

# PYTHON FROM BASICS

---

Alexandre Terrien, Martin Pozuelo

Mardi 11 octobre

ATILLA

- Algorithmes
- Fonctions
- Structures
- I/O

# LISTES EN COMPRÉHENSION

```
l = [i ** 2 for i in range(n)]
```

```
l = [i + 10 for i in l]
```

```
l = [i for i in l if i % 2]
```

Générer tous les triplets pythagoriciens.

$$0 < a \leq b \leq c < 50$$

$$(a, b, c) \in \mathbb{N}^3 \text{ tq } a^2 + b^2 = c^2$$

```
def triplets_pythagoriciens(n):  
    return [(a, b, c)  
            for a in range(1, n)  
            for b in range(a, n)  
            for c in range(b, n)  
            if a ** 2 + b ** 2 == c ** 2  
            ]
```

En python, tout est objet

- Attributs (variables de l'objet)
- Méthodes (fonctions sur l'objet)

```
>>> help(str)
>>> s = "Ceci est une phrase."
>>> s.split()
['Ceci', 'est', 'une', 'phrase.']
>>> s.split('e')
['C', 'ci ', 'st un', ' phras', '.']
```

## EXEMPLE : STR(2)

```
>>> l = s.split('e')
>>> "E".join(l)
'CEci Est unE phrasE.'
>>> '-'.join(l)
'C-ci -st un- phras-.'
>>> s.replace("e", "p")
'Cpci pst unp phrasp.'
```

## EN COMBINANT TOUT!

```
>>> [int(i) for i in input().split()]
845 785 45 632 1 5
[845, 785, 45, 632, 1, 5]
>>> max([int(i) for i in input().split()])
4558 458 63 215 7788 458 2
7788
```

```
>>> help(list)
>>> l = [845 785 45 632 1 5]
>>> l.append(78)
>>> l
[845, 785, 45, 632, 1, 5, 78]
>>> e = l.pop()
78
>>> l
[845 785 45 632 1 5]
```

```
>>> values = [7,45,8,658,7,25,45,36]
>>> while values:
...     e = values.pop()
...     doSomething(e)
```

```
>>> values=[7,45,8,658,7,25,45,36]
>>> sorted(values)
[7, 7, 8, 25, 36, 45, 45, 658]
>>> values
[7, 45, 8, 658, 7, 25, 45, 36]
>>> values.sort()
>>> values
[7, 7, 8, 25, 36, 45, 45, 658]
```

## DES FONCTIONS EN PARAMÈTRES(1)

```
>>> values=[7,1,6,9,8,45,21,56,-15,-4,-8,-62,-7]
>>> min(values)
-62
>>> min(values, key=abs)
1
```

## DES FONCTIONS EN PARAMÈTRES(2)

```
>>> def myFun(x):  
...     return(x**2 - 10*x + 5)  
>>> min(values, key=myFun)  
6
```

Courte définition d'une fonction

```
>>> def myFun(x):  
...     return(x ** 2 - 7 * x + 5)  
>>> myFun= lambda x: x ** 2 - 7 * x + 5  
>>> min(values, key= lambda x: x ** 2 - 10 * x + 5)  
6
```

On peut faire la même chose comme ça :

```
>>> keys=[myFun(x) for x in values]
>>> values[keys.index(min(keys))]
6
```

```
>>> sorted(values,key=lambda x: (abs(x), x))  
[1, -4, 6, 7, -7, 8, -8, 9, -15, 21, 45, 56, -62]  
>>> sorted(values,key=lambda x:x**2 - 10*x + 5)  
[6, 7, 8, 1, 9, -4, -7, -8, 21, -15, 45, 56, -62]
```

Tri de tuples de dates

```
>>> values = [  
    (22,11,1995),(23,10,1995),(2,7,1995),  
    (7,4,1994),(26,4,1996),(16,7,1994),(12,8,1996),  
    (10,2,1996),(14,12,1996)  
]
```

```
>>> sorted(values, key=lambda x: x[::-1])
```

```
# Avec liste
```

```
>>> max([int(i) for i in input().split()])
```

```
4558 458 63 215 7788 458 2
```

```
7788
```

```
# Avec générateur
```

```
>>> max(int(i) for i in input().split())
```

```
4558 458 63 215 7788 458 2
```

```
7788
```

# MODULES(1)

- Stocker des fonctions, classes
- Rassembler des éléments similaires

Ex : dans le fichier monModule.py

```
def fibo(n):  
# ...
```

```
def fact(n):  
# ...
```

Dans l'interpréteur (ou un autre fichier)

```
>>> import monModule
```

```
>>> monModule.fibo(5)
```

```
5
```

```
>>> from monModule import fact
```

```
>>> fact(3)
```

```
6
```

```
class Complex :  
    def __init__(self, reel, imag):  
        self.r = reel  
        self.i = imag  
    def add(self,c) :  
        return(Complex(self.r + c.r, self.i + c.i))
```

```
>>> x = Complex(7.5, -9.8)
>>> x.r
7.5
>>> x.i
-9.8
>>> y = Complex(1.0, 1.0)
>>> z = x.add(y)
>>> z.r
8.5
```

Méthodes pour les builtins python

```
class Complex:
    # ...
    def __repr__(self):
        return(str(self.r)+" "+str(self.i)+"i")

    def __str__(self):
        return(str(self.r)+" "+str(self.i)+"i")
```

```
class Complex:
    # ...
    def __abs__(self):
        return(self.r**2+self.i**2)

    def __add__(self,c) :
        return(Complex(self.r+c.r,self.i+c.i))
```

Fonction qui prend en paramètre une fonction et qui renvoie la même fonction modifiée de la manière suivante :

- Si la fonction n'a jamais été lancée avec ces paramètres, l'exécute, enregistre le résultat dans un dictionnaire et renvoie le résultat
- Si la fonction a déjà été lancée avec les mêmes paramètres, renvoie directement le résultat

- **class**
- `__init__()`
- `__call__()`
- Doc python

```
class Memoisation:
    def __init__(self, f):
        self.f = f
        self.memo = {}
    def __call__(self, *args):
        if not args in self.memo:
            self.memo[args] = self.f(*args)
        return self.memo[args]
```

- Codingame
- HackerRank
- Prologin (15 octobre)
- Google code jam