> & C:/Users/Gonzalo/AppData/Local/Programs/Python/Python311/python
2DAM/Sistema de gestión de empresa/segundo.py
Se acaba de crear una galleta amarga y marrón.
Se acaba de crear una galleta dulce y blanca.
PS C:\Users\Gonzalo>

```

**Destructor:** Sirve para eliminar el objeto y que limpie la memoria, cuando se cierra el programa todos los objetos se borran, pero también está esta función:



```

class Galleta:
    pass
galleta = Galleta()
del(galleta)

```

# Ficheros

El objeto file representa los ficheros del sistema operativo.

## Función open()

La estructura es open(ruta, modo)

Los modos

r	Solo lectura. El fichero solo se puede leer. Es el modo por defecto si no se indica.
w	Solo escritura. En el fichero solo se puede escribir. Si ya existe el fichero, machaca su contenido.
a	Adición. En el fichero solo se puede escribir. Si ya existe el fichero, todo lo que se escriba se añadirá al final del mismo.
x	Como 'w' pero si existe el fichero lanza una excepción.
r+	Lectura y escritura. El fichero se puede leer y escribir.

La función open sirve para abrir el fichero y una vez abierto poder escribir o leerlo

## Escribir:

Para escribir en un fichero usamos la función open(ruta, modo)

de tal forma

```
#Escribir en un fichero
with open("example.txt", "w") as file:
    # escribe texto en el archivo
    file.write("Hello, world!")
    file.write("Hello, world2!")
```

Así se nos creará un archivo en el que guardará el texto, si esta creado solo lo añadirá

## Leer:

Para Leer un archivo tenemos que escribir la función `read()` o `readline()`. La función `read` lee todo el archivo y lo imprime por la consola, el `readline` solo lee la línea especificada

```
#Leer una línea del archivo(readline)
with open("example.txt", "r") as file:
    contents = file.readline()
    print(contents)

#Lee todo el archivo (read)
with open("example.txt", "r") as file:
    contents = file.read()
    print(contents)
```

```
PS C:\Users\Equipo>
Hello, world!

Hello, world!
Hello, world2!
PS C:\Users\Equipo>
```

## Buscar:

En el caso de que queramos buscar la localización de un archivo tendremos que importar la librería `os` y llamar al método `getcwd()`

```
#Buscar la ruta del archivo
import os
with open("example.txt", "r") as file:
    print(os.getcwd())
```

```
PS C:\Users\Equipo>
C:\Users\Equipo
PS C:\Users\Equipo>
```

Si no especificamos una ruta concreta siempre nos lo guardara en el Usuario que estemos usando.