




UNIDAD 5

Módulos, paquetes y manejo de excepciones.

Módulos

Son ficheros `.py` que albergan un conjunto de funciones, variables o clases y que puede ser usado en nuevos programas o códigos.

Una vez definido, los módulos pueden ser usados o importados en otro archivos. Usando *import* es posible traer todo el contenido.

<pre># mimodulo.py def suma(a, b): return a + b def resta(a, b): return a - b</pre>		<pre># otromodulo.py import mimodulo print(mimodulo.suma(4, 3)) # 7 print(mimodulo.resta(10, 9)) # 1</pre>
--	---	--

Es posible importar únicamente los componentes que se necesiten usando la palabra clave *from*:

```
from mimodulo import suma, resta
print(suma(4, 3))    # 7
print(resta(10, 9)) # 1
```

Es posible importar todo el módulo haciendo uso de ***, para usar directamente las definiciones:

```
from mimodulo import *
print(suma(4, 3))    # 7
print(resta(10, 9)) # 1
```

Para hacer que un módulo esté visible para cualquier archivo o nuevo código se debe ubicar en la carpeta *Lib* dentro del directorio de instalación de Python.

Paquetes

Un paquete es una carpeta o directorio que contiene varios módulos. Para crearlo se debe crear el directorio y crear un fichero especial init vacío (`__init__.py`) en el directorio donde se tenga todos los módulos que se quiere agrupar. De esta forma cuando Python recorra este directorio será capaz de interpretar una jerarquía de módulos:

```
paquete/  
  __init__.py  
  saludos.py  
  script.py
```

```
matematica/  
  |-- __init__.py  
  |-- aritmetica.py  
  |-- geometria.py
```

Esta jerarquía se puede expandir tanto como se necesite creando subpaquetes, pero siempre añadiendo el fichero init en cada uno de ellos:

```
script.py  
paquete/  
  __init__.py  
  hola/  
    __init__.py  
    saludos.py  
  adios/  
    __init__.py  
    despedidas.py
```

De forma sencilla se puede ejecutar las funciones y métodos de los módulos de cada subpaquete:

```
from paquete.hola.saludos import saludar  
from paquete.adios.despedidas import Despedida  
  
saludar()  
Despedida()
```

Errores y excepciones.

Se trata de una forma de controlar el comportamiento de un programa cuando se produce un error o una forma anormal en la ejecución. Hay al menos dos tipos diferentes de errores: errores de sintaxis y excepciones

Errores de sintaxis

Los errores de sintaxis se deben a la interpretación dentro de Python.

```
>>> while True print 'Hola Mundo'
Traceback (most recent call last):
...
    while True print 'Hola Mundo'
                        ^
SyntaxError: invalid syntax
```

Excepciones

Son errores detectados durante la ejecución, y no son incondicionalmente fatales. La mayoría de las excepciones no son manejadas por los programas, y resultan en mensajes de error.

```
>>> 10 * (1/0)
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
ZeroDivisionError: integer division or modulo by zero
>>> 4 + spam*3
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
NameError: name 'spam' is not defined
>>> '2' + 2
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
TypeError: cannot concatenate 'str' and 'int' objects
```

Tipos de Excepciones

TypeError : Ocurre cuando se aplica una operación o función a un dato del tipo inapropiado.

ZeroDivisionError : Ocurre cuando se intenta dividir por cero.

OverflowError : Ocurre cuando un cálculo excede el límite para un tipo de dato numérico.

IndexError : Ocurre cuando se intenta acceder a una secuencia con un índice que no existe.

KeyError : Ocurre cuando se intenta acceder a un diccionario con una clave que no existe.

FileNotFoundError : Ocurre cuando se intenta acceder a un fichero que no existe en la ruta indicada.

ImportError : Ocurre cuando falla la importación de un módulo.

Manejo : try - except.

Para prevenir el fallo se debe poner el código propenso a error en un bloque *try* y luego encadenar un bloque *except* para tratar la situación excepcional mostrando que ha ocurrido un fallo:

```
try:
    n = float(input("Introduce un número: "))
    m = 4
    print(n/m)
except:
    print("Ha ocurrido un error, introduce bien el número")
```

Manejo else - finally.

Mediante el uso de else, agregando el código en el bloque se ejecutará si no ha ocurrido ninguna excepción

```
try:
    # Forzamos una excepción al dividir entre 0
    x = 2/0
except:
    print("Entra en except, ha ocurrido una excepción")
else:
    print("Entra en else, no ha ocurrido ninguna excepción")

#Entra en except, ha ocurrido una excepción
```

```
try:
    # La división puede realizarse sin problema
    x = 2/2
except:
    print("Entra en except, ha ocurrido una excepción")
else:
    print("Entra en else, no ha ocurrido ninguna excepción")

#Entra en else, no ha ocurrido ninguna excepción
```

Al usar *finally* el bloque se ejecutara siempre, haya o no haya habido excepción.

```
try:
    # Forzamos excepción
    x = 2/0
except:
    # Se entra ya que ha habido una excepción
    print("Entra en except, ha ocurrido una excepción")
finally:
    # También entra porque finally es ejecutado siempre
    print("Entra en finally, se ejecuta el bloque finally")

#Entra en except, ha ocurrido una excepción
#Entra en finally, se ejecuta el bloque finally
```



ESCUELA
POLITÉCNICA
NACIONAL



ESCUELA
POLITÉCNICA
NACIONAL



python™