Esta clase va a ser

grabada

Clase 15. PYTHON

Scripts, Módulos y Paquetes



Temario

14 Herencias Herencia Herencia múltiple Poliformismo

15 Scripts, Módulos y **Paquetes Scripts** <u>Módulo</u> <u>Paquete</u>

Git & Github Git Github Lorem ipsum



Objetivos de la clase

- Crear los primeros scripts
- **Conocer** la estructura de un módulo.

- Ejecutar módulos y paquetes en Python.
- Utilizar nuevas librerías.



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

Download Visual Studio Code

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

Free and built on open source. Integrated Git, debugging and extensions.



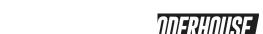




.tar.gz 64 bit ARM ARM 64
Snap Store



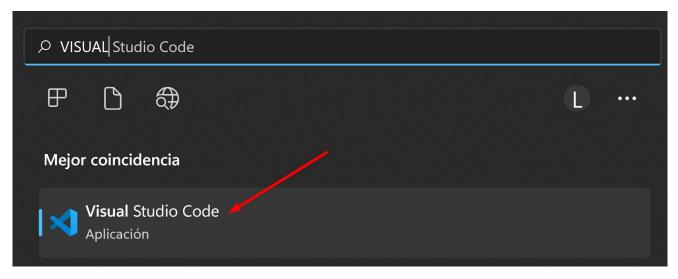
.zip Universal Intel Chip Apple Silicon





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

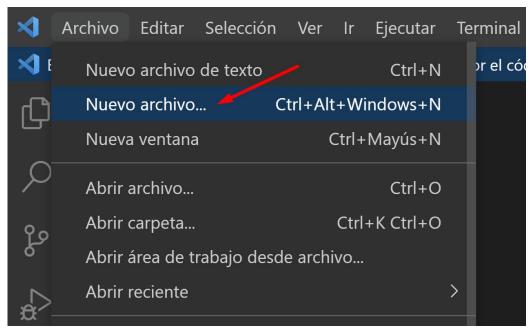
1. Abrimos VSC.







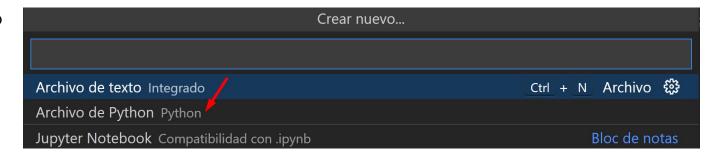
2. Creamos un archivo vacío.







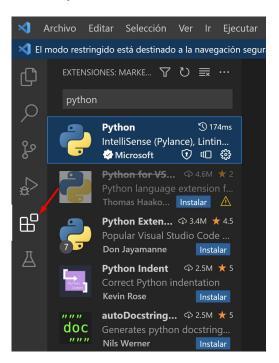
3. Elegimos un archivo formato tipo: Python.







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







4. Escribimos print('Hola Mundo')

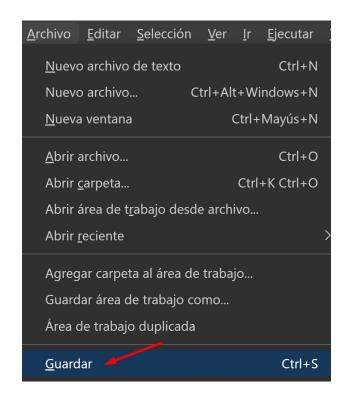
```
print('Hola Mundo') Untitled-1 •

1 print('Hola Mundo')
```





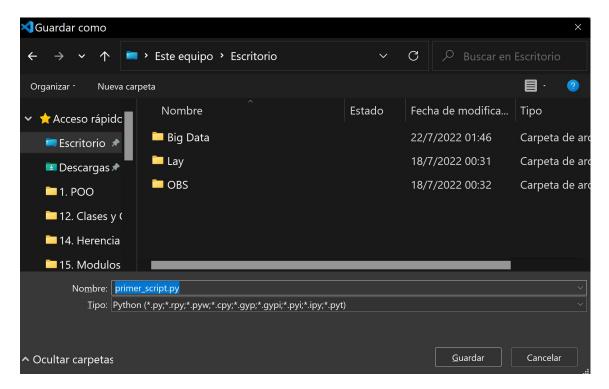
5. Guardamos el archivo y le damos un nombre.







6. Elegimos un directorio dentro de nuestro computador.

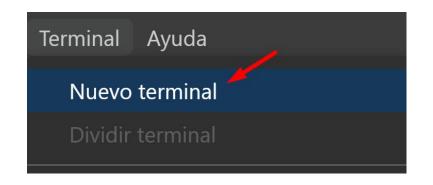






7. Ejecutamos el Script desde la terminal.

Comando: python primer_script.py



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



| Py pwsh + ∨ □

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







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[O]) 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]']
```





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[O]) es el propio **nombre** del **script**.

```
['primer_script.py', "'hola'", '1', '[1,2,3]']
```







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.







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)
```

For 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



Paso a paso



O. 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:

```
Welcome

✓ OPEN EDITORS

✓ Welcome

1 def sumar(n1, n2):
2 print (f"La suma me da: {n1+n2}")
3
4 def restar(n1, n2):
5 print (f"La resta me da: {n1-n2}")
```





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 *





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





Primer módulo

- 1. Hacer una clase fácil, como Alumno, con nombre y nota, con un método imprimir().
- 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







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

```
🗙 🕏 __init__.py mi_pri...
∨ EJEMPL... [‡ [‡ [〕 [☐

→ mi_primer_paquete

  __init__.py
```







2. Ahora hagamos dos módulos en el paquete.

```
mi_primer_paquete > * modulo2.py > * llamado_modulo2
OPEN EDITORS 1 UNSAVED mi_primer_paquete > 🏓 modulo1.py > ...
                             class Persona:
                                                                                                       def llamado modulo2():
GROUP 1
  X Welcome
                                                                                                           print("Acá estoy usando el modulo 2 :) ")
                                 def init (self, nombre, apellido):
   init_.py mi_pri...
                                     self.nombre = nombre
 modulo1.py mi_...
                                     self.apellido = apellido
GROUP 2
   modulo2.py mi ...
EJEMPL... [] E] U A
                                 def str (self):
                                     return f"NOMBRE {self.nombre}"
∨ mi_primer_paquete
__init__.py
                             #OJO esto es el primer ejemplo, siempre es aconsejable
modulo1.py
                             #modulos y paquetes tengan nombres representativos
modulo2.py
```







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 😍



```
menu.py > ...
∨ OPEN EDITORS
                               from mi_primer_paquete.modulo1 import Persona
  GROUP 1
                               from mi primer paquete.modulo2 import llamado modulo2
    Welcome
     _init_.py mi_pri...
     modulo1.py mi_...
                               persona1 = Persona("Nicolas", "Lopez")
  X 🕏 menu.py
  GROUP 2
                               print(personal)
     modulo2.py mi_...
                               llamado modulo2()
∨ EJEMPL... [th ch ひ 合
 ∨ mi_primer_paquete
  > _pycache_
  init_.py
                                            TERMINAL
  modulo1.py
  modulo2.py
                        PS C:\Users\nico \Desktop\ejemploPaquetes> & C:/Users/nico /AppData,
                         menu.py
 menu.py
                        NOMBRE Nicolas
                         Acá estoy usando el modulo 2 :)
                         PS C:\Users\nico \Desktop\ejemploPaquetes> []
```





¡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.

```
from setuptools import setup
   X Welcome
                                    setup(
    init .py mi primer paq...
   modulo1.py mi_primer_pa...
    menu.py
                                         name="mi primer paquete".
 X 🏺 setup.pv
                                         version = "1.0",
                                         description = "Estamos haciendo el primer paquete distribuido"
                                         author = "Clase 14 CODER - Python".
    modulo2.py mi primer pa...
                                         author email = "coder@coder.com",
FIFMPLOPAQUETES
) dist
                                         packages= ["mi primer paquete"]
> mi primer paquete
> mi_primer_paquete.egg-info
menu.py
setup.py
                                        OUTPUT TERMINAL DEBUG CONSOLE
                              PS C:\Users\nico \Desktop\ejemploPaquetes> python setup.py sdist
                              running sdist
                              running egg info
```



¿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?

回

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.

from collections import namedtuple

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



```
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
                             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>
```

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.







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.

• Eiemplos



Datetime



from datetime import datetime dt = datetime.now() # Fecha y hora actual print(dt) print(dt.year) # año print(dt.month) # mes print(dt.day) # dia print(dt.hour) # hora print(dt.minute) # minutos print(dt.second) # segundos print(dt.microsecond) # microsegundos

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.



```
from datetime import datetime

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

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

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

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

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



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 *F*



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





Algunos usos

Vamos a importarla con

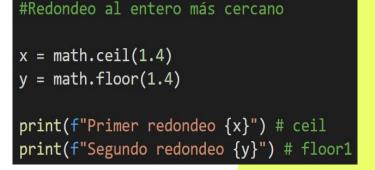
import math

```
#Raiz cuadrada
import math

x = math.sqrt(64.74646)

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







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 <u>random</u>





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))
```

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

print(random.choice(lista))

Coder
```

```
string = "Esta es una cadena de caracteres."
print(random.choice(string))
```





Segunda pre-entrega de tu Proyecto final

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



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.



¡Atención!

Recuerda instalar GIT para la próxima clase.



Ver tutorial





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





Recursos multimedia

Ejercicios de repaso

✓ <u>EjerciciosdeRepaso</u>



¿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