Esta clase va a ser

- grabada

Clase 14. PYTHON

Scripts, Módulos y Paquetes

Temario

13	**14**	15
Herencias	**Scripts, Módulos y Paquetes**	**Manejo de archivos y datos**
✓ Herencia	✓ Scripts	✓ Persistencia
✓ Herencia múltiple	✓ Módulo	✓ Archivos JSON
✓ Poliformismo	✓ Paquete	✓ Trabajo con datos reales

Objetivos de la clase

- **Crear** los primeros scripts
- **Conocer** la estructura de un módulo.
- **Ejecutar** módulos y paquetes en Python.
- **Utilizar** nuevas librerías.

Repositorio Github

Te dejamos el acceso al Repositorio de Github donde encontrarás todo el material complementario y scripts de la clase.

✓ Repositorio Python

CODERHOUSE

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).

CODERHOUSE

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?

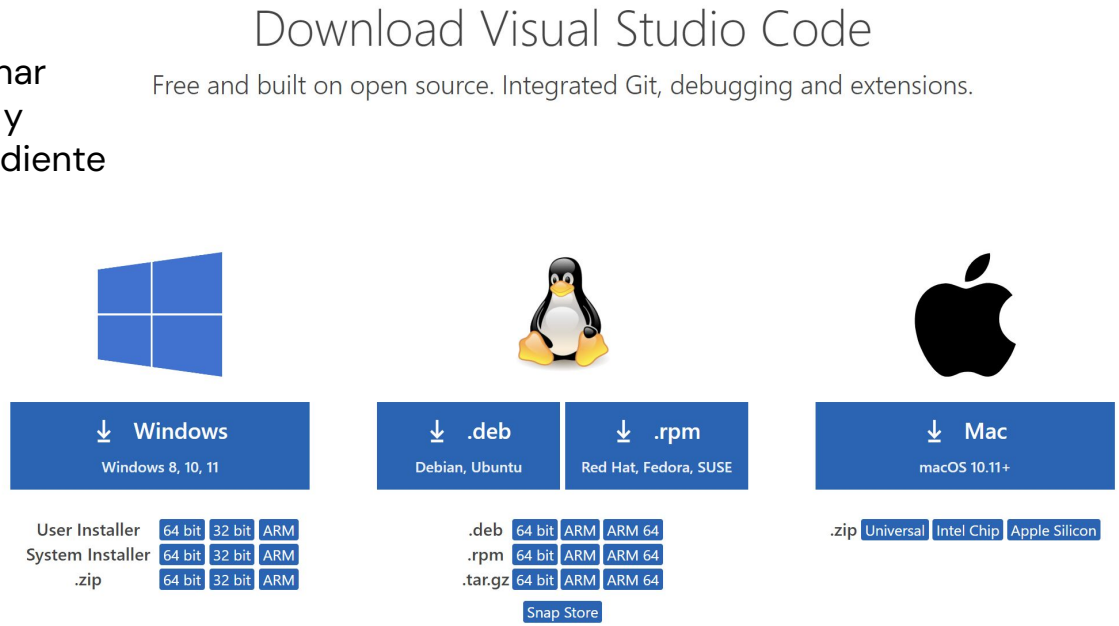


Y también están los editores de códigos que nos permiten escribir código de más de un lenguaje y tiene extensiones para poder “acercar” el editor a un IDE y sus funcionalidades.

Descarguemos VSC

- https://code.visualstudio.com/download

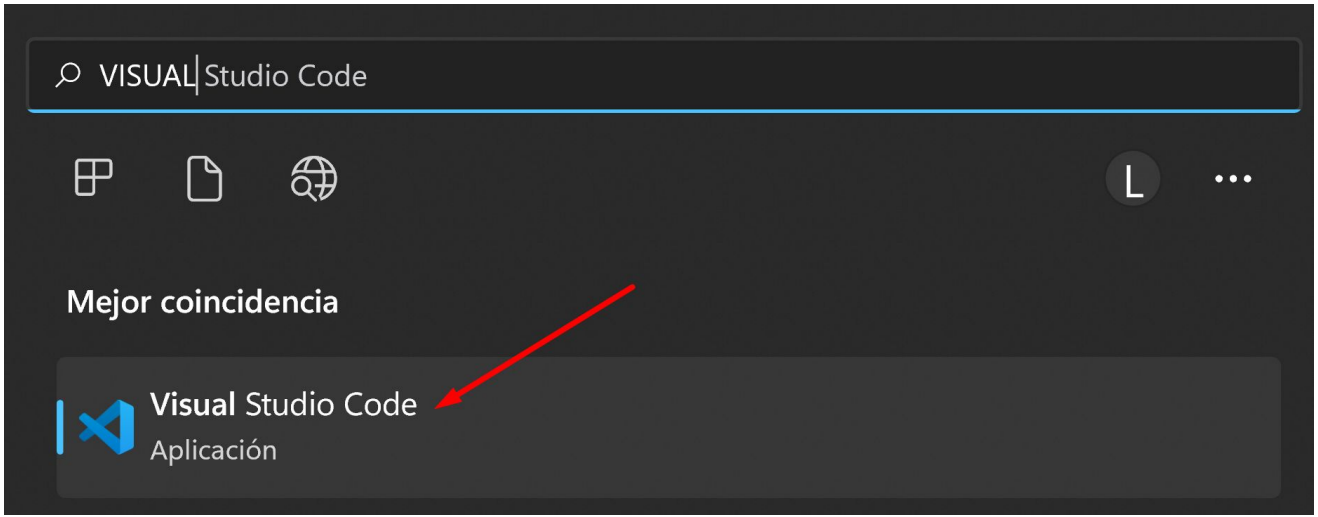
Deberemos seleccionar el S.O que tengamos y realizar su correspondiente Instalación.



Empecemos con VSC

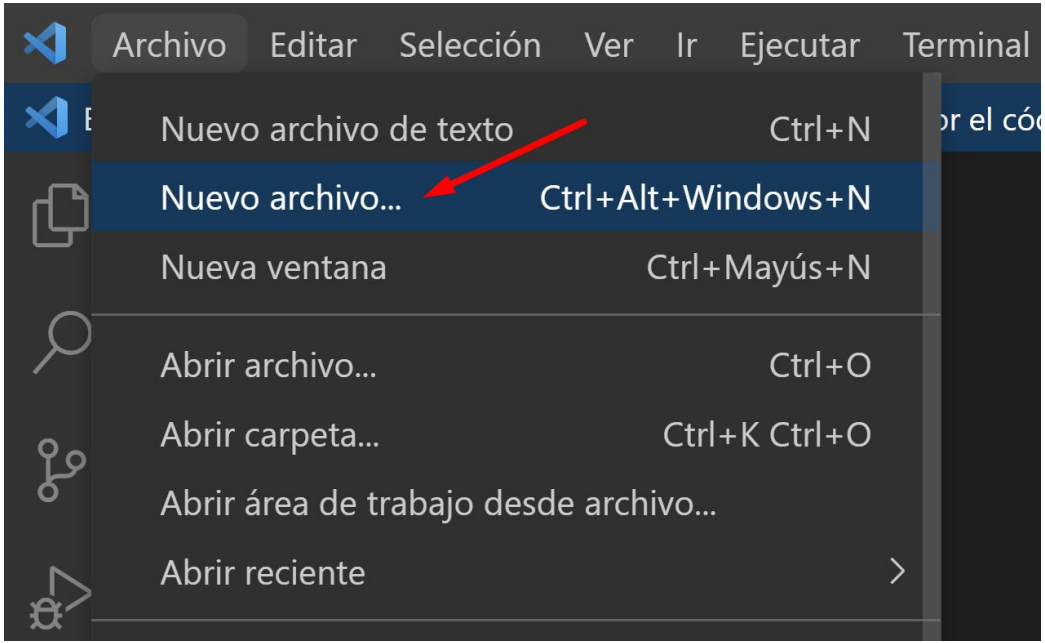
A continuación crearemos nuestro primer script en Python. Para ello, realizaremos los siguientes pasos:

1. Abrimos VSC.



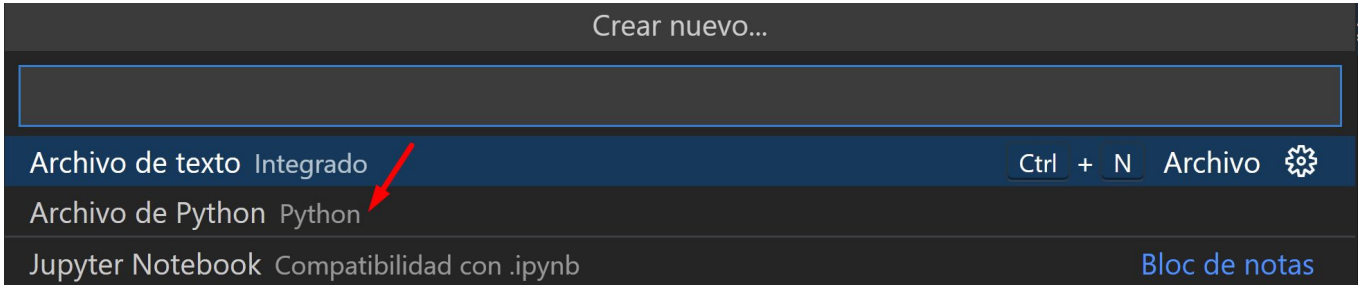
Empecemos con VSC

2. Creamos un archivo vacío.



Empecemos con VSC

3. Elegimos un archivo formato tipo: Python.



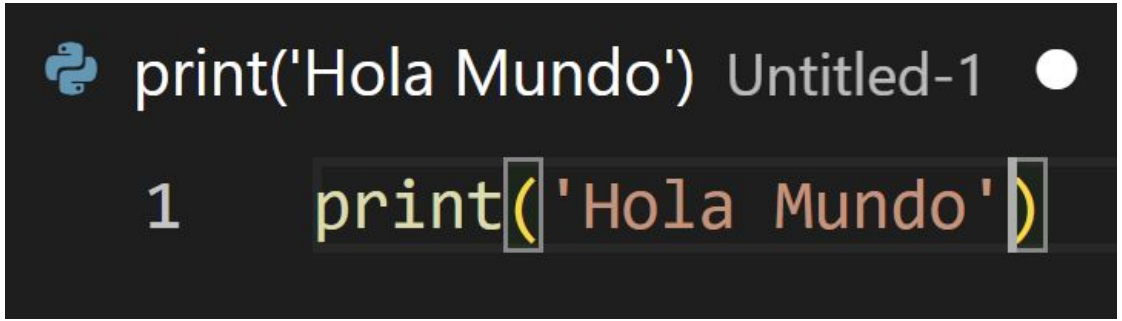
Empecemos con VSC

Importante

Si no aparece la opción de Python, es necesario descargar la extensión.
Se recomienda el Python Extension Pack que agrega un conjunto de extensiones que complementan nuestro desarrollo con Python

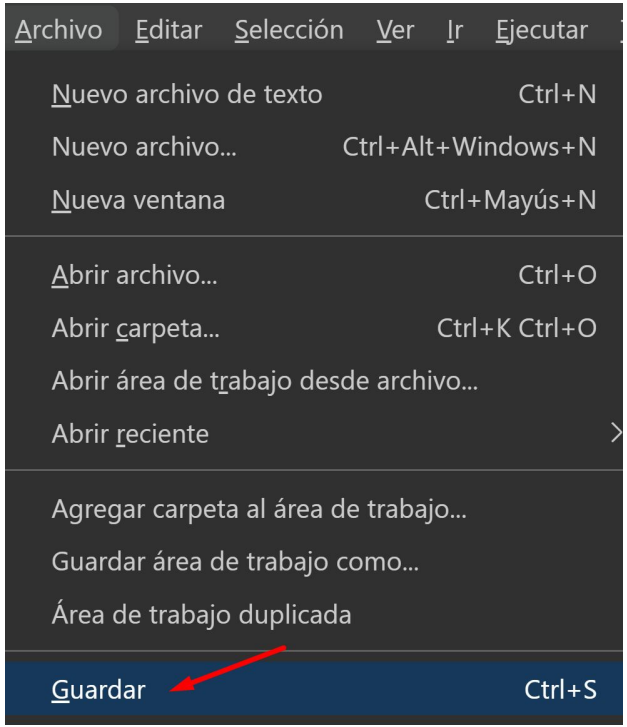
Empecemos con VSC

4. Escribimos print('Hola Mundo')



Empecemos con VSC

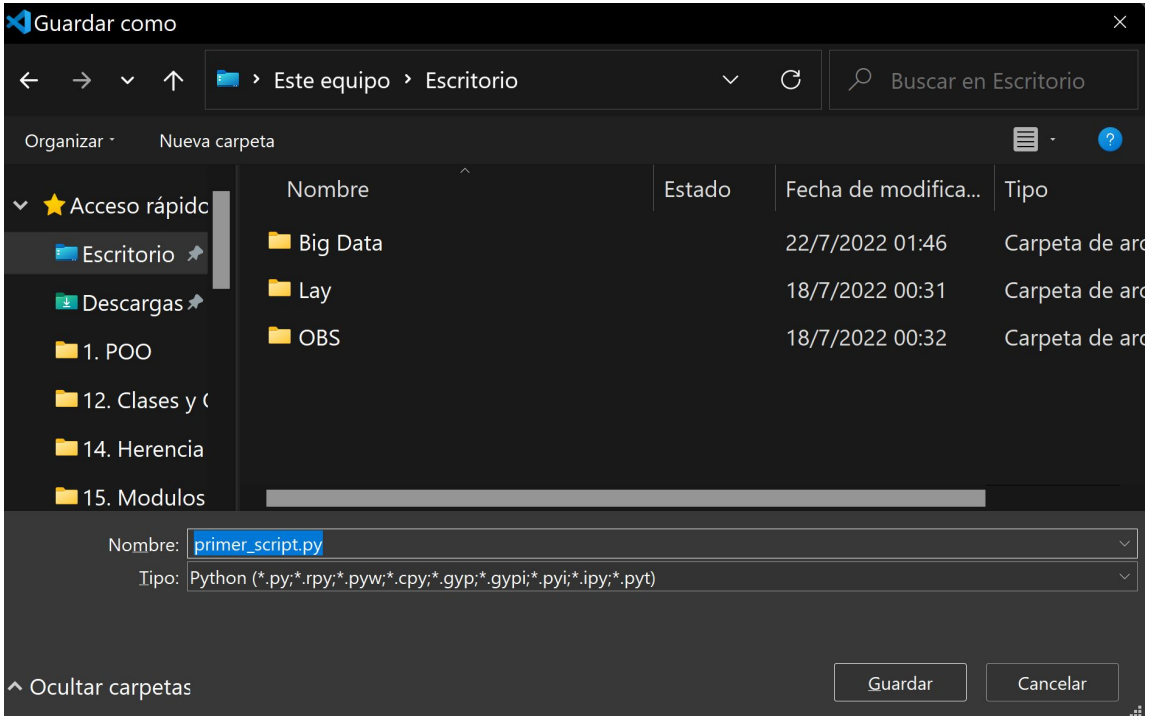
5. Guardamos el archivo y le damos un nombre.



CODERHOUSE

Empecemos con VSC

6. Elegimos un directorio dentro de nuestro computador.

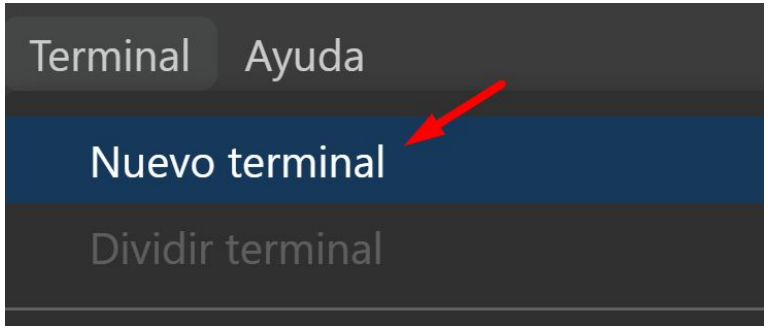


CODERHOUSE

Empecemos con VSC

7. Ejecutamos el Script desde la terminal.

Comando:
python primer_script.py



PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL JUPYTER pwsh

```
PS C:\Users\layla\Escritorio> python primer_script.py
Hola Mundo
PS C:\Users\layla\Escritorio>
```

CODERHOUSE

Scripts con argumentos

¿Qué son?

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

Los datos que se van a enviar se denominan **argumentos**. 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]
```

Ejemplo en vivo

Scripts con Argumentos

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)
```

Scripts con Argumentos

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]']
```

CODERHOUSE

Scripts con Argumentos

```
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')
```

Scripts con Argumentos

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]']
```

Scripts con Argumentos

Haremos un script que printee un argumento las veces que se indique por argumento, es decir, si paso los argumentos **“Hola” y 5, se imprimirá “Hola” 5 veces.**

Empecemos creando un nuevo archivo llamado **repeticion.py** y editemoslo.

Scripts con Argumentos

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.

Fuente: codigofuente.org

¿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.

Fuente: codigofuente.org

Ejemplo en vivo

Creación del módulo

CODERHOUSE

Paso a paso

0. Paso cero: por primera vez en el curso y de ahora en más dejamos de utilizar Colabs...

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

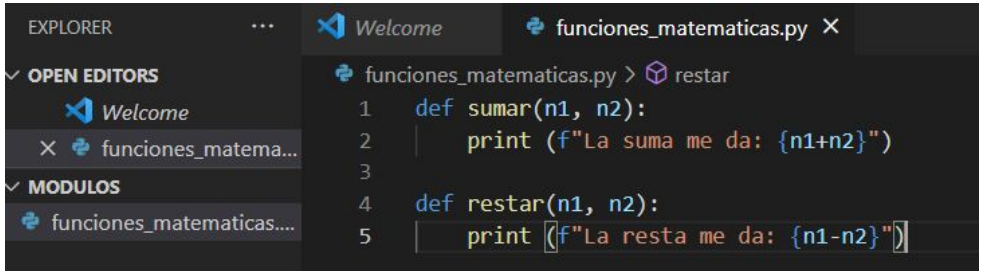
Paso a paso

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:



CODERHOUSE

PARA RECORDAR

Contenido destacado

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.

Paso a paso

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
```

Paso a paso

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
```

😎

Paso a paso

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 *
```

CODERHOUSE

Primer módulo

Crea un módulo que tenga una clase con atributos y métodos. Instanciar a la clase llamando al módulo desde otro archivo .py

Duración: **10 minutos**

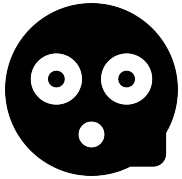
CODERHOUSE

ACTIVIDAD EN CLASE

Primer módulo

1. Hacer una clase fácil, como Alumno, con nombre y nota, con un método imprimir().
2. Crear una instancia de Alumno, mostrando sus datos y llamando al método desde otro módulo.

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.**

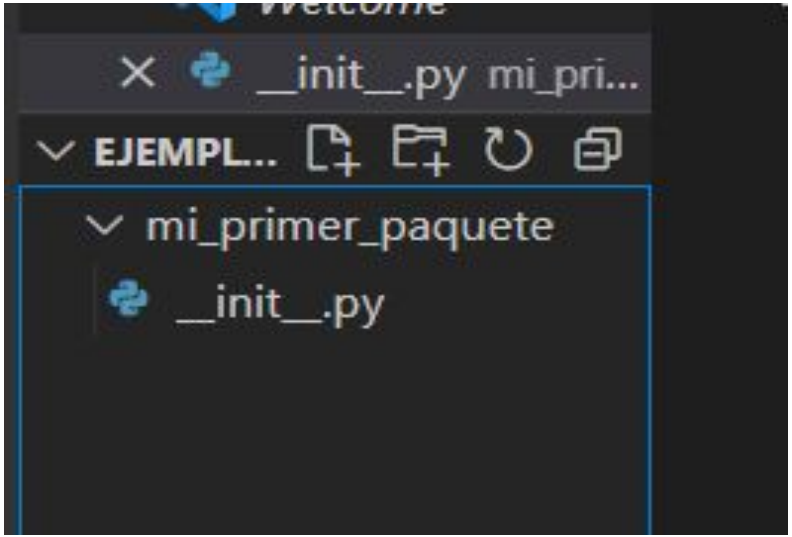
👉Crear un paquete es muy sencillo, solo tienes que crear una carpeta y dentro de ella deberas tener un archivo __init__.py

Fuente: codigofuente.org

Nuestro primer paquete

Paso a paso

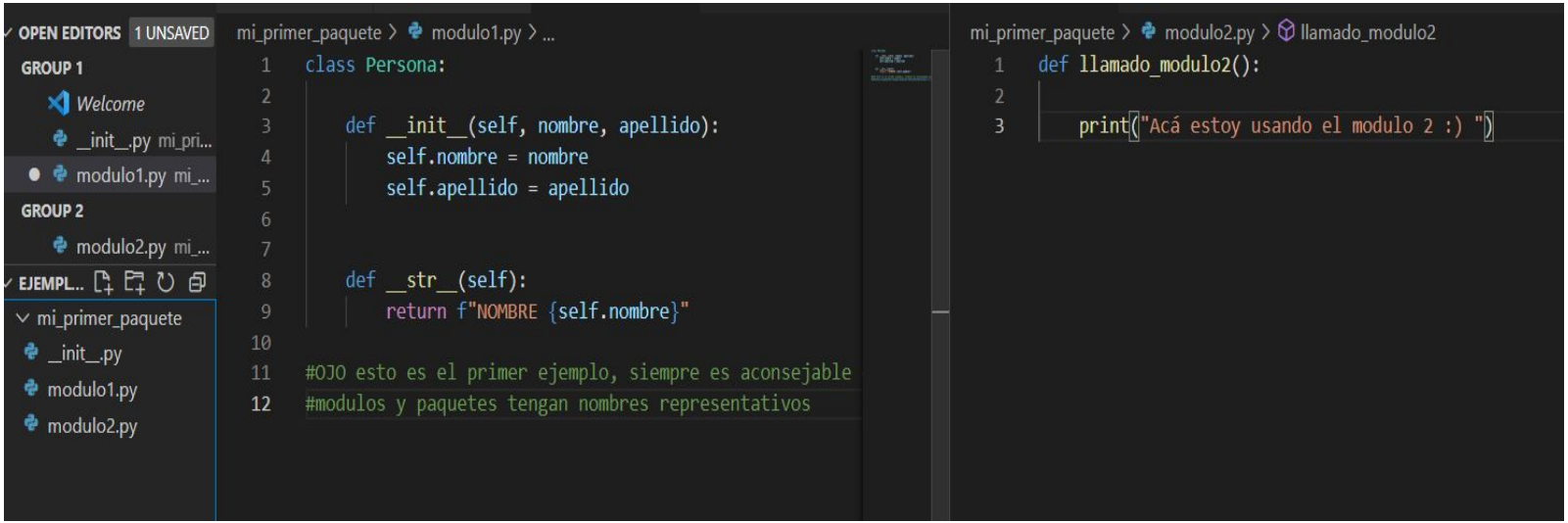
1. Podemos crear una carpeta llamada “mi_primer_paquete” con un archivo __init__.py



CODERHOUSE

Paso a paso

2. Ahora hagamos dos módulos en el paquete.



Paso a paso

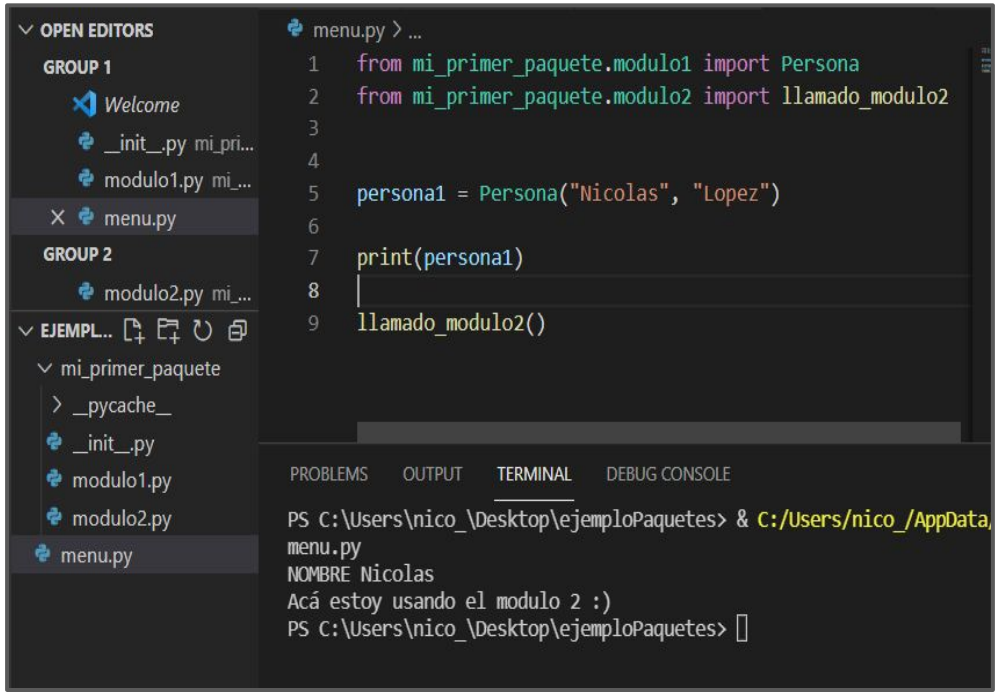
Solo nos falta llamar a alguno de nuestros módulos del paquete “mi_primer_paquete”.

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😍



Break

¡10 minutos y volvemos!

Paquetes redistribuibles

Definición

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

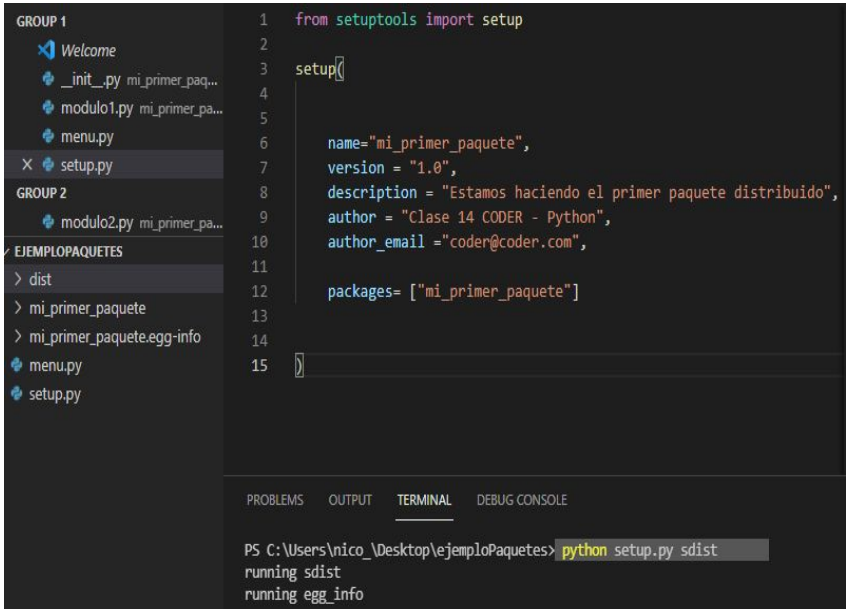
Esto nos permite poder llamar a un paquete sin importar su ubicación.

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.

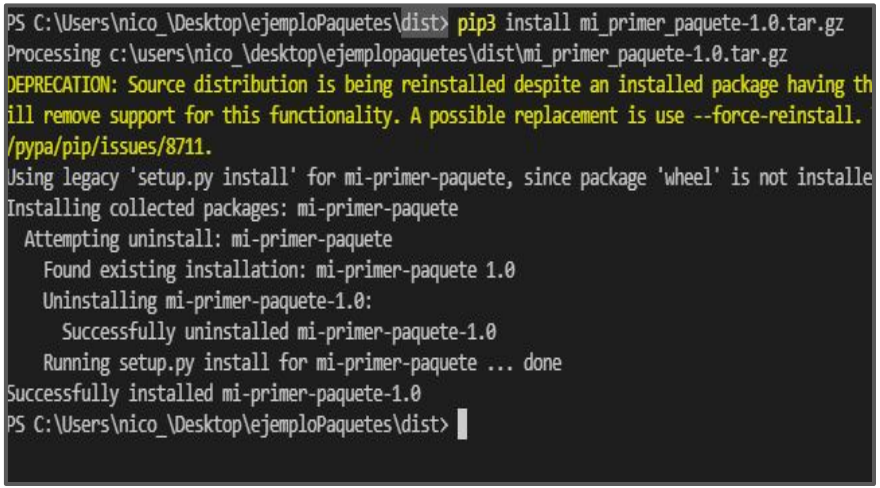


CODERHOUSE

¿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

```python
from collections import namedtuple

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

#Esto crea una clase Fish con los atributos publicos nombre,

miPrimerPez = Fish("Sammy", "Tibrón", "Tanque grande")

print(miPrimerPez)

#otra cosa útil es transforma una instancia de una clase en u
#diccionario

print(miPrimerPez._asdict())
```

```
PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE

PS C:\Users\nico_\Desktop\ejemploPaquetes> & C:/Users/nico_/AppData/L
/ejemploPaquetes/menu.py
Fish(name='Sammy', species='Tibrón', tank='Tanque grande')
{'name': 'Sammy', 'species': 'Tibrón', 'tank': 'Tanque grande'}
PS C:\Users\nico_\Desktop\ejemploPaquetes>
```

¡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.

```
16  from collections import Counter
17
18  l = [1,2,3,4,1,2,3,1,2,1]
19  print(Counter(l))
```

```
PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE

PS C:\Users\nico_\Desktop\ejemploPaquetes> & C:/Users/nico_/AppData/Loc
/ejemploPaquetes/menu.py
Fish(name='Sammy', species='Tibrón', tank='Tanque grande')
{'name': 'Sammy', 'species': 'Tibrón', 'tank': 'Tanque grande'}
Counter({1: 4, 2: 3, 3: 2, 4: 1})
PS C:\Users\nico_\Desktop\ejemploPaquetes>
```

Ejemplos con texto

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

```
21  estudiantes = "Nicolás Claudio Brenda Flor Nicolás Flor"
22  print(Counter(estudiantes))
23
24  print(Counter(estudiantes.split()))
```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE Python

```
Counter({1: 4, 2: 3, 3: 2, 4: 1})
Counter({'o': 5, 'l': 5, ' ': 5, 'i': 3, 'r': 3, 'N': 2, 'c': 2, 'á': 2, 's': 2, 'a': 2, 'd': 2, 'F': 2, 'C': 1, 'u': 1, 'B': 1, 'e'
: 1, 'n': 1})
PS C:\Users\nico_\Desktop\ejemploPaquetes> & C:/Users/nico_/AppData/Local/Microsoft/WindowsApps/python3.9.exe c:/Users/nico_/Desktop
/ejemploPaquetes/menu.py
Fish(name='Sammy', species='Tibrón', tank='Tanque grande')
{'name': 'Sammy', 'species': 'Tibrón', 'tank': 'Tanque grande'}
Counter({1: 4, 2: 3, 3: 2, 4: 1})
Counter({'o': 5, 'l': 5, ' ': 5, 'i': 3, 'r': 3, 'N': 2, 'c': 2, 'á': 2, 's': 2, 'a': 2, 'd': 2, 'F': 2, 'C': 1, 'u': 1, 'B': 1, 'e'
: 1, 'n': 1})
Counter({'Nicolás': 2, 'Flor': 2, 'Claudio': 1, 'Brenda': 1})
```

Otras funciones

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

👀 Ejemplos

Datetime

Datetime

```python
from datetime import datetime

dt = datetime.now()    # Fecha y hora actual

print(dt)
print(dt.year)         # año
print(dt.month)        # mes
print(dt.day)          # día

print(dt.hour)         # hora
print(dt.minute)       # minutos
print(dt.second)       # segundos
print(dt.microsecond)  # microsegundos
```

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

```
from datetime import datetime

dt = datetime(2000,1,1)
print(dt)
```

```
datetime.datetime(2000, 1, 1, 0, 0)
```

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

```
dt = dt.replace(year=3000)
print(dt)
```

```
datetime.datetime(3000, 1, 1, 0, 0)
```

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

Personalizar

```
dt.strftime("%A %d %B %Y %I:%M")
```

```
Saturday 18 June 2016 09:37
```

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

Otras combinaciones disponibles 👉
Formateos

Facilidad increíble

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

```
from datetime import datetime, timedelta

dt = datetime.now()
print(dt.strftime("%A %d de %B del %Y - %H:%M"))

# Generamos 14 días con 4 horas y 1000 segundos de tiempo
t = timedelta(days=14, hours=4, seconds=1000)

# Lo operamos con el datetime de la fecha y hora actual
dentro_de_dos_semanas = dt + t
```

Math

¿Qué es?

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

A continuación aprenderemos las funciones más importantes.
Pueden verlas a todas desde math 👇

CODERHOUSE

Algunos usos

Vamos a importarla con

`import math`

```python
#Raiz cuadrada
import math

x = math.sqrt(64.74646)

print(f"Raiz cuadrada:{x}")
```

```python
#Redondeo al entero más cercano

x = math.ceil(1.4)
y = math.floor(1.4)

print(f"Primer redondeo {x}") # ceil
print(f"Segundo redondeo {y}") # floor1
```

Random

¿Qué es?

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

A continuación aprenderemos las funciones más importantes.
Pueden verlas a todas desde random 👇

Algunos ejemplos

Con `import random` llamamos a nuestro módulo aleatorio.

```
#Aleatorios en un rango
print(random.randrange(20, 50))

#Aleatorios en un rango
print(random.randrange(20, 50,7))
```

```
42
27
```

```
lista = [1, 2, "Coder", -1, "Nico", 3]

print(random.choice(lista))
```

```
Coder
```

```
string = "Esta es una cadena de caracteres."

print(random.choice(string))
```

```
t
```

CODERHOUSE

Segunda pre-entrega de tu Proyecto final

Debes entregar la segunda pre-entrega de tu proyecto final.

CODERHOUSE

Segunda pre-entrega

Objetivo

- ✓ Practicar el concepto de Clases y Objetos.

Consigna

- ✓ Crear un programa que permita el modelamiento de Clientes en una página de compras. Se debe utilizar el concepto de Programación Orientada a Objetos y lo aprendido en clase.
- ✓ Se evaluará el uso correcto de atributos y métodos.
- ✓ Utilizar los conceptos aprendidos en la clase 15 y crear un paquete redistribuible con el programa creado."

Formato

- ✓ El proyecto debe ser un archivo comprimido del paquete. Formatos aceptados: .zip o .tar.gz bajo el nombre "**Segunda pre-entrega+Apellido**".

Segunda pre-entrega

Se debe entregar

- ✓ Se debe entregar todo el programa.

Sugerencias

- ✓ "La Clase Cliente debe tener mínimo 4 atributos y 2 métodos.
- ✓ Se debe utilizar el método __str__() para darle nombre a los objetos.
- ✓ Para crear el paquete distribuible también como adicional el archivo de la Pre entrega #1.
- ✓ Es opcional el uso de herencia."

Recuerden que en la clase 1 tienen disponible un proyecto modelo de un ex estudiante para tomar como inspiración.

#Codertraining

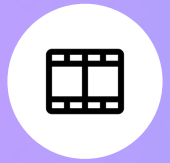
¡No dejes para mañana lo que puedes practicar hoy! Te invitamos a revisar la **Guía de Ejercicios Complementarios**, donde encontrarás un ejercicio para poner en práctica lo visto en la clase de hoy.

¡Atención!

Recuerda instalar GIT para la próxima clase.

Ver tutorial

CODERHOUSE



¿Quieres saber más?
Te dejamos material ampliado de la clase

CODERHOUSE

MATERIAL AMPLIADO

Recursos multimedia

Ejercicios de repaso

✓ EjerciciosdeRepaso

Disponible en nuestro repositorio.

CODERHOUSE

¿Preguntas?

Resumen
de la clase hoy

- ✓ Logramos crear nuestro primer scripts.
- ✓ Creación de módulo.
- ✓ Paquetes.
- ✓ Paquetes redistribuidos.
- ✓ Paquetes / Módulos externos.

Opina y valora esta clase

Muchas gracias.

#DemocratizandoLaEducación