

1)Module et Script

Module:

Un module en Python est un fichier qui contient du code Python et dont le nom termine par l'extension ".py" Pour importer un module dans l'interpreteur "Python"

```
>>> import 'nom_module'
```

Où nom_module est le nom du fichier correspondant au module sans l'extension ".py"

Pour ré-importer un module, il faut importer le module imp

```
>>> import imp
```

Ensuite on utilise la fonction reload

```
>>> imp.reload(nom_module)
```

"imp" est un module qui contient des fonctions pour l'import de module.

Pour appeler la fonction "f" du module "m"

```
>>> f.m(arg1, arg2,...)
```

Script:

Pour faire un Script Python (i.e écrire une application qui se lance directement depuis le shell) :
1ere ligne du fichier `which python3` (pour connaître le chemin)

```
#!/usr/bin/env python3
```

rendre le fichier exécutable

```
> chmod +x nom_fichier
```

La variable d'environnement PATH donne le chemin "absolu" jusqu'aux exécutables

```
> echo $PATH          # Pour voir les chemins qu'il contient  
> export PATH=$PATH:ref  # Pour ajouter ref aux variables d'environnement
```

La variable est définie pour la session, si on veut qu'à l'ouverture d'un shell cette définition soit prise en compte il faut mettre cette commande dans le fichier `.bashrc`(pour bash) ou `.zshrc`(pour zsh) du répertoire personnel

L'argument `__name__` est un argument prédéfini de chaque module qui vaut : En général le nom du module sans l'extension Lorsque le module est exécuté comme script, `__main__`

```
if __name__ == "__main__":  
    instruction exécuté si le module  
    est exécuté comme un script
```

Fonction `dir` et `help`

`*dir(nom-module)` : liste les variables globales et fonctions du module "nom_module" :

```
>>> s = "blabla"           # type String  
>>> dir(s)                 # liste variables globales et fonctions défini dans le module String .
```

`*help(fonc)` : aide sur la fonction "fonc".

```
>>> help(fonc)
```

2) Fonctions :

Définir une fonction :

```
def nom_fonc(arg1, arg2):  
    instruction
```

Les arguments et variables de la fonction sont locaux à la fonction. Pour définir une variable globale :

```
global nom_variable
```

ex:

```
def f():
.... global a
.... a+=1

>>> a = 5
>>>f()
>>>print(a)      # affiche 6

def g():
.... a = 7

>>> a = 5
>>> g()
>>>print(a)      # affiche 5
```

Return:

```
return (x,y,z)
```

ex :

```
def f(a,b):
.... return a+b, a*b, a/b

>>> x,y,z = f(4,6)  # x = 10, y = 24, z=2/3
```

`return` permet de retourner un n-uplet d'élément pas forcément du même type

3) Les Listes

une liste en Python est une suite (sequence) d'objets python mais pas forcément de même type

definir une liste :

```
>>> l = [3, "bonjour",3.5]
>>>type(l)      # 'list'
```

Mecanisme de "slice", Comment récupérer une sous-liste

`l[deb: fin: pas]`

```
>>> l=[1,2,3,4,5,6,7,8,9,10]

>>> l[2:8:2]      # on récupere [3,5,7]
>>> l[2:8]        # on recupere [3,4,5,6,7,8] # quand on n'indique pas le 'pas' celui-ci est automa
tiquement égal à 1
>>>l[2:]          # [3,4,5,6,7,8,9,10]
>>>l[:]           # [1,2,3,4,5,6,7,8,9,10]
>>>l[::-1]        # [10,9,8,7,6,5,4,3,2,1]
>>>l[3]           # 4 (4e élément)
>>>l[8:4:-1]      # [9,8,7,6]
>>>l[-1]          # 10 (dernier élément)
```

Ajouter un élément : (pour chacune des fonction la liste l = [1,2,3,4,5,6,7,8,9,10])

```
>>> l.extend(["toto","titi"])      # [1,2,3,4,5,6,7,8,9,10,"toto","titi"] concatenation
>>> l.append(["toto","titi"])      # [1,2,3,4,5,6,7,8,9,10,["toto","titi"]] ajout d'un élément
>>> l.insert(2,"pif")              # [1,2,"pif",3,4,5,6,7,8,9,10] ajout d'un élément a l'indice i
```

Ôter des élément :

```
>>>l = ['a','b','c']
>>>l.pop()          # retire le dernier élément de la liste
>>>l.remove('b')     # retire le premier b de la liste
>>> del l[0]         # retire l'élément 0 de la liste
```

Modifier un élément :

```
>>>l = [a,b,c]
>>> l[0] = 7        # [7, 'b', 'c']
```

Opérateur sur les listes :

- + concatenation
- * concatenation multiple
- in appartient

ex :

```
>>> l1 = ['a','b']
>>> l2 = [1,2,3]
>>> l1+l2          # ['a','b',1,2,3]
>>> l1*2           # ['a','b','a','b']
>>> c in l1        # False
```

Fonctions

4) Chaines de caractere :

Une chaines de caractere est une suite de caracteres le type Python est String

- mecanisme de slice identique aux listes
- operateur fonctionne comme pour les listes

Les chaînes sont des sequence non modifiables.

ex :

```
>>> s = 'bonjour'
>>> s[0] = 'B'
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'str' object does not support item assignment
```

Alors que :

```
>>> s = 'B' + s [1:]
>>> s          # 'Bonjour'
```

On peut aussi comparer deux chaines de caractères

```
>>> a = 'test'
>>> b = 'test'
>>> a == b      # True
```

Fonctions

- split :

Convertit une chaine en une liste de sous-chaîne

```
str.split(sep=None, maxsplit=-1)
```

```
>>> s = 'chaine de caractère'
>>> s1 = s.split()
>>> s1          #['chaine', 'de', 'caractère']

>>> s = 'chaine-de caractere'
>>> s1 = s.split('-')      #['chaine', 'de caractere']
```

- join

Convertit une liste en une chaîne de caractère

```
str.join(liste)
```

```
>>> s = 'chaîne de caractère'
>>> s1 = s.split()          # ['chaîne', 'de', 'caractère']
>>> s2 = "".join(s1)       # 'chainedecharacter'
>>> s2 = " ".join(s1)      # 'chaîne de caractère'
>>> s2 = "-".join(s1)      # 'chaîne-de-caractère'
>>> s2 = ":".join(s1)      # 'chaîne : de : caractère'
```

- count

Compte le nombre de sous-chaîne `ch` dans une chaîne `chaîne`

`chaîne.join(ch)`

```
>>> chaîne = 'chaîne, chainée chAîne'
>>> ch = 'ain'
>>> chaîne.count(ch)      # 2
```

- strip

Enlève les espaces de début et fin de chaîne

`chaîne.strip()`

```
>>> chaîne = ' chaîne '
>>> chaîne = chaîne.strip() # 'chaîne'
```

- replace

Remplace dans la chaîne `chaîne` toutes les sous-chaîne `ch1` par `ch2`

`chaîne.replace(ch1, ch2)`

```
>>> chaîne = 'chaîne de caractère'
>>> chaîne.replace(' ', '*')      # 'chaîne*de*caractère'
>>> chaîne.replace(' ', '_:_')    # 'chaîne_:_de_:_caractère'
```