Programación en Python

Clase 2.2: Uso de código externo

25/03/2026

Motivación

Supongamos que necesitamos:

- Calcular una raíz cuadrada

Motivación

Supongamos que necesitamos:

- Calcular una raíz cuadrada
- Generar números aleatorios

Motivación

Supongamos que necesitamos:

- Calcular una raíz cuadrada
- Generar números aleatorios
- Obtener un promedio

Motivación

Supongamos que necesitamos:

- Calcular una raíz cuadrada
- Generar números aleatorios
- Obtener un promedio

Problema

No hay función *built-in* que realice esas tareas en Python.

Motivación

Supongamos que necesitamos:

- Calcular una raíz cuadrada
- Generar números aleatorios
- Obtener un promedio

! Problema

No hay función *built-in* que realice esas tareas en Python.

? ¿Qué hacemos?

¿Tiene sentido programarlas desde cero?

Motivación

Supongamos que necesitamos:

- Calcular una raíz cuadrada
- Generar números aleatorios
- Obtener un promedio

! Problema

No hay función *built-in* que realice esas tareas en Python.

? ¿Qué hacemos?

¿Tiene sentido programarlas desde cero? **Por supuesto que no**.

Módulos y paquetes

Módulos y paquetes

- Cuando usamos código externo lo hacemos a través de módulos y paquetes.

Módulos y paquetes

- Cuando usamos código externo lo hacemos a través de módulos y paquetes.
- Un **módulo** es un archivo de Python, por ejemplo `modulo.py`.

Módulos y paquetes

- Cuando usamos código externo lo hacemos a través de módulos y paquetes.
- Un **módulo** es un archivo de Python, por ejemplo `modulo.py`.
- Un **paquete** es una colección de módulos organizados de manera particular.

Módulos y paquetes

- Cuando usamos código externo lo hacemos a través de módulos y paquetes.
- Un **módulo** es un archivo de Python, por ejemplo `modulo.py`.
- Un **paquete** es una colección de módulos organizados de manera particular.

Módulos y paquetes

- Cuando usamos código externo lo hacemos a través de módulos y paquetes.
- Un **módulo** es un archivo de Python, por ejemplo `modulo.py`.
- Un **paquete** es una colección de módulos organizados de manera particular.

¿Para qué existen?

Módulos y paquetes

- Cuando usamos código externo lo hacemos a través de módulos y paquetes.
- Un **módulo** es un archivo de Python, por ejemplo `modulo.py`.
- Un **paquete** es una colección de módulos organizados de manera particular.

¿Para qué existen?

- Para no repetir código.

Módulos y paquetes

- Cuando usamos código externo lo hacemos a través de módulos y paquetes.
- Un **módulo** es un archivo de Python, por ejemplo `modulo.py`.
- Un **paquete** es una colección de módulos organizados de manera particular.

¿Para qué existen?

- Para no repetir código.
- Para organizar un programa de mejor manera.

Módulos y paquetes

- Cuando usamos código externo lo hacemos a través de módulos y paquetes.
- Un **módulo** es un archivo de Python, por ejemplo `modulo.py`.
- Un **paquete** es una colección de módulos organizados de manera particular.

¿Para qué existen?

- Para no repetir código.
- Para organizar un programa de mejor manera.
- Para aprovechar código escrito por otras personas.

Librería estándar de Python

Colección de módulos y paquetes que viene incluida con Python.

Librería estándar de Python

Colección de módulos y paquetes que viene incluida con Python.

Ejemplos:

- `math`
- `random`
- `statistics`
- `os`
- …

Importar código

La sentencia `import`

La forma más simple de importar un módulo o paquete es:

```
import <nombre>
```

La sentencia `import`

La forma más simple de importar un módulo o paquete es:

```
import <nombre>
```

Y después accedemos a sus objetos con `math.nombre`.

```
>>> import math
```

La sentencia `import`

La forma más simple de importar un módulo o paquete es:

```
import <nombre>
```

Y después accedemos a sus objetos con `math.nombre`.

```
>>> import math
>>> math.sqrt(16)
4.0
```

La sentencia `import`

La forma más simple de importar un módulo o paquete es:

```
import <nombre>
```

Y después accedemos a sus objetos con `math.nombre`.

```
>>> import math
>>> math.sqrt(16)
4.0
>>> math.pi
3.141592653589793
```

La sentencia `import`

La forma más simple de importar un módulo o paquete es:

```
import <nombre>
```

Y después accedemos a sus objetos con `math.nombre`.

```
>>> import math
>>> math.sqrt(16)
4.0
>>> math.pi
3.141592653589793
>>> type(math.pi)
<class 'float'>
```

Comprendiendo los *namespaces*

Importar `math` disponibiliza de manera directa el módulo `math`, no los objetos que contiene en su *namespace*[1].

```
>>> import math
>>> sqrt(16)
```

[1] Puede ser pensado como el equivalente de un ambiente en R

Comprendiendo los *namespaces*

Importar `math` disponibiliza de manera directa el módulo `math`, no los objetos que contiene en su *namespace*[1].

```
>>> import math
>>> sqrt(16)
NameError: name 'sqrt' is not defined
```

[1] Puede ser pensado como el equivalente de un ambiente en R

Importar varios módulos o paquetes

Basta con escribir varios `import`:

```
>>> import math
>>> import random
```

Importar varios módulos o paquetes

Basta con escribir varios `import`:

```
>>> import math
>>> import random
>>> math.factorial(5)
120
```

Importar varios módulos o paquetes

Basta con escribir varios `import`:

```
>>> import math
>>> import random
>>> math.factorial(5)
120
>>> random.random()
0.34718286416366235
```

¿Qué hay adentro de un *namespace*?

La función `dir()` lista los nombres en el *namespace* de un objeto, e.g. de módulos y paquetes:

```
>>> import math
>>> dir(math)
['acos', 'asin', 'cos', 'pi', 'sqrt', ...]
```

¿Qué hay adentro de un *namespace*?

La función `dir()` lista los nombres en el *namespace* de un objeto, e.g. de módulos y paquetes:

```
>>> import math
>>> dir(math)
['acos', 'asin', 'cos', 'pi', 'sqrt', ...]
```

⚡ **¡`dir` funciona con cualquier objeto!**

```
>>> dir("hola :)")
['__add__', '__class__', '__contains__', '__delattr__', ...]
```

¿Qué hay adentro de un *namespace*?

La función `dir()` lista los nombres en el *namespace* de un objeto, e.g. de módulos y paquetes:

```
>>> import math
>>> dir(math)
['acos', 'asin', 'cos', 'pi', 'sqrt', ...]
```

⚡ **¡`dir` funciona con cualquier objeto!**

```
>>> dir("hola :)")
['__add__', '__class__', '__contains__', '__delattr__', ...]
```

En la práctica, el autocompletado del editor suele alcanzar.

`from ... import ...`

Se pueden importar objetos específicos:

```
>>> from math import log
```

`from ... import ...`

Se pueden importar objetos específicos:

```
>>> from math import log
>>> log(5)
1.6094379124341003
```

`from ... import ...`

Se pueden importar objetos específicos:

```
>>> from math import log
>>> log(5)
1.6094379124341003
```

También se pueden importar múltiples objetos en específico:

```
>>> from statistics import mean, median
>>> numeros = [4, 5, 9, 30, 3, 8, 6]
```

`from ... import ...`

Se pueden importar objetos específicos:

```
>>> from math import log
>>> log(5)
1.6094379124341003
```

También se pueden importar múltiples objetos en específico:

```
>>> from statistics import mean, median
>>> numeros = [4, 5, 9, 30, 3, 8, 6]
>>> mean(numeros)
9.285714285714286
```

`from ... import ...`

Se pueden importar objetos específicos:

```
>>> from math import log
>>> log(5)
1.6094379124341003
```

También se pueden importar múltiples objetos en específico:

```
>>> from statistics import mean, median
>>> numeros = [4, 5, 9, 30, 3, 8, 6]
>>> mean(numeros)
9.285714285714286
>>> median(numeros)
6
```

Uso de aliases

Podemos renombrar lo que importamos.

Uso de aliases

Podemos renombrar lo que importamos.

Vale con módulos:

```
>>> import math as mates
>>> mates.cos(mates.pi)
-1.0
```

Uso de aliases

Podemos renombrar lo que importamos.

Vale con módulos:

```
>>> import math as mates
>>> mates.cos(mates.pi)
-1.0
```

Y cualquier objeto en general:

```
>>> from math import sqrt as raiz
>>> raiz(81)
9.0
```

Importar todo

Es posible traer todos los objetos de un módulo/paquete al *namespace* actual.

Importar todo

Es posible traer todos los objetos de un módulo/paquete al *namespace* actual.

```
from <module> import *
```

Importar todo

Es posible traer todos los objetos de un módulo/paquete al *namespace* actual.

```
from <module> import *
```

Pero, es una **pésima idea** 👎

Importar todo

Es posible traer todos los objetos de un módulo/paquete al *namespace* actual.

```
from <module> import *
```

Pero, es una **pésima idea** 👎

- Se incorporan una cantidad desconocida (potencialmente grande) de nombres al ambiente actual.

Importar todo

Es posible traer todos los objetos de un módulo/paquete al *namespace* actual.

```
from <module> import *
```

Pero, es una **pésima idea** 👎

- Se incorporan una cantidad desconocida (potencialmente grande) de nombres al ambiente actual.
- Pueden ocasionar colisiones de nombres y sobrevive el último en ser cargado.

Importar todo

Es posible traer todos los objetos de un módulo/paquete al *namespace* actual.

```
from <module> import *
```

Pero, es una **pésima idea** 👎

- Se incorporan una cantidad desconocida (potencialmente grande) de nombres al ambiente actual.
- Pueden ocasionar colisiones de nombres y sobrevive el último en ser cargado.
- Después cuesta saber de dónde salió y qué es cada cosa.

Módulos propios

Un módulo propio

Un archivo llamado `funciones.py` también es un módulo. Supongamos:

Un módulo propio

Un archivo llamado `funciones.py` también es un módulo. Supongamos:

```
def es_par(n):
    if n % 2 == 0:
        return True
    return False

def es_primo(n):
    if n <= 1:
        return False
    for i in range(2, n):
        if n % i == 0:
            return False
    return True
```

Importar módulo propio

Si lo queremos importar, ¿dónde tiene que estar?

Importar módulo propio

Si lo queremos importar, ¿dónde tiene que estar?

Si el módulo está en el proyecto actual (en la raíz o un subdirectorio), Python puede encontrarlo.

Importar módulo propio

Si lo queremos importar, ¿dónde tiene que estar?

Si el módulo está en el proyecto actual (en la raíz o un subdirectorio), Python puede encontrarlo.

```
proyecto/
├── funciones.py
└── programa.py
```

Importar módulo propio

Desde `programa.py`:

```
>>> import funciones
```

Importar módulo propio

Desde `programa.py`:

```
>>> import funciones
>>> funciones.es_par(12)
True
```

Importar módulo propio

Desde `programa.py`:

```
>>> import funciones
>>> funciones.es_par(12)
True
>>> funciones.es_par(15)
False
```

Importar módulo propio

Desde `programa.py`:

```
>>> import funciones
>>> funciones.es_par(12)
True
>>> funciones.es_par(15)
False
>>> funciones.es_primo(11)
True
```

Importar objetos del módulo propio

También funciona así:

```
>>> from funciones import es_par, es_primo
```

Importar objetos del módulo propio

También funciona así:

```
>>> from funciones import es_par, es_primo
>>> es_par(12)
True
```

Importar objetos del módulo propio

También funciona así:

```
>>> from funciones import es_par, es_primo
>>> es_par(12)
True
>>> es_primo(15)
False
```

Paquetes externos

¿Qué cambia?

- Los módulos de la librería estándar ya vienen con Python.

¿Qué cambia?

- Los módulos de la librería estándar ya vienen con Python.
- Los paquetes externos se instalan aparte.

¿Qué cambia?

- Los módulos de la librería estándar ya vienen con Python.
- Los paquetes externos se instalan aparte.
- El gestor más común es `pip`.

¿Qué cambia?

- Los módulos de la librería estándar ya vienen con Python.
- Los paquetes externos se instalan aparte.
- El gestor más común es `pip`.
- El repositorio más usado es PyPI.

`pip`

`pip` se usa desde la terminal[1]

[1] Aunque Posit nos engaña y nos permite hacerlo desde la consola.

`pip`

`pip` se usa desde la terminal[1]

Consultar paquetes instalados:

```
pip list
```

[1] Aunque Posit nos engaña y nos permite hacerlo desde la consola.

`pip`

`pip` se usa desde la terminal[1]

Consultar paquetes instalados:

```
pip list
```

Instalar un paquete:

```
pip install nombre-paquete
```

[1] Aunque Posit nos engaña y nos permite hacerlo desde la consola.

`python -m pip`

También podemos ejecutar `pip` como módulo de Python:

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

`python -m pip`

También podemos ejecutar `pip` como módulo de Python:

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

> **¿Para qué sirve?**
>
> Ayuda a que `pip` quede asociado al Python que estamos usando.

Resumen

- `import modulo` trae el módulo.

Resumen

- `import modulo` trae el módulo.
- `from modulo import objeto` trae un objeto puntual.

Resumen

- `import modulo` trae el módulo.
- `from modulo import objeto` trae un objeto puntual.
- Un archivo `.py` propio, y sus objetos, también pueden importarse.

Resumen

- `import modulo` trae el módulo.
- `from modulo import objeto` trae un objeto puntual.
- Un archivo `.py` propio, y sus objetos, también pueden importarse.
- La herramienta `pip` permite instalar paquetes de terceros.