# Módulos

# Módulos en Python

- Las colecciones de funciones y constantes en Python se llaman módulos (en otros lenguajes se conocen como librerías).
- Los módulos aumentan significativamente la potencia del lenguaje.
- En la mayoría de los casos, los módulos se pueden instalar a través de los repositorios oficiales.

# Módulos estándar

- Existe una colección de módulos preinstalados en Python que conforman la librería estándar de Python.
- Algunos de estos módulos son:
  - math, statistics: matemáticas básicas.
  - collections: colecciones de estrucuras.
  - itertools: herramientas de iteración.
  - csv: lectura y escritura de archivos csv.
  - time, datetime, calendar:  gestión del tiempo.
  - Muchos más.

# Trabajar con módulos

- La forma más sencilla de trabajar con módulos es importarlos y acceder a sus miembros uno a uno con el operador de acceso:

```
import math

print(math.sqrt(4)) → 2
```

- Otra forma es importar solo los miembros de del módulo uno a uno:

```
from math import sqrt

print(sqrt(4)) → 2
```

# Importar con alias

- Podemos importar un módulo o miembro de un módulo y renombrarlo con un alias:

  `import numpy as np → np.array(...)`

  `from math import sqrt as root → root(...)`

- Es especialmente útil con nombres de módulos largos.

  `import matplotlib as mplot`

- Podemos importar todo el contenido de un módulo usando * pero no es recomendable:

  `from math import * → sqrt(...)`

# Estructura de un módulo

- Un módulo simple no es más que un script con una colección de funciones y constantes.
- Siendo así, podemos crear fácilmente nuestro propio módulo.
- Python solo puede importar los módulos contenidos en su path, que incluye:
  - Módulos en su directorio (se instalan a través del administrador de paquetes).
  - El directorio de trabajo.

# Módulo simple

- Vamos a crear un módulo simple:

```
# modulo.py
def comp(a, b):
    if a > b:
        return 1
    elif a < b:
        return -1
    else:
        return 0
```

```
# main.py
import modulo
a = input('Introduzca a:')
b = input('Introduzca b:')
c = modulo.comp(a, b)
```

- En este caso los script tiene que estar en la misma carpeta.

# Paquete ≠ módulo

- Cuando hablamos de módulos estamos hablando de scripts concretos.
- Un paquete se conforma de uno o más módulos que tienen una utilidad común. Los módulos de los paquetes suelen formar un árbol jerárquico al que se puede acceder a través del operador de acceso (.). Ejemplo de esto es por ejemplo el paquete scipy.

```
import scipy  # importamos el módulo de scipy.

import scipy.optimize  # importamos el módulo
scipy.optimize

import scipy.stats # importamos el módulos numpy.stats
```

- Todos esos módulos se instalan a la vez cuab.

# Instalar paquetes

- Python tiene un gestor de paquetes llamado pip que permite instalar rápidamente paquetes y disponer siempre de la última versión. Se usa desde la terminal (ej. en windows):

  ```
  >> python -m pip install <modulo>
  ```

- Además la distribución de anaconda tiene un repositorio propio llamado conda. La ventaja que tiene es que permite limitar los paquetes instalados al entorno de anaconda.

  ```
  >> conda install <modulo>
  ```

# Ejemplo de instalación

- Vamos a instalar un módulo de ejemplo. El módulo en cuestión es chempy.
- Chempy es una colección de utilidades relacionadas con la química.

  Con pip:

  ```
  >> python -m pip install chempy
  ```

  Con conda:

  ```
  >> conda install chempy
  ```

- Info del módulo: https://pypi.org/project/chempy/

# Ejemplo de instalación

- Ya podemos usar sin problema el paquete:

```
from chempy import Substance

ferricianuro =
Substance.from_formula('Fe(CN)6-3')

print(ferricianuro.unicode_name) → Fe(CN)₆³⁻

print(f'{ferricianuro.mass’) → 211.955
```

# Efecto de la importación

- Cuando importamos un módulo el flujo del programa pasa por el mismo incluyendo en el ámbito global todo lo que encuentra. Esto puede llevar a problemas, por ejemplo el programa:

modulo.py

```
a = 4

def foo(x):
    return x**2

print('hola')
```

main.py

```
import modulo

print('2^2 =', modulo.foo(2))
```

- Producirá la sigueinte salida:

```
hola
2^2 = 4
```

# Estructura del programa con módulos

- Para evitar que se ejecuten instrucciones indeseadas al importar un módulo todos nuestros scripts deben tener esta estructura:
  - Importación de módulos.
  - Definición de funciones.
  - Código principal con la siguiente estructura:

```
if __name__ == '__main__':
    # código principal.
```

# Estructura del programa con módulos

- __name__ es una variable que te indica el nombre del módulo en el que se encuentra el flujo.
- El nombre del script lanzado primero siempre se llama '__main__'.
- De esta forma si usamos esta estructura siempre podremos importar scripts como módulos sin que se ejecuten instrucciones indeseadas.

# Estructura del programa con módulos

- Realiza un programa que calcule el tiempo de ejecución de una función (módulo time).
- Compara tu método con el usado en el módulo timeit. [Con profesor]