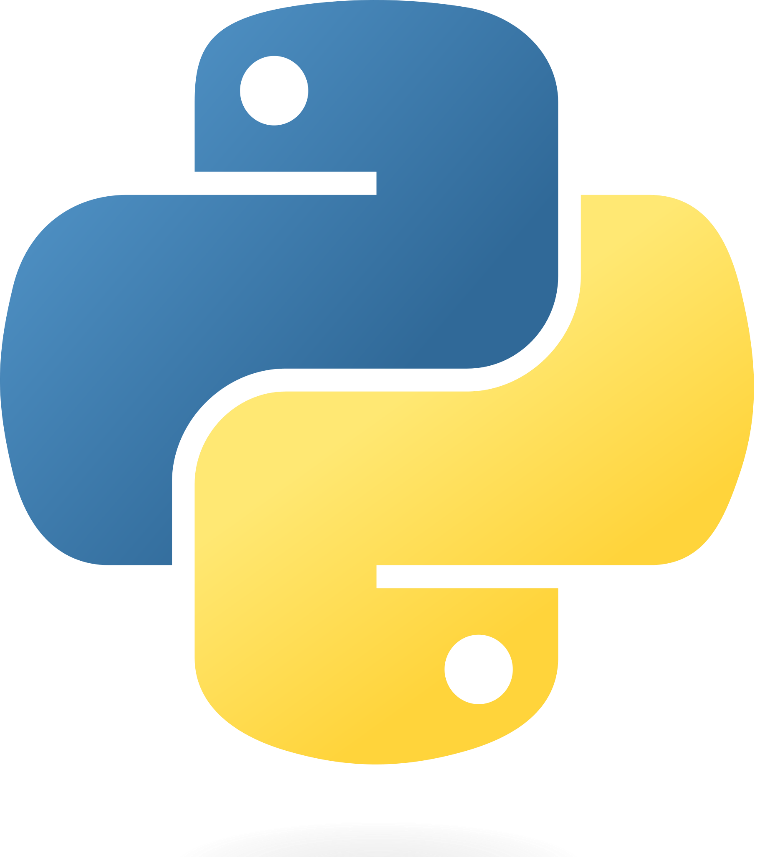
Manual de Python



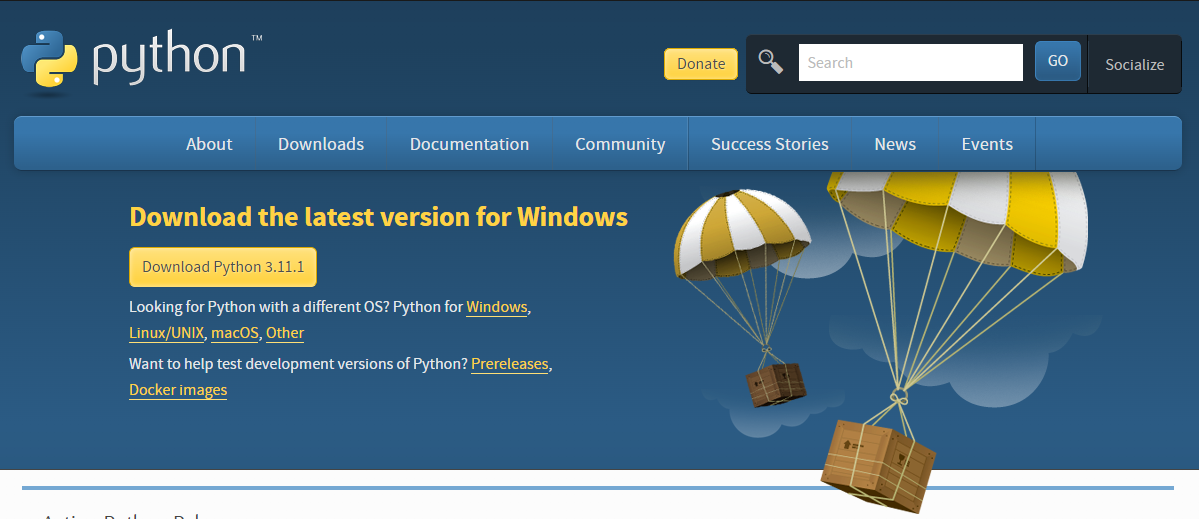
Gonzalo Pascual Romero

Instalación

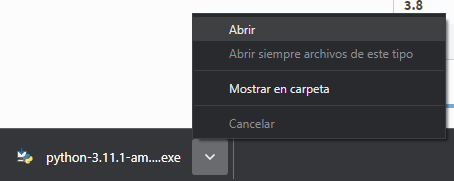
Para instalar Python primero debemos ir al siguiente enlace:

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

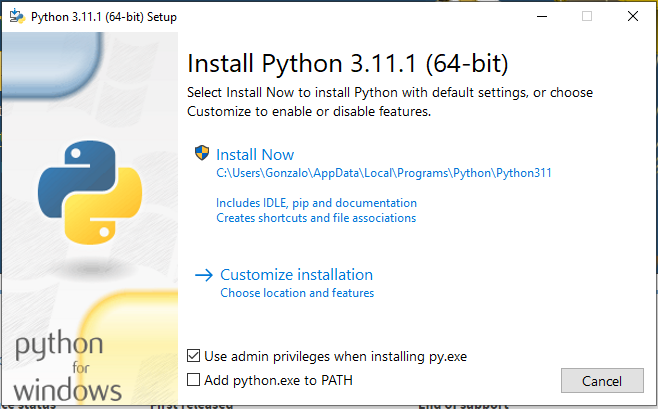
y darle a descargar la versión mas reciente, en este caso descargaremos la versión 3.11.1



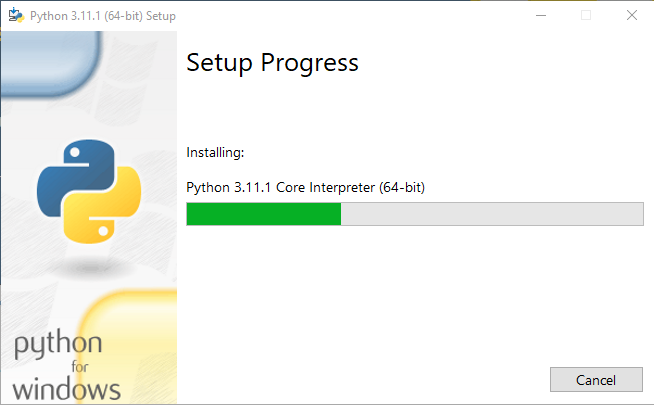
Una vez pulsado nos saldrá en la barra inferior cuanto le falta para descargarse y una vez descargado haremos Click sobre la flecha y pulsaremos abrir



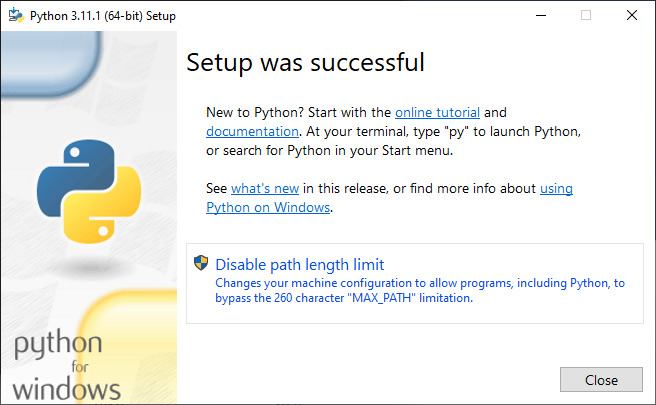
Comenzará la instalación en la que seleccionamos “Install Now” con las características predefinidas



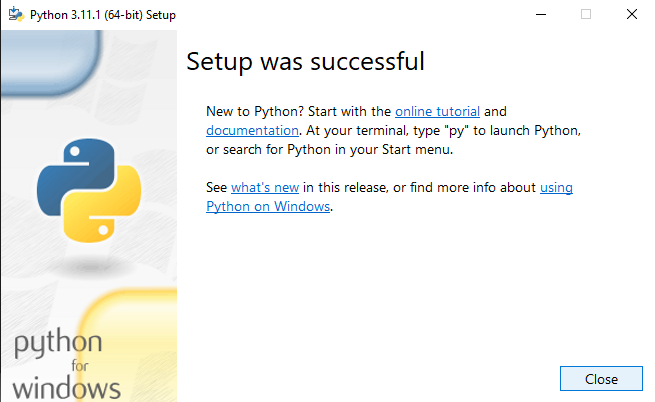
Y se empezara a instalar



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

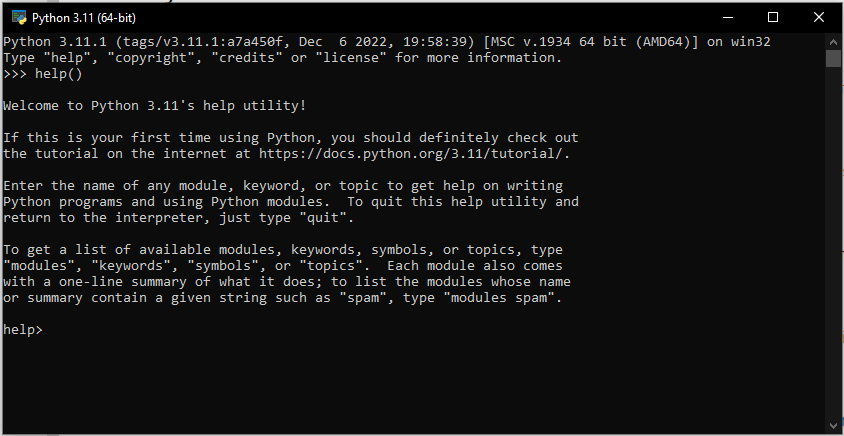


Y por último finalizaremos la instalación dándole a “Close”

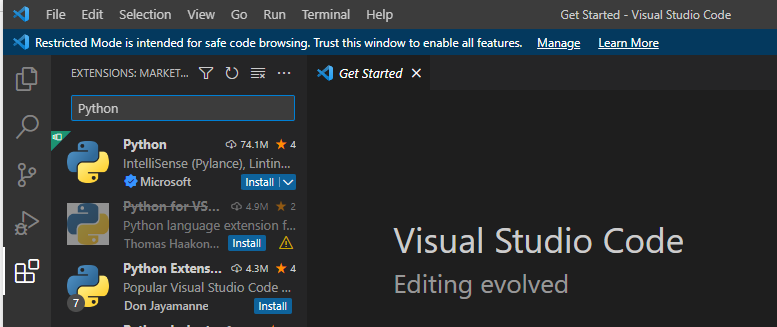


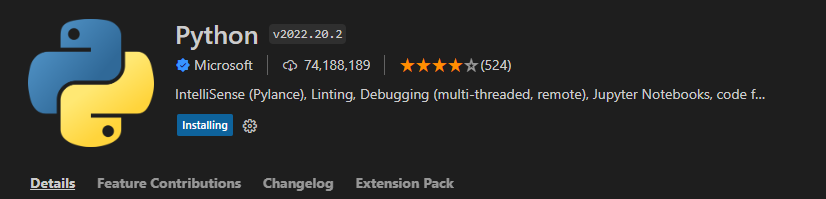
Para usarlo se puede abrir desde su terminal o como yo lo voy a hacer a través de visual studio code.

Terminal:

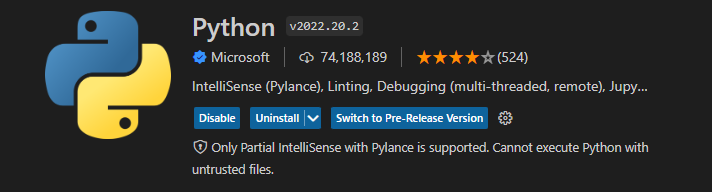


Para que reconozca Python en Visual Studio Code, tenemos que añadirle la extensión de Python. Por lo que entraremos en la aplicación, iremos a extensiones, buscaremos Python e instalaremos la primera que sea de Microsoft

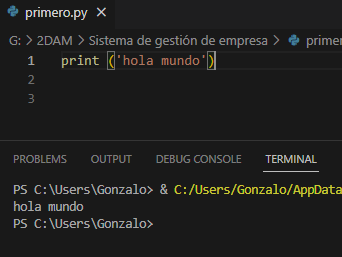




Y ya la tendríamos instalada.



Creamos un proyecto de Python y ya podríamos programar

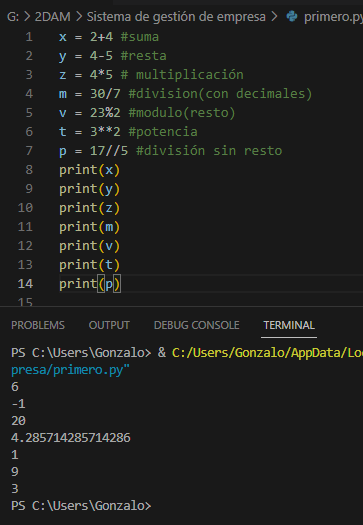


Operadores

* Aritméticos

Los operadores aritméticos que tenemos en Python son los siguientes:

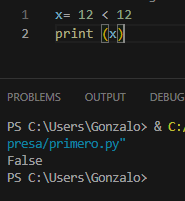
|  |  |  |
| --- | --- | --- |
| + | Suma | 12 + 3 = 15 |
| - | Resta | 12 - 3 = 9 |
| \* | Multiplicación | 12 \* 3 = 36 |
| / | División (con decimales) | 12 / 3 = 4 |
| % | Modulo (resto operación) | 16 % 3 = 1 |
| \*\* | Potencia = 12^3 | 12 \*\* 3 = 1728 |
| // | División sin decimales (entero)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



* Relacionales

Los operadores relacionales que tenemos en Python son los siguientes:

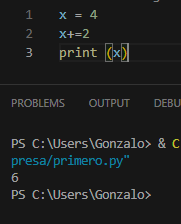
|  |  |  |
| --- | --- | --- |
| > | Mayor que | 12 > 3 |
| < | Menor que | 12 < 3 |
| == | Igual que | 12 == 3 |
| >= | Mayor o igual que | 12 >= 3 |
| <= | Menor o igual que | 12 <= 3 |
| != | Diferente a | 12 !=3 |



* Asignación

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

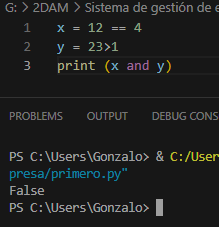
|  |  |
| --- | --- |
| = | x = 5. El valor 5 es asignado a la variable a |
| += | x += 5 es equivalente a x = x + 5 |
| -= | x -= 5 es equivalente a x = x - 5 |
| \*= | x \*= 3 es equivalente a x = x \* 3 |
| /= | x /= 3 es equivalente a x = x / 3 |
| %= | x %= 3 es equivalente a x = x % 3 |
| \*\*= | 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 |



* Lógicos

Los operadores lógicos que tenemos en Python son los siguientes:

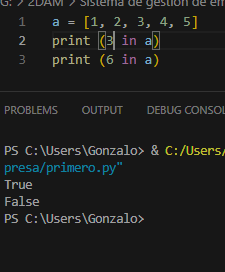
|  |  |  |
| --- | --- | --- |
| and | Devuelve True si ambos operandos son True | a and b |
| or | Devuelve True si alguno de los operandos es True | a or b |
| not | Devuelve True si alguno de los operandos False | not a |



* Pertenencia

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

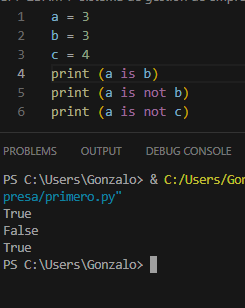
|  |  |
| --- | --- |
| In | Devuelve True si se encuentra en la secuencia |
| Not in | Devuelve True si no se encuentra en la secuencia |



* 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 |



* 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 /

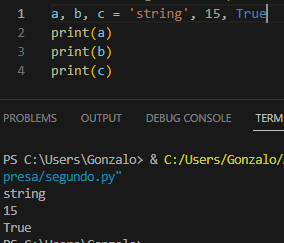
Para limpiar la pantalla: (solo funciona en visual code, en el idle no)

import os

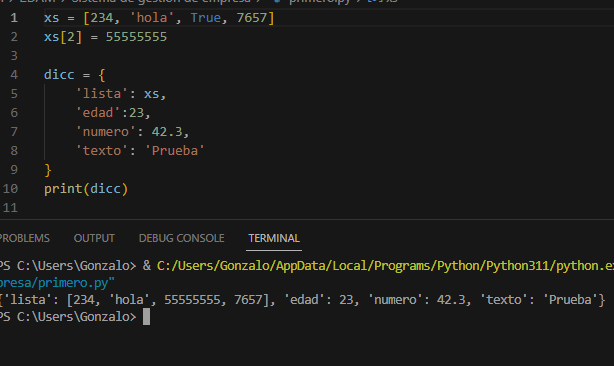
os.system('cls')

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:



|  |  |
| --- | --- |
| 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. |



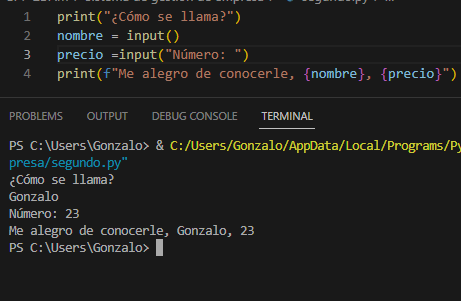
Entrada y salida

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.

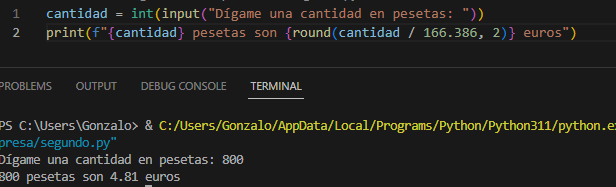
En el Print que lleva variables



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



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)



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” :

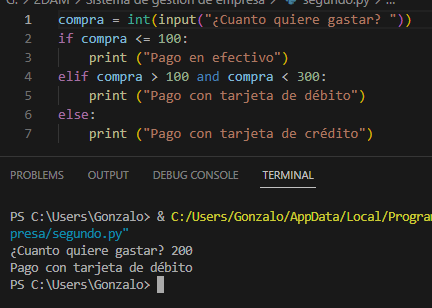
Resultado

Elif “condición”:

Resultado

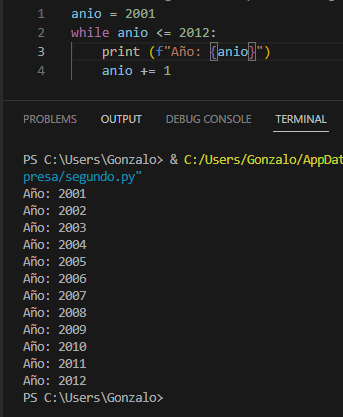
Else:

Resultado



* Repetitiva(While y For)

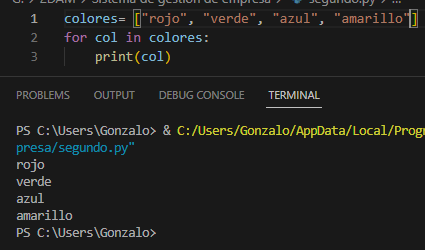
**While:** Ejecuta la misma acción mientras que alguna condición se cumpla:



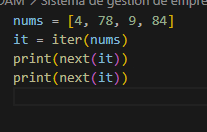
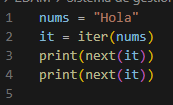
**For:** El bucle for se utiliza para recorrer los elementos de un objeto iterable (lista, tupla, conjunto, diccionario, …)

For (elemento) in (iterador):

Print(elemento)

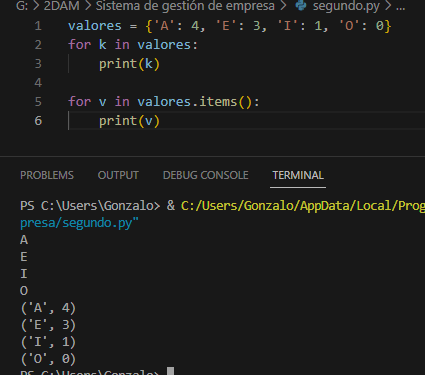


**Iterable:** es un objeto que permite recorrer sus elementos uno a uno. Se le puede pasar la función iter().

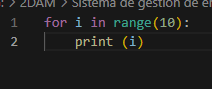
 

Los tipos son: List, tupla, dict, set o string

**For en los diccionarios:** En los diccionarios podemos recorrer tantos 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:



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



Se pueden poner distintos parámetros:

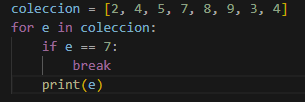
Range(Max)

Range(Min, Max)

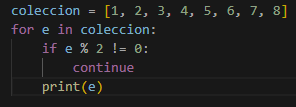
Range(Min, Max, Step)

**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

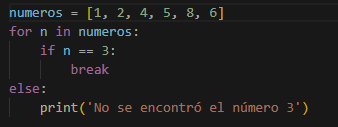


Solo imprimirá los números pares



For… Else: es una estructura for que se ejecuta siempre y cuando no se haya ejecutado la sentencia break dentro del for

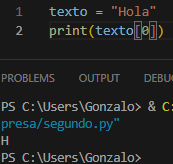
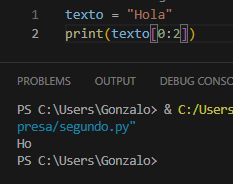
Si encuentra el número 3 entonces terminara en el break y saldrá sin imprimir nada, en el caso de que no lo encuentre, imprimirá que no ha encontrado el número 3



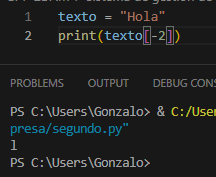
Cadenas

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

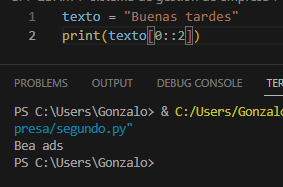
Se pueden buscar caracteres

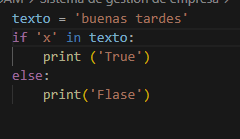
Si queremos buscar de derecha a izquierda lo hacemos con números negativos



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



**Método In:**



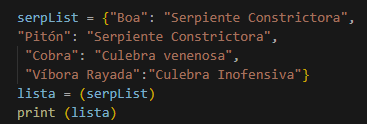
Listas

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



Se pueden meter bibliotecas en listas

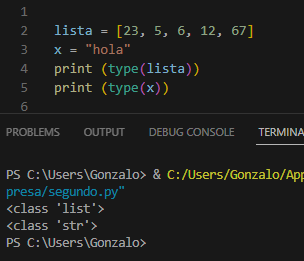


**Funciones:**

Función Len para contar el número de elementos



Función Type para que indique que tipo es



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 +)



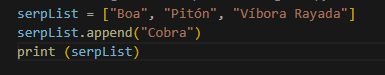
Mostrar rango: [num:num]



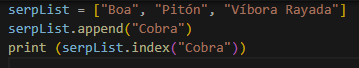
Repetir valores de la lista: \*3



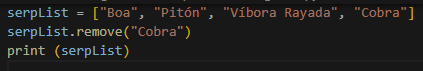
Añadir elementos: .append



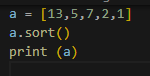
Buscar índice de elemento: .index



Eliminar de la lista: .remove



Ordenar: .sort

.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.

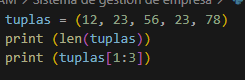
.reverse(): invierte el orden de la lista.

Tuplas

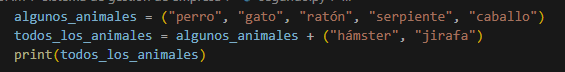
Las tuplas son como las listas solo que no dejan editar sus elementos y una vez se ha cargado de contenido, permanece inalterable y no puede ser eliminada. Se pueden hacer con cualquier tipo de dato.



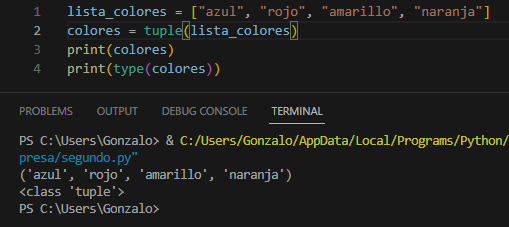
Y se pueden utilizar funciones como len(), range



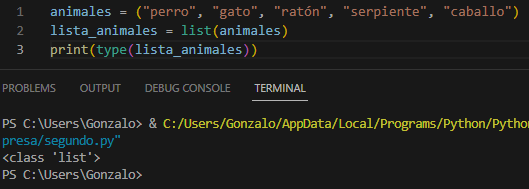
Se puede modificar una tupla creando otra nueva y añadiéndole datos nuevos mediante la concatenación: +



Pasar una lista a tupla con el método tuple



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

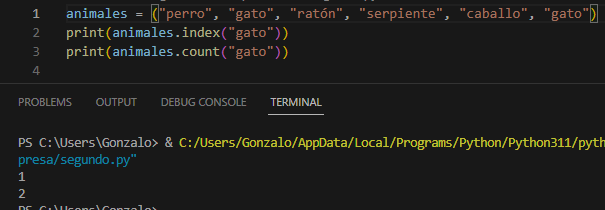


Se puede eliminar una tupla entera con el método del



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

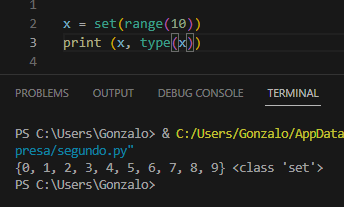


Conjuntos

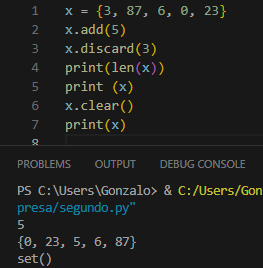
Colecciones **desordenadas** de valores de datos únicos (Pueden ser cualquier tipo de dato salvo que sean mutables y hashables como puede ser una lista, una lista no se puede añadir, pero una tupla sí. Ignora los valores repetidos). Es como el diccionario, pero sin claves. Y los valores son mutables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



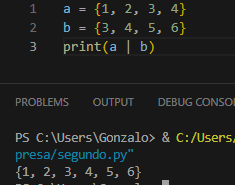
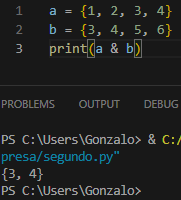
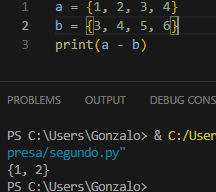
También se puede crear un conjunto de esta forma:



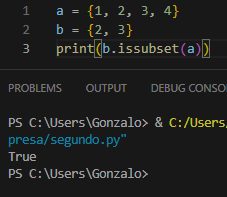
Al igual que en otros tipos de datos, tenemos funciones como: .add() para añadir, .discard() para elimar un elemento, .clear() para eliminar todos los elementos del conjunto y len() para saber el número de elementos



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)

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

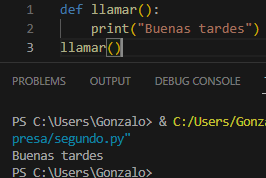


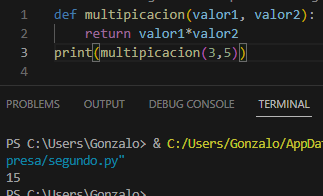
Funciones

Es un bloque de código reutilizable que se puede utilizar muchas veces. Se define con la palabra clave Def a continuación el nombre, los paréntesis con los parámetros de entrada (opcionales), dos puntos, la función y la sentencia de retorno, aunque no es obligatoria.

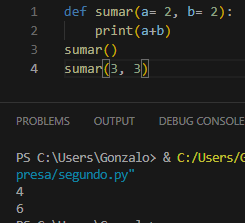
Para llamar a una función escribimos el nombre de la función con paréntesis y los parámetros si son necesarios.

Nota: si la función tiene parámetros se deben proporcionar la misma cantidad de parámetros que hay, salvo que haya datos predefinidos.



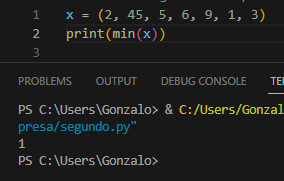


También se pueden poner valores definidos en los parámetros por si no se proporcionan ninguno

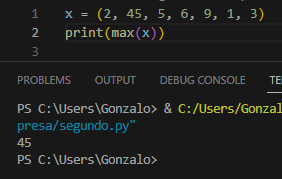


**Funciones incorporadas en Python:**

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

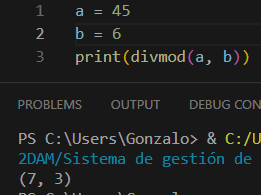


**Función max:** Igual que la función max, pero devolviendo el valor máximo de la iterable o de dos elementos

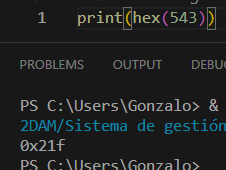


**Función divmod:** Devuelve el cociente y el resto al dividir el numero a por el número b, el argumento no puede ser un número complejo

(cociente, resto)



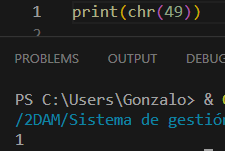
Función Hex(x): Convierte un número entero en una cadena hexadecimal en minúsculas con el prefijo “Ox”



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

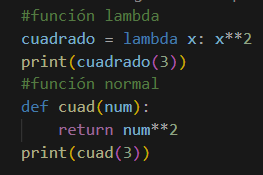


**Función Chr:** Convierte el número entero que representa en el código Unicode en una cadena que representa un carácter:



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.



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



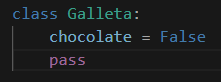
**Creación de objetos:**

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

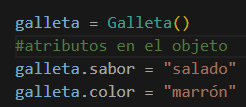


**Atributos:**

Se pueden hacer atributos de la clase:

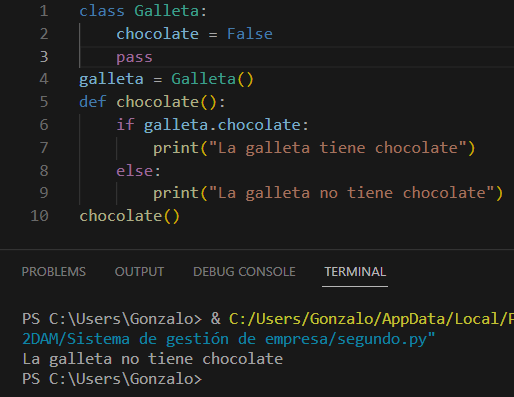


O en el objeto:

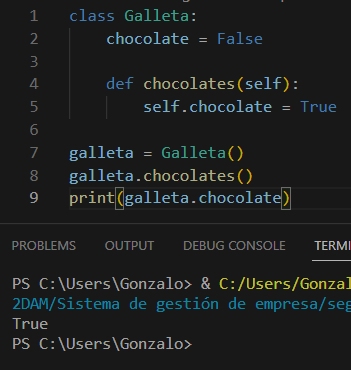


**Métodos:**

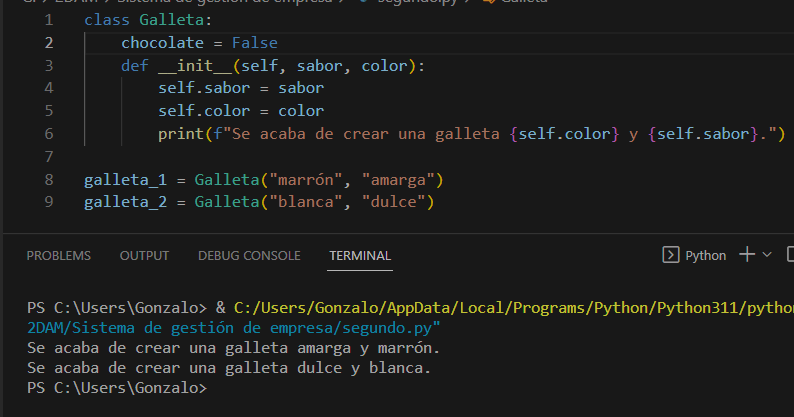
Se hacen igual que las funciones de antes

****

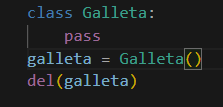
**Self:** Nos permite acceder al propio objeto y a sus atributos desde un método



**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

****

**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:



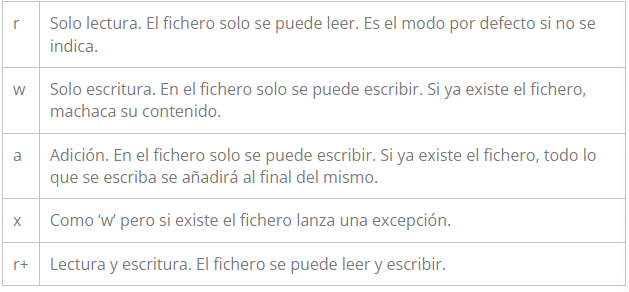
Ficheros

El objeto file representa los ficheros del sistema operativo.

**Función open()**

La estructura es open(ruta, modo)

Los modos

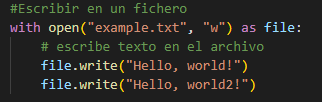
****

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)

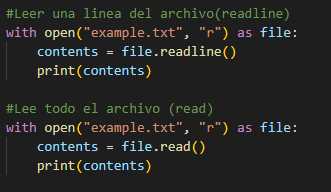
de tal forma

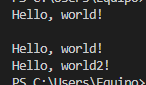


Así se nos creará un archivo en el que guardará el texto, si esta creado solo lo añadirá

**Leer:**

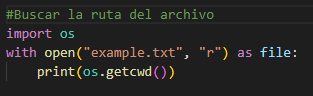
Para Leer un archivo tenemos que escribir la función read() o readline(). La función read lee todo el archivo y lo imprime por la consola, el readline solo lee la línea especificada





**Buscar:**

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





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