Creado por:

Isabel Maniega

Importar módulos y administrar paquetes de Python usando PIP

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).

Math

1) Ejemplo importar el modulo **math** de Python

In [1]:
```
import math
```

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

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

Out[2]:
```
1.0
```

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

2) Otro ejemplo de importar modulo math en Python

In [3]:
```
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.

In [4]:
```
sin(pi/2)
```

Out[4]:
```
1.0
```

Se llaman directamente sin necesidad de declarar math de nuevo

3) Otro ejemplo de importar modulo math en Python

In [5]:
```
from math import *
```

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

Importando un módulo usando la palabra reservada *as*

In [6]:
```
# pip install pandas
```

In [7]:
```
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.

DIR

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

In [8]:
```
dir(math)
```

Out[8]:
```
['__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']
```

In [9]:
```
dir(pd)
```

Out[9]:
```
['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.

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.

1) Crear un modulo

In [10]:
```
# 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
```

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.

2) mostrar la información de module.py

In [11]:
```
# Podemos repetir los pasos anteriores pero en este punto añadimos al scr
# 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
```

3) '_ _ *name* _ _ '

In [12]:
```
# Podemos repetir los pasos anteriores pero en este punto añadimos al scr
# 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
    # Cuando un archivo se importa como un módulo, su variable __name__ s
```

4) main

In [13]:
```
# Podemos repetir los pasos anteriores pero en este punto añadimos al scr
# 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
    # Cuando un archivo se importa como un módulo, su variable __name__ s
```

5) Contador de llamada de funciones

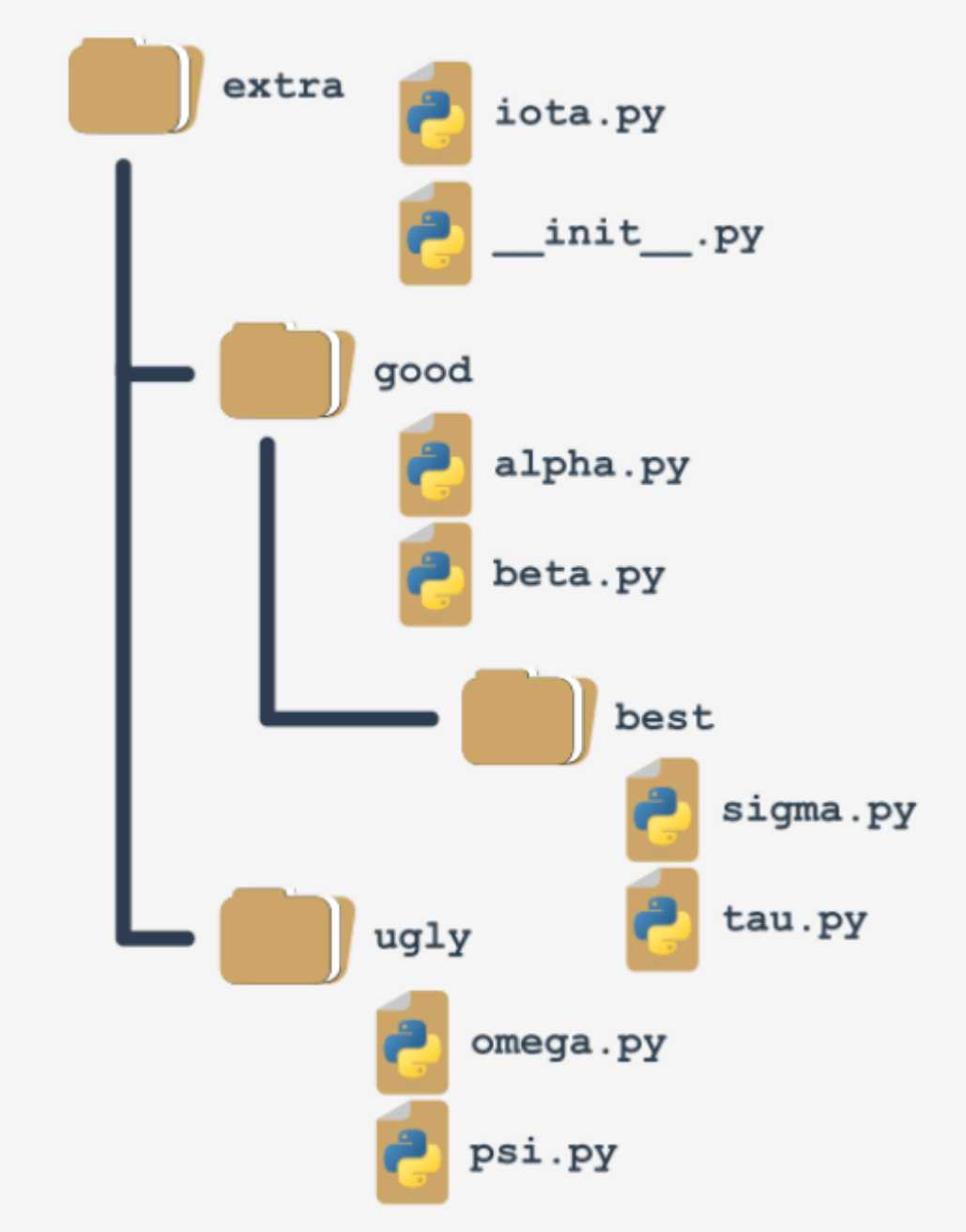
In [14]:
```
# Podemos repetir los pasos anteriores pero en este punto añadimos al scr
# por los tanto los pasos serían:

# 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
# ¿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 var

# A diferencia de muchos otros lenguajes de programación,
# Python no tiene medios para permitirte ocultar tales variables a los oj

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

# Esto se hace anteponiendo al nombre de la variable _ (un guión bajo) o
# pero recuerda, es solo un acuerdo. Los usuarios de tu módulo pueden obe
```

6) Árbol carpetas

In [15]:
```
from IPython import display

display.Image("arbol_carpetas.png")
```

Out[15]:



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.

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/

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

In [16]:
```
pip --version
```

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

In [17]:
```
pip help
```

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

Commands:
  install                     Install packages.
  download                    Download packages.
  uninstall                   Uninstall packages.
  freeze                      Output installed packages in requirements fo
rmat.
  inspect                     Inspect the python environment.
  list                        List installed packages.
  show                        Show information about installed packages.
  check                       Verify installed packages have compatible de
pendencies.
  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 i
ndexes.
  wheel                       Build wheels from your requirements.
  hash                        Compute hashes of package archives.
  completion                  A helper command used for command completio
n.
  debug                       Show information useful for debugging.
  help                        Show help for commands.

General Options:
  -h, --help                  Show help.
  --debug                     Let unhandled exceptions propagate outside t
he
                              main subroutine, instead of logging them to
                              stderr.
  --isolated                  Run pip in an isolated mode, ignoring
                              environment variables and user configuratio
n.
  --require-virtualenv        Allow pip to only run in a virtual environme
nt;
                              exit with an error otherwise.
  --python <python>           Run pip with the specified Python interprete
r.
  -v, --verbose               Give more output. Option is additive, and ca
n be
                              used up to 3 times.
  -V, --version               Show version and exit.
  -q, --quiet                 Give less output. Option is additive, and ca
n be
                              used up to 3 times (corresponding to WARNIN
G,
                              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 wh
ich
                              mechanism to use [disabled, import, subproce
ss].
                              (default: disabled)
  --proxy <proxy>             Specify a proxy in the form
```

```
                              scheme://[user:passwd@]proxy.server:port.
  --retries <retries>         Maximum number of retries each connection sh
ould
                              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)bor
t.
  --trusted-host <hostname>   Mark this host or host:port pair as trusted,
                              even though it does not have valid or any HT
TPS.
  --cert <path>               Path to PEM-encoded CA certificate bundle. I
f
                              provided, overrides the default. See 'SSL
                              Certificate Verification' in pip documentati
on
                              for more information.
  --client-cert <path>        Path to SSL client certificate, a single fil
e
                              containing the private key and the certifica
te
                              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 fo
r
                              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 backwa
rd
                              incompatible.
  --use-deprecated <feature>  Enable deprecated functionality, that will b
e
                              removed in the future.
Note: you may need to restart the kernel to use updated packages.
```

Para saber los paquetes instalados usamos:

- pip list

In [18]:
```
pip list
```

```
Package                   Version
------------------------- -------------
adagio                    0.2.6
aiofiles                  22.1.0
aiosqlite                 0.21.0
altair                    5.5.0
annotated-types           0.7.0
antlr4-python3-runtime    4.11.1
anyio                     4.9.0
appdirs                   1.4.4
argon2-cffi               23.1.0
argon2-cffi-bindings      21.2.0
arrow                     1.3.0
asttokens                 3.0.0
async-lru                 2.0.5
attrs                     25.3.0
babel                     2.17.0
beautifulsoup4            4.13.4
bleach                    6.2.0
blinker                   1.9.0
bokeh                     3.4.0
branca                    0.8.1
cachetools                5.5.2
Cartopy                   0.24.1
certifi                   2025.4.26
cffi                      1.17.1
charset-normalizer        3.4.2
chart-studio              1.1.0
click                     8.2.1
comm                      0.2.2
contourpy                 1.3.2
cycler                    0.12.1
dacite                    1.9.2
debugpy                   1.8.14
decorator                 5.2.1
defusedxml                0.7.1
entrypoints               0.4
executing                 2.2.0
fastjsonschema            2.21.1
Flask                     3.1.1
flexcache                 0.3
flexparser                0.4
folium                    0.19.6
fonttools                 4.58.1
fqdn                      1.5.1
fs                        2.4.16
fsspec                    2025.5.1
fugue                     0.9.1
fugue-jupyter             0.2.3
fugue-sql-antlr           0.2.2
geographiclib             2.0
geopy                     2.4.1
gitdb                     4.0.12
GitPython                 3.1.44
google-api-core           2.25.0
google-api-python-client  2.171.0
google-auth               2.40.3
google-auth-httplib2      0.2.0
google-auth-oauthlib      1.2.2
googleapis-common-protos  1.70.0
```

```
h11                          0.16.0
htmlmin                      0.1.12
httpcore                     1.0.9
httplib2                     0.22.0
httpx                        0.28.1
idna                         3.10
ImageHash                    4.3.1
importlib_resources          6.5.2
ipykernel                    6.29.5
ipython                      9.2.0
ipython-genutils             0.2.0
ipython_pygments_lexers      1.1.1
ipywidgets                   8.1.7
isoduration                  20.11.0
itables                      2.4.2
itsdangerous                 2.2.0
jedi                         0.19.2
Jinja2                       3.1.6
joblib                       1.5.1
json5                        0.12.0
jsonpointer                  3.0.0
jsonschema                   4.23.0
jsonschema-specifications 2025.4.1
jupyter_client               7.4.9
jupyter_core                 5.8.0
jupyter-events               0.12.0
jupyter-lsp                  2.2.5
jupyter_server               2.16.0
jupyter_server_fileid        0.9.3
jupyter_server_terminals     0.5.3
jupyter_server_ydoc          0.8.0
jupyter-ydoc                 0.2.5
jupyterlab                   4.4.3
jupyterlab-lsp               3.10.2
jupyterlab_pygments          0.3.0
jupyterlab_server            2.27.3
jupyterlab_widgets           3.0.15
kiwisolver                   1.4.8
llvmlite                     0.44.0
MarkupSafe                   3.0.2
matplotlib                   3.10.0
matplotlib-inline            0.1.7
MetPy                        1.7.0
mistune                      3.1.3
MouseInfo                    0.1.3
multimethod                  1.12
narwhals                     1.41.0
nbclassic                    1.3.1
nbclient                     0.10.2
nbconvert                    7.16.6
nbformat                     5.10.4
nest-asyncio                 1.6.0
networkx                     3.5
notebook                     7.4.3
notebook_shim                0.2.4
numba                        0.61.0
numpy                        2.1.3
oauthlib                     3.2.2
overrides                    7.7.0
packaging                    24.2
```

```
pandas                    2.2.3
pandas-bokeh              0.5.5
pandocfilters             1.5.1
parso                     0.8.4
patsy                     1.0.1
pexpect                   4.9.0
phik                      0.12.4
pillow                    11.2.1
Pint                      0.24.4
pip                       24.0
platformdirs              4.3.8
plotly                    6.1.2
pooch                     1.8.2
prometheus_client         0.22.0
prompt_toolkit            3.0.51
proto-plus                1.26.1
protobuf                  6.31.1
psutil                    7.0.0
ptyprocess                0.7.0
pure_eval                 0.2.3
puremagic                 1.29
pyarrow                   20.0.0
pyasn1                    0.6.1
pyasn1_modules            0.4.2
PyAutoGUI                 0.9.54
pycparser                 2.22
pydantic                  2.11.6
pydantic_core             2.33.2
pydeck                    0.9.1
PyGetWindow               0.0.9
Pygments                  2.19.1
PyMsgBox                  1.0.9
pyparsing                 3.2.3
pyperclip                 1.9.0
pyproj                    3.7.1
PyRect                    0.2.0
PyScreeze                 1.0.1
pyshp                     2.3.1
python-dateutil           2.9.0.post0
python-json-logger        3.3.0
python3-xlib              0.15
pytweening                1.2.0
pytz                      2025.2
PyWavelets                1.8.0
pywhatkit                 5.4
PyYAML                    6.0.2
pyzmq                     26.4.0
qpd                       0.4.4
referencing               0.36.2
requests                  2.32.3
requests-oauthlib         2.0.0
retrying                  1.3.4
rfc3339-validator         0.1.4
rfc3986-validator         0.1.1
rpds-py                   0.25.1
rsa                       4.9.1
scikit-learn              1.7.0
scipy                     1.15.3
seaborn                   0.13.2
Send2Trash                1.8.3
```

```
setuptools               80.8.0
shapely                  2.1.1
six                      1.17.0
smmap                    5.0.2
sniffio                  1.3.1
soupsieve                2.7
sqlglot                  26.28.0
stack-data               0.6.3
statsmodels              0.14.4
streamlit                1.45.1
sweetviz                 2.3.1
tenacity                 9.1.2
terminado                0.18.1
threadpoolctl            3.6.0
tinycss2                 1.4.0
toml                     0.10.2
tornado                  6.5.1
tqdm                     4.67.1
traitlets                5.14.3
triad                    0.9.8
tropycal                 1.3
typeguard                4.4.3
types-python-dateutil    2.9.0.20250516
typing_extensions        4.14.0
typing-inspection        0.4.1
tzdata                   2025.2
uri-template             1.3.0
uritemplate              4.2.0
urllib3                  2.4.0
visions                  0.8.1
watchdog                 6.0.0
wcwidth                  0.2.13
webcolors                24.11.1
webencodings             0.5.1
websocket-client         1.8.0
Werkzeug                 3.1.3
widgetsnbextension       4.0.14
wikipedia                1.4.0
wordcloud                1.9.4
xarray                   2025.4.0
xyzservices              2025.4.0
y-py                     0.6.2
ydata-profiling          4.16.1
ypy-websocket            0.8.4
Note: you may need to restart the kernel to use updated packages.
```

Para mostrar más información sobre un paquete:

- pip show nombre del paquete

In [19]:
```
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_DS/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

In [20]:
```
# 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

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:

In [21]:
```
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.

In [22]:
```
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.

In [23]:
```
import os

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

```
['3.4_dashboard', '3.1 Gestión de módulos y paquete.ipynb', '3.3_Integracion APIs.ipynb', '.ipynb_checkpoints', 'my_first_directory', 'arbol_carpetas.png', '3.2_Scripts para el procesamiento de alertas y notificaciones.ipynb', 'prod.log']
```

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'](y todo el contenido del catálogo de trabajo actual).

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.

In [24]:
```
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*.

In [25]:
```
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_DS/3. Automatizacion de Procesos y a
lertas/my_first_directory/my_first_directory
/home/isabelmaniega/Documentos/Python_DS/3. Automatizacion de Procesos y a
lertas/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.

In [26]:
```
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:

In [27]:
```
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.

In [28]:
```
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