11_ModulosYPaquetes

June 18, 2025

Creado por:

Isabel Maniega

[1]:
```
from IPython import display
```

1 Módulos

Módulo es un archivo que contiene definiciones y sentencias de Python, que se pueden importar más tarde y utilizar cuando sea necesario.

Importar modulos

Para que un módulo sea utilizable, hay que importarlo (piensa en ello como sacar un libro del estante). La importación de un módulo se realiza mediante una instrucción llamada import. Nota: import es también una palabra clave reservada (con todas sus implicaciones).

2 Math

2.0.1 1) Ejemplo importar el modulo math de Python

[2]:
```
import math
```

Dentro del modulo math podemos importar la función seno para calcular el valor de pi entre dos:

[3]:
```
math.sin(math.pi/2)
```

[3]:
```
1.0
```

Lo realizamos llamando: **modulo + ‘.’ + nombre de la entidad** ej. math.sin()

2.0.2 2) Otro ejemplo de importar modulo math en Python

[4]:
```
from math import sin, pi
```

- La palabra clave reservada **from**.
- El nombre del módulo a ser (selectivamente) importado.
- La palabra clave reservada **import**.
- El nombre o lista de nombres de la entidad o entidades las cuales estan siendo importadas al namespace.

[5]:
```
sin(pi/2)
```

[5]:
```
1.0
```

Se llaman directamente sin necesidad de declarar math de nuevo

2.0.3 3) Otro ejemplo de importar modulo math en Python

[6]:
```
from math import *
```

Esto quiere decir que importa todas las entidades que conforman el paquete **math**

2.1 Importando un módulo usando la palabra reservada *as*

[7]:
```
# pip install pandas
```

[8]:
```
import pandas as pd
```

La palabra **as** nos permite usar a lo largo del código la palabra asignada sin necesidad de usar el nombre completo. En este ejemplo **pandas** a partir de este momento pasará a llamarse **pd** a lo largo del código.

import modulo **as** alias

El "module" identifica el nombre del módulo original mientras que el "alias" es el nombre que se desea usar en lugar del original.

2.2 DIR

La función devuelve una lista ordenada alfabéticamente la cual contiene todos los nombres de las entidades disponibles en el módulo

[9]:
```
dir(math)
```

[9]:
```
['__doc__',
 '__loader__',
 '__name__',
 '__package__',
 '__spec__',
 'acos',
 'acosh',
```

```
'asin',
'asinh',
'atan',
'atan2',
'atanh',
'cbrt',
'ceil',
'comb',
'copysign',
'cos',
'cosh',
'degrees',
'dist',
'e',
'erf',
'erfc',
'exp',
'exp2',
'expm1',
'fabs',
'factorial',
'floor',
'fmod',
'frexp',
'fsum',
'gamma',
'gcd',
'hypot',
'inf',
'isclose',
'isfinite',
'isinf',
'isnan',
'isqrt',
'lcm',
'ldexp',
'lgamma',
'log',
'log10',
'log1p',
'log2',
'modf',
'nan',
'nextafter',
'perm',
'pi',
'pow',
```

```
 'prod',
 'radians',
 'remainder',
 'sin',
 'sinh',
 'sqrt',
 'sumprod',
 'tan',
 'tanh',
 'tau',
 'trunc',
 'ulp']
```

math.pi → constant value.

math.e → Euler's number value.

math.sqrt(x) → the square root of x.

math.sin(x) → the sine of x.

math.cos(x) → the cosine of x.

math.tan(x) → the tangent of x.

math.asin(x) → the arcsine of x.

math.acos(x) → the arccosine of x.

math.atan(x) → the arctangent of x.

math.radians(x) → a function that converts x from degrees to radians.

math.degrees(x) → acting in the other direction (from radians to degrees).

math.sinh(x) → the hyperbolic sine.

math.cosh(x) → the hyperbolic cosine.

math.tanh(x) → the hyperbolic tangent.

math.asinh(x) → the hyperbolic arcsine.

math.acosh(x) → the hyperbolic arccosine.

math.atanh(x) → the hyperbolic arctangent.

math.exp(x) → finds the value of ex.

math.log(x) → the natural logarithm of x.

math.log(x, b) → the logarithm of x to base b.

math.log10(x) → the decimal logarithm of x (more precise than math.log(x, 10)).

log2(x) → the binary logarithm of x (more precise than math.log(x, 2)).

math.ceil(x) → the ceiling of x (the smallest integer greater than or equal to x).

math.floor(x) → the floor of x (the largest integer less than or equal to x).

math.trunc(x) → the value of x truncated to an integer (be careful – it's not an equivalent of either ceil or floor).

math.factorial(x) → returns x! (x has to be an integral and not a negative).

math.hypot(x, y) → returns the length of the hypotenuse of a right-angle triangle with the leg lengths equal to x and y (the same as math.sqrt(pow(x, 2) + pow(y, 2)) but more precise).

[10]:
```
dir(pd)
```

[10]:
```
['ArrowDtype',
 'BooleanDtype',
 'Categorical',
 'CategoricalDtype',
 'CategoricalIndex',
 'DataFrame',
 'DateOffset',
 'DatetimeIndex',
 'DatetimeTZDtype',
 'ExcelFile',
 'ExcelWriter',
 'Flags',
 'Float32Dtype',
 'Float64Dtype',
 'Grouper',
 'HDFStore',
 'Index',
 'IndexSlice',
 'Int16Dtype',
 'Int32Dtype',
 'Int64Dtype',
 'Int8Dtype',
 'Interval',
 'IntervalDtype',
 'IntervalIndex',
 'MultiIndex',
 'NA',
 'NaT',
 'NamedAgg',
 'Period',
 'PeriodDtype',
 'PeriodIndex',
 'RangeIndex',
 'Series',
 'SparseDtype',
 'StringDtype',
 'Timedelta',
```

```
'TimedeltaIndex',
'Timestamp',
'UInt16Dtype',
'UInt32Dtype',
'UInt64Dtype',
'UInt8Dtype',
'__all__',
'__builtins__',
'__cached__',
'__doc__',
'__docformat__',
'__file__',
'__git_version__',
'__loader__',
'__name__',
'__package__',
'__path__',
'__spec__',
'__version__',
'_built_with_meson',
'_config',
'_is_numpy_dev',
'_libs',
'_pandas_datetime_CAPI',
'_pandas_parser_CAPI',
'_testing',
'_typing',
'_version_meson',
'annotations',
'api',
'array',
'arrays',
'bdate_range',
'compat',
'concat',
'core',
'crosstab',
'cut',
'date_range',
'describe_option',
'errors',
'eval',
'factorize',
'from_dummies',
'get_dummies',
'get_option',
'infer_freq',
```

```
'interval_range',
'io',
'isna',
'isnull',
'json_normalize',
'lreshape',
'melt',
'merge',
'merge_asof',
'merge_ordered',
'notna',
'notnull',
'offsets',
'option_context',
'options',
'pandas',
'period_range',
'pivot',
'pivot_table',
'plotting',
'qcut',
'read_clipboard',
'read_csv',
'read_excel',
'read_feather',
'read_fwf',
'read_gbq',
'read_hdf',
'read_html',
'read_json',
'read_orc',
'read_parquet',
'read_pickle',
'read_sas',
'read_spss',
'read_sql',
'read_sql_query',
'read_sql_table',
'read_stata',
'read_table',
'read_xml',
'reset_option',
'set_eng_float_format',
'set_option',
'show_versions',
'test',
'testing',
```

```
 'timedelta_range',
 'to_datetime',
 'to_numeric',
 'to_pickle',
 'to_timedelta',
 'tseries',
 'unique',
 'util',
 'value_counts',
 'wide_to_long']
```

¿Has notado los nombres extraños que comienzan con ___ al inicio de la lista? Se hablará más sobre ellos cuando hablemos sobre los problemas relacionados con la escritura de módulos propios.

[11]:
```
# the floor of x (the largest integer less than or equal to x).

math.floor(3.999999)
```

[11]: 3

[12]:
```
#  the ceiling of x (the smallest integer greater than or equal to x).

math.ceil(3.000000001)
```

[12]: 4

[13]:
```
# Returns the value of x raised to power y.
# This method converts both arguments into a float.

math.pow(2, 3)
```

[13]: 8.0

[14]:
```
# If 'x' is negative and 'y' is not an integer, it returns a ValueError.

math.pow(-2, 1.2)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[14], line 3
      1 # If 'x' is negative and 'y' is not an integer, it returns a ValueError
----> 3 math.pow(-2, 1.2)

ValueError: math domain error
```

[15]:
```
# the square root of x.

```

```
math.sqrt(8)
```

[15]: 2.8284271247461903

[16]:
```
# the square root of x.

math.sqrt(-8)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[16], line 3
      1 # the square root of x.
----> 3 math.sqrt(-8)

ValueError: math domain error
```

[17]:
```
#   constant value.

math.pi
```

[17]: 3.141592653589793

[18]:
```
value = math.pi < math.floor(3.14)
value
```

[18]: False

[19]:
```
value = math.pi < math.ceil(3.14)
# value = 3.14 < 4
# True => 1
int(value)
```

[19]: 1

[20]:
```
# returns x! (x has to be an integral and not a negative).

math.factorial(5)
```

[20]: 120

[21]:
```
# returns x! (x has to be an integral and not a negative).

math.factorial(-5)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[21], line 3
```

```
      1 # returns x! (x has to be an integral and not a negative).
----> 3 math.factorial(-5)

ValueError: factorial() not defined for negative values
```

[22]:
```
math.floor(3.00), math.ceil(3.00)
```

[22]:
```
(3, 3)
```

[23]:
```
print(math.floor(-15.9999999))
print(math.floor(-15.0000001))
```

```
-16
-16
```

[24]:
```
#set perpendicular and base
parendicular = 3
base = 4

#print the hypotenuse of a right-angled triangle
print(math.hypot(parendicular, base))
```

```
5.0
```

[25]:
```
# Return the truncated integer parts of different numbers

print(math.trunc(18.77))
print(math.trunc(25.343))
print(math.trunc(-991.5))
```

```
18
25
-991
```

3 Random

Nos permite obtener número **pseudoaleatorios**, esto quiere decir que los algoritmos, que generan los números, no son aleatorios, son deterministas y predecibles.

Los números random, necesitan determinar una **semilla**, calcula un número "aleatorio" basado en él (el método depende de un algoritmo elegido). El valor de la semilla inicial, establecido durante el inicio del programa, determina el orden en que aparecerán los valores generados.

3.0.1 Ejemplo 1:

Generamos números pseudoaleatorios de 0.0 a 1.0

[26]:
```
from random import random
```

```
for i in range(5):
    print(random())
```

```
0.5484158491912323
0.9549224958905835
0.5428495675273918
0.07927621626847103
0.35380421939125595
```

La semilla en este caso no está definida, por lo tanto es dificil saber porque valor empieza y presentará una aleatoriedad.

3.0.2 Ejemplo 2: seed

Añadimos una semilla de 0

[27]:
```
from random import random, seed

seed(0)

for i in range(5):
    print(random())
```

```
0.8444218515250481
0.7579544029403025
0.420571580830845
0.25891675029296335
0.5112747213686085
```

Debido al hecho de que la semilla siempre se establece con el mismo valor, la secuencia de valores generados siempre se ve igual.

3.0.3 Ejemplo 3: randrange, randint

randrange y ranint: genera números enteros aleatorios en un rango determinado:.

- randrange(inicio, fin, incremento). Ejemplo de 0 a 2 (No incluido)
- randint(izquierda, derecha). Ejemplo de 0 a 2 (incluido)

[28]:
```
from random import randrange, randint

print(randrange(100), end=' ')
print(randrange(0, 100), end=' ')
print(randrange(0, 100, 1), end=' ')
print(randint(0, 100))
```

```
51 38 61 45
```

Nota: Observa que los datos generados no son único

[29]:
```
from random import randint

for i in range(10):
    print(randint(1, 10), end=', ')
```

```
10, 4, 9, 3, 5, 3, 2, 10, 5, 9,
```

3.0.4 Ejemplo 4: Choice, sample

- choice(secuencia)
- sample(secuencia, elementos_a_elegir=1)

choice elige un elemento "aleatorio" de la secuencia de entrada y lo devuelve.

sample crea una lista (una muestra) que consta del elemento elementos_a_elegir (que por defecto es 1) "sorteado" de la secuencia de entrada.

[30]:
```
from random import choice, sample

my_list = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

print(choice(my_list))
print(sample(my_list, 5))
print(sample(my_list, 10))
```

```
10
[3, 5, 2, 6, 1]
[6, 8, 2, 3, 4, 7, 9, 5, 10, 1]
```

3.0.5 Ejemplo 5: Shuffle

[31]:
```
from random import shuffle

my_list2 = [4, 5, 8, 14, 95, 74, 41]

# Reorganiza los elementos aleatoriamente:
shuffle(my_list2)

my_list2
```

[31]:
```
[5, 14, 74, 4, 8, 95, 41]
```

4 Estructura de ejecución de código

- Tu código quiere crear un archivo, por lo que invoca una de las funciones de Python.
- Python acepta la orden, la reorganiza para cumplir con los requisitos del sistema operativo local, es como poner el sello "aprobado" en una solicitud y lo envía (esto puede recordarte una cadena de mando).

- El SO comprueba si la solicitud es razonable y válida (por ejemplo, si el nombre del archivo se ajusta a algunas reglas de sintaxis) e intenta crear el archivo. Tal operación, aparentemente es muy simple, no es atómica: consiste de muchos pasos menores tomados por:
- El hardware, el cual es responsable de activar los dispositivos de almacenamiento (disco duro, dispositivos de estado sólido, etc.) para satisfacer las necesidades del sistema operativo.

4.1 Módulo Platform

El módulo platform permite acceder a los datos de la plataforma subyacente, es decir, hardware, sistema operativo e información sobre la versión del intérprete.

Simplemente devuelve una cadena que describe el entorno; por lo tanto, su salida está más dirigida a los humanos que al procesamiento automatizado (lo verás pronto).

[32]:
```
from platform import platform

print(platform())
print(platform(1))
print(platform(0, 1))
```

```
Linux-6.8.0-60-generic-x86_64-with-glibc2.39
Linux-6.8.0-60-generic-x86_64-with-glibc2.39
Linux-6.8.0-60-generic-x86_64-with-glibc2.39
```

- aliased → cuando se establece a True (o cualquier valor distinto a cero) puede hacer que la función presente los nombres de capa subyacentes alternativos en lugar de los comunes.
- terse → cuando se establece a True (o cualquier valor distinto a cero) puede convencer a la función de presentar una forma más breve del resultado (si lo fuera posible).

4.1.1 1) machine()

Para conocer el nombre genérico del procesador que ejecuta el sistema operativo junto con Python y el código, una función llamada machine() te lo dirá.

[33]:
```
from platform import machine

print(machine())
```

```
x86_64
```

4.1.2 2) processor()

La función processor() devuelve una cadena con el nombre real del procesador (si lo fuese posible).

[34]:
```
from platform import processor

print(processor())
```

```
x86_64
```

4.1.3 3) System

Una función llamada system() devuelve el nombre genérico del sistema operativo en una cadena.

[35]:
```
from platform import system

print(system())
```

```
Linux
```

4.1.4 4) version()

La versión del sistema operativo se proporciona como una cadena por la función version().

[36]:
```
from platform import version

print(version())
```

```
#63-Ubuntu SMP PREEMPT_DYNAMIC Tue Apr 15 19:04:15 UTC 2025
```

4.1.5 5) python_implementation y python_version_tuple

Las funciones python_implementation y python_version_tuple

Si necesitas saber que versión de Python está ejecutando tu código, puedes verificarlo utilizando una serie de funciones dedicadas, aquí hay dos de ellas:

- **python_implementation()** → devuelve una cadena que denota la implementación de Python (espera CPython aquí, a menos que decidas utilizar cualquier rama de Python no canónica).
- **python_version_tuple()** → devuelve una tupla de tres elementos la cual contiene, ej 3.8.18:
  - La parte mayor de la versión de Python.
  - La parte menor.
  - El número del nivel de parche.

[37]:
```
from platform import python_implementation, python_version_tuple

print(python_implementation())

for atr in python_version_tuple():
    print(atr)
```

```
CPython
3
12
3
```

5 Paquetes

- Un módulo es un contenedor lleno de funciones - puedes empaquetar tantas funciones como desees en un módulo y distribuirlo por todo el mundo.
- Por supuesto, no es una buena idea mezclar funciones con diferentes áreas de aplicación dentro de un módulo (al igual que en una biblioteca: nadie espera que los trabajos científicos se incluyan entre los cómics), así que se deben agrupar las funciones cuidadosamente y asignar un nombre claro e intuitivo al módulo que las contiene (por ejemplo, no le des el nombre videojuegos a un módulo que contiene funciones destinadas a particionar y formatear discos duros).
- Crear muchos módulos puede causar desorden: tarde que temprano querrás agrupar tus módulos de la misma manera que previamente has agrupado funciones: ¿Existe un contenedor más general que un módulo?
- Sí lo hay, es un paquete: en el mundo de los módulos, un paquete juega un papel similar al de una carpeta o directorio en el mundo de los archivos.

```
[any directory]


        package             ← directory

            gearbox.py      ← file: contains turn_on() function

            __init__.py     ← empty file

            subpackage      ← directory

                engine.py   ← file: contains start() function
```

Podemos importar script `gearbox.py` y su función `turn_on()` de la siguiente forma:

```
import package.gearbox

package.gearbox.turn_on()
```

Otra forma de importar la función es:

```
from package.gearbox import turn_on

turn_on()
```

Otra forma más de importarlo:

```
from package.gearbox import *

start()
```

5.0.1 1) Crear un modulo

[38]:
```
# 1) Crea un script con nombre module.py
# 2) Crea un script con nombre main.py
# 3) Dentro del script main, llama al script module:
    # import module
# 4) Ejecuta el script main.py y no deberias de ver como salida nada,
# si no ha realizado ningún error, ha realizado con éxito la importación del␣
 ↪modulo (module.py)
```

Al ejecutar el script main.py verás que ha aparecido una nueva subcarpeta, ¿puedes verla? Su nombre es **pycache**. Echa un vistazo adentro. ¿Qué es lo que ves?

Hay un archivo llamado (más o menos) module.cpython-xy.pyc donde x y y son dígitos derivados de tu versión de Python (por ejemplo, serán 3 y 8 si utilizas Python 3.8).

El nombre del archivo es el mismo que el de tu módulo. La parte posterior al primer punto dice qué implementación de Python ha creado el archivo (CPython) y su número de versión. La ultima parte (pyc) viene de las palabras Python y compilado.

Puedes mirar dentro del archivo: el contenido es completamente ilegible para los humanos. Tiene que ser así, ya que el archivo está destinado solo para uso el uso de Python.

Cuando Python importa un módulo por primera vez, traduce el contenido a una forma algo compilada.

El archivo no contiene código en lenguaje máquina: es código semi-compilado interno de Python, listo para ser ejecutado por el intérprete de Python. Como tal archivo no requiere tantas comprobaciones como las de un archivo fuente, la ejecución comienza más rápido y también se ejecuta más rápido.

Gracias a eso, cada importación posterior será más rápida que interpretar el código fuente desde cero.

Python puede verificar si el archivo fuente del módulo ha sido modificado (en este caso, el archivo pyc será reconstruido) o no (cuando el archivo pyc pueda ser ejecutado al instante). Este proceso es completamente automático y transparente, no tiene que ser tomando en cuenta.

5.0.2 2) mostrar la información de module.py

[39]:
```
# Podemos repetir los pasos anteriores pero en este punto añadimos al script␣
 ↪module.py un print,
# por los tanto los pasos serían:

# 1) Crea un script con nombre mudule.py
# 2) Dentro del script module.py realiza un print:
    # print("Me gusta ser un módulo.")
# 3) Crea un script con nombre main.py
# 4) Dentro del script main, llama al script module:
    # import module
# 5) Ejecuta el script main.py en este punto verás que la salida muestra:
```

```
    # Me gusta ser un módulo.
# Visualizamos el contenido del módulo module.py
```

5.0.3 3) '__ __ *name* __ __'

[40]:
```
# Podemos repetir los pasos anteriores pero en este punto añadimos al script␣
 ↪module.py un print que muestre '__name__',
# por los tanto los pasos serían:

# 1) Crea un script con nombre module.py
# 2) Dentro del script module.py realiza un print:
    # print("Me gusta ser un módulo.")
    # print(__name__)
# 3) Crea un script con nombre main.py
# 4) Dentro del script main, llama al script module:
    # import module
# 5) Ejecuta el script main.py en este punto verás que la salida muestra:
    # Me gusta ser un módulo.
    # __main__ ó module (si es un modulo como es este ejemplo)

# Podemos decir que:

    # Cuando se ejecuta un archivo directamente, su variable __name__ se␣
 ↪establece a __main__.
    # Cuando un archivo se importa como un módulo, su variable __name__ se␣
 ↪establece al nombre del archivo (excluyendo a .py).
```

5.0.4 4) main

[41]:
```
# Podemos repetir los pasos anteriores pero en este punto añadimos al script␣
 ↪module.py un print que muestre '__name__',
# por los tanto los pasos serían:

# 1) Crea un script con nombre mudule.py
# 2) Dentro del script module.py realiza un print:
    # print("Me gusta ser un módulo.")
    # if __name__ == "__main__":
        # print("Yo prefiero ser un módulo")
    # else:
        # print("Me gusta ser un módulo")
# 3) Crea un script con nombre main.py
# 4) Dentro del script main, llama al script module:
    # import module
# 5) Ejecuta el script main.py en este punto verás que la salida muestra:
    # Me gusta ser un módulo.
    # __main__
```

```
# Podemos decir que:

    # Cuando se ejecuta un archivo directamente, su variable __name__ se␣
 ↪establece a __main__.
    # Cuando un archivo se importa como un módulo, su variable __name__ se␣
 ↪establece al nombre del archivo (excluyendo a .py).
```

5.0.5 5) Contador de llamada de funciones

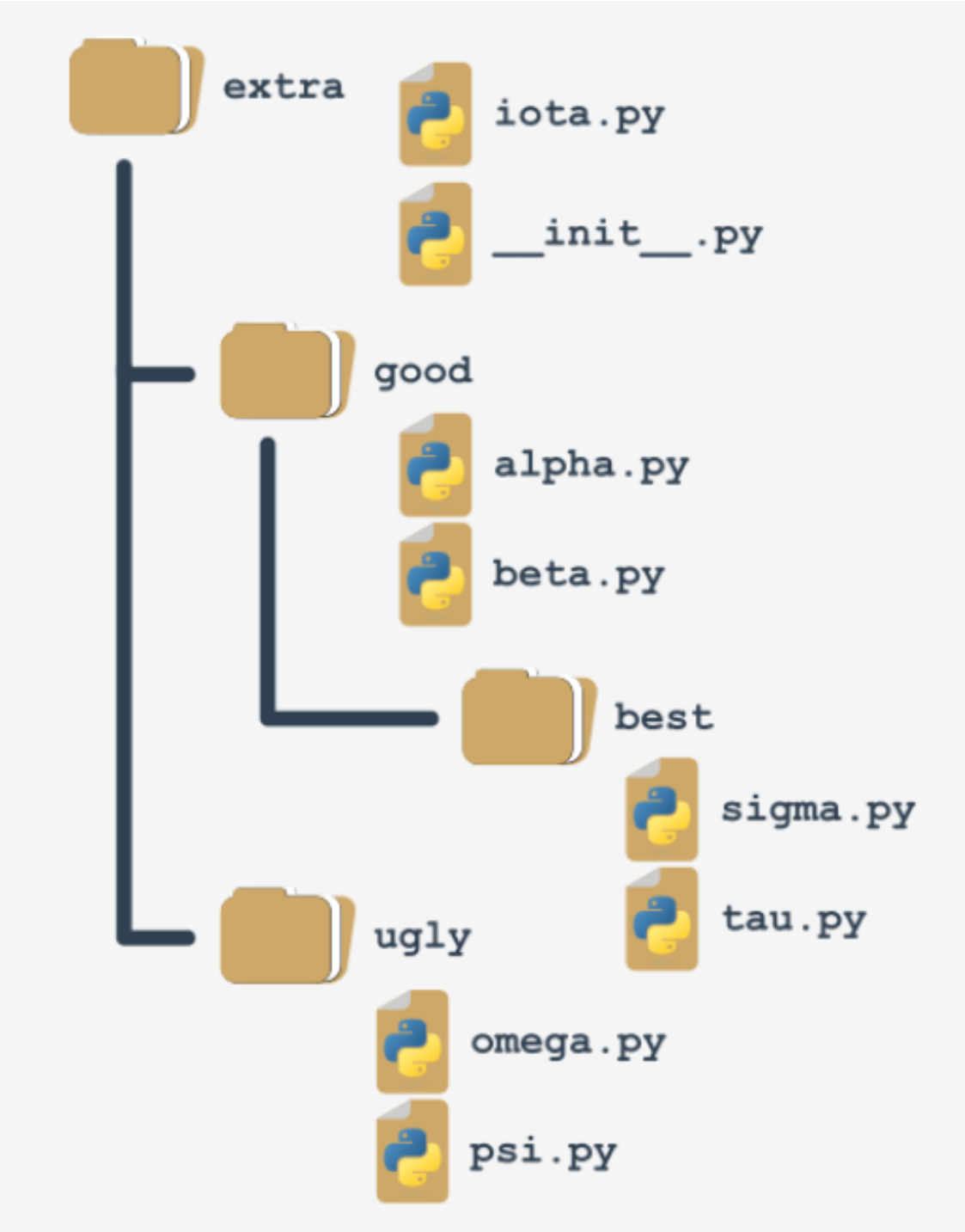
[42]:
```
# Podemos repetir los pasos anteriores pero en este punto añadimos al script␣
 ↪module.py un print que muestre '__name__',
# por los tanto los pasos serían:

# 1) Crea un script con nombre mudule.py
# 2) Dentro del script module.py realiza un print:
    # count = 0
    # if __name__ == "__main__":
        # print("Yo prefiero ser un módulo")
    # else:
        #print("Me gusta ser un módulo")
# 3) Crea un script con nombre main.py
# 4) Dentro del script main, llama al script module:
    # import module
    # print(module.counter)
# 5) Ejecuta el script main.py en este punto verás que la salida muestra:
    # Yo prefiero ser un módulo.
    # 0

# Como puedes ver, el archivo principal intenta acceder a la variable de␣
 ↪contador del módulo.
# ¿Es esto legal? Sí lo es. ¿Es utilizable? Claro. ¿Es seguro?

# Eso depende: si confías en los usuarios de tu módulo, no hay problema;
# sin embargo, es posible que no desees que el resto del mundo vea tu variable␣
 ↪personal o privada.

# A diferencia de muchos otros lenguajes de programación,
# Python no tiene medios para permitirte ocultar tales variables a los ojos de␣
 ↪los usuarios del módulo.

# Solo puedes informar a tus usuarios que esta es tu variable, que pueden␣
 ↪leerla,
# pero que no deben modificarla bajo ninguna circunstancia.

# Esto se hace anteponiendo al nombre de la variable _ (un guión bajo) o __␣
 ↪(dos guiones bajos),
```

```
# pero recuerda, es solo un acuerdo. Los usuarios de tu módulo pueden␣
  ↪obedecerlo o no.
```

5.0.6 6) Árbol carpetas

[43]:
```
display.Image("./images/arbol_carpetas.png")
```

[43]:



Nota: no solo la carpeta raiz puede contener el archivo **init.py**, también puedes ponerlo dentro de cualquiera de sus subcarpetas (subpaquetes). Puede ser útil si algunos de los subpaquetes requieren tratamiento individual o un tipo especial de inicialización.

Para acceder a la función funT() ubicado en el archivo tau pondremos:

```
extra.good.best.tau.funT()
```

Y para acceder a la función funP() del archivo psi:

```
extra.ugly.psi.funP()
```

Entonces para importarlo como sería:

```
import extra.good.best.tau as t
import extra.ugly.psi as p
```

otra forma:

```
from extra.good.best.tau import funT
from extra.ugly.psi import funP

print(t.funT())
print(p.funP())
print(funT())
print(funP())
```

Los nombres shabang, shebang, hasbang, poundbang y hashpling describen el dígrafo escrito como **#!**, se utiliza para instruir a los sistemas operativos similares a Unix sobre cómo se debe iniciar el archivo fuente de Python. Esta convención no tiene efecto en MS Windows.

6 Modulo sys

[44]:
```
import sys

for p in sys.path:
    print(p)
```

```
/usr/lib/python312.zip
/usr/lib/python3.12
/usr/lib/python3.12/lib-dynload

/home/isabelmaniega/Documentos/Python_ETL/env/lib/python3.12/site-packages
```

Ten en cuenta también que: hay un archivo zip listado como uno de los elementos de la ruta, esto no es un error. Python puede tratar los archivos zip como carpetas ordinarias, esto puede ahorrar mucho almacenamiento.

Deseas evitar que el usuario de tu módulo ejecute tu código como un script ordinario. ¿Cómo lograrías tal efecto?

[45]:
```
import sys

if __name__ == "__main__":
    print("¡No hagas eso!")
    sys.exit()
```

```
¡No hagas eso!
```

```
An exception has occurred, use %tb to see the full traceback.

SystemExit
```

```
/home/isabelmaniega/Documentos/Python_ETL/env/lib/python3.12/site-
packages/IPython/core/interactiveshell.py:3680: UserWarning: To exit: use
'exit', 'quit', or Ctrl-D.
  warn("To exit: use 'exit', 'quit', or Ctrl-D.", stacklevel=1)
```

[46]:
```
# Ejecutamos el script index.py: python index.py Hello 2

# from sys import argv

# print(argv[0])

# Output: index.py
```

7 Pypi

El repositorio de Python es **PyPI** (es la abreviatura de Python Package Index) y lo mantiene un grupo de trabajo llamado Packaging Working Group, una parte de la Python Software Foundation, cuya tarea principal es apoyar a los desarrolladores de Python en la diseminación de código eficiente.

https://wiki.python.org/psf/PackagingWG

https://pypi.org/

pip es package installer for Python (https://es.wikipedia.org/wiki/Pip_(administrador_de_paquetes))

Para descargar los paquetes del repositorio de Pypi necesitas usar *pip*.

Para verificar su instalación usamos:

- pip3 –version –> Si tenemos python 2 instalado en el sistema
- pip –version

Para pedir ayuda a pip se realiza con:

- pip help

[47]:
```
pip --version
```

```
pip 24.0 from /home/isabelmaniega/Documentos/Python_ETL/env/lib/python3.12/site-
packages/pip (python 3.12)
Note: you may need to restart the kernel to use updated packages.
```

[48]:
```
pip help
```

```
Usage:
  /home/isabelmaniega/Documentos/Python_ETL/env/bin/python -m pip <command>
[options]

Commands:
  install                     Install packages.
  download                    Download packages.
  uninstall                   Uninstall packages.
  freeze                      Output installed packages in requirements format.
  inspect                     Inspect the python environment.
  list                        List installed packages.
  show                        Show information about installed packages.
  check                       Verify installed packages have compatible
dependencies.
  config                      Manage local and global configuration.
  search                      Search PyPI for packages.
  cache                       Inspect and manage pip's wheel cache.
  index                       Inspect information available from package
indexes.
  wheel                       Build wheels from your requirements.
  hash                        Compute hashes of package archives.
  completion                  A helper command used for command completion.
  debug                       Show information useful for debugging.
  help                        Show help for commands.

General Options:
  -h, --help                  Show help.
  --debug                     Let unhandled exceptions propagate outside the
                              main subroutine, instead of logging them to
                              stderr.
  --isolated                  Run pip in an isolated mode, ignoring
                              environment variables and user configuration.
  --require-virtualenv        Allow pip to only run in a virtual environment;
                              exit with an error otherwise.
  --python <python>           Run pip with the specified Python interpreter.
  -v, --verbose               Give more output. Option is additive, and can be
                              used up to 3 times.
  -V, --version               Show version and exit.
  -q, --quiet                 Give less output. Option is additive, and can be
                              used up to 3 times (corresponding to WARNING,
                              ERROR, and CRITICAL logging levels).
```

```
  --log <path>                Path to a verbose appending log.
  --no-input                  Disable prompting for input.
  --keyring-provider <keyring_provider>
                              Enable the credential lookup via the keyring
                              library if user input is allowed. Specify which
                              mechanism to use [disabled, import, subprocess].
                              (default: disabled)
  --proxy <proxy>             Specify a proxy in the form
                              scheme://[user:passwd@]proxy.server:port.
  --retries <retries>         Maximum number of retries each connection should
                              attempt (default 5 times).
  --timeout <sec>             Set the socket timeout (default 15 seconds).
  --exists-action <action>    Default action when a path already exists:
                              (s)witch, (i)gnore, (w)ipe, (b)ackup, (a)bort.
  --trusted-host <hostname>   Mark this host or host:port pair as trusted,
                              even though it does not have valid or any HTTPS.
  --cert <path>               Path to PEM-encoded CA certificate bundle. If
                              provided, overrides the default. See 'SSL
                              Certificate Verification' in pip documentation
                              for more information.
  --client-cert <path>        Path to SSL client certificate, a single file
                              containing the private key and the certificate
                              in PEM format.
  --cache-dir <dir>           Store the cache data in <dir>.
  --no-cache-dir              Disable the cache.
  --disable-pip-version-check
                              Don't periodically check PyPI to determine
                              whether a new version of pip is available for
                              download. Implied with --no-index.
  --no-color                  Suppress colored output.
  --no-python-version-warning
                              Silence deprecation warnings for upcoming
                              unsupported Pythons.
  --use-feature <feature>     Enable new functionality, that may be backward
                              incompatible.
  --use-deprecated <feature>  Enable deprecated functionality, that will be
                              removed in the future.
Note: you may need to restart the kernel to use updated packages.
```

Para saber los paquetes instalados usamos:

- pip list

[49]:
```
pip list
```

```
Package                   Version
------------------------- --------------
aiosqlite                 0.21.0
alembic                   1.16.2
```

annotated-types	0.7.0
anyio	4.9.0
apprise	1.9.3
argon2-cffi	25.1.0
argon2-cffi-bindings	21.2.0
arrow	1.3.0
asgi-lifespan	2.1.0
asttokens	3.0.0
async-lru	2.0.5
asyncpg	0.30.0
attrs	25.3.0
babel	2.17.0
beautifulsoup4	4.13.4
bleach	6.2.0
cachetools	6.1.0
certifi	2025.6.15
cffi	1.17.1
charset-normalizer	3.4.2
click	8.1.8
cloudpickle	3.1.1
colorama	0.4.6
comm	0.2.2
contourpy	1.3.2
coolname	2.2.0
cramjam	2.10.0
cryptography	45.0.4
cycler	0.12.1
dateparser	1.2.1
debugpy	1.8.14
decorator	5.2.1
defusedxml	0.7.1
docker	7.1.0
et_xmlfile	2.0.0
exceptiongroup	1.3.0
executing	2.2.0
fastapi	0.115.13
fastjsonschema	2.21.1
fastparquet	2024.11.0
fonttools	4.58.4
fqdn	1.5.1
fsspec	2025.5.1
graphviz	0.21
greenlet	3.2.3
griffe	1.7.3
h11	0.16.0
h2	4.2.0
hpack	4.1.0
httpcore	1.0.9

```
httpx                       0.28.1
humanize                    4.12.3
hyperframe                  6.1.0
idna                        3.10
importlib_metadata          8.7.0
ipykernel                   6.29.5
ipython                     9.3.0
ipython_pygments_lexers     1.1.1
isoduration                 20.11.0
jedi                        0.19.2
Jinja2                      3.1.6
jinja2-humanize-extension   0.4.0
json5                       0.12.0
jsonpatch                   1.33
jsonpointer                 3.0.0
jsonschema                  4.24.0
jsonschema-specifications   2025.4.1
jupyter_client              8.6.3
jupyter_core                5.8.1
jupyter-events              0.12.0
jupyter-lsp                 2.2.5
jupyter_server              2.16.0
jupyter_server_terminals    0.5.3
jupyterlab                  4.4.3
jupyterlab_pygments         0.3.0
jupyterlab_server           2.27.3
kiwisolver                  1.4.8
Mako                        1.3.10
Markdown                    3.8
markdown-it-py              3.0.0
MarkupSafe                  3.0.2
matplotlib                  3.10.3
matplotlib-inline           0.1.7
mdurl                       0.1.2
mistune                     3.1.3
nbclient                    0.10.2
nbconvert                   7.16.6
nbformat                    5.10.4
nest-asyncio                1.6.0
notebook                    7.4.3
notebook_shim               0.2.4
numpy                       2.3.0
oauthlib                    3.3.0
openpyxl                    3.1.5
opentelemetry-api           1.34.1
orjson                      3.10.18
overrides                   7.7.0
packaging                   25.0
```

pandas	2.3.0
pandocfilters	1.5.1
parso	0.8.4
pathspec	0.12.1
pendulum	3.1.0
pexpect	4.9.0
pillow	11.2.1
pip	24.0
platformdirs	4.3.8
prefect	3.4.6
prometheus_client	0.22.1
prompt_toolkit	3.0.51
psutil	7.0.0
ptyprocess	0.7.0
pure_eval	0.2.3
pycparser	2.22
pydantic	2.11.7
pydantic_core	2.33.2
pydantic-extra-types	2.10.5
pydantic-settings	2.9.1
Pygments	2.19.1
pyodbc	5.2.0
pyparsing	3.2.3
python-dateutil	2.9.0.post0
python-dotenv	1.1.0
python-json-logger	3.3.0
python-slugify	8.0.4
python-socks	2.7.1
pytz	2025.2
PyYAML	6.0.2
pyzmq	27.0.0
readchar	4.2.1
referencing	0.36.2
regex	2024.11.6
requests	2.32.4
requests-oauthlib	2.0.0
rfc3339-validator	0.1.4
rfc3986-validator	0.1.1
rich	14.0.0
rpds-py	0.25.1
ruamel.yaml	0.18.14
ruamel.yaml.clib	0.2.12
seaborn	0.13.2
Send2Trash	1.8.3
setuptools	80.9.0
shellingham	1.5.4
six	1.17.0
sniffio	1.3.1

```
soupsieve                2.7
SQLAlchemy               2.0.41
stack-data               0.6.3
starlette                0.46.2
terminado                0.18.1
text-unidecode           1.3
tinycss2                 1.4.0
toml                     0.10.2
tornado                  6.5.1
traitlets                5.14.3
typer                    0.16.0
types-python-dateutil    2.9.0.20250516
typing_extensions        4.14.0
typing-inspection        0.4.1
tzdata                   2025.2
tzlocal                  5.3.1
ujson                    5.10.0
uri-template             1.3.0
urllib3                  2.4.0
uv                       0.7.13
uvicorn                  0.34.3
wcwidth                  0.2.13
webcolors                24.11.1
webencodings             0.5.1
websocket-client         1.8.0
websockets               15.0.1
xlsxwriter               3.2.5
zipp                     3.23.0
Note: you may need to restart the kernel to use updated packages.
```

Para mostrar más información sobre un paquete:

- pip show nombre del paquete

[50]:
```
pip show pip
```

```
Name: pip
Version: 24.0
Summary: The PyPA recommended tool for installing Python packages.
Home-page:
Author:
Author-email: The pip developers <distutils-sig@python.org>
License: MIT
Location: /home/isabelmaniega/Documentos/Python_ETL/env/lib/python3.12/site-
packages
Requires:
Required-by:
Note: you may need to restart the kernel to use updated packages.
```

Para buscar un paquete determinado:

- pip search anystring

[51]:
```
# pip search pip
# Da un error en jupyter
```

pip emplea una opción dedicada llamada –user (observa el guión doble). La presencia de esta opción indica a pip que actúe localmente en nombre de tu usuario sin privilegios de administrador.

Como administrador la instalación es: pip install pygame Como usuario sin derechos de administrador es: pip install –user pygame

El comando **pip install** tiene dos habilidades adicionales importantes:

Es capaz de actualizar un paquete instalado localmente; por ejemplo, si deseas asegurarte de que estás utilizando la última versión de un paquete en particular, puedes ejecutar el siguiente comando:

pip install -U nombre_del_paquete

Es capaz de instalar una versión seleccionada por el usuario de un paquete (pip instala por defecto la versión más nueva disponible); para lograr este objetivo debes utilizar la siguiente sintaxis:

pip install nombre_del_paquete==versión_del_paquete

Si alguno de los paquetes instalados actualmente ya no es necesario y deseas deshacerte de el, pip también será útil. Su comando uninstall ejecutará todos los pasos necesarios.

pip uninstall nombre_del_paquete

---

8 Modulo os

En esta sección, aprenderás sobre un módulo llamado os, que te permite interactuar con tu sistema operativo usando Python.

Proporciona funciones que están disponibles en sistemas Unix y/o Windows. Si estás familiarizado con la consola de comandos, verás que algunas funciones dan los mismos resultados que los comandos disponibles en los sistemas operativos.

Un buen ejemplo de esto es la función **mkdir**, que te permite crear un directorio como el comando mkdir en Unix y Windows.

Además de las operaciones de archivos y directorios, el módulo os te permite:

- Obtener información sobre el sistema operativo.
- Manejar procesos.
- Operar en streams de E/S usando descriptores de archivos.

Antes de crear tu primera estructura de directorios, verás cómo puedes obtener información sobre el sistema operativo actual. Esto es realmente fácil porque el módulo os proporciona una función llamada uname, que devuelve un objeto que contiene los siguientes atributos:

- systemname: almacena el nombre del sistema operativo.
- nodename: almacena el nombre de la máquina en la red.
- release: almacena el release (actualización) del sistema operativo.

- version: almacena la versión del sistema operativo.
- machine: almacena el identificador de hardware, por ejemplo, x86_64.

Veamos cómo es en la práctica:

[52]:
```
import os
print(os.uname())
```

```
posix.uname_result(sysname='Linux', nodename='isabelmaniega',
release='6.8.0-60-generic', version='#63-Ubuntu SMP PREEMPT_DYNAMIC Tue Apr 15
19:04:15 UTC 2025', machine='x86_64')
```

El módulo os te permite distinguir rápidamente el sistema operativo mediante el atributo **name**, que soporta uno de los siguientes nombres:

- posix: obtendrás este nombre si usas Unix.
- nt: obtendrás este nombre si usas Windows.
- java: obtendrás este nombre si tu código está escrito en Jython.

[53]:
```
import os
print(os.name)
```

```
posix
```

NOTA: En los sistemas Unix, hay un comando llamado uname que devuelve la misma información (si lo ejecutas con la opción -a) que la función uname.

El módulo os proporciona una función llamada *mkdir*, la cual, como el comando *mkdir* en Unix y Windows, te permite crear un directorio. La función *mkdir* requiere una ruta que puede ser relativa o absoluta. Recordemos cómo se ven ambas rutas en la práctica:

- my_first_directory: esta es una ruta relativa que creará el directorio my_first_directory en el directorio de trabajo actual.
- ./my_first_directory: esta es una ruta relativa que apunta explícitamente al directorio de trabajo actual. Tiene el mismo efecto que la ruta anterior.
- ../my_first_directory: esta es una ruta relativa que creará el directorio my_first_directory en el directorio superior del directorio de trabajo actual.
- /python/my_first_directory: esta es una ruta absoluta que creará el directorio my_first_directory, que a su vez está en el directorio raíz de python.

[54]:
```
import os

os.mkdir("my_first_directory")
print(os.listdir())
```

```
['7_Más sobre diccionarios.ipynb', '18_Introducción a orquestadores y
planificación (Airflow, Prefect).ipynb', '15_Pandas.ipynb',
'Ejer_Python_Básico', '13_Técnicas de preparación de datos.ipynb', '.gitignore',
'PDFs', 'images', '.ipynb_checkpoints', '19_ETL_Titanic.ipynb', 'files',
'my_first_directory', '4_Introducción a Python.ipynb', '17_Casos prácticos de
migración desde SAS.ipynb', '2_Primeros_Pasos.ipynb', '9_Clases y
Funciones.ipynb', '17_Logging.ipynb', '5_Bucles.ipynb',
```

```
'6_Más_sobre_Listas,tuplas,matrices.ipynb', 'data', '8_Clase y Funciones.ipynb',
'11_ModulosYPaquetes.ipynb', 'env', 'Presentation.pptx', 'Presentation.pdf',
'10_Excepciones.ipynb', 'contacts.csv', 'venv38', '12_PEP.ipynb',
'14_Redimientos formato de Archivos.ipynb', '3_Introducción a Python.ipynb',
'1_EntornosVirtuales.ipynb', '16_SQL para analistas de datos.ipynb']
```

Muestra un ejemplo de cómo crear el directorio my_first_directory usando una ruta relativa. Esta es la variante más simple de la ruta relativa, que consiste en pasar solo el nombre del directorio.

Si pruebas tu código aquí, generará el directorio recién creado 'my_first_directory'.

La función mkdir crea un directorio en la ruta especificada. Ten en cuenta que ejecutar el programa dos veces generará un FileExistsError.

Esto significa que no podemos crear un directorio si ya existe. Además del argumento de la ruta, la función mkdir puede tomar opcionalmente el argumento mode, que especifica los permisos del directorio. Sin embargo, en algunos sistemas, el argumento mode se ignora.

Para cambiar los permisos del directorio, recomendamos la función chmod, que funciona de manera similar al comando chmod en sistemas Unix. Puedes encontrar más información al respecto en la documentación.

En el ejemplo anterior, se usa otra función proporcionada por el módulo os llamada *listdir*. La función *listdir* devuelve una lista que contiene los nombres de los archivos y directorios que se encuentran en la ruta pasada como argumento.

Si no se le pasa ningún argumento, se utilizará el directorio de trabajo actual (como en el ejemplo anterior). Es importante que el resultado de la función *listdir* omita las entradas '.' y '..', que se muestran, por ejemplo, cuando se usa el comando ls -a en sistemas Unix.

NOTA: Tanto en Windows como en Unix, hay un comando llamado mkdir, que requiere una ruta de directorio. El equivalente del código anterior que crea el directorio my_first_directory es el comando mkdir my_first_directory.

Creación recursiva de directorios

La función mkdir es muy útil, pero ¿qué sucede si necesitas crear otro directorio dentro del directorio que acabas de crear? Por supuesto, puedes ir al directorio creado y crear otro directorio dentro de él, pero afortunadamente el módulo os proporciona una función llamada *makedirs*, que facilita esta tarea.

La función *makedirs* permite la creación recursiva de directorios, lo que significa que se crearán todos los directorios de la ruta.

[55]:
```
import os

os.makedirs("my_first_directory/my_second_directory")
os.chdir("my_first_directory")
print(os.listdir())
```

```
['my_second_directory']
```

El primero de ellos se crea en el directorio de trabajo actual, mientras que el segundo en el directorio my_first_directory.

No tienes que ir al directorio my_first_directory para crear el directorio my_second_directory, porque la función makedirs hace esto por ti. En el ejemplo anterior, vamos al directorio my_first_directory para mostrar que el comando makedirs crea el subdirectorio my_second_directory.

Para moverte entre directorios, puedes usar una función llamada *chdir*, que cambia el directorio de trabajo actual a la ruta especificada. Como argumento, toma cualquier ruta relativa o absoluta. En nuestro ejemplo, le pasamos el nombre del primer directorio.

NOTA: El equivalente de la función makedirs en sistemas Unix es el comando mkdir con el indicador -p, mientras que en Windows, simplemente el comando mkdir con la ruta:

- Sistemas tipo Unix:

mkdir -p my_first_directory/my_second_directory

- Windows:

mkdir my_first_directory/my_second_directory

Ubicación directorios

El módulo *os* proporciona una función que devuelve información sobre el directorio de trabajo actual. Se llama *getcwd*.

[56]:
```
import os

os.makedirs("my_first_directory/my_second_directory")
os.chdir("my_first_directory")
print(os.getcwd())
os.chdir("my_second_directory")
print(os.getcwd())
```

```
/home/isabelmaniega/Documentos/Python_ETL/my_first_directory/my_first_directory
/home/isabelmaniega/Documentos/Python_ETL/my_first_directory/my_first_directory/
my_second_directory
```

Creamos el directorio my_first_directory y el directorio my_second_directory dentro de él. En el siguiente paso, cambiamos el directorio de trabajo actual al directorio my_first_directory y luego mostramos el directorio de trabajo actual (primera línea del resultado).

A continuación, vamos al directorio my_second_directory y nuevamente mostramos el directorio de trabajo actual (segunda línea del resultado). Como puedes ver, la función getcwd devuelve la ruta absoluta a los directorios.

NOTA: En sistemas tipo Unix, el equivalente de la función getcwd es el comando pwd, que imprime el nombre del directorio de trabajo actual.

Eliminar directorios

El módulo os también te permite eliminar directorios. Te da la opción de borrar un solo directorio o un directorio con sus subdirectorios. Para eliminar un solo directorio, puedes usar una función llamada *rmdir*, que toma la ruta como argumento.

[57]:
```
import os

os.mkdir("my_first_directory")
print(os.listdir())
os.rmdir("my_first_directory")
print(os.listdir())
```

```
['my_first_directory']
[]
```

El ejemplo anterior es realmente simple. Primero, se crea el directorio my_first_directory y luego se elimina usando la función rmdir. La función listdir se utiliza como prueba de que el directorio se ha eliminado correctamente. En este caso, devuelve una lista vacía. Al eliminar un directorio, asegúrate de que exista y esté vacío; de lo contrario, se generará una excepción.

Para eliminar un directorio y sus subdirectorios, puedes utilizar la función *removedirs*, que requiere que se especifique una ruta que contenga todos los directorios que deben eliminarse:

[58]:
```
import os

os.makedirs("my_first_directory/my_second_directory")
os.removedirs("my_first_directory/my_second_directory")
print(os.listdir())
```

```
[]
```

Al igual que con la función rmdir, si uno de los directorios no existe o no está vacío, se generará una excepción.

NOTA: Tanto en Windows como en Unix, hay un comando llamado *rmdir*, que, al igual que la función rmdir, elimina directorios. Además, ambos sistemas tienen comandos para eliminar un directorio y su contenido. En Unix, este es el comando rm con el indicador -r.

system()

Todas las funciones pueden ser reemplazadas por una función llamada system, que ejecuta un comando que se le pasa como una cadena.

La función system está disponible tanto en Windows como en Unix. Dependiendo del sistema, devuelve un resultado diferente.

En Windows, devuelve el valor devuelto por el shell después de ejecutar el comando dado, mientras que en Unix, devuelve el estado de salida del proceso.

[59]:
```
import os

returned_value = os.system("mkdir my_first_directory")
print(returned_value)
```

```
0
```

El ejemplo anterior funcionará tanto en Windows como en Unix. En nuestro caso, recibimos el estado de salida 0, que indica éxito en los sistemas Unix.

Esto significa que se ha creado el directorio my_first_directory. Como parte del ejercicio, intenta enumerar el contenido del directorio donde se creó el directorio my_first_directory.

Creado por:

Isabel Maniega