Clase 15 - Scripts, Módulos y Paquetes

# **Scripts**

**¿Qué son?**

¿Qué es un script? Un script es un “guión” con instrucciones de código, (básicamente, lo que venimos haciendo hasta ahora) guardado con un nombre y ejecutado desde el intérprete (ide), estos scripts pueden tomar datos (argumentos) en el momento de la ejecución.

Pueden tomar estos datos desde el exterior y tener distintos comportamientos.

**IDE**

Un entorno de desarrollo integrado (IDE) es un sistema de software para el diseño de aplicaciones que combina herramientas comunes para desarrolladores en una sola interfaz de usuario gráfica (GUI).

**IDE**

Cualquier IDE debe tener una serie de características básicas que garanticen que la experiencia del usuario será satisfactoria. Todo IDE debe contar con:

* Editor de código.
* Compilador.
* Depurador o debugger.
* Refactorización de código.

**¿Y qué IDEs existen?**

****

**Descarguemos VSC**

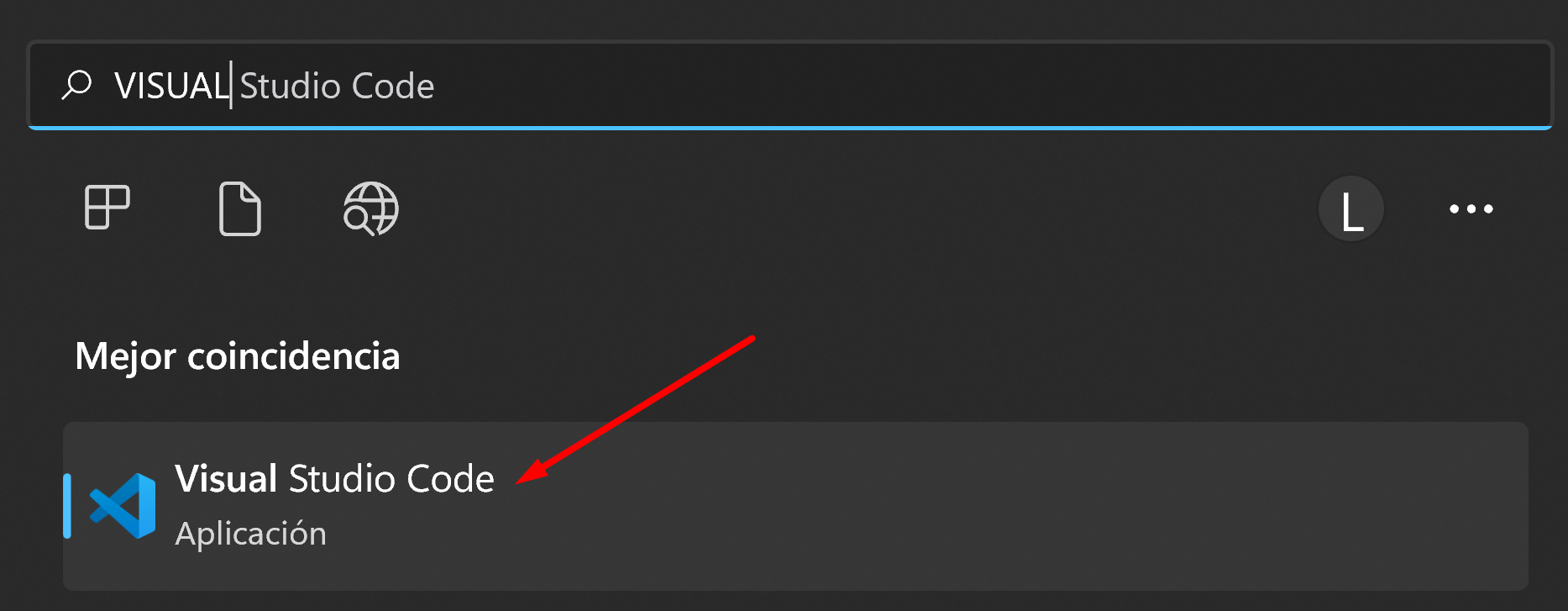
* <https://code.visualstudio.com/download>

Deberemos seleccionar el S.O que tengamos y realizar su correspondiente Instalación.

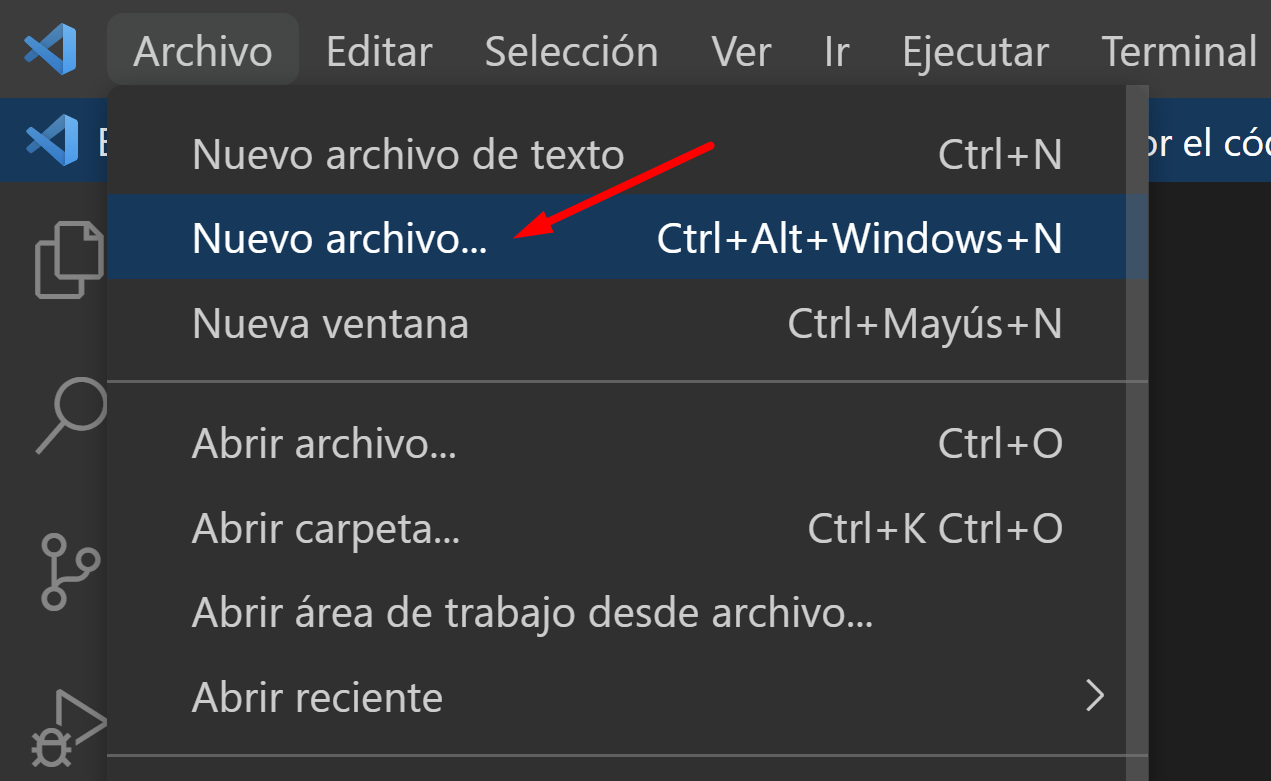
**Empecemos con VSC**

A continuación crearemos nuestro primer script en Python. Para ello, realizaremos los siguientes pasos:

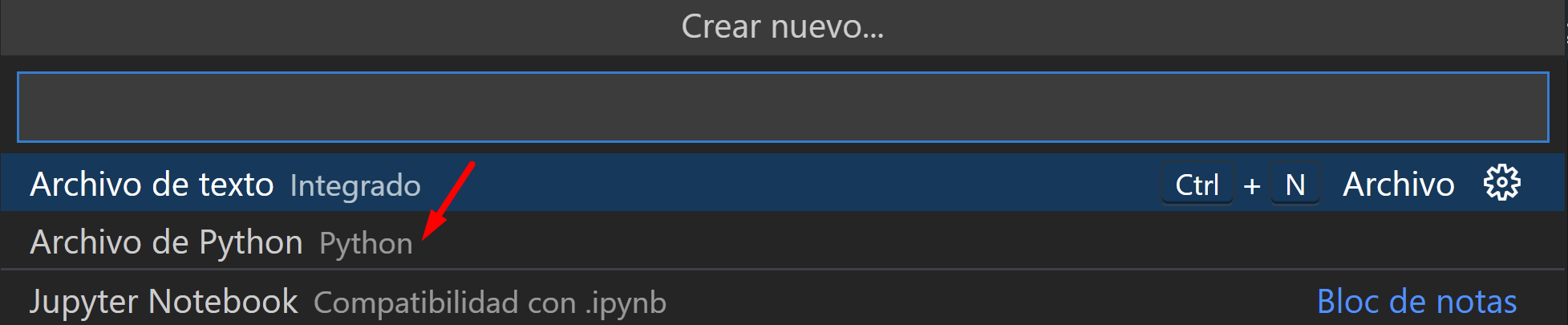
1. Abrimos VSC.



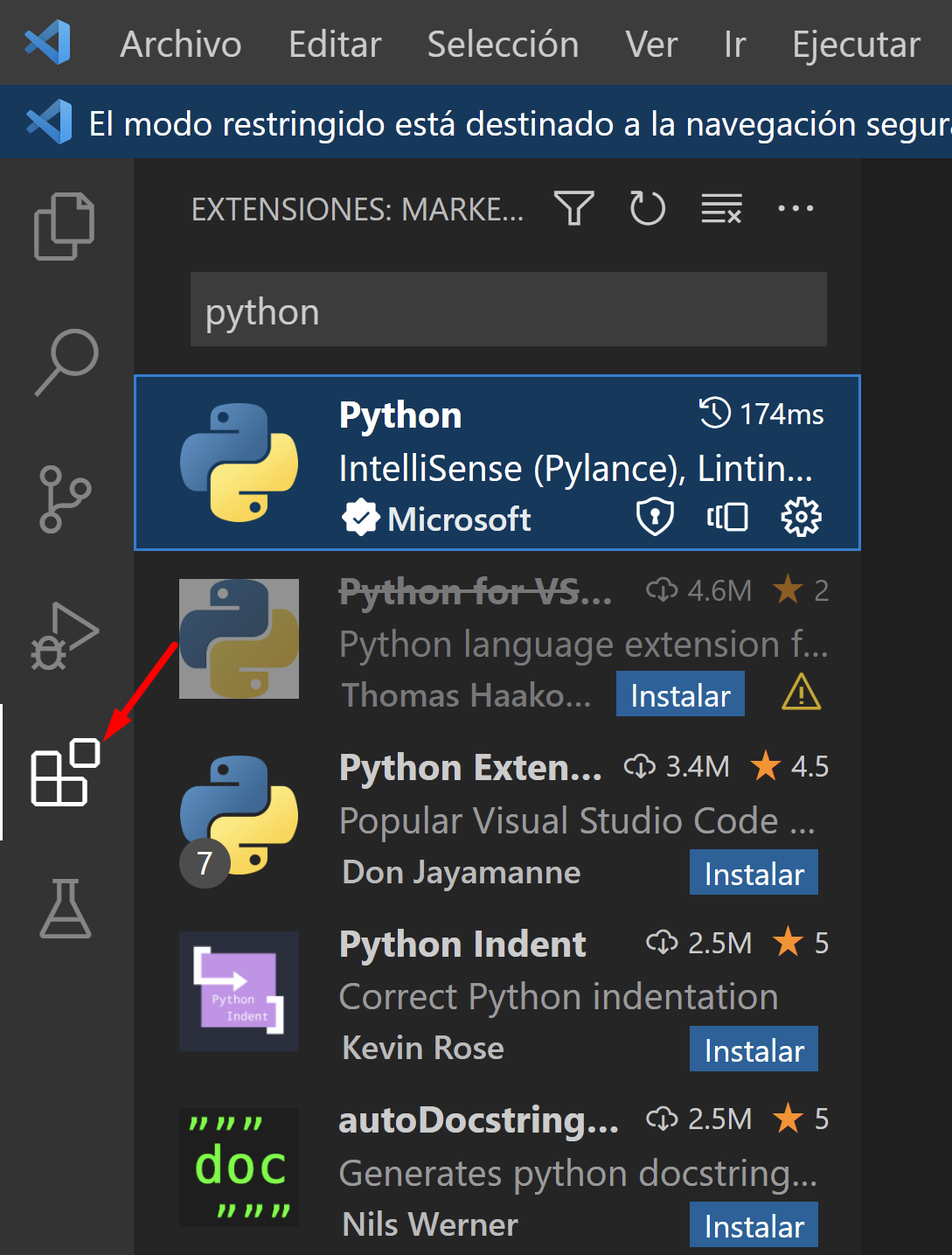
2. Creamos un archivo vacío.



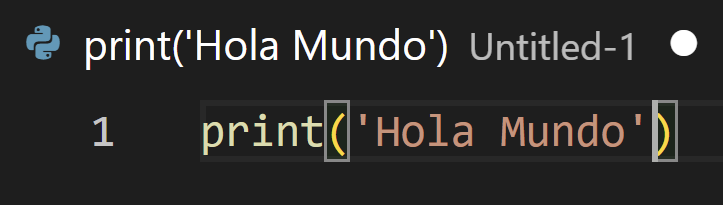
3. Elegimos un archivo formato tipo: Python.



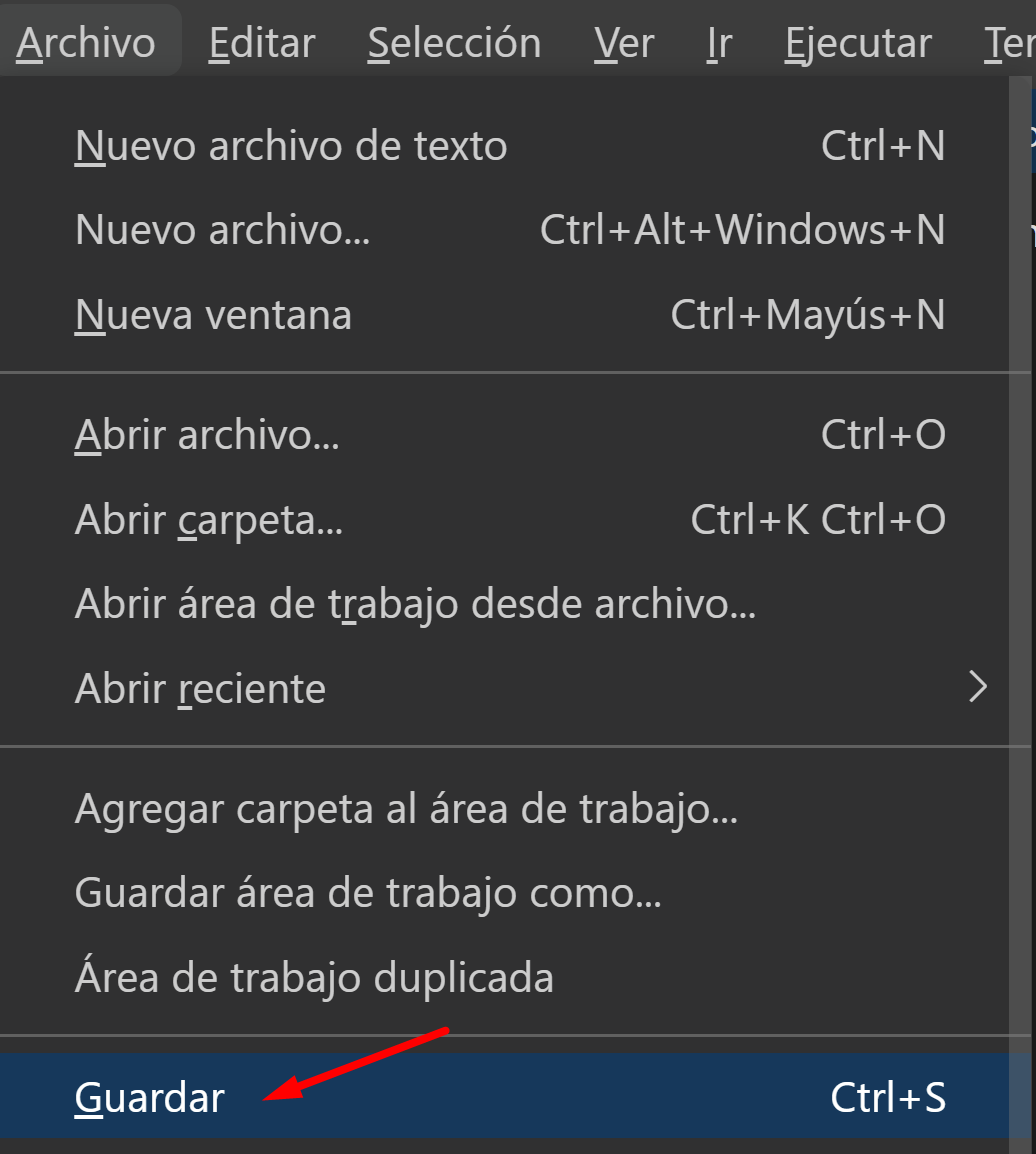
Importante: Si no aparece la opción de Python, es necesario descargar la extensión.



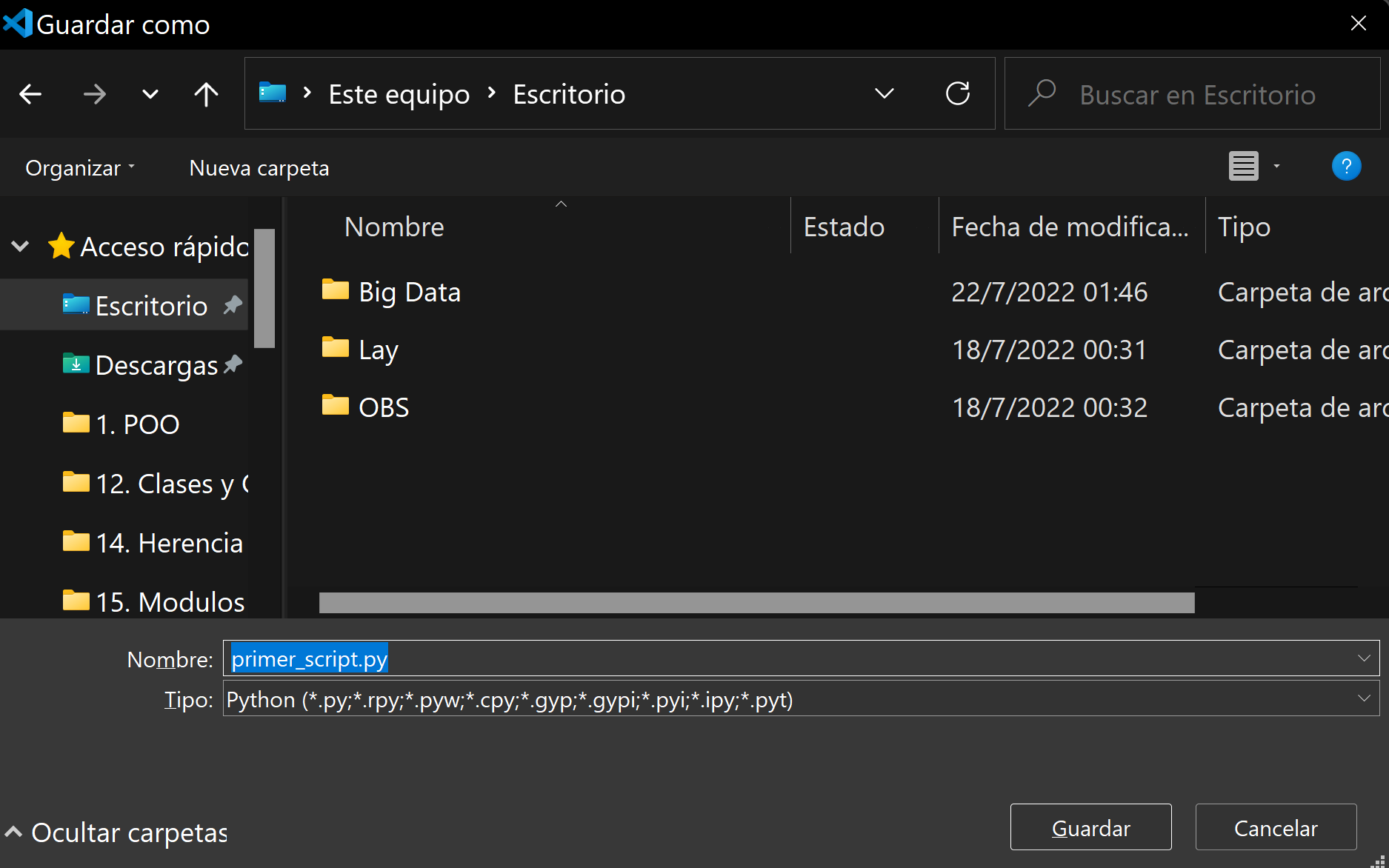
4. Escribimos print('Hola Mundo')



5. Guardamos el archivo y le damos un nombre.



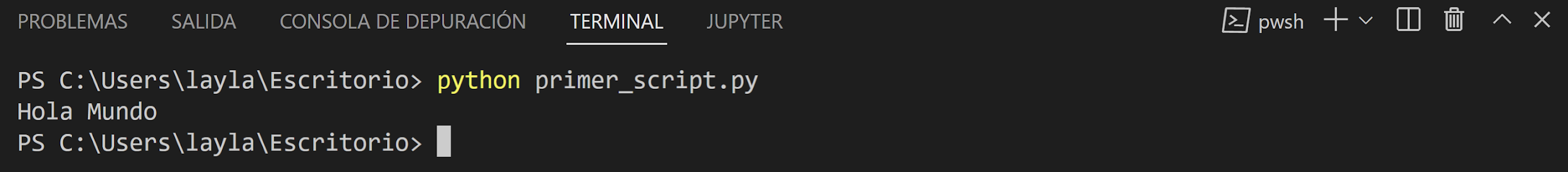
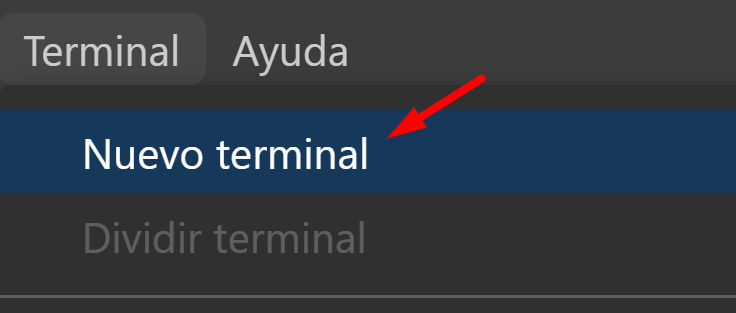
6. Elegimos un directorio dentro de nuestro computador.



7. Ejecutamos el Script desde la terminal.

Comando:

*python primer\_script.py*



**Scripts con argumentos**

**¿Qué son?**

Ahora sabemos como crear scripts, pero la idea era tener una entrada de datos, ¿no?

Los datos que se van a enviar se denominan **argumentos**, y son valores que se pasan al ejecutar un script para que este mismo después los modifique o haga lo que programemos con estos datos.

**Pasar Argumentos a un Script**

Mediante consola (cmd), lo pasamos como si fuera un texto extra a añadir:

**python primer\_script.py argumentos**

Todo se enviará como texto, pero, podemos enviar varios valores.

**Enviar Strings**

En este caso, primero debemos llamar a nuestro script separado por espacios y entre comillas, de la siguiente forma:

**python primer\_script.py “Cadena” 5 [1,2,3,4]**

**Scripts con Argumentos**

Tenemos que ir al script y editarlo. Primero importamos una librería llamada sys y después imprimimos los argumentos usando la librería.

**import sys**

**print(sys.argv)**

Al ejecutar el script podemos ver que devuelve una lista con una cadena que contiene el nombre del script y los argumentos que le pasamos. Siempre el **primer argumento** de la lista sys.argv **(sys.argv[0])** es el propio **nombre** del **script** y lo siguiente son los argumentos que hemos mandado, como **string, int** y **list** respectivamente.

**['primer\_script.py', "'hola'", '1', '[1,2,3]']**

**import sys**

**# Comprobación de seguridad, ejecutar sólo si se reciben 2 argumentos reales**

**if len(sys.argv) == 3:**

**texto = sys.argv[1]**

**repeticiones = int(sys.argv[2])**

**for r in range(repeticiones):**

**print(texto)**

**else:**

**print("Error - Introduce los argumentos correctamente")**

**print('Ejemplo: escribir\_lineas.py "Texto" 5')**

Al ejecutar el script podemos ver que devuelve una lista con una cadena que contiene el nombre del script y los argumentos que le pasamos. Siempre el **primer argumento** de la lista sys.argv (sys.argv[0]) es el propio **nombre** del **script.**

**['primer\_script.py', "'hola'", '1', '[1,2,3]']**

Al ejecutar el script podemos ver que devuelve una lista con una cadena que contiene el nombre del script y los argumentos que le pasamos. Siempre el **primer argumento** de la lista sys.argv (sys.argv[0]) es el propio **nombre** del **script.**

**['primer\_script.py', "'hola'", '1', '[1,2,3]']**

Si escribimos el siguiente script estaría funcionando:

**import sys**

**cadena = sys.argv[1]**

**repeticiones = int(sys.argv[2])**

**for repeticion in range(repeticiones):**

**print(cadena)**

**👉 Nota:** Si no pasamos los argumentos que pide el script, nos va a fallar. Para eso debemos comprobar que existen los argumentos o comprobar la longitud de la lista validandolo con if-else

**Módulo**

**¿Qué es?**

Un módulo es un archivo que consta de código Python. Puede definir funciones, clases y variables. También puede incluir código ejecutable.

Se trata de un archivo con extensión **.py** o **.pyc** (Python compilado) que posee su propio espacio de nombres y que puede contener variables, funciones, clases e incluso otros módulos.

**¿Para qué sirve?**

Un módulo permite organizar lógicamente el código Python. Agrupar el código relacionado en un módulo hace que el código sea más fácil de entender y usar. Un módulo es un objeto de Python con atributos de nombres arbitrarios que puede enlazar y luego referenciar.

Sirve para organizar y reutilizar el código (modularización y reutilización). Se genera uno creando un archivo con extensión **.py** (o **.pyc** o archivo en C) y guardándolo donde nos interese.

Básicamente utilizaremos módulos para organizar y reutilizar nuestro código.

**Creación del módulo**

**0.** Paso cero: por primera vez en el curso y de ahora en más nos desprendemos de Colabs…

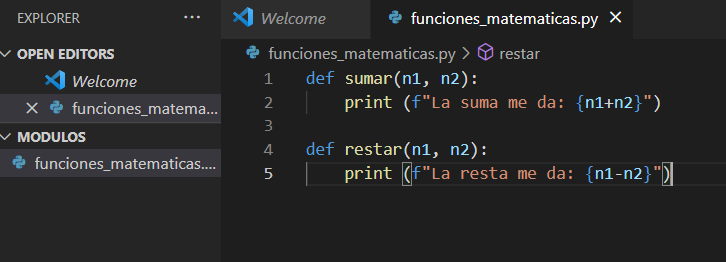
Y pasamos a un IDE más robusto y con más funcionalidades. En nuestro caso Visual Studio Code.

**1.** Creamos una carpeta, donde más cómodo te sientas.

**2.** Abrimos nuestro editor (VCS) y entramos a la carpeta del punto 1.

**3.** Creamos un archivo con la extensión .py. En el ejemplo se llamará funciones\_matematicas.py.

**4.** Creamos algunas funciones simples en ese módulo, veamos mi ejemplo:



Hasta ahora parece que no es nada nuevo, ¿Verdad?

Efectivamente es lo mismo de siempre. Pero, aquí viene lo nuevo e interesante: Vamos a llamar a esas funciones desde otro archivo.

**5.** Crear otro archivo .py, se llamara uso.py

**6.** Ahora viene lo nuevo, si queremos que desde uso.py se puede acceder a todo lo realizado en funciones\_matematicas.py tenemos que escribir arriba de todo:

**import funciones\_matematicas**

Ahora al ejecutar, estarás viendo el resultado de la suma o de cualquier otra función que hayas creado en tu módulo. Para evitar escribir a cada rato el nombre del módulo, hay una alternativa muy útil: Podemos reemplazar el **import** por:

**from funciones\_matematicas import suma**

😎

La opción anterior, es una buena alternativa cuando son pocas las funciones que a usar de un módulo.

**Si vamos a usar muchas, o todas, será mejor que usemos:**

**from funciones\_matematicas import \***

**Paquete**

**Definición**

Guardar todos los módulos en la misma “carpeta” puede ser muy tedioso y difícil de administrar. ¡Imaginen si tienen 100 módulos, esa carpeta estará plagada de archivos .py. !

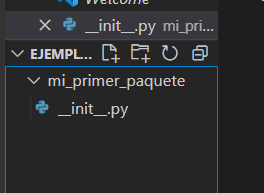
Para eso aparecen los **paquetes**, no son otra cosa que **carpetas para organizar los módulos, por categorías.**

👉Crear un paquete es muy sencillo, solo tienes que crear una carpeta y dentro de ella deberas tener un archivo \_\_init\_\_.py

**Nuestro primer paquete**

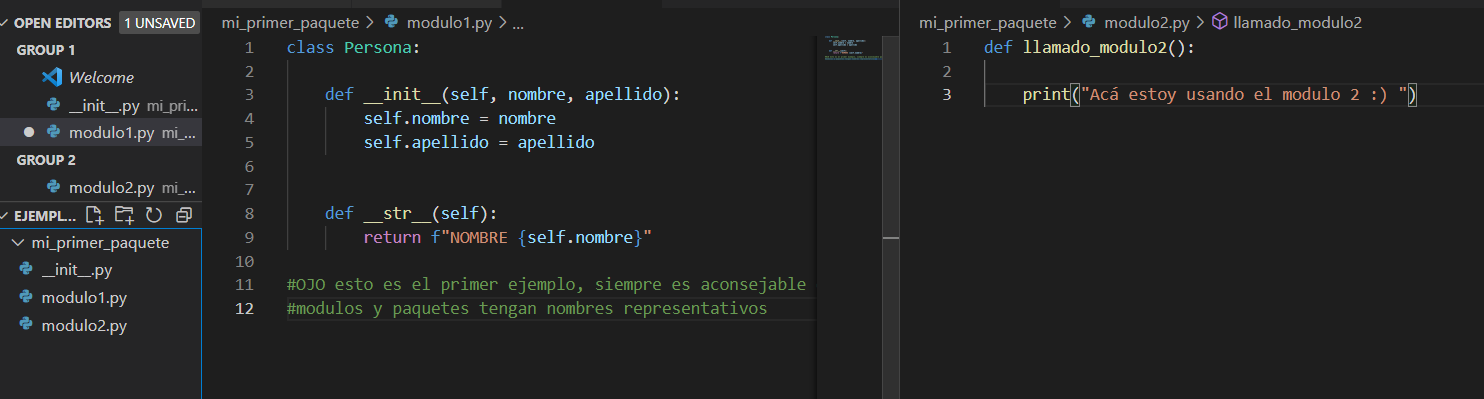
**Paso a paso**

**1.** Podemos crear una carpeta llamada “mi\_primer\_paquete” con un archivo \_\_init\_\_.py



**Paso a paso**

**2.** Ahora hagamos dos módulos en el paquete.



**Paso a paso**

Solo nos falta llamar a alguno de nuestros módulos del paquete “mi\_primer\_paquete”.

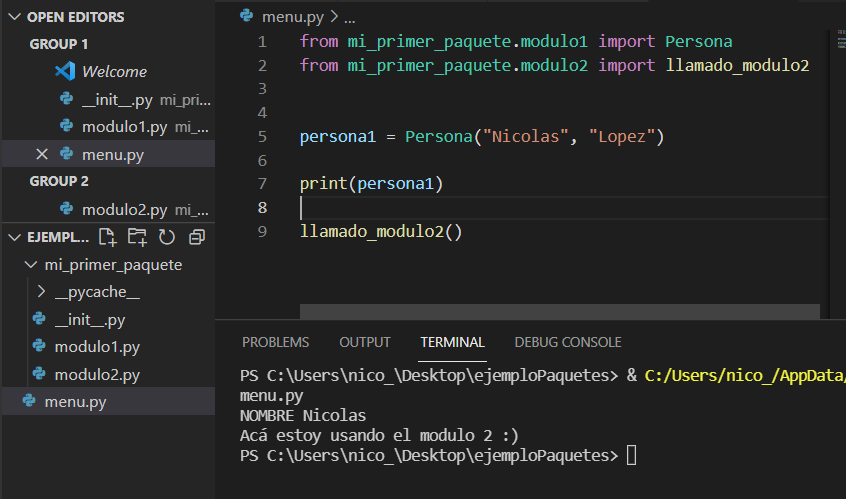
**3.** Para esto creamos un archivo en el directorio raiz, lo llamaremos “menu.py”.

**4.** Importamos al paquete y los módulos de la siguiente manera:

**from mi\_primer\_paquete.modulo1 import Persona**

**from mi\_primer\_paquete.modulo2 import llamado\_modulo2**

**Así nos queda nuestro código😍**

****

**Paquetes redistribuibles**

**Definición**

Es básicamente un paquete para compartir y que lo pueda usar cualquier otro desarrollador o incluso tú mismo en otro proyecto.

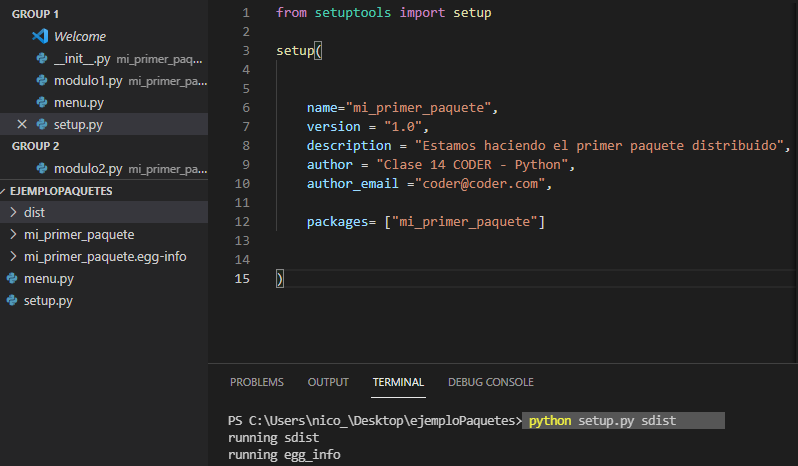
Esto nos permite poder llamar a un paquete sin importar su ubicación.

Veamos a continuación…

**Llamar a un paquete sin importar su ubicación**

**1.** Crear en el directorio raíz un archivo setup.py con algunos datos del paquete y luego desde la terminal ejecutar python setup dist.

**2.** Se crean dos carpetas, en la carpeta dist tendremos nuestro paquete para compartir.

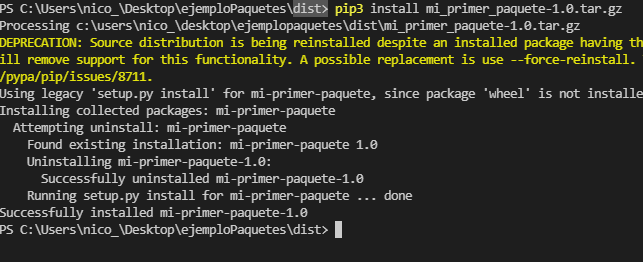


**¿Cómo uso el paquete distribuido de alguien?**

Simple, nos ubicamos en el directorio donde tenemos el paquete distribuido que alguien te paso, y ejecutamos:

**pip3install**

**nombrePaquete.tar.gz**



**Paquetes/módulos externos**

**Collections**

**¿Qué es?**

Python 3 tiene varias estructuras de datos integradas, incluyendo tuplas, diccionarios y listas.

Las estructuras de datos nos proporcionan una forma de organizar y almacenar datos. El módulo collections nos ayuda a completar y manipular las estructuras de datos de forma eficiente.

Para usarlo debemos importarlo, ya que es un paquete con módulos: from collections import \*

**Namedtuple**

Permite añadir nombres explícitos a cada elemento de una tupla para hacer que estos significados sean claros en su programa Python.

👉 Vamos a usar namedtuple para generar una clase que claramente denomine a cada elemento de la tupla de peces:

**from collections import namedtuple**

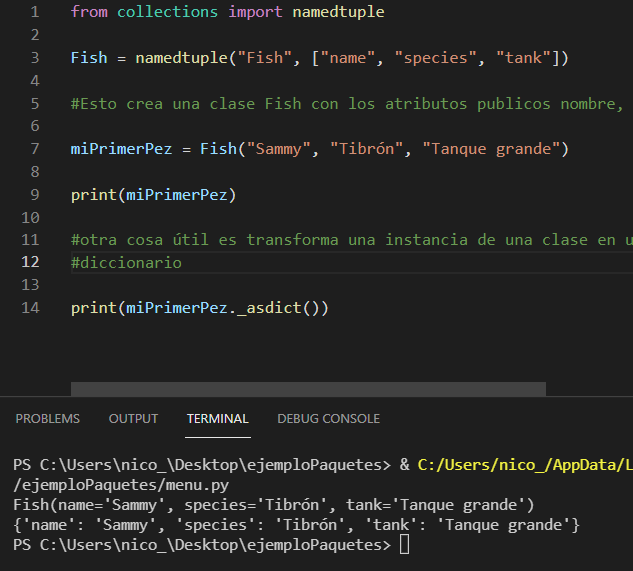
**Fish = namedtuple("Fish", ["name", "species", "tank"])**

**Namedtuple**

**¡Lo hemos logrado!**

**👏**

Creamos una clase y la instanciamos en dos renglones.

****

**Counter**

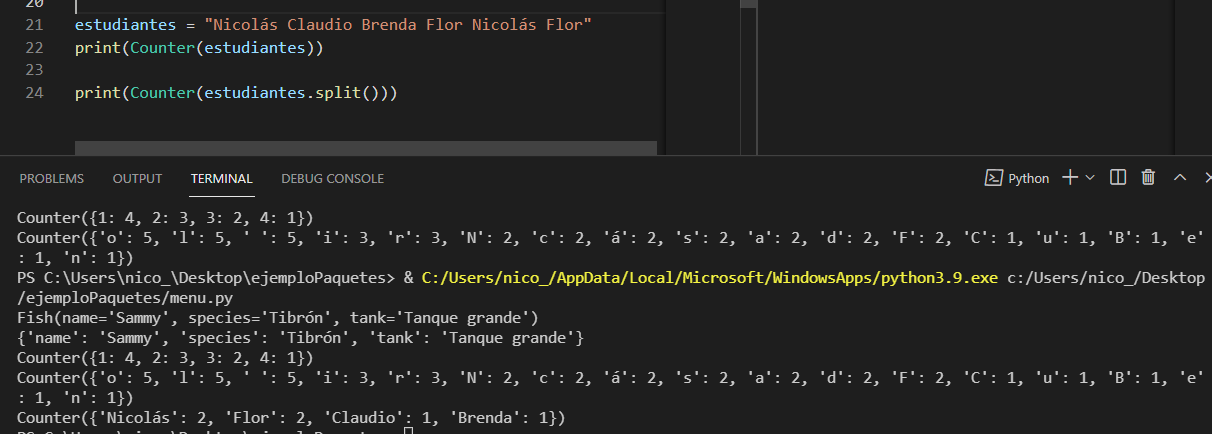
**¿Qué es?**

La clase Counter es una subclase de diccionario utilizada para realizar cuentas con diccionarios y listas.



**Ejemplos con texto**

Podemos contar y transformar a diccionario los conteos por letras o palabras.



**Otras funciones**

Con este paquete podemos, entre otras cosas, también **ordenar, agrupar y rankear registros de un diccionario.**

👀 [Ejemplos](https://docs.hektorprofe.net/python/modulos-y-paquetes/modulo-collections/)

**Datetime**

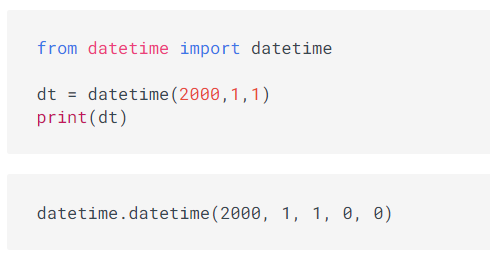
Esta es una clase para manejar fechas y horas de una forma muy simple, nos será super útil de acá en adelante.

**Por ejemplo:** de la siguiente forma podemos acceder al año, mes, dia, hora, minuto, segundo y microsegundo del instante en el que creamos una instancia.

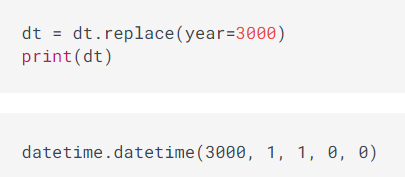
****

**Datetime**

No solo se puede trabajar con una hora puntual, podemos crear nuestra propia fecha usando el constructor de esta clase.



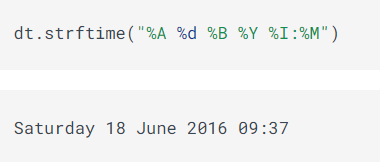
Si queremos cambiar uno de los parámetros, debemos usar la función replace.



**Personalizar**

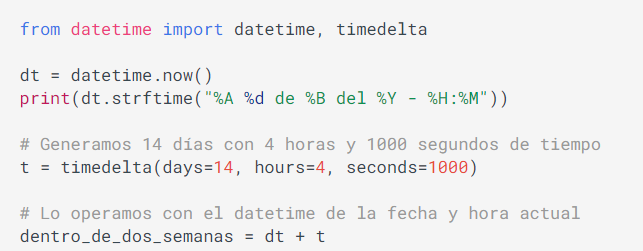
Es muy sencillo cambiar la forma en la que presentamos un datetime.

Otras combinaciones disponibles 👉 [Formateos](https://docs.python.org/3/library/datetime.html#strftime-and-strptime-behavior)



**Facilidad increíble**

Con la función timedelta () también puedes sumar o restar tiempos.



**Math**

**¿Qué es?**

A continuación aprenderemos las funciones más importantes.

Pueden verlas a todas desde [math](https://www.w3schools.com/python/module_math.asp)

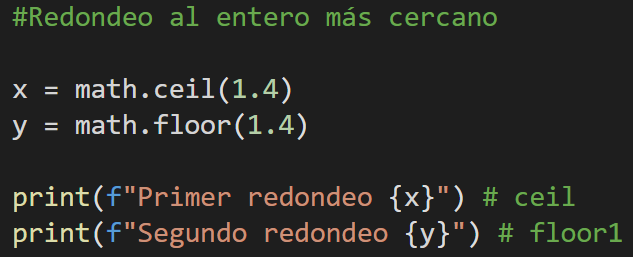
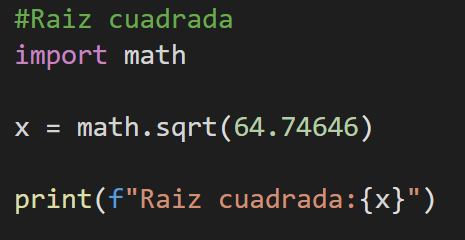
👇

Python tiene un módulo incorporado que puede usarse para tareas matemáticas, este módulo se llama Math.

**Algunos usos**

Vamos a importarla con

**import math**



**Random**

**¿Qué es?**

A continuación aprenderemos las funciones más importantes.

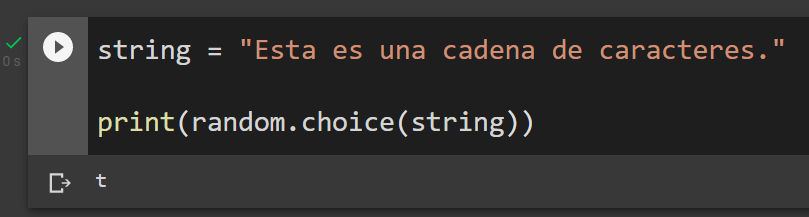
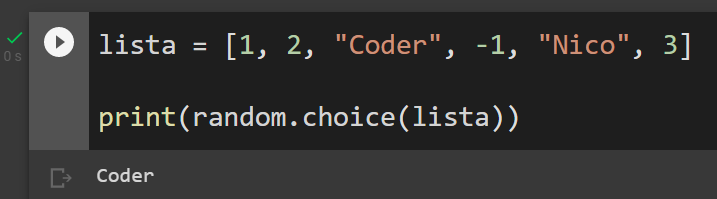
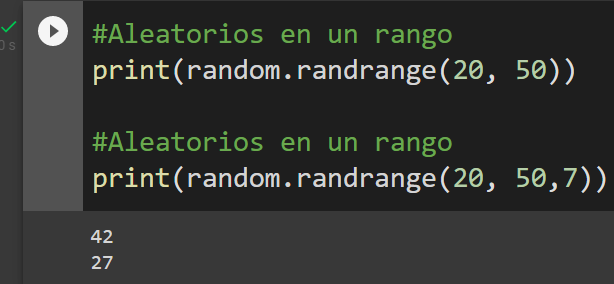
Pueden verlas a todas desde [random](https://www.geeksforgeeks.org/random-numbers-in-python/)

👇

Este nuevo módulo nos permite obtener valores al azar o aleatorios. Por eso lo conocemos como módulo aleatorio.

**Algunos ejemplos**

Con **import random**  llamamos a nuestro módulo aleatorio.



**Recuerda instalar** GIT **para la próxima clase.**

[Ver tutorial](https://docs.google.com/presentation/d/1fkD3xOR35lTDyTg6xSzupEUyarKOVKnkXovvNl-rmXU/edit?usp=sharing)