Creado por:

Isabel Maniega

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

In [6]:
```
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 [7]:
```
dir(math)
```

Out[7]:
```
['__doc__',
 '__loader__',
 '__name__',
 '__package__',
 '__spec__',
 'acos',
 'acosh',
 'asin',
 'asinh',
 'atan',
 'atan2',
 'atanh',
 'ceil',
 'comb',
 'copysign',
 'cos',
 'cosh',
 'degrees',
 'dist',
 'e',
 'erf',
 'erfc',
 'exp',
 'expm1',
 'fabs',
 'factorial',
 'floor',
 'fmod',
 'frexp',
 'fsum',
 'gamma',
 'gcd',
 'hypot',
 'inf',
 'isclose',
 'isfinite',
 'isinf',
 'isnan',
 'isqrt',
 'ldexp',
 'lgamma',
 'log',
 'log10',
 'log1p',
 'log2',
 'modf',
 'nan',
 'perm',
 'pi',
 'pow',
 'prod',
 'radians',
 'remainder',
 'sin',
 'sinh',
 'sqrt',
 'tan',
 'tanh',
 'tau',
 'trunc']
```

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

Out[8]:
```
['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__',
 '_config',
 '_is_numpy_dev',
 '_libs',
 '_testing',
```

```
 '_typing',
 '_version',
 '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.

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.

Ejemplo 1:

Generamos números pseudoaleatorios de 0.0 a 1.0

In [20]:
```
from random import random

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

```
0.9081128851953352
0.5046868558173903
0.28183784439970383
0.7558042041572239
0.6183689966753316
```

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

Ejemplo 2: seed

Añadimos una semilla de 0

In [18]:
```
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.

Ejemplo 3: randrange, randint

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

- randrange(inicio, fin, incremento)
- randint(izquierda, derecha)

In [22]:
```
from random import randrange, randint

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

```
0 0 0 0
```

Nota: Observa que los datos generados no son único

In [27]:
```
from random import randint

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

```
6, 9, 4, 10, 9, 10, 5, 8, 2, 10,
```

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.

In [28]:
```
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))
```

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

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.

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 [ ]:
```
# 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 [ ]:
```
# 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.")
# 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 [ ]:
```
# 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__

# 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 [ ]:
```
# 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 [ ]:
```
# 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:
    # counter = 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
```

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 [29]:
```
pip --version
```

```
pip 23.3.2 from /home/isabelmaniega/Documentos/Python_Básico_cas/env/lib/p
ython3.8/site-packages/pip (python 3.8)
Note: you may need to restart the kernel to use updated packages.
```

In [30]:
```
pip help
```

```
Usage:
  /home/isabelmaniega/Documentos/Python_Básico_cas/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 [31]:
```
pip list
```

```
Package                   Version
------------------------- --------------
anyio                     4.2.0
appdirs                   1.4.4
argon2-cffi               23.1.0
argon2-cffi-bindings      21.2.0
arrow                     1.3.0
asttokens                 2.4.1
async-lru                 2.0.4
attrs                     23.2.0
Babel                     2.14.0
backcall                  0.2.0
beautifulsoup4            4.12.3
bleach                    6.1.0
certifi                   2023.11.17
cffi                      1.16.0
charset-normalizer        3.3.2
comm                      0.2.1
contourpy                 1.1.1
cycler                    0.12.1
debugpy                   1.8.0
decorator                 5.1.1
defusedxml                0.7.1
exceptiongroup            1.2.0
executing                 2.0.1
fastjsonschema            2.19.1
fonttools                 4.48.1
fqdn                      1.5.1
idna                      3.6
importlib-metadata        7.0.1
importlib-resources       6.1.1
ipykernel                 6.29.0
ipython                   8.12.3
isoduration               20.11.0
jedi                      0.19.1
Jinja2                    3.1.3
json5                     0.9.14
jsonpointer               2.4
jsonschema                4.21.1
jsonschema-specifications 2023.12.1
jupyter_client            8.6.0
jupyter_core              5.7.1
jupyter-events            0.9.0
jupyter-lsp               2.2.2
jupyter_server            2.12.5
jupyter_server_terminals  0.5.2
jupyterlab                4.0.11
jupyterlab_pygments       0.3.0
jupyterlab_server         2.25.2
kiwisolver                1.4.5
MarkupSafe                2.1.4
matplotlib                3.7.4
matplotlib-inline         0.1.6
mistune                   3.0.2
nbclient                  0.9.0
nbconvert                 7.14.2
nbformat                  5.9.2
nest-asyncio              1.6.0
notebook                  7.0.7
notebook-as-pdf           0.5.0
```

```
notebook_shim             0.2.3
numpy                     1.24.4
overrides                 7.6.0
packaging                 23.2
pandas                    2.0.3
pandocfilters             1.5.1
parso                     0.8.3
pexpect                   4.9.0
pickleshare               0.7.5
pillow                    10.2.0
pip                       23.3.2
pkgutil_resolve_name      1.3.10
platformdirs              4.1.0
prometheus-client         0.19.0
prompt-toolkit            3.0.43
psutil                    5.9.8
ptyprocess                0.7.0
pure-eval                 0.2.2
pycparser                 2.21
pyee                      8.2.2
Pygments                  2.17.2
pyparsing                 3.1.1
PyPDF2                    2.12.0
pyppeteer                 1.0.2
python-dateutil           2.8.2
python-json-logger        2.0.7
pytz                      2023.3.post1
PyYAML                    6.0.1
pyzmq                     25.1.2
referencing               0.32.1
requests                  2.31.0
rfc3339-validator         0.1.4
rfc3986-validator         0.1.1
rpds-py                   0.17.1
Send2Trash                1.8.2
setuptools                69.0.3
six                       1.16.0
sniffio                   1.3.0
soupsieve                 2.5
stack-data                0.6.3
terminado                 0.18.0
tinycss2                  1.2.1
tomli                     2.0.1
tornado                   6.4
tqdm                      4.66.1
traitlets                 5.14.1
types-python-dateutil     2.8.19.20240106
typing_extensions         4.9.0
tzdata                    2023.4
uri-template              1.3.0
urllib3                   1.26.18
wcwidth                   0.2.13
webcolors                 1.13
webencodings              0.5.1
websocket-client          1.7.0
websockets                10.4
wheel                     0.42.0
zipp                      3.17.0

[notice] A new release of pip is available: 23.3.2 -> 24.0
```

```
[notice] To update, run: pip install --upgrade pip
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 [32]:
```
pip show pip
```

```
Name: pip
Version: 23.3.2
Summary: The PyPA recommended tool for installing Python packages.
Home-page: https://pip.pypa.io/
Author: The pip developers
Author-email: distutils-sig@python.org
License: MIT
Location: /home/isabelmaniega/Documentos/Python_Básico_cas/env/lib/python
3.8/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 [34]:
```
# 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

Módulo Re

RegEx, Regular Expression, este módulo proporciona operaciones de coincidencia de expresiones regulares similares a las encontradas en Perl.

Las expresiones regulares usan el carácter de barra inversa ('') para indicar formas especiales o para permitir el uso de caracteres especiales sin invocar su significado especial.

In [35]:
```
# Python para imprimir una barra necesitamos usar dos:
print('\\')
```

```
\
```

In [36]:
```
# Si sólo usamos una única barra nos arroja un error de Syntaxis:
print('\')
```

```
  Cell In[36], line 2
    print('\')
                ^
SyntaxError: EOL while scanning string literal
```

La solución es usar la notación de cadena raw de Python para los patrones de expresiones regulares; las barras inversas no se manejan de ninguna manera especial en un literal de cadena prefijado con 'r'. Así que r"\n" es una cadena de dos caracteres que contiene '' y 'n', mientras que "\n" es una cadena de un carácter que contiene una nueva línea. Normalmente los patrones se expresan en código Python usando esta notación de cadena raw.

In [37]:
```
print(r'\n')
```

```
\n
```

In [38]:
```
print('Antes de...')
print('\n')
print('Salto de línea')
```

```
Antes de...


Salto de línea
```

Una expresión regular (o RE, por sus siglas en inglés) especifica un conjunto de cadenas que coinciden con ella; las funciones de este módulo permiten comprobar si una determinada cadena coincide con una expresión regular dada (o si una expresión regular dada coincide con una determinada cadena, que se reduce a lo mismo).

In [39]:
```
import re # Importamos el paquete re

# Comprobar que la frase empieza por "The" y acaba por "Spain":

txt = "The rain in Spain"
x = re.search("^The.*Spain$", txt)

if x:
  print("YES! We have a match!")
else:
  print("No match")
```

```
YES! We have a match!
```

Funciones de RegEx

- findall: retorna una lista que contiene todas las coincidencias
- search: retorna la coincidencia del objeto.
- split: Divide la string («cadena») por el número de ocurrencias del pattern («patrón»).
- sub: Retorna la cadena obtenida reemplazando las ocurrencias no superpuestas del pattern («patrón») en la string («cadena») por el reemplazo de repl.

Metacaracteres

- [] Conjunto de caracteres, ejemplo: "[a-m]"
- \ Caracter de escape, ejemplo: "\d"
- . Cualquier carácter (excepto carácter de nueva línea), ejemplo: "he..o"
- ^ Empezar por, ejemplo: "^hello"
- \$ Acabar con, ejemplo: "planet\$"
- * Cero o más ocurrencias, ejemplo: "he.*o"
- + Una o más concurrencias, ejemplo: "he.+o"
- ? cero o una concurrencia, ejemplo: "he.?o"
- {} Exactamente el número especificado de ocurrencias, ejemplo: "he.{2}o"
- | Cualquiera o, ejemplo: "falls|stays"
- () Capturar y grupo, ejemplo:

In [40]:
```
import re

txt = "The rain in Spain"

# Encontrar todas las minúsculas alfabéticas entre la "a" y "m":

x = re.findall("[a-m]", txt)
print(x)
```

```
['h', 'e', 'a', 'i', 'i', 'a', 'i']
```

In [41]:
```
import re

txt = "That will be 59 dollars"
```

```
# Encontrar los números en la frase:

x = re.findall("\d", txt)
print(x)
```

```
['5', '9']
```

In [42]:
```
import re

txt = "hello planet"

# Busque una secuencia que comience con "he" (cocidencias exactas, he, no

x = re.findall("he..o", txt)
print(x)
```

```
['hello']
```

In [43]:
```
# /s --> espacio en blanco, buscar la primera posición en blanco:

import re

txt = "The rain in Spain"
x = re.search("\s", txt)

print("The first white-space character is located in position:", x.start(
```

```
The first white-space character is located in position: 3
```

In [44]:
```
#  Buscar si esta Portugal en la frase:

import re

txt = "The rain in Spain"
x = re.search("Portugal", txt)
print(x)
```

```
None
```

In [45]:
```
# Dividir la frase cuando encuentre un espacio en blanco (\s):

import re

txt = "The rain in Spain"
x = re.split("\s", txt)
print(x)
```

```
['The', 'rain', 'in', 'Spain']
```

In [46]:
```
# Sustituir espacio en blanco por 9:

import re

txt = "The rain in Spain"
x = re.sub("\s", "9", txt)
print(x)
```

```
The9rain9in9Spain
```

Creado por:

Isabel Maniega