Programador de Videojuegos

CLASE 13/08/25

Importar métodos

Importar métodos o archivos

Importar métodos o archivos significa traer código que está en otro archivo (o módulo) para poder usarlo en el archivo actual, sin tener que reescribirlo. Es muy común cuando querés organizar tu código en módulos.

Conceptos claves a tener en cuenta:

- ➔ **Módulo:** cualquier archivo .py que contenga código reutilizable (funciones, clases, variables).
- ➔ **Paquete**: una carpeta que contiene varios módulos y un archivo __init__.py
  - ◆ ¿Qué es un archivo __init__.py? Es un archivo necesario para que Python entienda que determinada carpeta es un paquete. Puede estar vacío. Desde Python 3.3, no siempre es obligatorio, pero sigue siendo buena práctica.
- ➔ **Importar**: el proceso de decirle a Python que queremos usar un módulo o parte de él.

Para importar se utilizarán la palabra reserva **import** lo cual permite traer el módulo entero y utilizar sus elementos escribiendo el nombre_modulo.nombre_elemento

```
import nombre_modulo
```

o

```
import nombre_paquete.nombre_modulo
```

También se puede utilizar from en el import, para traer métodos específicos y utilizarlos directamente.

```
from nombre_modulo import nombre_elemento, nombre_elemento2
#Sería como decir, de este módulo específico, traé solo estas partes.
```

Importar módulo

Para importar un módulo completo en otro archivo, se debe utilizar:

import nombre_modulo_a_importar

Para usar algún método del módulo importado, se tiene que escribir: nombre_modulo_a_importar.elemento

Teniendo los siguientes archivos:

- utilidades.py: contiene funciones a utilizar
- principal.py: archivo donde se importaran las funciones

Código de los archivos:

utilidades.py

```python
def saludar(nombre):
    print(f"Hola, {nombre}")

```

principal.py

```python
import utilidades # Importa el módulo completo

def main():
    # Para usar algo del archivo importado, se debe poner: nombreModulo.funcion()
    utilidades.saludar("Probando")

if __name__ == "__main__":
    #Solo ejecuta este bloque si este archivo se está corriendo directamente,
    # no si se está importando
    main()

```

Importar módulo con alias

Al módulo importado se le puede asignar un alias.Esto es útil cuando el nombre del módulo es muy largo.

import nombre_modulo_a_importar as alias_del_modulo

Para usar algún método del módulo importado que tiene un alias, se tiene que escribir:
alias_modulo.elemento

utilidades.py

```python
def saludar(nombre):
    print(f"Hola, {nombre}")

```

principal_importando_modulo_con_alias.py

```python
import utilidades as ut # Importa el módulo completo asignándole un alias

def main():
    # Para usar algo del archivo importado, se debe poner: alias_modulo.funcion()
    ut.saludar("Probando")

if __name__ == "__main__":
    #Solo ejecuta este bloque si este archivo se está corriendo directamente,
    # no si se está importando
    main()

```

Importar funciones específicas

Para importar un módulo completo en otro archivo, se debe utilizar:

from nombre_modulo import nombre_elemento

o

from nombre_paquete.nombre_modulo import nombre_elemento

Esto carga solo la función especificada, no todo el módulo.
Para utilizar el elemento importado, se usa directamente sin el prefijo del módulo.

- También se le puede adicionar un alias: **from nombre_modulo import nombre_elemento as alias_elemento**
- Y para adicionar varias funciones, se las separa con coma:

from nombre_modulo import nombre_elemento1, nombre_elemento2

utilidades.py

```python
def saludar(nombre):
    print(f"Hola, {nombre}")

```

principal_import_funcion.py

```python
from utilidades import saludar # Importa la función saludar del módulo utilidades

def main():
    # Simplemente llamamos a la función importada
    saludar("Probando")

if __name__ == "__main__":
    #Solo ejecuta este bloque si este archivo se está corriendo directamente,
    # no si se está importando
    main()

```

Importar módulo desde paquete/carpeta

Para importar un módulo completo en otro archivo, se debe utilizar:

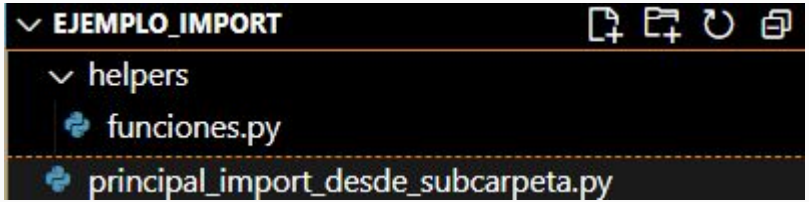
import nombre_paquete.nombre_modulo

o

from nombre_paquete. nombre_modulo import nombre_elemento

Teniendo los siguientes archivos:

- funciones.py: archivo dentro de la carpeta helpers, el mismo contiene funciones a utilizar
- principal.py: archivo donde se importaran las funciones



Código de los archivos:

utilidades.py

```python
def saludar(nombre):
    print(f"Hola, {nombre}")

```

principal_import_desde_subcarpeta.py

```python
#Como el archivo funciones está dentro de una carpeta, pongo el nombre del paquete.archivo
from helpers.funciones import saludar

def main():
    # Ejecuto la función saludar
    saludar("Probando función importada desde subcarpeta")

if __name__ == "__main__":
    #Solo ejecuta este bloque si este archivo se está corriendo directamente,
    # no si se está importando
    main()
```

__name__

__name__ hace alusión al nombre que Python le asigna al archivo (módulo) que se está ejecutando o importando. Es decir, hace alusión al rol que está cumpliendo el archivo en ese momento.
Es como una etiqueta interna que le pregunta a Python "¿quién soy yo en este momento?". El valor de esa etiqueta cambia según cómo se está usando el archivo.

Si el valor de **__name__** es igual a:

- "__main__" : el archivo/módulo lo estás ejecutando como programa principal.
- "nombre_del_modulo" : estás usando el archivo/módulo como módulo importado.

Por eso, en el siguiente caso:

__name__ == "__main__"

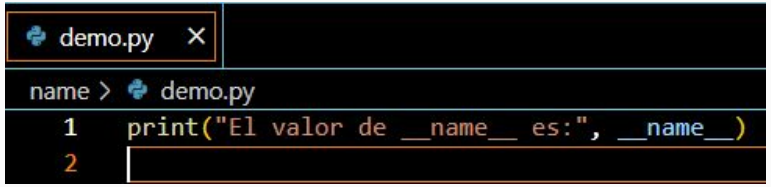
"__main__" es un nombre especial reservado que indica “este es el punto de entrada del programa”.

Esto sirve para evitar que se ejecute main() si este archivo es importado por otro.

__name__

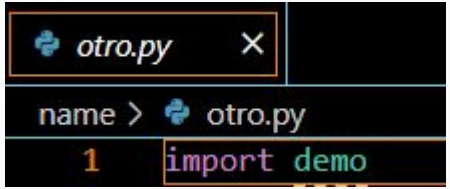
Ejemplo para ver qué valores toma name

Teniendo el archivo demo, el cual contiene un print de name. Al ejecutarlo, vemos que name es igual a main.




```
El valor de __name__ es: __main__
```

En cambio, si importamos demo desde otro archivo y ejecutamos ese archivo, el valor de name es igual a demo, es decir, es igual al nombre del módulo importado.




```
El valor de __name__ es: demo
```

Ejemplo de estructura de un proyecto en Python

- assets: Carpeta que contiene los recursos del juego
  - images/ : Sprites, fondos, iconos
  - sounds/ : Efectos de sonido, música
  - fonts/ : Tipografías usadas
- src/ : Contiene todo el código fuente de la aplicación.
  - main.py : Punto de entrada del juego, inicia el juego
  - settings.py : Centraliza las configuraciones globales (parámetros como resolución, velocidad, colores, etc.)
  - classes/ : Carpeta que contiene las clases del paradigma orientado a objetos
    - game.py : Clase con la lógica del juego
    - player.py : Clase del jugador
    - enemy.py : Clase del enemigo
  - helpers/ : Carpeta que agrupa archivos de funciones auxiliares.
    - utils.py : Archivo con funciones auxiliares
    - matematicas.py : Archivo con funciones auxiliares de calculos matemáticos
- tests/ : Carpeta que agrupa pruebas de lógica (no de renderizado). Contendrá archivos de pruebas (tests) usando unittest, pytest, etc.
  - test_player.py
- requirements.txt : Lista de dependencias del proyecto (pygame, etc.). Es decir, los paquetes necesarios para ejecutar el proyecto.
- README.md: Descripción general del proyecto.
- .gitignore: Archivos/carpetas que Git (repositorio) debe ignorar.

Algunas buenas prácticas

- Usar src/ o el nombre del proyecto como carpeta principal del código fuente.
- Organización por responsabilidades: dividir en módulos según la lógica del programa.
- Evitar archivos gigantes.
- Reutilizar código.
- Colocar pruebas en una carpeta tests/ paralela al código.
- Usar __init__.py para declarar paquetes.
- Evitar código suelto en el nivel raíz del proyecto.
- Crear un main.py o app.py como punto de entrada.

Ejercitación: Import

Ejercicio 1: Importar un módulo

1. Crear el archivo jugador.py, el cual contendrá dos métodos uno con un mensaje de bienvenida y otro con un menú de inicio el cual tiene las opciones iniciar juego y salir.
2. Crear un archivo main.py, importar todo el módulo jugadores usando import.
3. Llamar a los métodos del módulo importado.

Ejercicio 2: Importar función específica

4. Crear otro archivo e importar sólo la función del mensaje de bienvenida.
5. Ejecutar la segunda función sin haberla importado y ver qué sucede.

Ejercicio integrador

Ejercicio integrador: Piedra, papel o tijera

Desarrollar el código para generar el juego del piedra, papel o tijera donde el usuario compita contra la computadora teniendo hasta 3 intentos.

Reglas:

- El jugador elige entre las opciones "piedra", "papel" o "tijera".
- La computadora elige aleatoriamente entre las tres opciones.
- Se determina quién gana cada ronda según las reglas clásicas:
  - Piedra gana a tijera.
  - Tijera gana a papel.
  - Papel gana a piedra.
- El juego termina después de 3 rondas. Se debe informar:
  - Cuántas ganó el jugador.
  - Cuántas ganó la PC.
  - Cuántas hubo empate.
  - Y finalmente quién ganó.

Particionar el código, generando las funciones necesarias en el archivo funciones.py e importarlas al archivo main.py. Cabe mencionar que las funciones contenidas en el archivo funciones.py no se llamarán desde dicho archivo, solo se ejecutarán desde main.py.