Premières fonctions

La fonction main:

La fonction main est le point d'entrée du programme. C'est elle qui est appelée lorsqu'on lance l'exécution du code.
Copier coller la fonction ci-dessous :

```
if __name__ == '__main__':
    print('Hello World')
```

On note que l'indentation fait partie intégrante du langage Python
C'est toujours la dernière méthode du fichier.

Fonction :

```
def test(n):
        return "test" + n

def doNothing():
        return
```

La fonction doNothing retourne None. Si une méthode ne retourne rien, même si aucun return n'est présent, elle retournera None. Ainsi ces trois méthodes qui suit sont équivalente :

```
def foo(n):
        print(n)

def foo(n):
        print(n)
        return

def foo(n):
        print(n)
        return None
```

Maths :

```
2 + 2
4
>>> 50 - 5*6
20
```

```
>>> (50 - 5*6) / 4
5.0
>>> 8 / 5  # division always returns a floating point number
1.6
17 / 3  # classic division returns a float
5.666666666666667
>>>
>>> 17 // 3  # floor division discards the fractional part
5
>>> 17 % 3  # the % operator returns the remainder of the
division
2
>>> 5 * 3 + 2  # floored quotient * divisor + remainder
17
>>> 5 ** 3 # 5 power 3
```

Condition :

if :

```
def foo(x):
      if x < 0:
            print('Negative')
      elif x == 0:
            print('Zero')
      else:
            print('Positive')
```

for :

```
def foo():
      words = ['cat', 'window', 'defenestrate']
      for w in words:
            print(w, len(w))
```

```
Le résultat de l'appel à cette méthode est :
cat 3
window 6
defenestrate 12
```

```
words = ['cat', 'window', 'defenestrate'] permet de définir la
liste words
```

```
words.append(25) ajoute 25 à la fin de la liste
```

range() :

```
for i in range(10):
            print(i, end="")
```

Affiche 0123456789 -> 10 est exclu

```
for i in range(1,10):
            print(i, end="")
Affiche 123456789 -> 1 est inclu
for i in range(1,10,2):
            print(i, end="")
Affiche 13579 -> 2 est la step, on monte de 2 en 2
Et ça marche aussi à l'envers :
for i in range(9,-1,-1):
            print(i, end="")
Affiche 9876543210
```

match :

```
def http_error(status):
    match status:
        case 400:
            return "Bad request"
        case 404:
            return "Not found"
        case 418:
            return "I'm a teapot"
        case _:
            return "Something's wrong with the internet"
```

comme un if elif elif else mais plus lisible
Ici ce match serait strictement équivalent à :

```
def http_error(status):
    if status == 400:
            return "Bad request"
    elif status == 404:
            return "Not found"
    elif status == 418:
            return "I'm a teapot"
    else :
            return "Something's wrong with the internet"
```

While :

```
while i < 6:
      print(i)
      i += 1
```

tant que i < 6 on continue la boucle. Comme à la fin de la boucle on incrémente i de 1 on finira par avoir i = 6 après avoir print 1 2 3 4 et 5, et on sortira de la boucle sans écrire 6.

Capture d'entrée utilisateur :

```
username = input("Enter username:")
print("Username is: " + username)
```

On peut aussi capturé à l'entrée du programme :
d'abord il faut ajouter en haut du code :

```
import sys
```

Puis :

```
if __name__ == '__main__':
   var = sys.argv[1]
```

Attention : la liste sys.argv commence à 0 mais l'élément 0 correspond à l'emplacement du fichier python en train d'être exécuté.
Attention aussi : si l'argument numéro x de sys.argv[x] n'existe pas une exception sera levée. On parlera plus en détail des exceptions dans le deuxième cours.

Lecture et écriture de fichier :

`filin = open("myFile.txt", "r")`
Si le ficher myFile.txt est dans le même dossier que le code exécuté
Sinon /home/path/myFile.txt (Linux/Mac OS) ou C:\path\myFile.txt
Ensuite :

```
lignes = filin.readlines()
for ligne in lignes:
      print(ligne)
```

Permet de naviguer ligne par ligne, on peut aussi itérer sur ligne si besoin.
Et enfin :

`filin.close()`

Il faut toujours finir par fermer le fichier, sinon c'est ce qu'on appelle une fuite mémoire, on accumule les fichiers ouverts dans la mémoire et on sature l'ordinateur.

Pour simplifier et assurer le .close() il existe le with :

```
with open("myFile.txt", "r") as filin:
    lignes = filin.readlines()
     for ligne in lignes:
          print(ligne)
```

Même si des erreurs arrivent, le with appellera .close() de l'objet (marche aussi avec des types autres que les fichiers)

Ecriture dans un fichier :

Lors de l'appel open, le "r" signifie read, si on veut écrire on met "w" (si on veut faire les deux "r+")

```
with open("myFile.txt", "w") as filout:
     filout.write("test")
```

Commentaire et documentation Python :

"""Ceci est une documentation"""

#Ceci est un commentaire

Maintenant on évite les commentaires, c'est un aveux de faiblesse, on verra des moyens de faire un code propre sans commentaires.

La documentation n'est pas un commentaire, elle est utilisée par les IDE, pour générer des sites de documentation à l'usage de développeur, …