Creado por:

Isabel Maniega

2.2.1 – Importar módulos y administrar paquetes de Python usando PIP

1. Math
2. PyPI

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

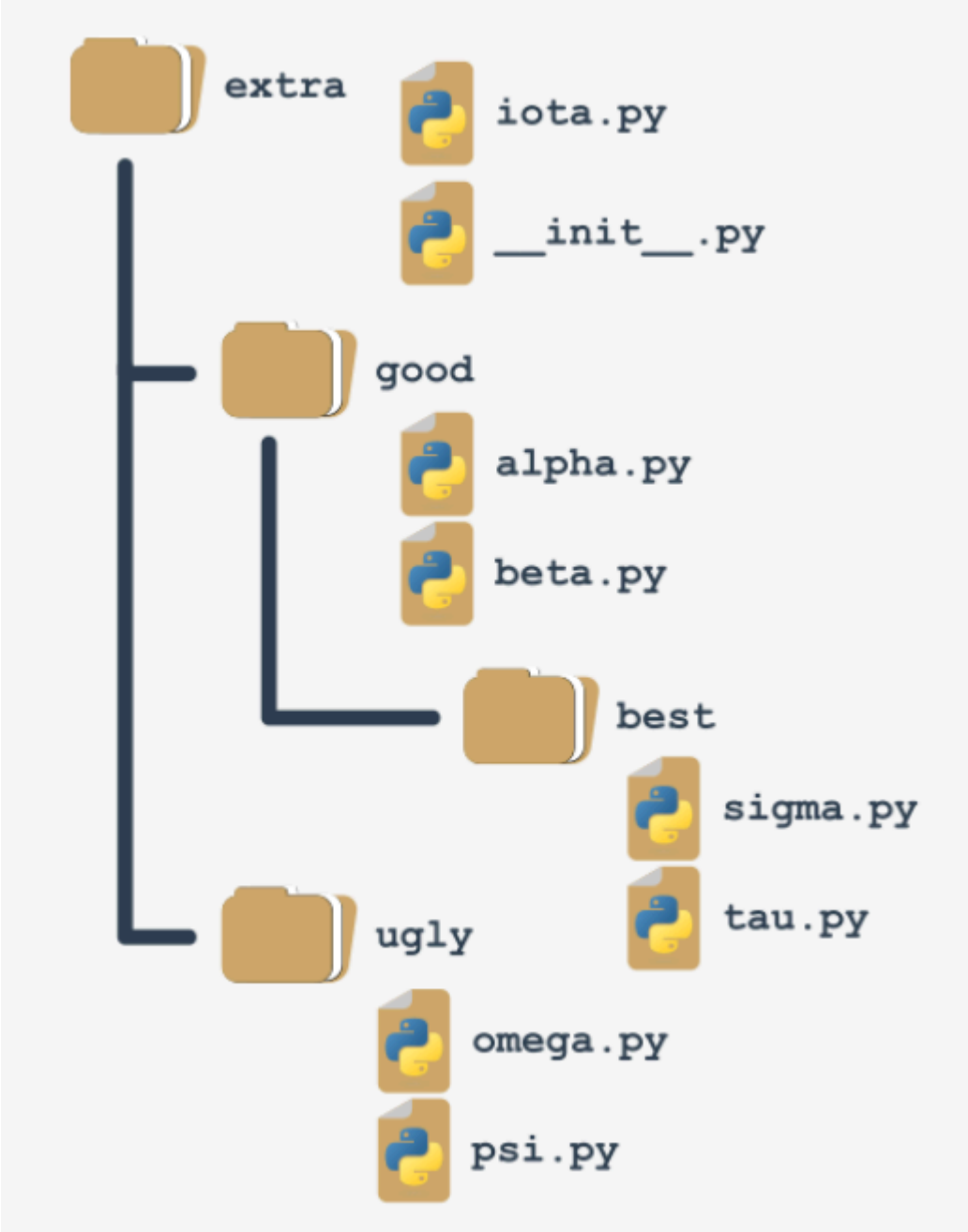
In [14]:
```
# Podemos repetir los pasos anteriores pero en este punto añadimos al scr
# por los tanto los pasos serían:

# 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/PCAD_Data Analyst/env/lib/pyt
hon3.12/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/PCAD_Data Analyst/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
-------------------------- -------------
anyio                      4.9.0
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
certifi                    2025.4.26
cffi                       1.17.1
charset-normalizer         3.4.2
comm                       0.2.2
contourpy                  1.3.2
cycler                     0.12.1
debugpy                    1.8.14
decorator                  5.2.1
defusedxml                 0.7.1
executing                  2.2.0
fastjsonschema             2.21.1
fonttools                  4.58.0
fqdn                       1.5.1
h11                        0.16.0
httpcore                   1.0.9
httpx                      0.28.1
idna                       3.10
ipykernel                  6.29.5
ipython                    9.2.0
ipython_pygments_lexers    1.1.1
isoduration                20.11.0
jedi                       0.19.2
Jinja2                     3.1.6
json5                      0.12.0
jsonpointer                3.0.0
jsonschema                 4.23.0
jsonschema-specifications 2025.4.1
jupyter_client             8.6.3
jupyter_core               5.7.2
jupyter-events             0.12.0
jupyter-lsp                2.2.5
jupyter_server             2.16.0
jupyter_server_terminals   0.5.3
jupyterlab                 4.4.2
jupyterlab_pygments        0.3.0
jupyterlab_server          2.27.3
kiwisolver                 1.4.8
MarkupSafe                 3.0.2
matplotlib                 3.10.3
matplotlib-inline          0.1.7
mistune                    3.1.3
nbclient                   0.10.2
nbconvert                  7.16.6
nbformat                   5.10.4
nest-asyncio               1.6.0
notebook                   7.4.2
notebook_shim              0.2.4
numpy                      2.2.6
```

```
overrides                 7.7.0
packaging                 25.0
pandas                    2.2.3
pandocfilters             1.5.1
parso                     0.8.4
patsy                     1.0.1
pexpect                   4.9.0
pillow                    11.2.1
pip                       24.0
platformdirs              4.3.8
prometheus_client         0.22.0
prompt_toolkit            3.0.51
psutil                    7.0.0
ptyprocess                0.7.0
pure_eval                 0.2.3
pycparser                 2.22
Pygments                  2.19.1
pyparsing                 3.2.3
python-dateutil           2.9.0.post0
python-json-logger        3.3.0
pytz                      2025.2
PyYAML                    6.0.2
pyzmq                     26.4.0
referencing               0.36.2
requests                  2.32.3
rfc3339-validator         0.1.4
rfc3986-validator         0.1.1
rpds-py                   0.25.0
scipy                     1.15.3
seaborn                   0.13.2
Send2Trash                1.8.3
setuptools                80.7.1
six                       1.17.0
sniffio                   1.3.1
soupsieve                 2.7
stack-data                0.6.3
statsmodels               0.14.4
terminado                 0.18.1
tinycss2                  1.4.0
tornado                   6.5
traitlets                 5.14.3
types-python-dateutil     2.9.0.20250516
typing_extensions         4.13.2
tzdata                    2025.2
uri-template              1.3.0
urllib3                   2.4.0
wcwidth                   0.2.13
webcolors                 24.11.1
webencodings              0.5.1
websocket-client          1.8.0
Note: you may need to restart the kernel to use updated packages.
```

Para mostrar más información sobre un paquete:

- pip show nombre del paquete

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/PCAD_Data Analyst/env/lib/python
3.12/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

2.2.2 – Aplicar manejo básico de excepciones y mantener la solidez del script.

-1- Excepciones: en la web de Python

In [21]:
```
# https://docs.python.org/3/library/exceptions.html
```

-2- Ejemplo básico try-except

In [22]:
```
# Imagina que tenemos 2 variables
```

In [23]:
```
x = 5
# 'y' no la tenemos definida
```

In [24]:
```
print(x)
```

```
5
```

In [25]:
```
# print(y)
# NameError: name 'y' is not defined
# OBVIAMENTE DA ERROR, AL NO TENERLA DEFINIDA
```

In [26]:
```
try:
    print(x)
except:
    print("No tenemos definida la variable:")
```

```
5
```

In [27]:
```
try:
    print(y)
except:
    print("No tenemos definida la variable:")
```

```
No tenemos definida la variable:
```

In [28]:
```
# De esta forma podemos conseguir que un código funcione saltando un erro

# Pero OJO! en donde colocamos este Try-Except. Porque si es algo crítico

# Solo sirve cuando es algo que necesitamos saltar,

# (para que el código ejecute en un momento que sabemos que algo no va)
```

-3- Forma básica de crear una excepción

-3.1- En la División por cero

In [29]:
```
a = 5
b = 0
# a/b
```

```
# Si descomentamos "a/b" nos sale:
# ZeroDivisionError: division by zero

# No es posible dividir un número por cero. (Daria infinito)
```

Podemos hacer lo siguiente, sin excepciones

In [30]:
```
def funcion_dividir_1(a, b):
    if b != 0:
        print(a / b)
    else:    # b = 0  -> no puede dividir
        print("el denominador es 0, no podemos dividir")
```

In [31]:
```
funcion_dividir_1(2,3)
```

```
0.6666666666666666
```

In [32]:
```
funcion_dividir_1(2,0)
```

```
el denominador es 0, no podemos dividir
```

el mismo ejercicio cambiando el operador

In [33]:
```
def funcion_dividir_1(a, b):
    if b == 0:
        print("el denominador es 0, no podemos dividir")
    else:      # b es distinto de 0
        print(a/b)
```

In [34]:
```
funcion_dividir_1(2,3)
```

```
0.6666666666666666
```

In [35]:
```
funcion_dividir_1(2,0)
```

```
el denominador es 0, no podemos dividir
```

-4- Uso de raise

In [36]:
```
# Podemos lanzar excepciones, no lo usaremos
```

-5- Try-Except-Else

In [37]:
```
# me creo una función para comprobar más casos.

def funcion_division(a,b):
    try:
        division = a / b
        print('estamos en try y hemos calculado a/b')
    except ZeroDivisionError:
        print("Un número dividido por 0 sale infinito")
        print("No pongas un 0 en el deniminador!")
```

```
    else:
        print('estamos en el else')
        print('valor de la división:', division)
```

In [38]:
```
funcion_division(1,0)
```

```
Un número dividido por 0 sale infinito
No pongas un 0 en el deniminador!
```

In [39]:
```
funcion_division(1,2)
```

```
estamos en try y hemos calculado a/b
estamos en el else
valor de la división: 0.5
```

-6- try-except-else con archivos

Un archivo que no existe

(o no se encuentra en ese lugar)

In [40]:
```
try:
    f = open('archivo_excepciones.txt')  # El fichero no existe
except FileNotFoundError:
    print('¡El fichero no existe!')
else:
    print(f.read())
```

```
¡El fichero no existe!
```

In [41]:
```
# lo cerramos, si esta abierto
# f.close()
```

Un archivo que SI existe

(y lo encuentra en esa ubicación)

In [42]:
```
try:
    f = open('archivo_excepciones_2.txt')  # El fichero si existe
except FileNotFoundError:
    print('¡El fichero no existe!')
else:
    print(f.read())
    f.close()
```

```
¡El fichero no existe!
```

-7- Errores cuando sumamos strings en vez de números

In [43]:
```
# print(2+"2")
# TypeError: unsupported operand type(s) for +: 'int' and 'str'

# al hacer esa operación nos devuelve un error
```

In [44]:
```
3
```

Out[44]: 3

In [45]:
```
type(3)
```

Out[45]: int

In [46]:
```
str(3)
```

Out[46]: '3'

In [47]:
```
'3'
```

Out[47]: '3'

In [48]:
```
type('3')
```

Out[48]: str

In [49]:
```
def funcion_formatos_diferentes(a,b):
    try:
        suma = a + b
        print(suma)
    except TypeError:
        print("revisa el formato de los números, porque no es correcto")
```

In [50]:
```
funcion_formatos_diferentes(2,"3")
```

```
revisa el formato de los números, porque no es correcto
```

In [51]:
```
funcion_formatos_diferentes(2,3)
```

```
5
```

-8- except Exception

Otra forma si no sabes que excepción puede saltar, puedes usar la clase genérica Exception. Sirve para cualquier tipo de excepción. De hecho todas las excepciones heredan de Exception

except Exception: Ejemplo 1

In [52]:
```
def funcion_suma_2(a,b):
    try:
        suma = a + b
        print("la suma es: ", suma)
    except Exception:
        print("Ha habido una excepción")
```

In [53]:
```
funcion_suma_2(2,0)
```

```
la suma es:  2
```

In [54]:
```
funcion_suma_2(2,"2")
```

```
Ha habido una excepción
```

except Exception: Ejemplo 2

In [55]:
```
def funcion_division_3(a,b):
    try:
        division = a / b
        print("la division es: ", division)
    except Exception:
        print("Ha habido una excepción")
```

In [56]:
```
funcion_division_3(1,3)
```

```
la division es:  0.3333333333333333
```

In [57]:
```
funcion_division_3(2,0)
```

```
Ha habido una excepción
```

In [58]:
```
funcion_division_3(2,"2")
```

```
Ha habido una excepción
```

-9- except Exception as e (una de las mejores opciones)

In [59]:
```
def funcion_division_4(a,b):
    try:
        division = a / b
        print("la division es: ", division)
    except Exception as e:
        print("Ha habido una excepción")
        print("tipo del error: ", type(e))
        print('str(e):', str(e))
```

In [60]:
```
funcion_division_4(1,3)
```

```
la division es:  0.3333333333333333
```

In [61]:
```
funcion_division_4(2,0)
```

```
Ha habido una excepción
tipo del error:  <class 'ZeroDivisionError'>
str(e): division by zero
```

In [62]:
```
funcion_division_4(2,"2")
```

```
Ha habido una excepción
tipo del error:  <class 'TypeError'>
str(e): unsupported operand type(s) for /: 'int' and 'str'
```

-10- except Exception as e (otra posibilidad: try-except-else)

In [63]:
```
def funcion_division_5(a,b):
    try:
        division = a / b
        print("la division es: ", division)
    except Exception as e:
        print("Ha habido una excepción")
        print("tipo del error: ", type(e))
    else:
        print("estamos en else, no hubo excepciones")
```

In [64]:
```
funcion_division_5(1,3)
```

```
la division es:  0.3333333333333333
estamos en else, no hubo excepciones
```

In [65]:
```
funcion_division_5(2,0)
```

```
Ha habido una excepción
tipo del error:  <class 'ZeroDivisionError'>
```

In [66]:
```
funcion_division_5(2,"2")
```

```
Ha habido una excepción
tipo del error:  <class 'TypeError'>
```

-11- except Exception as e (otra posibilidad: try-except-finally)

Este bloque se suele usar si queremos ejecutar algún tipo de acción de limpieza.

Si por ejemplo estamos escribiendo datos en un fichero pero ocurre una excepción,

tal vez queramos borrar el contenido que hemos escrito con anterioridad,

para no dejar datos inconsistenes en el fichero.

In [67]:
```
def funcion_division_6(a,b):
    try:
        division = a / b
        print("la division es: ", division)
        print("\n")
    except Exception as e:
        print("Ha habido una excepción")
        print("tipo del error: ", type(e))
        print("\n")
    finally:
```

```
        print("estamos en finally")
        print("esto se ejecuta SIEMPRE haya o no excepciones")
```

In [68]:
```
funcion_division_6(1,3)
```

```
la division es:  0.3333333333333333


estamos en finally
esto se ejecuta SIEMPRE haya o no excepciones
```

In [69]:
```
funcion_division_6(2,0)
```

```
Ha habido una excepción
tipo del error:  <class 'ZeroDivisionError'>


estamos en finally
esto se ejecuta SIEMPRE haya o no excepciones
```

In [70]:
```
funcion_division_6(2,"2")
```

```
Ha habido una excepción
tipo del error:  <class 'TypeError'>


estamos en finally
esto se ejecuta SIEMPRE haya o no excepciones
```

-12- Ejemplo de excepciones con archivos

In [71]:
```
def funcion_lectura(archivo):
    try:
        with open(archivo) as file:
            lectura_archivo = file.read()
            print(lectura_archivo)
    except Exception as e:
        print("no se pudo abrir")
        print("Tipo de error:", type(e))
        print(str(e))
```

In [72]:
```
funcion_lectura('archivo_excepciones_1.txt')  # no lo encuentra
```

```
no se pudo abrir
Tipo de error: <class 'FileNotFoundError'>
[Errno 2] No such file or directory: 'archivo_excepciones_1.txt'
```

In [73]:
```
funcion_lectura('archivo_excepciones_2.txt') # si lo encuentra

# (si lo coloco yo previamente este archivo)
# SE ENCUENTRA EN LA MISMA RUTA
```

```
no se pudo abrir
Tipo de error: <class 'FileNotFoundError'>
[Errno 2] No such file or directory: 'archivo_excepciones_2.txt'
```

Más sobre Excepciones...

Ordenadas las excepciones por orden de preferencia:

```
BaseException
├── BaseExceptionGroup
├── GeneratorExit
├── KeyboardInterrupt
├── SystemExit
└── Exception
    ├── ArithmeticError
    │    ├── FloatingPointError
    │    ├── OverflowError
    │    └── ZeroDivisionError
    ├── AssertionError
    ├── AttributeError
    ├── BufferError
    ├── EOFError
    ├── ExceptionGroup [BaseExceptionGroup]
    ├── ImportError
    │    └── ModuleNotFoundError
    ├── LookupError
    │    ├── IndexError
    │    └── KeyError
    ├── MemoryError
    ├── NameError
    │    └── UnboundLocalError
    ├── OSError
    │    ├── BlockingIOError
    │    ├── ChildProcessError
    │    ├── ConnectionError
    │    │    ├── BrokenPipeError
    │    │    ├── ConnectionAbortedError
    │    │    ├── ConnectionRefusedError
    │    │    └── ConnectionResetError
    │    ├── FileExistsError
    │    ├── FileNotFoundError
    │    ├── InterruptedError
    │    ├── IsADirectoryError
    │    ├── NotADirectoryError
    │    ├── PermissionError
    │    ├── ProcessLookupError
    │    └── TimeoutError
    ├── ReferenceError
    ├── RuntimeError
    │    ├── NotImplementedError
    │    └── RecursionError
    ├── StopAsyncIteration
    ├── StopIteration
    ├── SyntaxError
    │    └── IndentationError
    │         └── TabError
```

```
        ├── SystemError
        ├── TypeError
        ├── ValueError
        │    └── UnicodeError
        │         ├── UnicodeDecodeError
        │         ├── UnicodeEncodeError
        │         └── UnicodeTranslateError
        └── Warning
             ├── BytesWarning
             ├── DeprecationWarning
             ├── EncodingWarning
             ├── FutureWarning
             ├── ImportWarning
             ├── PendingDeprecationWarning
             ├── ResourceWarning
             ├── RuntimeWarning
             ├── SyntaxWarning
             ├── UnicodeWarning
             └── UserWarning
```

Tipos de excepciones más relevantes

Para capturar cualquier excepción podemos usar Exception o BaseException:

In [74]:
```
try:
    30* (2/0)
except BaseException as e:
    print('Error %s' % str(e))
```

```
Error division by zero
```

In [75]:
```
try:
    30* (2/0)
except Exception as e:
    print('Error %s' % str(e))
```

```
Error division by zero
```

ArithmeticError

- Division entre 0: **ZeroDivisionError**

In [76]:
```
30 * (2/0)
```

```
---------------------------------------------------------------------------
ZeroDivisionError                         Traceback (most recent call last)
Cell In[76], line 1
----> 1 30 * (2/0)

ZeroDivisionError: division by zero
```

In [77]:
```
try:
    30* (2/0)
```

```
except ZeroDivisionError as e:
    print('Error %s' % str(e))
```

```
Error division by zero
```

AttributeError

- Error en el uso: **AttributeError**

In [78]:
```
num = 10
num.append(6)
print(num)
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
Cell In[78], line 2
      1 num = 10
----> 2 num.append(6)
      3 print(num)

AttributeError: 'int' object has no attribute 'append'
```

In [79]:
```
try:
    num= 10
    num.append(6)
    print(num)
except AttributeError as e:
    print('Error %s' % str(e))
```

```
Error 'int' object has no attribute 'append'
```

ImportError

- Error al importar un módulo: **ImportError**

In [80]:
```
from pandas import hola
```

```
---------------------------------------------------------------------------
ImportError                               Traceback (most recent call last)
Cell In[80], line 1
----> 1 from pandas import hola

ImportError: cannot import name 'hola' from 'pandas' (/home/isabelmaniega/Documentos/PCAD_Data Analyst/env/lib/python3.12/site-packages/pandas/__init__.py)
```

In [81]:
```
try:
    from pandas import hola
except ImportError as e:
    print('Error %s' % str(e))
```

```
Error cannot import name 'hola' from 'pandas' (/home/isabelmaniega/Documentos/PCAD_Data Analyst/env/lib/python3.12/site-packages/pandas/__init__.py)
```

Error en importar un modulo será: **ModuleNotFoundError**

In [82]:
```
try:
    import hola
except ModuleNotFoundError as e:
    print('Error %s' % str(e))
```

```
Error No module named 'hola'
```

LookupError

- Error de índice: **IndexError**

In [83]:
```
L = [10, 50, 60]
L[3]
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
Cell In[83], line 2
      1 L = [10, 50, 60]
----> 2 L[3]

IndexError: list index out of range
```

In [84]:
```
try:
    L = [10, 50, 60]
    L[3]
except IndexError as e:
    print('Error %s' % str(e))
```

```
Error list index out of range
```

Si en vez de poner un número entero ponemos un string el error sería de tipo:

In [85]:
```
try:
    L = [10, 50, 60]
    L['3']
except IndexError as e:
    print('Error Index %s' % str(e))
except TypeError as e:
    print('Error TypeError %s' % str(e))
```

```
Error TypeError list indices must be integers or slices, not str
```

- Error de clave en un diccionario: **KeyError**

In [86]:
```
ages = {'Juan': 25, 'Luis':36, 'Pedro':41}
ages['Maria']
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
Cell In[86], line 2
      1 ages = {'Juan': 25, 'Luis':36, 'Pedro':41}
----> 2 ages[       ]

KeyError: 'Maria'
```

In [87]:
```
try:
    ages = {'Juan': 25, 'Luis':36, 'Pedro':41}
    ages['Maria']
except KeyError as e:
    print('Error %s' % str(e))
```

```
Error 'Maria'
```

NameError

- Nombre no definido: **NameError**

In [88]:
```
4 + w*3
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[88], line 1
----> 1 4 + w*3

NameError: name 'w' is not defined
```

In [89]:
```
try:
    4 + w*3
except NameError as e:
    print('Error %s' % str(e))
```

```
Error name 'w' is not defined
```

OSError

- Archivo no encontrado: **FileNotFoundError**

In [90]:
```
try:
    file = open('data.csv')
except FileNotFoundError as e:
    print('Error %s' % str(e))
```

```
Error [Errno 2] No such file or directory: 'data.csv'
```

- Archivo no encontrado: **NotADirectoryError**

In [91]:
```
try:
    # crear una carpeta vacia llamada "solucion" al lado del archivo Exce
    file = open('./solucion')
```

```
except IsADirectoryError as e:
    print('Error %s' % str(e))
```

```
---------------------------------------------------------------------------
FileNotFoundError                         Traceback (most recent call last)
Cell In[91], line 3
      1 try:
      2     # crear una carpeta vacia llamada "solucion" al lado del archivo Excepciones.ipynb
----> 3     file = open(            )
      4 except IsADirectoryError as e:
      5     print('Error %s' % str(e))

File ~/Documentos/PCAD_Data Analyst/env/lib/python3.12/site-packages/IPython/core/interactiveshell.py:326, in _modified_open(file, *args, **kwargs)
    319 if file in {0, 1, 2}:
    320     raise ValueError(
    321         f"IPython won't let you open fd={file} by default "
    322         "as it is likely to crash IPython. If you know what you are doing, "
    323         "you can use builtins' open."
    324     )
--> 326 return io_open(file, *args, **kwargs)

FileNotFoundError: [Errno 2] No such file or directory: './solucion'
```

SyntaxError

- Error en la identación o sintaxis: **SyntaxError**

IndentationError

In [92]:
```
name = 'Pepe'

if name == 'Pepe':
print('El nombre es Pepe')
```

```
  Cell In[92], line 4
    print('El nombre es Pepe')
    ^
IndentationError: expected an indented block after 'if' statement on line 3
```

In [93]:
```
name = 'Pepe'

try:
    if name == 'Pepe':
    print('El nombre es Pepe')
except IndentationError as e:
    print('Error %s' % str(e))
```

```
  Cell In[93], line 5
    print('El nombre es Pepe')
    ^
IndentationError: expected an indented block after 'if' statement on line
4
```

Este no se puede capturar, solo si se realiza con dos scripts y se importa uno en otro, podremos realizar la excepción

In [94]:
```
# test1.py
try:
    import test2
except IndentationError as ex:
    print(ex)

# test2.py
def f():
    pass
        pass # error
```

```
  Cell In[94], line 10
    pass # error
    ^
IndentationError: unexpected indent
```

SyntaxError

Por ejemplo si definimos mal un string, lista, etc se nos olvida el cierre

In [95]:
```
name = 'Pepe
```

```
  Cell In[95], line 1
    name = 'Pepe
           ^
SyntaxError: unterminated string literal (detected at line 1)
```

In [96]:
```
try:
    name = 'Pepe
except SyntaxError as e:
    print('Error %s' str(e))
```

```
  Cell In[96], line 2
    name = 'Pepe
           ^
SyntaxError: unterminated string literal (detected at line 2)
```

Pasa lo mismo que con la identación, no se puede capturar.

TypeError

- Error de tipo de variable: **TypeError**

In [97]:
```
'4' + 2
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[97], line 1
----> 1 '4' + 2

TypeError: can only concatenate str (not "int") to str
```

In [98]:
```
try:
    '4' + 2
except TypeError as e:
    print('Error %s' % str(e))
```

```
Error can only concatenate str (not "int") to str
```

ValueError

- Error al recibir un error de tipo o de valor inapropiado: **ValueError**

In [99]:
```
import math

x = -3

print(f'Square Root of {x} is {math.sqrt(x)}')
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[99], line 5
      1 import math
      3 x = -3
----> 5 print(f'Square Root of {x} is {math.sqrt(x)}')

ValueError: math domain error
```

In [100…
```
x = -3

try:
    print(f'Square Root of {x} is {math.sqrt(x)}')
except ValueError as ve:
    print(f'You entered {x}, which is not a positive number.')
    print('Error %s' % str(ve))
```

```
You entered -3, which is not a positive number.
Error math domain error
```

Múltiples excepciones

En el caso de declarar multiples excepciones se tomarán por orden de preferencia según la primera tabla al ser detectados, para declararlos se pone except y entre paréntesis las excepciones:

In [101…
```
try:
    # Error al pulsar enter sin insertar dato o insertar un texto, error
    value = input('Inserte un número: ')
```

```
    result = 25/int(value)
    print(f'El resultado es: {result}')
    print(f'Square Root of {x} is {math.sqrt(int(value))}')
    L = [10, 5, 6]
    print(f'El valor en la lista es: {L[value]}')
except (IndexError, TypeError, ValueError) as e:
    print('Error %s' % str(e))
```

```
Error invalid literal for int() with base 10: ''
```

In [102…

```
try:
    # Error al poner un número. el index es un string
    value = input('Inserte un número: ')
    result = 25/int(value)
    print(f'El resultado es: {result}')
    print(f'Square Root of {x} is {math.sqrt(int(value))}')
    L = [10, 5, 6]
    print(f'El valor en la lista es: {L[value]}')
except (IndexError, TypeError, ValueError) as e:
    print('Error %s' % str(e))
```

```
El resultado es: 12.5
Square Root of -3 is 1.4142135623730951
Error list indices must be integers or slices, not str
```

In [103…

```
try:
    # Error al poner un número negativo
    value = input('Inserte un número: ')
    result = 25/int(value)
    print(f'El resultado es: {result}')
    print(f'Square Root of {x} is {math.sqrt(int(value))}')
    L = [10, 5, 6]
    print(f'El valor en la lista es: {L[value]}')
except (IndexError, TypeError, ValueError) as e:
    print('Error %s' % str(e))
```

```
El resultado es: -12.5
Error math domain error
```

Raise

También se puede usar raise directamente con las excepciones:

In [104…

```
# Error al poner un número un número negativo
value = input('Inserte un número: ')

if not type(value) is int:
    raise TypeError('Error en el index')
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call las
t)
Cell In[104], line 5
      2 value = input('Inserte un número: ')
      4 if not type(value) is int:
----> 5     raise TypeError('Error en el index')

TypeError: Error en el index
```

Creado por:

Isabel Maniega