Python

Les operateurs en Python sont: `+ - * / // % == != < <= > >= ^ **`

Les operateurs logiques: `and or not`

```
x = int(input("give me the value of x: "))
print("the value of x is:", x, end="\n")
```

```
for x in range(start, end, step):
    pass
else:
    pass
```

```
while True:
    pass
else:
    pass
```

```
def add(a, b = 2):
    return a + b
```

```
add = lambda a, b = 2: a + b
```

```
global_variable = 5
def function1():
    local_variable = 10
    print(local_variable) # works
    print(global_variable) # works
    global_variable = 20 # doesn't work

def function2():
    global global_variable
    local_variable = 10
    print(local_variable) # works
    print(global_variable) # works
    global_variable = 20 # works
```

Les tableaux

```
from array import array
A = array("i", [1, 2]) # array of ints
B = array("f", [0.5, 0.7]) # array of floats
A[index] = value
A.append(value)
len(A)
del(A[0])
A.insert(0, 5)
A.reverse()
sorted(A)
```

```
array.append(value)
len(array)
del(array[index])
array.reverse()
array.insert(index, value)
sorted(array)
```

Les chaines de caracteres

```
string = "hello"
string[0] = "H" # doesn't work because strings are immutable
print(string + " world")
len(string)
string[1:len(string)-1] #=> "ell"
"he" in string
```

```
str(8)
int("24")
float("19.5")
bool("True")
ord("A")
chr(65)
```

```
"ko" * 2
list(string)
string.split(word)
string.count(word)
string.endswith(word)
string.startswith(word)
string.lower()
string.upper()
string.find(word) # if not found returns - 1
string.index(word) # if not found throws ValueError exception
string.islower()
string.isupper()
string.isdegit()
string.isalpha()
string.isalnum()
string.replace(word, new_word) # replaces all occurrences
string.isnumeric()
```

```
string.isspace()
string.istitle()
string.join(list_of_strings)
string.splitlines()
string.capitalize()
string.isprintable()
string.isidentifier()
string.swapcase()
```

Les listes

```
L = []
L = list(range(start[, end, step]))
L = [x ** 2 for x in range(start[, end, step]) if x % 3 == 0]
L[index] = value
L[start_index:end_index]
del(L[index])
del(L[start_index:end_index])
del(L[start_index:])
del(L[:end_index])
L[index1:index2] = [value*]
L.sort()
L.append(value)
L.pop(index)
L.reverse()
L.index(value, start_index, end_index)
L.remove(value)
L.extend(other_list)
L.insert(index, value)
```

```
L1 + L2
L * n
T = [0] * 3 #=> [0, 0, 0]
x in L
len(L)
min(L)
max(L)
```

Les dictionnaires

```
d = {}
d = dict()
d = dict(sequence)
# dict([[1, 2], [3, 4]]) => {1: 2, 3: 4}
d = {"key": "value"}
d[key] = value # key values are immutable
```

```
del(d[key])
del(d)
len(d)
d.pop(key)
d.clear()
d.keys()
d.values()
d.items()
d.has_key()
d.copy()
```

```
for key in d:
    pass
```

Les tuples

```
t = tuple()
t = ()
t = tuple(seq) # tuples are immutable
t = (value,)
t[start_index, end_index]
t[index] = value
t + (3,4)
t * 2
a, b = 1, 2
a, b = b, a
c = a, b
```

Les ensembles

```
s = set(seq)
f = frozenset(seq) # an immutable type that can be used as a dictionary key
in
not in
== != # check elements appearances in each set
A < B # sous-ensemble strict A ⊂ B
A <= B # sous-ensemble A ⊆ B
A > B # sur-ensemble strict A ⊃ B
A >= B # sur-ensemble A ⊇ B
& # intersection ∩
| # union ∪
- # difference -
^ # difference symetrique Δ
del(s)
```

```
set(d) # set of keys or any immutable sequence
set(d.keys())
set(d.values())
set(d.items())
```

```
s.add(value)
s.update(sequence)
s.remove(value)
s.pop(value)
s.copy()
```

Fichiers

```
import os
cwd = os.getcwd()
os.chdir(path)
```

```
file = open(path, mode = "r|w|a")
file.read()
file.read(number_of_chars_to_read)
file.readline()
file.readlines()
file.write(string)
file.tell() # return the current cursor position
file.seek(index) # change the position of the cursor
file.close()
```

Les exceptions

```
try:
    pass
except ValueError as e:
    pass
else:
    pass
finally:
    pass
```

```
IOError
ZeroDivisionError
IndexError
ValueError
```

Les bibliotheques mathematiques

cmath

```
import cmath
num = complex(num1, num2)
cmath.polar(num)
abs(num)
num.conjugate()
num.real
num.imag
```

numpy

```
import numpy as np

T = np.array([1, 2]) # array([1, 2])
T = np.array([[1, 2], [3, 4]]) # array([[1, 2], [3, 4]])
T.trace() # 1 + 4 = 5
T.transpose() # array([[1, 3], [2, 4]])
T.shape # (2, 2)
T.ndim # 2
T.dtype # dtype('float64')
T[i_index][j_index]
T[i_index, j_index]
T[:, j_index]
T[i_index, :]
T * 2
T / 2
T + K
```

```
np.arange(start, end, float_step)
np.arrange(1, 3, 0.5) # array([1, 1.5, 2, 2.5])

np.linspace(start, end_inclusive, length)
np.linspace(1, 3, 4) # array([1, 1.66, 2.33, 3])

np.zeros(length)
np.ones((w,h))
np.zeros_like(T) # np.zeros(T.shape)
np.ones(3) # array([1., 1., 1.])
np.zeros((2,2)) # array([[0., 0.], [0., 0.]])

np.random.rand(2) # array([0.43407251, 0.81413197])
np.random.randn(2) # array([0.43407251, -0.81413197])
np.random.rand(2, 2) # array([[0.55062666, 0.65620799], [0.24119092,
0.2263176]])
np.random.randn(2, 2) # array([[-0.55062666, 0.65620799], [0.24119092,
-0.2263176]])

np.random.randint(start, end, length)
np.random.randint(start, end, (w,h))
np.random.randint(2, 10, 3) # array([4, 2, 7])
np.random.randint(2, 6, (2,2)) # array([[4, 3], [5, 2]])

np.diag(X, k)
np.diag(T, k=0) # array([1, 4])
np.diag(T, k=1) # array([2])
np.diag(T, k=-1) # array([3])

np.eye(n)
np.eye(3) # array([[1., 0., 0.],
          #        [0., 1., 0.],
          #        [0., 0., 1.]])

np.reshape(T, new_shape)
np.reshape(T, (1, 4))
```

```
np.dot(A, B)
np.outer(A, B) # 2 dimension array of A x B
np.concatenate((A, B), axis)
```

linalg

```
from numpy import linalg

linalg.det(T)
linalg.norm(T)
linalg.inv(T)

# 4x + 2y = 0
# x - y = 5
A = np.array([[4, 2], [1, -1]])
B = np.array([[0], [5]])
x = linalg.solve(A, B)
```

matplotlib.pyplot

```
import matplotlib.pyplot as plt
import numpy as np

g = lambda x: sin(x) * x ** 2
f = lambda x: x ** 2
t = np.arange(0, 10, 0.1)
Y1 = f(t)
Y2 = g(t)

plt.plot(t, Y1, 'r')
plt.plot(t, Y2, 'b')

plt.xlabel("temps(s)")
plt.ylabel("y(temps)")
plt.title("graph title")
plt.legend(('x->x**2','x->sin(x)*x**2'))
show()
```

copy

```
import copy

# shallow copy
L = [1, 2, 3]
K = list(L)
K = L[:]
K = copy.copy(L)
# deep copy
K = copy.deepcopy(L) # works with other objects too
```

Connexion a une base de donne

```
import sqlite3
```

```
db="path/db.sqlite"
connection = None
cursor = None
try:
    connection = sqlite3.connect(db)
    cursor = connection.cursor()
    cursor.execute(query)
    cursor.executemany(query)
    rows = cursor.fetchall()
    row = cursor.fetchone()
    rowcount = cursor.rowcount
    connection.commit()
except sqlite3.Error as e:
    print(e)
finally:
    if cursor != None:
        cursor.close()
    if connection != None:
        connection.close()
```

OOP

```
class ClassName(BaseName):
    counter = 0 # counter is a static attribute
    """Class description"""
    def __init__(self, x, y):
        """Method description"""
        super().__init__(x, y)
        self.__name = "salah" # __name is a private attribute
        ClassName.counter += 1
        self.__class__.counter += 1
        pass

    def getName(self):
        return self.__name

    def setName(self, name):
        self.__name = name

    def __[add|sub|mult|div]__(self, obj):
        return ClassName(self.x + obj.x, self.y + obj.y)

    def __del__(self):
        self.__class__.counter -= 1
```

tkinter

```
import tkinter as tk

window = tk.Tk()
window["bg"] = "white"
window.title("title")
window.geometry("300x200")
```

```
tk.Label(window, text="text").pack()

tk.Button(window, text="button", command=window.quit).pack()

checkvar = tk.StringVar() # checkvar.get()
tk.Checkbutton(window, text="checkbutton", variable=checkvar, onvalue="yes",
offvalue="no").pack()

entryvar = tk.StringVar()
Entry(cadre, textvariable=entryvar).pack()

radiovar = tk.IntVar()
tk.Radiobutton(window, text="true", variable=radiovar, value=1).pack()
tk.Radiobutton(window, text="false", variable=radiovar, value=2).pack()

listbox = tk.Listbox(window)
listbox.insert(1, "A")
listbox.insert(2, "B")
listbox.pack() # listbox.get(listbox.curselection())

spinbox = Spinbox(window, from_=0, to=10) # spinbox.get()
spinbox.pack()

scalevar = tk.DoubleVar()
tk.Scale(window, variable=scalevar).pack()

frame = tk.Frame(window, borderwidth=2,
                 relief=[tk.RAISED, tk.SUNKEN, tk.FLAT, tk.RIDGE, tk.GROOVE,
tk.SOLID])
frame.pack(side=tk.LEFT, padx=30, pady=30)

labelFrame = LabelFrame(window, text="text", padx=20, pady=20,  relief?)
labelFrame.pack(fill="both", expand="yes")

menubar = tk.Menu(window)
menu = tk.Menu(menubar)
menu.add_command(label="label", command=alert)
menu.add_separator()
menu.add_command(label="quit", command=window.quit)
menubar.add_cascade(label="label", menu=menu)
window.config(menu=menubar)

def keyboard(event):
    key = event.keysym
    print(key)
canvas = tk.Canvas(window, width=150, height=120, background="yellow")
canvas.focus_set()
canvas.bind("<[Key|KeyPress|Button-1..3|Double-Button-
1..2|Return|Escape|KeyPress-
a..Z|Down|Up|Right|Left|ButtonRelease|Leave|Enter|Motion|B1..2-
Motion|Map|Unmap|MouseWheel]>", keyboard)
canvas.create_line(x1, y1, x2, y2)
canvas.create_text(x, y, text="text", font="Arial 16 italic", fill="blue")
canvas.pack()

import tkinter.messagebox as mb
mb.askyesno("title", "message")
mb.show[warning|info|error]("title", "message")
```

```
window.mainloop()
```

```
menubar = tk.Menu(window)
menu = tk.Menu(menubar)
menu.add_command(label="label", command=alert)
menu.add_separator()
menu.add_command(label="quit", command=window.quit)
menubar.add_cascade(label="label", menu=menu)
window.config(menu=menubar)

def keyboard(event):
    key = event.keysym
    print(key)
canvas = tk.Canvas(window, width=150, height=120, background="yellow")
canvas.focus_set()
canvas.bind("<[Key|KeyPress|Button-1..3|Double-Button-
1..2|Return|Escape|KeyPress-
a..Z|Down|Up|Right|Left|ButtonRelease|Leave|Enter|Motion|B1..2-
Motion|Map|Unmap|MouseWheel]>", keyboard)
canvas.create_line(x1, y1, x2, y2)
canvas.create_text(x, y, text="text", font="Arial 16 italic", fill="blue")
canvas.pack()

import tkinter.messagebox as mb
mb.askyesno("title", "message")
mb.show[warning|info|error]("title", "message")

window.mainloop()
```

Classification supervisee et non supervisee

KNN(k-nearest-neighbors)

- set `k` the number of nearest neighbors
- Calculate the distance between the **query-instance** and all the **training samples**

Euclidean $\sqrt{\sum_{i=1}^{k}(x_i - y_i)^2}$

- Select nearest neighbors based on the `k-th` minimum distance
- Select and choose the major class as the prediction value

```python
from sklearn.neighbors import KNeighborsClassifier

positions = [[0, 0], [1, 1]] # [x, y]
classes = ["A", "B"]
classifier = KNeighborsClassifier(n_neighbors=1) # pour k > 1 si le nombre de classe proche sont egaux en prendre la premier classe
classifier.fit(positions, classes)
print(classifier.predict([[0, 0.5]])) # ['A']
print(classifier.predict([[1, 0.5]])) # ['B']
print(classifier.predict([[0, 0.5], [1, 0.5]])) # ['A', 'B']
```

```python
from sklearn.neighbors import KNeighborsClassifier

positions = [[0], [1], [2]] # [x, y]
classes = ["A", "B", "B"]
classifier = KNeighborsClassifier(n_neighbors=3)
classifier.fit(positions, classes)
print(classifier.predict([[0.4]])) # ['B']
# pour k=1 ou k=2 le resultat est ['A']
```

Decision Tree

```python
from sklearn.tree import DecisionTreeClassifier

samples = [[0], [1]]
labels = ["A", "B"]
classifier = DecisionTreeClassifier()
classifier.fit(samples, labels)
print(classifier.predict([[0.4]])) # ['A']
```

---

k-means

L'algorithme **k-means** est en 4 étapes :

1. Choisir `k` objets formant ainsi `k` clusters
2. (Ré)affecter chaque objet `O` au cluster `Ci` de centre `Mi` tel que `d(O,Mi)` est minimal

Manhattan $\sum_{i=1}^{k} |x_i - y_i|$

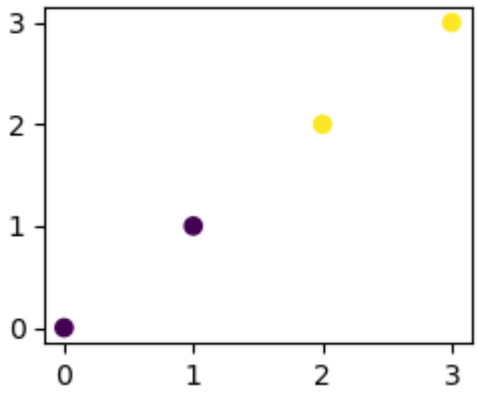
3. Recalculer `Mi` de chaque cluster *(le barycentre)*
4. Aller à l'étape 2 si on vient de faire une affectation

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans

samples = np.array([[0, 0], [1, 1], [2, 2], [3, 3]]) # [x, y]
km = KMeans(n_clusters=2)
km.fit(samples)

clusters = km.predict(samples)
print(clusters) # [0 0 1 1]

plt.scatter(samples[:, 0], samples[:, 1], c=clusters) # plt.scatter(x, y,
c=colors)
plt.show()
```




```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans

samples = np.random.rand(10000, 2)
km = KMeans(n_clusters=30)
km.fit(samples)

clusters = km.predict(samples)

plt.scatter(samples[:, 0], samples[:, 1], c=clusters, s=2)
plt.show()
```