On rappelle quelques commandes usuelles sous Python.
Vous pourrez trouver tous les détails à l'adresse [https://docs.python.org/3.3/library/index.html] ou [http://www.python-course.eu/python3_course.php].

Types de variables

Python n'est pas un langage typé au sens où le type des variables n'est pas défini avant l'initialisation de celles-ci.

Lorsque cela fait sens, une variable peut être affichée par la fonction `print` et convertie en chaîne de caractères par la fonction `str`. Inversement une chaîne de caractères peut être convertie en nombre par la fonction `int`.

```
nb = 0                      # 'nb' est ici un nombre
b = 0==1                    # 'b' est ici un boolen
mot = "un langage"          # 'mot' est ici une chaine de caracteres
l = [3,4,"dix",12]          # 'l' est ici une liste
```

nombre

Il existe trois grandes familles de nombres sous Python : les entiers, les réels (float) et les complexes (la partie imaginaire s'introduit alors par `j`). Par défaut sur Python, un réel est codé sur 64 bits avec une mantisse codée sur 52. On ne peut donc coder de manière exacte qu'un sous-ensemble fini de nombres ; 0,1 ne vaut pas exactement 0,1 et par défaut sous Python 0.1+0.1+0.1==0.3 renvoie False (!). Heureusement certains modules (ou librairies) permet de palier en grande partie ses limitations (*e.g.* decimal ou fractions).

On peut appliquer les opérateurs suivants aux nombres : `+`, `-`, `*`, `/`, `//`, `%` et `**` (//, % et ** correspondent respectivement au quotient de la divison euclidienne, au reste de la division euclidienne et à la puissance).

```
nb = 0                      # 'nb' est ici un nombre
nb2=1+2j                    # nb2 est un nombre imaginaire
nb+= 3
print("Le nombre vaut {}".format(nb), end='')   # pas de saut de ligne
nb=nb**2
print("et son carre modulo 5 vaut {}".format(nb%5))
```

exo : écrire la division euclidienne d'un nombre n par m (*e.g.* pour (n ;m)=(10 ;3), $10 = 3 \times 3 + 1$)

rq : plusieurs moyens existent pour afficher des nombres. Comparer ses formats et résultats.

```
n = 12.345
m = 0.067
print(n, m, sep=" et ")
print("%2d et %0.2f" %(n,m))
print("{0:2d} et {1:0.2f}".format(int(n),m)) # 0: ou 1: refere aux
    arguments de format()
```

booléen

On peut créer un booléen à l'aide d'une comparaison : `==`, `!=`, `<`, `>`, `<=`, `>=`, `is` et `in`.
On peut alors appliquer les opérateurs suivants aux booléens : `and`, `or` et `not`.

```
bf = 0==1                   # 'bf' est un boolen de valeur faux
bv = 0==0                   # 'bv' a pour valeur vraie
bnew = 0<3 and 0 in [0,1,2]
print("la valeur de bnew est {}".format(bnew))
```

séquences

Une séquence est un ensemble d'éléments statique ou dynamique. Différentes types de séquences existent sous Python mais on peut toutes leur appliquer les opérateurs suivants : `len`, `*` et `[]`.

On rappelle que les éléments sont rangés de `0` à `len()-1`.

tableau et chaîne de caractères (string)

Un tableau est une suite statique d'éléments indexées. De même les mots et chaînes de caractères sont considérés par Python comme des tableaux et on peut leur appliquer les fonctions usuelles des tableaux.

Une chaîne peut être définie entre guillemets simples ou doubles. On peut alors définir un mot composé de guillemets en utilisant les autres guillemets pour le définir (*e.g.* `"Elle l'appela"` ou `'Il dit : "coucou"'`) ou bien en utilisant un charactère d'échappement (*e.g.* `'Elle l\'appela'`).

```
mot = "un langage"          # 'mot' est ici une chaine de caracteres
mott= 'un langage'
tab = 10*[None]             # 'tab' est tableau vide
print(mot+' est-il egal a un autre ?', mot is mott)
print(mot[3],mot(-3))
print(tab[2] is None)
print(mot*3)
"une phrase avec des mots".split()      # renvoie la liste des mots
```

La commande `split('-')` va séparer la chaîne en liste composée des parties séparaarées par le séparateur '-' (par défaut le séparateur est l'espace). Ainsi `"un-deux-trois-quatre cinq".split('-')` renvoie la liste `['un', 'deux','trois','quatre cinq']`.

liste

Une liste est une séquence dynamique, elle peut donc changer de dimension.
Outre les opérateurs des séquences, on peut appliquer aux listes les opérateurs suivants :

- ◇ `liste[n:m]` : renvoie les éléments de la liste entre n et m-1
- ◇ `del liste[n:m]` : efface les éléments de la liste entre n et m-1
- ◇ `clear` : efface toute la liste (⟺ `del liste[:]`)
- ◇ `copy` : copie les valeurs d'une liste (⟺ `list(liste)`)
- ◇ `insert(i,V)` : insert V en position i à la liste
- ◇ `append(V)` : insert une valeur en fin de liste (⟺ `insert(len(liste),V)`)
- ◇ `extend(liste2) : concatène les deux listes (⟺ liste1+=liste2)`
- ◇ `pop(i)` : renvoie et supprime l'élément en position i
- ◇ `index(V)` : renvoie l'indice de la première occurence de V
- ◇ `remove(V)` : retire la première occurence de V (⟺ `pop(index(V))`)
- ◇ `count(V)` : compte le nombre d'occurence de l'élément V
- ◇ `reverse` : inverse l'ordre des éléments
- ◇ `sort` : lorsque cela est possible trie la liste
- ◇ `".join(liste)` : renvoie la liste en tant que string (" sont 2 guillemets simples)

```
tab=[None]          # 'tab' est un tableau vide
l = []              # 'l' est ici une liste
l.extend('32')
print(l)            # renvoie ['3','2']
l.append(32)        # remarquerez que extend() est different de append()
print(l)            # renvoie ['3','2',32]
del l[0:2]          # equivaut a pop(0) et pop(1)
l.append('dix')
l.pop(0)
print(l)
m=l
n=l.copy()
m.append(2)         # l est egalement affecte mais pas n !
print(l,m,n)
```

La commande `join()` est la réciproque de la commande `split()`.
Ainsi la commande `'-'.['un','deux','trois','quatre cinq']` renvoie `"un-deux-trois-quatre cinq"` (chaîne qui est la concaténation de chaque élément de la liste auxquels on ajoute le séparateur '-' entre chacun).

tuple

Un t-uplet (ou tuple) est une séquence statique d'éléments.

```
tup = 2,3,'six','quinze'    # 'tup' est ici une t-uplet
```

Instructions de bases

condition

L'instruction conditionnelle se forme comme suit

```
n=int(input())
if n==0:
    print('n est nul')
elif n>0:
    print('n est strictement positif')
else:
    print('n est strictement negatif')
```

boucles

La boucle `for` et la boucle `while` se forment ainsi

```
mot=list(input())
for i in range(len(mot)):
    mot[i]=mot[i]+'java'

i=0
while i<len(mot):
    print(i)
    i+=1

print(mot)
```

fonction

La définition d'une fonction commence par le token `def` et se termine à la fin de l'indentation

```
def nom_fonction(var1,var2):
    var1 += 5
    return var1*var2
```

lecture de fichier

De nombreuses solutions existent pour lire un fichier ; notamment `read()` (peu efficace en terme mémoire) ou bien en utilisant le fait que `file` soit un objet itérable (`for lign in file :`). Dans la solution suivante, on laisse l'utilisateur gérer la lecture du fichier ligne après ligne.

⚠ on prendra garde que `readline()` renvoie toute la ligne, y compris le retour chariot '\n').

```
file = open("fichier.txt","r")    # ouvre le fichier en lecture seule
line = file.readline()            # renvoie 1 ligne du fichier comme 1 string
file.close()                      # ferme le fichier
```

rq : la syntaxe `with open("fichier.txt") as file` semble de plus en plus conseillée.