Python en terminal:

Se puede usar _ para referir al ultimo resultado

```
>>> 10 * 10
100
>>> 10 + _
110
```

Strings:

```
>>> print('"Isn\'t," they said.')
```

raw string (r''): sin formatear caracteres especiales con \

```
>>> print(r'C:\some\name')  # note the r before the quote
```

Para hacer multiple linia:

manteniendo format, se puede usar """ """ o ''' '''.

Para que no genere el salto de linia extra, se usa \

```
print("""\
   Usage: thingy [OPTIONS]
        -h                        Display this usage message
        -H hostname               Hostname to connect to\
""")
```

Strings de literals separados por espacio, se juntan

```
text = ('Put several strings within parentheses '
        'to have them joined together.')
# Put several strings within parentheses to have them joined
together.
```

Concatenando variables:

Formated strings (f'')

```
>>> year = 2016
>>> event = 'Referendum'
>>> f'Results of the {year} {event}'
```

Strings se pueden repetir con * y concatenar con +

```
>>> 3 * 'un' + 'ium'
'unununium'
```

Strings se pueden indexar, index negativo va al reves (desde la ultima letra)

```
word = 'Python'
word[1] #y
word[-1] # n
```

Tambien se puede hacer slicing "devolver una parte del string"

```
>>> word[2:5]  # characters from position 2 (included) to 5
(excluded)
'tho'
```

Slicing indexs tienen defaults, delante a 0 y detras a la ultima pos

```
>>> word[:2] + word[2:]
'Python'
```

Los strings son inmutables, no pueden cambiar, por tanto nose puede hacer esto:

```
>>> word[0] = 'J'
TypeError: 'str' object does not support item assignment
```

Tamaño del string:

```
len(word)
```

Conversion de tipos:

Cast to string: str(text)

Cast to int: `int(text)`

to json: `json.dumps([1, 'simple', 'list'])`

Condicionales

If elif else:

```
>>> x = int(input("Please enter an integer: "))
Please enter an integer: 42
>>> if x < 0:
...     x = 0
...     print('Negative changed to zero')
... elif x == 0:
...     print('Zero')
... elif x == 1:
...     print('Single')
... else:
...     print('More')
```

Comparadores

Se pueden encadenar:

```
a < b == c
```

and, or, >, <, ==, is, in, not

Iteradores:

Los iteradores sirven para usarlos en un for o creacion de listas.

Range function: range(start, end, step) / range(size)

Devuelve un iterable que se puede user con fors

```
range(0, 10, 3)
   0, 3, 6, 9
```

Crear lista de un range:

```
>>> list(range(4))
[0, 1, 2, 3]
```

Map

Para crear un iterador a partir de una lambda

```
list(map(lambda x: x**2, range(10)))
```

Lists:

Se puede incluir y borrar data by slicing

```
letters = ['a', 'b', 'c', 'd', 'e', 'f', 'g']
letters[2:5] = ['C', 'D', 'E'] # ['a', 'b', 'C', 'D', 'E',
'f', 'g']
letters[2:5] = [] # ['a', 'b', 'f', 'g']
```

Se pueden crear a partir de un iterador con list(iterator):

```
list(range(4))
```

Creacion a partir de list comprehension:

```
squares = [x**2 for x in range(10)]
```

Estructura list comprehension:

La primera parte es el valor de retorno, luego pueden venir multiples for y if, todo entre []

[(x, y) for x in [1,2,3] if x> 2 for y in [3,1,4] if x != y]

Nested list comprensions:

```
>>> matrix = [
...     [1, 2, 3, 4],
...     [5, 6, 7, 8],
...     [9, 10, 11, 12],
... ]
>>> [[row[i] for row in matrix] for i in range(4)]
[[1, 5, 9], [2, 6, 10], [3, 7, 11], [4, 8, 12]]
==
>>> transposed = []
>>> for i in range(4):
...     transposed.append([row[i] for row in matrix])
...
>>> transposed
[[1, 5, 9], [2, 6, 10], [3, 7, 11], [4, 8, 12]]
```

Metodos:

Añadir al final: `list.append(x)`

Añadir coleccion al final: `list.extend(iterable)`

Añadir item en i posicion: `list.insert(i, x)`

Eliminar item: del list[x]

Eliminar item con x valor: `list.remove(x)`

Elimina item de i posicion: `list.pop([i])`

Elimina todos los items: `list.clear()` = `del a[:]`.

Buscar el index de un item: `list.index(x[, start[, end]])` se puede marcar un start y end donde buscar para optimizar velocidad de busqueda

Numero de ocurrencias de x: `list.count(x)`

Cambiar el orden de la lista al reves: `list.reverse()`

Copiar una lista: `list.copy()` / `a[:]`

Otros tipos de listas:

Que

```
>>> from collections import deque
>>> queue = deque(["Eric", "John", "Michael"])
>>> queue.append("Terry")           # Terry arrives
>>> queue.append("Graham")          # Graham arrives
>>> queue.popleft()                 # The first to arrive now
leaves
'Eric'
>>> queue.popleft()             # The second to arrive now leaves
'John'
```

Tuplas

Son inmutables, se puede hacer destructuracion

```
x, y, z = tupla
```

Sets:

Los valores son unicos, no pueden repetirse. No se puede acceder a los elementos por index

```
b = set([1, 2, 3, 4, 4, 3, 2])
# {1, 2, 3, 4}
```

Set comprehension:

```
a = {x for x in 'abracadabra' if x not in 'abc'}
```

Dictionaries

```
tel = {'jack': 4098, 'sape': 4139}
dict([('sape', 4139), ('guido', 4127), ('jack', 4098)])
```

Dict comprehension:

```
{x: x**2 for x in (2, 4, 6)}
```

Iterando valores de arrays:

En diccionario

```
nights = {'gallahad': 'the pure', 'robin': 'the brave'}
>>> for k, v in knights.items():
...     print(k, v)
```

Para obtener valor y index en los otros tipos de listas:

```
>>> for i, v in enumerate(['tic', 'tac', 'toe']):
...     print(i, v)
```

Para juntar varias listas en un for se usa zip:

```
>>> questions = ['name', 'quest', 'favorite color']
>>> answers = ['lancelot', 'the holy grail', 'blue']
>>> for q, a in zip(questions, answers):
...     print('What is your {0}?  It is {1}.'.format(q, a))
```

Iterar con reversed() y sorted():

```
for i in reversed(range(1, 10, 2)):
for f in sorted(set(basket)):
```

Modules: “ficheros .py”

__name__: toma el valor del nombre del fichero, o “__main__” si es ejecutado desde el terminal.

Imports:

```
from fibo import fib as fibonacci
```

Compiled files

Python genera ficheros de cache, para aumentar la velocidad de ejecucion, no cachea el fichero inicial de ejecucion.

```
__pycache__
```

Functions:

Parametros:

Con numero de argumentos variable: definiendo valores por defecto

```
def ask_ok(prompt, retries=4, reminder='Please try again!'):
```

Se puede llarmar a la funcion, por keyword argumentos:

```
ask_ok(prompt=1000000, retries=10)
```

Parametros de tipo *args y **keys :

*args = numero variable de parametros al cual se puede acceder como lista

**keys = numero variable de params en forma de dic al cual se accede como lista

```
def cheeseshop(kind, *arguments, **keywords):
    print("-- Do you have any", kind, "?")
    print("-- I'm sorry, we're all out of", kind)
    for arg in arguments:
        print(arg)
    print("-" * 40)
    for kw in keywords:
        print(kw, ":", keywords[kw])
```

Llamada:

```
cheeseshop("Limburger", "It's very runny, sir.",
           "It's really very, VERY runny, sir.",
```

```
            shopkeeper="Michael Palin",
            client="John Cleese",
            sketch="Cheese Shop Sketch")
```

Resultado:

```
-- Do you have any Limburger ?
-- I'm sorry, we're all out of Limburger
It's very runny, sir.
It's really very, VERY runny, sir.
----------------------------------------
shopkeeper : Michael Palin
client : John Cleese
sketch : Cheese Shop Sketch
```

Lambda expressions: como las arrows functions

```
pairs.sort(key= lambda pair: pair[1] )
```

pass: Para cuando una parte de codigo no haga nada sin dar error

```
>>> def initlog(*args):
...     pass   # Remember to implement this!
```

Leer fichero

```
open(filename, mode) # mode optional, default 'r'
>>> f = open('workfile', 'w')
```

with

Abrir con with para cerrar el fichero correctamente despues de su uso, sustituye al try catch

```
>>> with open('workfile') as f:
...     read_data = f.read()
```

Funciones del fichero

```
f.read(size) #size of bytes optional, default all file
f.readline()
```

Convertir fichero a array de strings:

`list(f)` or `f.readlines()`.

Escribir en fichero:

```
f.write('This is a test\n')
```

Loop:

```
>>> for line in f:
...     print(line, end='')
```

Errors and exceptions:

try except

```
try:
...     this_fails()
... except ZeroDivisionError as err:
```

```
...     print('Handling run-time error:', err)
```

Generate exception: raise exception

```
raise NameError('HiThere')
```

Obtener el error (traceback)

```
formatted_lines = traceback.format_exc().splitlines()
```

Clases:

Constructor (__init__)

Los parametros de init, son los que se usan para instanciar la clase

```
def __init__(self, param1, param2):
    self.data = []
```

Variables de clase

No definirlas en scope global de la clase o seran compartidas con otras instancias de la clase, definirlas dentro del constructor.

Variables privadas

Iniciar el nombre de variable con __

```
def __update(self, iterable):
```

Herencia

```
class DerivedClassName(Base1, Base2, Base3):
```

Referenciar a funciones de la clase padre

Con super_(), es necessario en el constructor.

```
super().__init__()
```

Standard Library:

Documentation, help / dir

```
>>> import os
>>> dir(os)
<returns a list of all module functions>
>>> help(os)
<returns an extensive manual page created from the module's
docstrings>
```

os (operative system)

Per a cridar comandes de terminal via python, com mkdir, chdir, pwd, etc..

```
>>> import os
>>> os.getcwd()      # Return the current working directory
'C:\\Python38'
```

```
>>> os.chdir('/server/accesslogs')   # Change current working
directory
>>> os.system('mkdir today')   # Run the command mkdir in the
system shell
```

File directory management

```
>>> import shutil
>>> shutil.copyfile('data.db', 'archive.db')
'archive.db'
>>> shutil.move('/build/executables', 'installdir')
'installdir'
```

Command line arguments

command: `python demo.py one two three`

```
import sys
>>> print(sys.argv)
['demo.py', 'one', 'two', 'three']
```

Regular expressions (re)

```
>>> import re
>>> re.findall(r'\bf[a-z]*', 'which foot or hand fell fastest')
['foot', 'fell', 'fastest']
>>> re.sub(r'(\b[a-z]+) \1', r'\1', 'cat in the the hat')
'cat in the hat'
```

Dates and times:

```
>>> # dates are easily constructed and formatted
>>> from datetime import date
>>> now = date.today()
>>> now
datetime.date(2003, 12, 2)
>>> now.strftime("%m-%d-%y. %d %b %Y is a %A on the %d day of
%B.")
'12-02-03. 02 Dec 2003 is a Tuesday on the 02 day of December.'

>>> # dates support calendar arithmetic
>>> birthday = date(1964, 7, 31)
>>> age = now - birthday
>>> age.days
14368
```

Unit testing (unittest)

```
import unittest
```

```
class TestStatisticalFunctions(unittest.TestCase):

    def test_average(self):
        self.assertEqual(average([20, 30, 70]), 40.0)
        self.assertEqual(round(average([1, 5, 7]), 1), 4.3)
        with self.assertRaises(ZeroDivisionError):
            average([])
        with self.assertRaises(TypeError):
            average(20, 30, 70)

unittest.main()  # Calling from the command line invokes all
tests
```

Virtual Env:

Para poder tener una version de librerias distinta por proyecto

virtualEnv nombreproyecto
cd nombreproyecto/Scripts && activate

Instalar virtual env con distinta version de python

virtualEnv nombreProyecto -p path/python.exe

Instalar librerias marcadas en un fichero de requerimientos

pip install -r requirements.txt