

MÓDULO 4. PAQUETES, MÓDULOS Y NAMESPACES

Loreto Pelegrín Castillo



MINISTERIO
DE EDUCACIÓN
Y FORMACIÓN PROFESIONAL



Índice:

1. Introducción
2. Módulos, paquetes y namespaces
3. Algunos módulos útiles
4. Página principal o main
5. Docstrings

1

Introducción

1. Introducción

- En esta unidad, se va a aprender a programar utilizando módulos, paquetes y namespaces.
- Es una manera de organizar el código, ya que conforme más complejo sea el programa, una mayor organización, será mas útil y clara.

2

Módulos, paquetes y namespaces

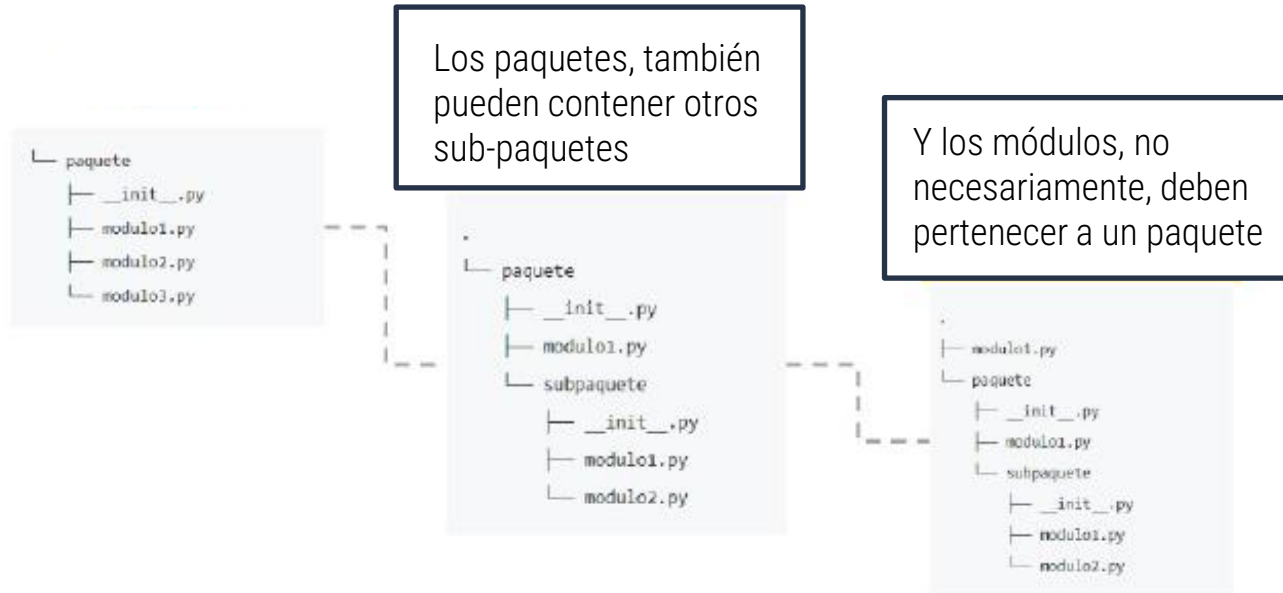
2.1. Módulo

- Un **módulo** es un archivo de Python cuya extensión es `.py`.
- Los objetos (funciones, clases, excepciones, etc.) que contiene dicho archivo pueden ser accedidos desde otro archivo.
- Se trata simplemente de una forma de organizar grandes códigos.

2.2. Paquete

- Un **paquete** es una carpeta que contiene archivos .py, pero para que una carpeta pueda ser considerada un paquete debe contener un archivo de inicio llamado `__init__.py`.
- Este archivo, no necesita contener ninguna instrucción, puede estar completamente vacío.

2.2. Paquete



2.3. Importando módulos

- El contenido de cada módulo puede ser utilizado a la vez por otros módulos.
- Para ellos es necesario importar los módulos que se quieran utilizar.
- Para importar, se utiliza la palabra reservada **import**:

```
import modulo # Importar un modulo dentro del mismo paquete
```

```
import paquete1.modulo1 #Importar un modulo que esta dentro de un paquete
```

```
import paquete1.subpaquete.modulo1 #Importar un modulo que esta dentro de un paquete y a su vez dentro de otro subpaquete
```

```
EjemploModulo > paquete1 > subpaquete >  modulo1.py
```

2.4. Namespaces

- En Python, el concepto de namespace, es el nombre que se ha indicado luego de la palabra import, es decir, la ruta (namespace) del módulo.
- Para acceder a cualquier elemento del módulo importado, se realiza mediante el namespace seguido de un punto (.) y el nombre del elemento que se desea obtener.

Namespace

```
import modulo  
import paquete.modulo1  
import paquete.subpaquete.modulo1
```

Elementos de los módulos importados

```
print modulo.CONSTANTE_1  
print paquete.modulo1.CONSTANTE_1  
print paquete.subpaquete.modulo1.CONSTANTE_1
```

2.4.1. Namespaces: Alias

- Se puede abreviar los namespaces mediante un alias. Para ellos, durante la importación, se asigna la palabra clave **as** seguida del alias asociado al namespace.

```
import modulo as m
import paquete.modulo1 as pm
import paquete.subpaquete.modulo1 as psm
```

```
print m.CONSTANTE_1
print pm.CONSTANTE_1
print psm.CONSTANTE_1
```

3

Algunos módulos utiles

3. Módulos de Python

- En la biblioteca de Python hay módulos ya contruidos que se actualizan continuamente y cuyo numero crece cada día.
- Para poder usarlos, es necesario importarnos dentro de nuestros archivos.

3.1. Módulo random

- El módulo **random** en Python es una herramienta esencial cuando necesitas introducir un elemento de **aleatoriedad** en tus programas.
- Este módulo proporciona una variedad de funciones que te permiten generar números pseudoaleatorios de diferentes tipos y distribuciones.
- <https://docs.python.org/es/3.10/library/random.html>

3.1. Módulo random

Función	Descripción
<code>random.choice()</code>	Retorna un valor aleatorio dentro de una secuencia que le pasemos como argumento
<code>random.randint()</code>	Devuelve el valor aleatorio dentro de un intervalo formado por 2 números enteros que pasamos como argumento. El primer numero debe de ser menor que el anterior o dará error.
<code>random.shuffle()</code>	Realiza una mezcla los elementos de una lista en un orden aleatorio.
<code>random.random()</code>	Devuelve un número aleatorio en el intervalo real que incluye el 0 y un número menor que 1.

3.1. Módulo random

```
import random

lista_numeros = [125,624,56.7,-98,42]
numero_aleatorio = random.choice(lista_numeros)
print(numero_aleatorio) #56.7

lista_paises = ["España","Francia","Italia","Grecia","Alemania","Belgica"]
pais_aleatorio = random.choice(lista_paises)
print(pais_aleatorio) #España

numero_aleatorio = random.randint(1,500)
print(numero_aleatorio) #201

numero_aleatorio = random.random()
print(numero_aleatorio) #0.6817194127678036
```


3.2. Modulo math

- Este módulo provee de funciones matemáticas muy útiles para trabajar con números reales.
- Devolverá casi siempre números del tipo float.
- <https://docs.python.org/es/3.10/library/math.html>

3.2. Módulo math

Función	Descripción
<code>math.pi()</code>	Nos provee del valor de la constante pi.
<code>math.sin()</code>	Esta función nos devuelve el seno del ángulo cuyo valor pasemos como argumento a radianes.
<code>math.pow()</code>	Elevará el número a la potencia.

3.2. Módulo math

```
import math

print(math.pi) #3.141592653589793

print(math.sin(89)) #0.8600694058124532

print(math.pow(2,4)) #16
```

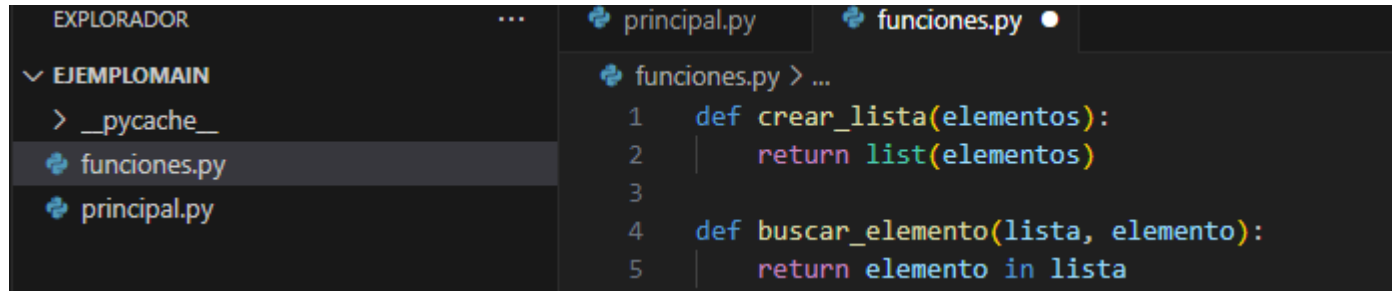
3.3. Más módulos

- **Modulo string:** Permite trabajar con cadenas de texto más fácilmente.
 - ▷ <https://docs.python.org/es/3/library/string.html>
- **Modulo re:** Este módulo sirve para gestionar las expresiones regulares.
 - ▷ <https://docs.python.org/es/3/library/re.html>
- **Modulo datetime:** proporciona clases para manipular fechas y horas.
 - ▷ <https://docs.python.org/es/3.9/library/datetime.html>

4

**Página principal o
main**

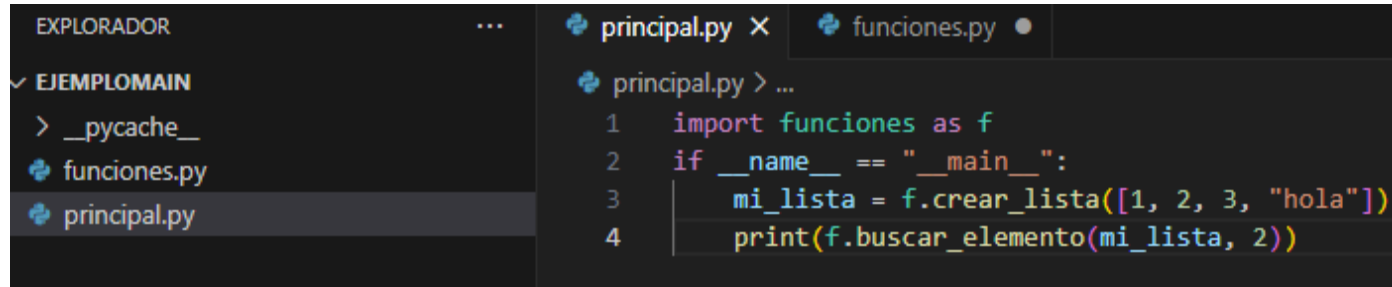
4. Página principal o main



This screenshot shows the Visual Studio Code editor with the 'funciones.py' file open. The Explorer sidebar on the left shows the project structure under 'EJEMPLMAIN', including '__pycache__', 'funciones.py', and 'principal.py'. The 'funciones.py' file is selected, and its content is displayed in the editor window.

```
EXPLORADOR    ...    principal.py    funciones.py ●
└─ EJEMPLMAIN
  > __pycache__
  funciones.py
  principal.py

funciones.py > ...
1  def crear_lista(elementos):
2      return list(elementos)
3
4  def buscar_elemento(lista, elemento):
5      return elemento in lista
```



This screenshot shows the Visual Studio Code editor with the 'principal.py' file open. The Explorer sidebar on the left shows the project structure under 'EJEMPLMAIN', including '__pycache__', 'funciones.py', and 'principal.py'. The 'principal.py' file is selected, and its content is displayed in the editor window.

```
EXPLORADOR    ...    principal.py ✕    funciones.py ●
└─ EJEMPLMAIN
  > __pycache__
  funciones.py
  principal.py

principal.py > ...
1  import funciones as f
2  if __name__ == "__main__":
3      mi_lista = f.crear_lista([1, 2, 3, "hola"])
4      print(f.buscar_elemento(mi_lista, 2))
```

4. Página principal o main

■ El ejemplo anterior muestra en el archivo principal.py la expresión:

▷ `if __name__ == "__main__"`

■ Esta expresión es necesaria para ejecutar el programa, solamente en el modulo donde se encuentre, ya que toma por defecto que ese modulo se va a llamar main.

4. Página principal o main

- **Cuando se ejecuta un script directamente:** La variable `__name__` toma el valor `"__main__"`. Esto significa que el código dentro del bloque `if` se ejecutará.
- **Cuando se ejecuta en otro módulo:** La variable `__name__` toma el nombre de ese módulo. En este caso, la condición `__name__ == "__main__"` será falsa y el código dentro del bloque `if` no se ejecutará.
 - ▶ En el ejemplo anterior, si se pulsa ejecutar en el módulo `funciones.py`, no ocurrirá nada en el programa.

5

Docstrings

5. Docstrings

- Una buena practica dentro de la programación consiste en comentar el código que se realiza.
- Y para ser más específicos, cuando se comentan funciones se utiliza el termino **docstrings**.
- Son comentarios explicativos que van dentro de 3 comillas dobles, al inicio de la función.

5. Docstrings

Se suele escribir la documentación después de crear la función, la información que se declara es:

- ▶ Que funcionalidad tiene la función.
- ▶ Los argumentos o parámetros que recibe.
- ▶ Y la devolución que realiza.

```
def crear_lista(elementos):  
    """  
    Crear una lista nueva  
  
    Argumentos o Parametros o Args:  
    |     elementos: objeto iterable (lista, tupla, cadena)  
  
    Return:  
    |     lista nueva creada a partir de elementos  
    """  
    return list(elementos)
```

5. Docstrings

- En caso de que sea una función que no reciba parámetros o que no devuelva nada. Esa información no es necesaria.

```
def hola():  
    """Funcion para imprimir Hola"""  
    print("Hola")
```

5. Docstrings

- Se puede escribir en la documentación, la información como se desee.
- Para ver la documentación:
 - A la hora de llamarla se pasa el ratón por encima del nombre de la función aparece la información en una ventana al lado de esta.

```
principal.py > ...
1  import funciones as f
2
3  if __name__ == "__main__":
4      mi_lista = f.crear_lista([1, 2, 3, 4])
5      print(f.busca
6
7
8
```

(function) def crear_lista(elementos: Any) -> list

Crear una lista nueva

Argumentos o Parametros o Args:

elementos: objeto iterable (lista, tupla, cadena)

Return:

lista nueva creada a partir de elementos

5. Docstrings

- ▶ Otra forma de acceder a la documentación, es a través de la función `__doc__`, detrás de nuestra función. En este caso, mostrará la información escrita en la función.

```
principal.py > ...
1  import funciones as f
2
3  if __name__ == "__main__":
4      mi_lista = f.crear_lista([1, 2, 3, 4])
5      print(f.buscar_elemento(mi_lista, 2))
6
7      print(f.crear_lista.__doc__)
```

```
PS C:\Users\narue\Desktop\Academia Avanza\Python\1.Programación Python - IFCD32\Unidad_4\EjemploMain>
Python\1.Programación Python - IFCD32\Unidad_4\EjemploMain\principal.py"
True

    Crear una lista nueva

Argumentos o Parametros o Args:
    elementos: objeto iterable (lista, tupla, cadena)

Return:
    lista nueva creada a patir de elementos
```

5. Docstrings

- ▶ Con la función **help**, también se puede acceder a la información de la función documentada.

```
principal.py > ...  
1  import funciones as f  
2  
3  if __name__ == "__main__":  
4      mi_lista = f.crear_lista([1, 2, 3, 4])  
5      print(f.buscar_elemento(mi_lista, 2))  
6  
7      help(f.crear_lista)
```

```
Help on function crear_lista in module funciones:  
  
crear_lista(elementos)  
    Crear una lista nueva  
  
Argumentos o Parametros o Args:  
    elementos: objeto iterable (lista, tupla, cadena)  
  
Return:  
    lista nueva creada a partir de elementos
```

5. Docstrings

- ▶ La función `help` también funciona para funciones nativas de Python. Por ejemplo:

```
help(len)
```

- ▶ Y por pantalla nos muestra lo que realiza la función.

```
Help on built-in function len in module builtins:  
  
len(obj, /)  
    Return the number of items in a container.
```




¡Gracias!