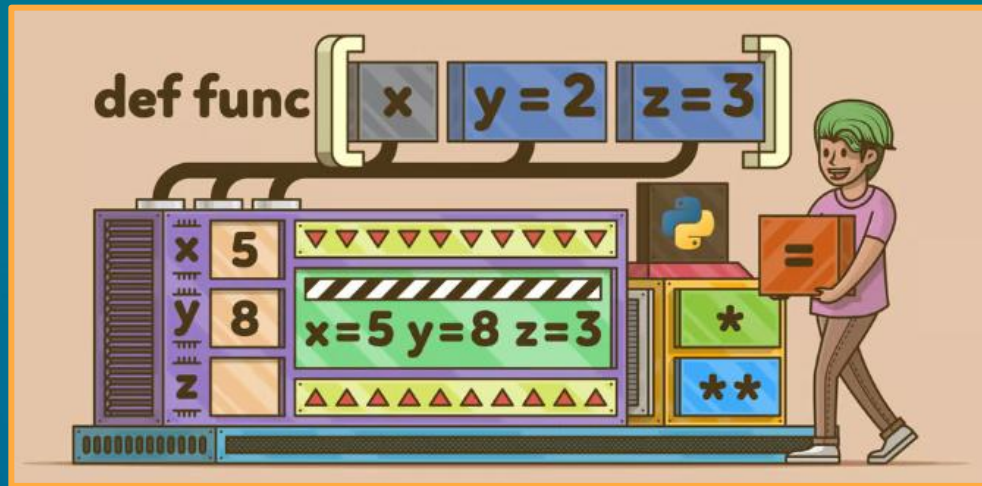


Funciones



Introducción

En Python, una función es un bloque de código que realiza una tarea específica.

¿Qué es una función?

A medida que los programas crecen en extensión y complejidad la resolución se torna más **complicada** y su **depuración** y **modificaciones** futuras resultan casi imposibles. Para resolver este tipo de problemas lo que se hace es **dividir el programa** en módulos más pequeños que cumplan **una tarea simple y bien acotada** llamados funciones.

¿Para qué sirve una función?

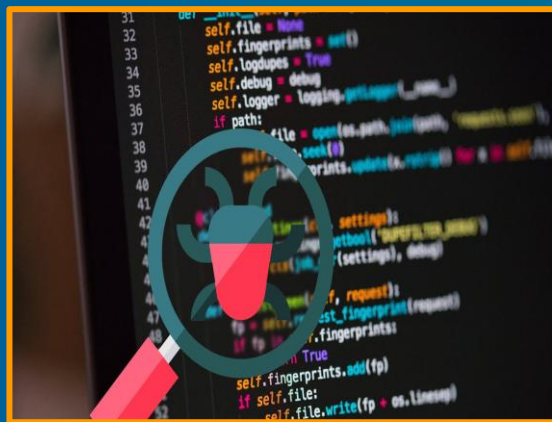
- Las funciones sirven para **descomponer una tarea específica en un bloque de código** que se puede llamar varias veces desde cualquier parte del programa.
- Las funciones permiten que el código sea **modular, legible y fácil de mantener**.

Objetivos de una función:



Minificación: Logramos que el programa sea más simple de comprender ya que cada función se dedica a realizar una tarea en particular.

Objetivos de una función:



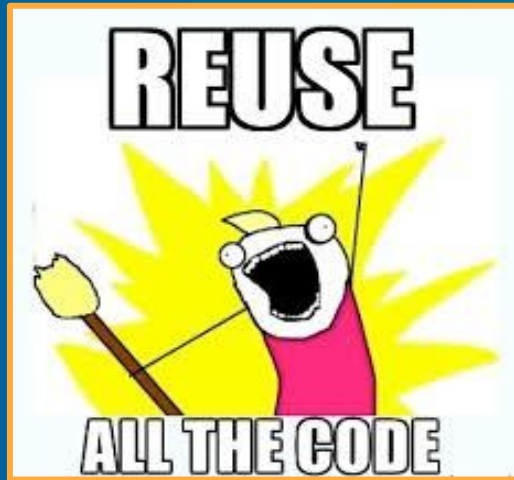
Depuración: La depuración queda acotada a cada función.

Objetivos de una función:



Modificación: Las modificaciones al programa se reducen a cada módulo.

Objetivos de una función:



Reutilización: Cuando cada función está bien probada, se la puede usar las veces que se quiera y así reutilizar código.

Objetivos de una función:



Independencia: Se obtiene una autonomía del código donde cada función es independiente de otra.

Componentes de una función:

Palabra reservada

Nombre

Parámetro

Parámetro opcional

```
def calcular_precio_con_iva(valor_sin_iva, iva=21):
```

```
    '''
```

```
    Documentacion
```

```
    '''
```

```
    resultado = valor_sin_iva * (1 + (iva / 100))
```

```
    return resultado
```

Documentación

Variable local

Valor del retorno

Desarrollo e invocación

```
def calcular_precio_con_iva(valor_sin_iva, iva = 21):  
    resultado = valor_sin_iva * (1 + iva / 100)  
    return resultado
```

Definición y
desarrollo

```
print(calcular_precio_con_iva(100))    # 121.0  
print(calcular_precio_con_iva(100, 10.5)) # 110.5
```

Invocación o
llamada

Palabras reservadas: **def** y **return**

El uso de **def** nos permite crear una función.
Podemos utilizar **return** para devolver un valor.

```
def sumar(numero_a, numero_b):  
    suma = numero_a + numero_b  
    return suma  
  
resultado = sumar(33, 2)  
print("El resultado de la suma es", resultado)
```

Nombre

Las funciones y las variables se deberán escribir en formato **snake_case**.

Las palabras separadas entre sí por barra baja (underscore) en vez de espacios y las letras en minúscula.

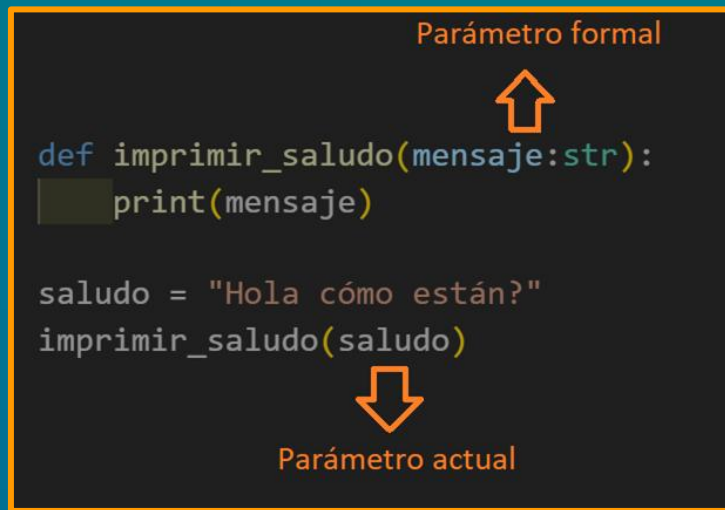
Las funciones representan una acción, por lo tanto, el nombre debe contener un verbo.

calcular_precio_con_iva()

Tipos de parámetros

Parámetros Formales: son variables locales declaradas en la definición de la función, que se utilizarán para recibir datos con los que va a operar la misma.

Parámetros Actuales: son las variables o datos que recibe la función al momento de ser invocada.



Parámetros por posición

La forma más intuitiva de trabajar con parámetros es pasandolos respetando la posición.

```
def restar(numero_a, numero_b):  
    diferencia = numero_a - numero_b  
    return diferencia  
  
print(restar(15, 5)) # 10
```

Parámetros por nombre

Otra forma de pasar parámetros a una función es asignando valores a las variables utilizando sus nombres.

```
def restar(numero_a, numero_b):  
    diferencia = numero_a - numero_b  
    return diferencia  
  
print(restar(numero_b = 5, numero_a = 15)) # 10
```


Parámetros opcionales

Python permite trabajar con parámetros opcionales o por defecto, estos deben ir siempre después de los obligatorios.

```
def restar(numero_a, numero_b = 5):  
    diferencia = numero_a - numero_b  
    return diferencia  
  
print(restar(15)) # 10  
print(restar(15, 7)) # 8
```

Parámetros y retorno con tipos

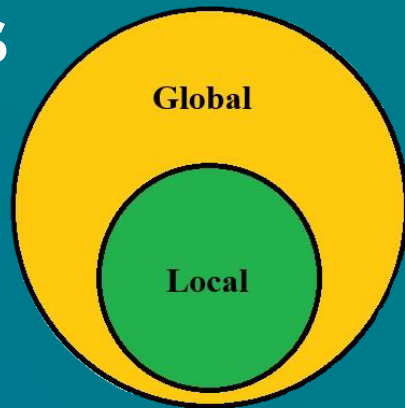
Se recomienda indicar que tipo de dato se espera que reciban los parámetros y que tipo de dato devuelve la función. * En Python estas indicaciones NO son restrictivas.

```
def restar(numero_a:int, numero_b:int = 5)-> int:  
    diferencia = numero_a - numero_b  
    return diferencia
```

```
print(restar(15)) # 10
```

Variables locales y globales

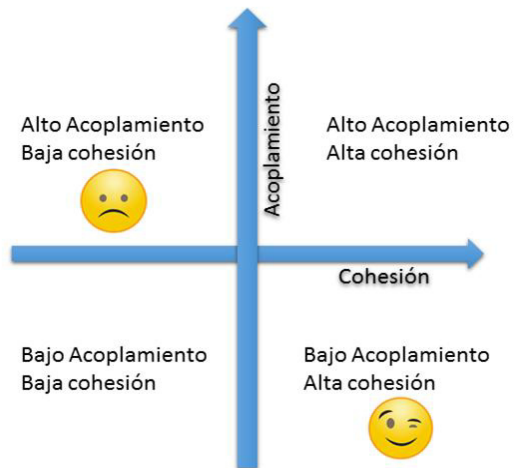
- Las variables **locales** son aquellas que se definen **dentro de una función** y solo son accesibles dentro de esa función.
- Las variables **globales** son aquellas que se definen **fuera de todas las funciones** y son accesibles desde cualquier parte del programa.



El **scope** (ámbito de la función) es donde las variables **locales** nacen y mueren, este mismo está separado por **sangría** al igual que las estructuras lógicas y repetitivas.

Nota: las variables globales, dentro de una función, pueden ser modificadas solo si se las marca como globales dentro del scope de la misma.

Cohesión y Acoplamiento :



Cohesión: Implica una conexión o unión que asegura que las partes de un todo estén bien integradas y funcionen juntas de manera efectiva.

Acoplamiento: Es la dependencia que tienen las funciones entre sí.

¿Qué es una Referencia?

Una referencia en programación es una forma de acceder a un valor mediante su **dirección de memoria** en lugar de interactuar directamente con el valor. Esto permite manipular el dato original sin crear copias adicionales, lo cual es eficiente en términos de recursos. Al utilizar una referencia, cualquier modificación realizada a través de ella afectará directamente al valor almacenado en esa dirección de memoria.

Pasaje por valor y por referencia en general :

En principio los conceptos de paso por **valor** y por **referencia** que aplican a la hora de **cómo** trata una función a los parámetros que se le pasan como entrada.

- Por VALOR: la función creará una copia local de la variable y cualquier modificación sobre ella no tendrá ningún efecto sobre la original.
- Por REFERENCIA: la función recibirá la dirección de memoria de la variable original y podrá realizar modificaciones directamente sobre ella.

Pasaje por valor y por referencia en Python :

En Python, **los parámetros de una función se pasan por referencia**, pero esta referencia es en realidad una referencia al objeto, no a la variable original. Esto significa que cuando se pasa una variable a una función, la función recibe una referencia al objeto al que apunta la variable.

Sin embargo, no puede modificar la variable original si el objeto al que apunta es **inmutable**, como números o cadenas de texto. En cambio, si el objeto es **mutable**, como una lista, un diccionario o una instancia de una clase, los cambios realizados dentro de la función afectarán al objeto original.

Pasaje por valor y por referencia

pass by reference



fillCup()

pass by value



fillCup()

Documentación

La documentación de una función es fundamental para la claridad, comprensión, y mantenibilidad del código, facilitando tanto el trabajo individual como la colaboración en equipos de desarrollo.

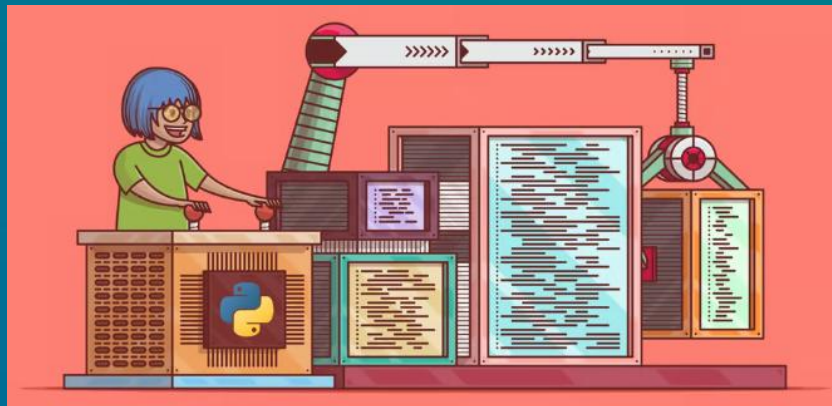
```
def sumar(numero_a:int, numero_b:int) -> int:
    """Que debe hacer la función
    Args:
        Que parametros recibe
    Returns:
        Que retorna
    """
    suma = numero_a + numero_b
    return suma
```

```
def sumar(numero_a:int, numero_b:int) -> int:
    """Funcion que se encarga de sumar dos
    números enteros

    Args:
        numero_a (int): Primer número
        numero_b (int): Segundo Número

    Returns:
        int: Devuelve la suma entre estos dos
        números
    """
    suma = numero_a + numero_b
    return suma
```

Módulos y paquetes



Módulos

A medida que los programas crecen, es necesario separarlo en varios archivos para que el mantenimiento sea más sencillo. Quizás también sea necesario usar una función ya escrita en otro programa sin la necesidad de copiar su definición en cada programa que la quiera utilizar. Para ello, los módulos son la solución.

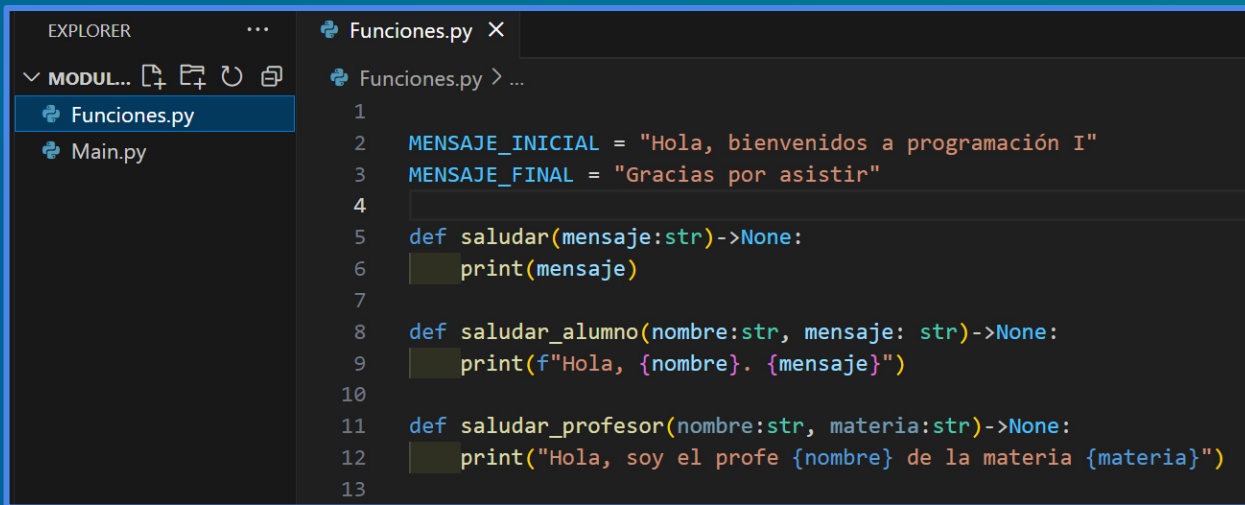


Generalidades

- Un módulo en Python es un archivo que contiene definiciones de funciones, clases y variables, junto con cualquier otro código ejecutable.
- Tienen la extensión .py.
- Pueden ser importados y utilizados en otros programas de Python.
- Ayudan a organizar y reutilizar el código al dividirlo en componentes más manejables y específicos.

Creación

Supongamos que definimos las siguientes funciones y variables para saludar en un archivo .py (en este caso nuestro módulo)

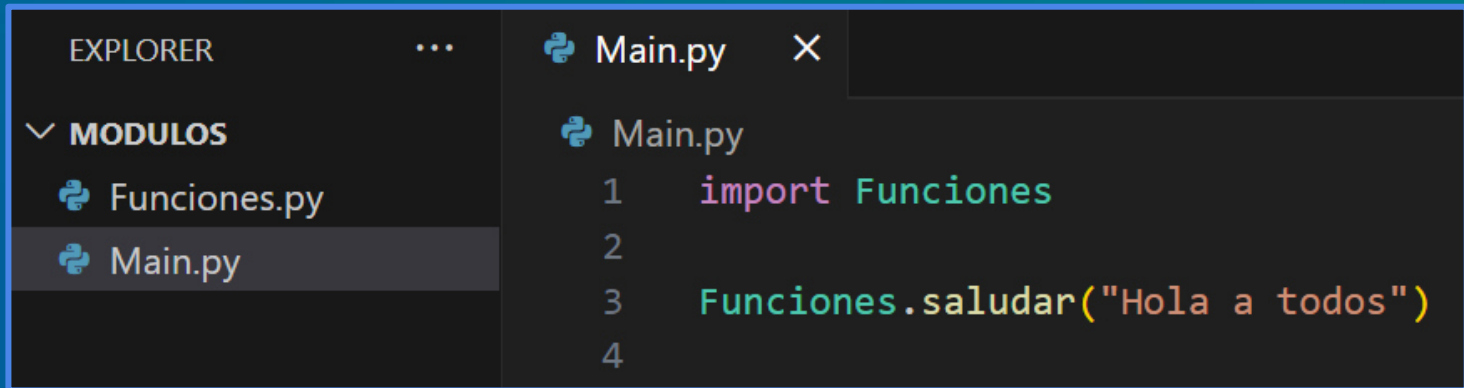


```
EXPLORER
└─ MODUL...
  └─ Funciones.py
  └─ Main.py

Funciones.py
1
2 MENSAJE_INICIAL = "Hola, bienvenidos a programación I"
3 MENSAJE_FINAL = "Gracias por asistir"
4
5 def saludar(mensaje:str)->None:
6     print(mensaje)
7
8 def saludar_alumno(nombre:str, mensaje: str)->None:
9     print(f"Hola, {nombre}. {mensaje}")
10
11 def saludar_profesor(nombre:str, materia:str)->None:
12     print("Hola, soy el profe {nombre} de la materia {materia}")
13
```

Importación

Para poder utilizar las funciones definidas en el módulo debemos importar dicho módulo en el archivo en donde queremos utilizar las funciones:



The screenshot shows a code editor with a dark theme. On the left, the 'EXPLORER' sidebar is visible, showing a folder named 'MODULOS' containing two files: 'Funciones.py' and 'Main.py'. 'Main.py' is selected. The main editor area shows the code in 'Main.py':

```
1 import Funciones
2
3 Funciones.saludar("Hola a todos")
4
```

En este caso, para utilizar cualquiera de las funciones definidas en el módulo, tenemos que anteponer el nombre del mismo antes de llamar a la función, seguido del operador punto.

Importación

Observemos las siguientes alternativas para importar módulos y sus componentes:

```
Main.py X
Main.py
1  from Funciones import saludar
2
3  saludar("Hola a todos")
4
```

Caso 1: desde el módulo Funciones, únicamente importamos la función saludar.

```
Main.py X
Main.py
1  from Funciones import saludar, saludar_alumno
2
3  saludar("Estamos probando modulos")
4
5  saludar_alumno("Pepe", "Tenes que estudiar mas!!!")
6
```

Caso 2: desde el módulo Funciones, importamos las funciones saludar y saludar_alumno.

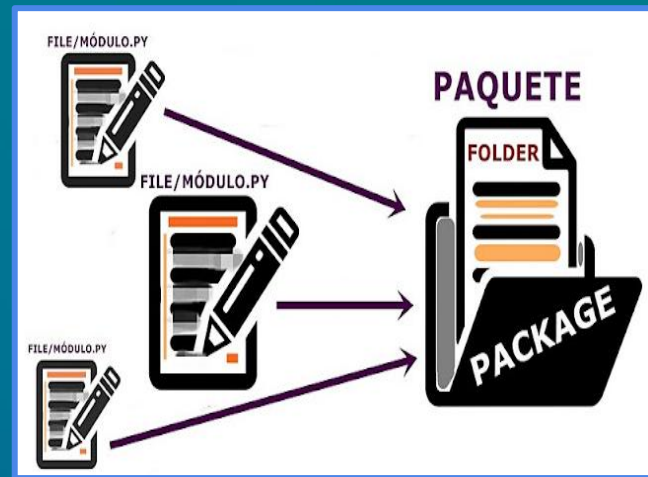
```
Main.py X
Main.py
1  from Funciones import *
2
3  saludar(MENSAJE_INICIAL)
4
5  saludar_profesor("Juan", "Química")
6
7  saludar_alumno("Pepe", "Tenes que estudiar mas!!!")
8
9  saludar(MENSAJE_FINAL)
```

*Caso 3: desde el módulo Funciones, importamos todas las funciones y variables globales con el operador *.*

Paquetes

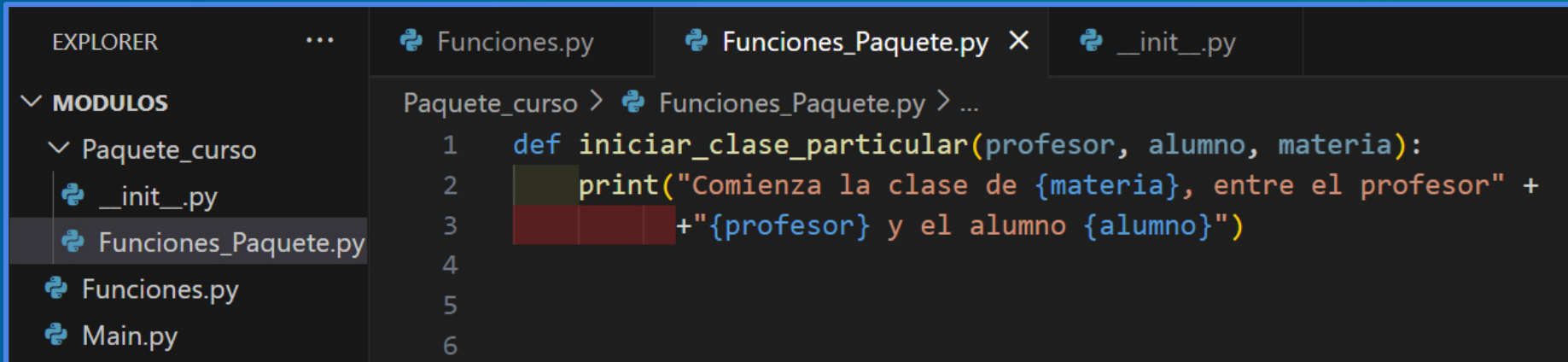
Un paquete en Python es una forma de estructurar y organizar módulos relacionados.

Permiten reutilizar código de manera más efectiva, al agrupar funcionalidades relacionadas en un solo lugar. Esto facilita la gestión de proyectos grandes y complejos al proporcionar una estructura jerárquica y modular para el código.



Creación

Para crear un paquete debemos crear una carpeta junto con un archivo `__init__.py` (de esta manera el intérprete de python entenderá que la carpeta se trata de un paquete). Luego agregamos en el mismo los módulos que necesitemos agrupar.



```
EXPLORER  ...  Funciones.py  Funciones_Paquete.py X  __init__.py

▼ MODULOS
  ▼ Paquete_curso
    __init__.py
    Funciones_Paquete.py
    Funciones.py
    Main.py

Paquete_curso > Funciones_Paquete.py > ...
1  def iniciar_clase_particular(profesor, alumno, materia):
2      print("Comienza la clase de {materia}, entre el profesor" +
3          "{profesor} y el alumno {alumno}")
4
5
6
```

Importación

De la misma forma que importamos módulos independientes, podremos importar el/los módulos necesarios que se encuentran en un paquete.

```
Main.py X
Main.py
1  #UTILIZAMOS UN ALIAS
2  import Paquete_curso.Funciones_Paquete as PC
3
4  PC.iniciar_clase_particular(profesor = "Pedro", alumno = "Juan", materia = "Matemática")
5
```

```
Main.py X
Main.py
1  from Funciones import * # Importando todas las funciones del módulo Funciones
2
3  # Importando la funcion iniciar_clase_particular del modulo Funciones_Paquete
4  # del paquete Paquete_curso
5  from Paquete_curso.Funciones_Paquete import iniciar_clase_particular
6
7  iniciar_clase_particular(profesor = "Pedro", alumno = "Juan", materia = "Matemática")
```

Comunicar módulos

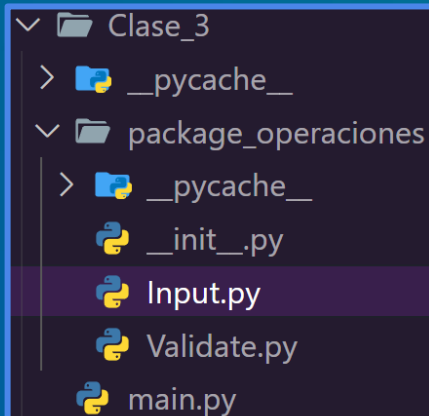
Si necesitamos comunicar 2 módulos de un mismo paquete como por ejemplo Input y Validate:

Desde Input deberemos importar Validate de la siguiente manera.

```
from .Validate import *
```

Esto se debe a que Python utiliza la convención de puntos para organizar los módulos en paquetes.

Comunicar módulos



```
1 from .Validate import * # Manera de importar entre módulos
2
3 def get_int():
4     validate_int() # Función de el módulo Validate
```

package

module

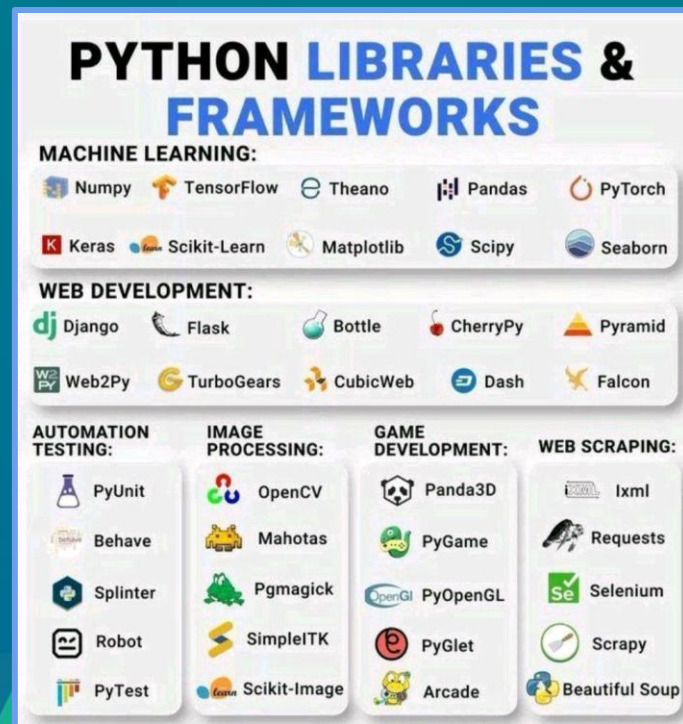
function

Son un conjunto de módulos y paquetes que se distribuyen junto con Python. Muchas de las operaciones más comunes ya se implementan en ellas.



Módulos de terceros

Es inmensurable la cantidad de módulos y paquetes desarrollados por la comunidad de python, cada uno de ellos con propósitos específicos.



Instalando módulos de terceros

Cada uno de estos paquetes está identificado por su nombre. Para instalar alguno de ellos utilizaremos una herramienta llamada pip que se incluye con la instalación de Python.

```
pip install paquete
```

En caso de que tu SO no reconozca pip:

```
python -m pip install paquete
```



```
py -m pip install paquete
```



[Si no pudiste solucionarlo, consulta esta página](#)



Probemos instalando el módulo pygame
Para asegurarnos que funciona correctamente,
corramos el siguiente comando:

```
[python | py] -m pygame.examples.aliens
```

