LS4 23/01/2014 Anne Micheli anne.micheli@liafa.univ-paris-diderot.fr

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 moduel 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 pyhton3
```

rendre le fichier exécutable

```
chmod +x nom_fichier
```

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

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

La variable est definie 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 .barshrc(pour bash) ou .zshrc(pour zsh) du repertoire personnel

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

```
if __name__ == "__main__":
    instruction éxécuté si le mdule
    est éxécuté 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 argument et variable de la fonction sont local 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 forcement du même type

3) Les Listes

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

definir une liste :

```
>>> l = [3, "bonjour",3.5]
>>>type(l)              # affiche '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 automatiquement égal à 1
>>>l[2:]               # on recupere [2,3,4,5,6,7,8,9,10]
>>>l[:]                # on recupere [1,2,3,4,5,6,7,8,9,10]
>>>l[::-1]             # on recupere [10,9,8,7,6,5,4,3,2,1]
>>>l[2::2]             # on recupere [3,5,7,9]
>>>l[3]                # on recupere 4 (élément)
>>>l[8:4:-1]           # on recupere [9,8,7,6]
>>>l[-1]               # on recupere 10 (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
```

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
```

4) Chaines de caractere :

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

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