Séquence 0 – Rappels sur Python

Objectifs

- Affectation de variables
- P-uplets (ou tuple)
- Tableau indexé (ou list), tableau donné en compréhension, tableaux de tableaux
- Dictionnaires par clés et valeurs (méthodes keys(), values () et items ())
- Fonctions et appels de fonction
- Séquences, affectation conditionnelles
- Boucles bornées, boucles non bornées

La 1ère partie de ce document est disponible sous forme de Notebook Jupyter :
http://ninoo.fr/LC/Term_NSI/seq0_rappels_python/0_rappels_python.ipynb

Dans le même document vous pouvez lire le cours et tester les commandes python.

Pour ce faire, vous pouvez installer et utiliser Jupyter-Notebook sur votre ordinateur, ou alors vous pouvez l' utiliser en ligne :
https://flaustriat.frama.io/basthon-notebook/

Sinon, vous pouvez suivre le cours à partir de ce document pdf :
http://ninoo.fr/LC/Term_NSI/seq0_rappels_python/0_rappels_python_cours_eleves.pdf
et tester les commandes python sur un logiciel éditeur de Python ou IDE (Spyder, Edupython, …) ou en ligne : https://console.basthon.fr/

1 Rappel - Deux commandes de base Python

1.1 La commande print()

Cette commande permet d'afficher (imprimer) un message.
Attention ! Ce message doit être mis entre guillemets : " … "

A faire vous même 1.

- Ecrivez ceci :

```
>>> print(Voici un premier message)
```

- Que se passe-t-il ? Quel est le problème ? Comment le résoudre ? Ecrivez la bonne commande

  ……

- Ecrivez ceci :

```
>>> print("Affichage du deuxième message")
```

- Tapez la commande pour afficher : Python c'est super !
- Essayez les touches flèches haut / flèches bas. Que se passe-t-il ?

  ……

1.2 La commande input()

Cette commande permet de demander à un « utilisateur » de saisir un texte.

A faire vous même 2.

- Ecrivez ceci :

```
>>> input("Comment t'appelle-tu ?")
```

- Que se passe-t-il ? Que faire ?
- Tapez la commande pour afficher la question : Quelle matière préfère-tu ?
- Essayez les touches flèches haut / flèches bas. Que se passe-t-il ?

2 Notion de variable en Python

2.1 Rappel - Affectation de variable

A faire vous même 3.

- Ecrivez ceci :

```
>>> toto
```

- Que se passe-t-il ?
- Ecrivez ceci :

```
>>> toto=2005
>>> toto
```

- La variable toto est créé et contient la valeur 2005
- Ecrivez ceci :

```
>>> toto=1998
>>> toto
```

- La variable toto contient une autre valeur
- Créez une variable poids_de_mon_chat en lui affectant 6.125
- Créez une variable nom_de_mon_chien en lui affectant Médor
- Trouvez commande Python qui affiche/imprime une variable déjà créé

2.2 Variable et commande input()

Nous pouvons combiner l'affectation de variable et la commande input.

A faire vous même 4.

- Ecrivez ceci :

```
>>> prenom=input("Quel est votre prénom")
>>> prenom
```

- Que se passe-t-il ?
- Ecrivez une commande permettant de demander la couleur préférée et de la mettre dans une variable couleur
- Ecrivez une commande permettant de demander l'âge et de la mettre dans une variable age
- Testez les commandes suivantes :

```
>>> lycee="Les Cordeliers"
>>> print("Mon lycée c'est ", lycee)
```

- Trouvez une commande qui permette d'afficher une phrase (que vous inventerez) qui intègre les 3 variables (prenom, couleur et age)

2.3 Aller plus loin

- Pas de déclaration de type (Java, C/C++, C#, ...)

```
>>> symbole = 10 * 20
>>> symbole
200
>>> type(symbole)
<class 'int'>
>>> assert type(symbole)==int #Rien ne se passe assertion vraie
>>> assert type(symbole)==float # Message d'erreur
…
AssertionError
```

- original est un symbole référençant un objet de type liste

```
>>> original = [0, 1]
```

- alias référence le même objet que original

```
>>> alias = original
```

- La preuve

```
>>> alias[0] = 10
>>> assert original == [10, 1]
>>> assert alias is original # Identité d'objet
```

- Affectations multiples

```
>>> a = b = 1 + 2
```

... est équivalent à ...

```
>>> b= 1 + 2
>>> a= b
```

- Affectations multiples bis

```
>>> a, b = 1, 2
>>> a, b = (1, 2)
```

... sont équivalents à ...

```
>>> a = 1
>>> b = 2
```

- Astuce : la substitution devaleurs

```
>>> a, b = b, a
```

- Affectation conditionnelle

```
>>> b = 1
>>> a = 5 if b > 2 else 10
>>> assert a == 10
```

2.4 Rappels sur les booléens

```
>>> vrai, faux = True, False
>>> assert vrai or faux == True
>>> assert vrai and faux == False
>>> assert not vrai == False
>>> assert not false == True
```

2.5 Rappels sur les chaines de caractères

```
>>> chaine = 'simple' # Avec apostrophe
>>> chaine = "double" # Avec guillemets
>>> multiligne = u"""Cette chaîne unicode
s'étend sur plusieurs
lignes.""" # Peut aussi être encadrée avec '''
>>> jointure_automatique = (
"ligne1"
"ligne2"
)
>>> assert jointure_automatique == "ligne1ligne2"
>>> assert type(chaine) is str # Types de base des chaines
```

- Caractères spéciaux avec \xxx

```
>>> chaine = """\
ignore""" # Le saut de ligne est ignoré
>>> assert chaine == "ignore"
>>> chaine = "\"" # Guillemets
>>> chaine = '\'' # Apostrophe
>>> chaine = "\r\n\t" # Carriage return, line feed, tab
>>> chaine = "\\" # Le caractère "\" lui-même
>>> chaine = r"chaine \ brute" # Les "\" font partie de la chaine
```

- Opérations sur les chaines

```
>>> assert "foo" + "bar" == "foobar" # Concaténation
>>> assert "foo" * 3 == "foofoofoo"
>>> assert "ab" in "zabc" # Appartenance
>>> assert "xy" not in "zabc" # Non appartenance
>>> assert "abcd"[0] == "a" #N-ième item d'une chaine,partant de 0
>>> assert "abcd"[-1] == "d" # N-ième item à droite d'une chaine,
partant de -1
```

- Slices

```
>>> chaine = "abcdefgh"
>>> assert chaine[1:] == "bcdefgh" #Troncature à gauche
>>> assert chaine[:] == chaine #Ce qui permet de faire une copie
>>> assert chaine[:-2] == "abcdef" #Troncature à droite
>>> assert chaine[::2] == "aceg" #Extraction par pas de 2
>>> assert chaine [2:-2:2] == "ce" #Du 3ème à l'avant-avant
dernier par pas de 2
```

3 Les séquences - for … in range():

Quand on veut répéter une commande un certain nombre de fois (3 fois 7 fois, 632 fois, …), on utilise une séquence.

A faire vous même 5.

- Ecrivez ceci sans vous trompez :

```
>>> for i in range(3) :
```

```
>>> .... print("Coucou")
```

- Que se passe-t-il ?

  .....

- Ecrivez ceci sans vous trompez :

```
>>> for i in range(5) :
        .... print("A la ", i)
```

- Que se passe-t-il ?

.....

4 Calculer avec Python

- Quotient seuil ou quotient entier de division euclidienne :

```
>>> 27 // 7
3
```

- Modulo ou reste de division euclidienne :

```
>>> 27 % 7
6
```

- Puissance :

```
>>> 5 ** 3
125
```

5 Les scripts

La fenêtre avec les >>> s'appelle la *console*.
La console est intéressante quand on veut tester des commandes simples.
Dès qu'il faut enchaîner plusieurs commandes, il vaut mieux changer de sous-fenêtre et aller à droite pour écrire un *script*.

A faire vous même 6.

- Dans la fenêtre script écrivez une deuxième fois :

```
# -*- coding: UTF-8 -*-
print("Début du script")
for i in range(5) :
    print("A la ", i)
print("Fin du script")
```

- ATTENTION ! METTEZ 4 ESPACES AU DEBUT DE LA 3e LIGNE !
- Enregistrez ce script quelque part
- Exécutez-le
- A noter : La 1ère ligne permet simplement d'utiliser les caractères accentués (é, è, ë, ê, à, …). Elle est à ajouter à chaque début de script.

A faire vous même 7.

- Écrivez un script Python qui réalise les choses suivantes :
  - Afficher un message : Début de programme
  - Demander à un utilisateur : Jusqu'à combien dois-je compter ?
  - Afficher le comptage (Et de 1. Et de 2. Et de 3 ...)
  - Afficher un message : Fin de programme

A faire vous même 8.

- Écrivez un programme qui affiche les 20 premiers termes de la table de multiplication par 7.

A faire vous même 9.

- Écrivez un programme qui calcule le volume d'un parallélépipède rectangle dont on demande au départ la largeur, la hauteur et la profondeur.
  A noter : la fonction int() convertit une donnée saisie en entier :

```
>>>int('21')
21
```

A faire vous même 10. A faire hors classe

Écrivez un programme qui convertisse un nombre entier de secondes fourni au départ, en un nombre d'années, de mois, de jours, de minutes et de secondes. (Utilisez l'opérateur modulo : %).

6 Comparez deux valeurs

- Inférieur

```
>>> a = 1
>>> b = 2
>>> a < b
True
```

- Inférieur ou égal

```
>>> a = 1
>>> b = 2
>>> 2*a <= b
True
```

- Egalité de valeur

```
>>> a = 1
>>> b = 2
>>> 2*a == b
```

True

- Supérieur

```
>>> a = 1
>>> b = 2
>>> a > b
False
```

- Supérieur ou égal

```
>>> a = 1
>>> b = 2
>>> 2*a >= b
True
```

- Différence de valeur

```
>>> a = 1
>>> b = 2
>>> 2*a != b
```

False

7 Instructions conditionnelles

A faire vous même 11.

- Dans la fenêtre script écrivez :

```
1. # -*- coding: UTF-8 -*-
2. nombre1 = 55
3. nombre2=input("Quel est votre nombre ?")
4. nombre2=int(nombre2)
5. if nombre2 > nombre1 :
6.     print("Votre nombre est plus grand que le nombre du programme")
7. elif nombre2 < nombre1 :
8.     print("Votre nombre est plus petit que le nombre du programme")
9. else :
10.        print("Bravo ! Vous avez trouvé !")
```

- Attention ! Les lignes 6, 8 et 10 on 4 espaces en début de ligne
- Testez ce script
- Si vous avez compris, vous passez à la suite. Sinon vous appelez le professeur.

A faire vous même 12.

- L'ordi joue un dé. Le joueur joue un dé aussi. Le script dit qui est le vainqueur.
- Dans la fenêtre script écrivez ce début de script :

```
from random import randint
valeur_de1=randint(1,6)
```

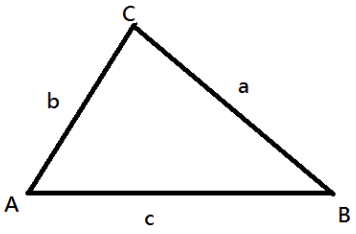
- Compléter le script avec :
  - Afficher la valeur du dé de l'ordi
  - Lancer et afficher la valeur du dé du joueur
  - Comparer les 2 valeurs
  - Imprimer qui a gagné

A faire vous même 13. A faire à la maison

- Écrivez un programme qui calcule les 50 premiers termes de la table de multiplication par 13, mais n' affiche que ceux qui sont des multiples de 7.

A faire vous même 14. A faire à la maison

ABC est un triangle dont on connaît les longueurs en cm, des côtés.
Écrire le script python qui demande à l'utilisateur 3 nombres (les longueurs des 3 côtés) et qui affiche si le triangle est rectangle ou pas.



8 Affectation augmentée

- Exemple simple :

« a += 1 » est équivalent à « a = a + 1 »

```
>>> a = 15
>>> a += 1
>>> a
16
```

- Opérateurs compatibles : + - * /

```
>>> a = 15
>>> a *= 2
>>> a
30
```

9 Boucles non bornées - while

Une boucle non bornée ne s'arrête que quand une condition est remplie.

A faire vous même 15.

- Dans la fenêtre script écrivez

```
# -*- coding: UTF-8 -*-
choix = "o"
nombre=1
while choix=="o" :
    nombre = nombre *2
    print("Puissance de 2 : ", nombre)
    choix = input("Tapez o pour continuer ")
print("AU REVOIR !")
```

- A noter : Se trouvent dans la boucle les 3 commandes qui sont décalés de 4 vers la droite

10 IMPORTANT ! L'indentation dans Python

L'indentation, c'est-à-dire le décalage vers la droite, des lignes est fondamental en Python. C'est cela qui détermine les blocs de commandes qui s'applique à tel ou tel instruction conditionnelle ou qui appartient à telle ou telle boucle.

11 Les fonctions

Il est possible de mettre des instructions que l'on utilise plusieurs fois dans une fonction. Une fonction est définie avec le mot-clé : **def**

11.1 Les fonctions mathématiques

Les fonctions python sont définies grâce au mot clé *def*.
Chaque fonction prend des paramètres (que l'on retrouve dans les parenthèses et retourne des valeur (juste après le mot-clé *return*).

A faire vous même 16.

F est la fonction définie sur $\mathbb{R}$ par :

$$\begin{cases} si\ x \leqslant 0\ alors\ f(x) = x^2 \\ si\ 0 < x \leqslant 1\ alors\ f(x) = x \\ si\ 1 < x\ alors\ f(x) = -2x + 3 \end{cases}$$

- Écrivez le programme python d'une fonction f qui donne pour résultat f(x) suivant la valeur de x.

```
premiere_fonction.py
def f(x) :
    if x<=0 :
        resultat=x**2
    elif x<=1 :
        resultat=x
    else :
        resultat=-2*x+3
    return resultat
```

- Puis testez

```
>>> from premiere_fonction import f
>>> toto=f(5)
>>> toto
-7
>>> toto=f(0.5)
>>> toto
0.5
```

A faire vous même 17.

- Sur une droite graduée, A et B sont deux points d'abscisses a et b.

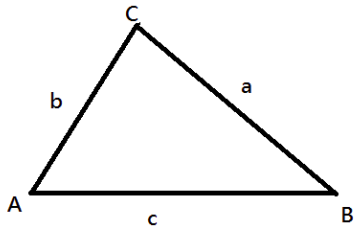
a) Exprimez la distance AB en fonction de a et b
b) Écrivez la fonction python d'une fonction `dictanceDeuxPoints(a,b)` qui donne pour résultat la distance AB

A faire vous même 18.

Même chose que l'exercice précédent mais dans le plan avec A(xa, ya) et B(xb, yb)

A faire vous même 19.

ABC est un triangle dont on connaît les longueurs en cm, des côtés.
Écrire le script python d'une fonction rectangleOuPas(a, b, c) qui donne pour résultat True (vrai) si le triangle est rectangle ou False (faux) sinon.



11.2 Précision sur les retours de fonctions

```
def returnValeurSimple():
    return 5
>>> resultat = returnValeurSimple()
>>> assert resultat == 5
def returnMultiple():
    return 2, 3
>>> resultat = returnMultiple()
>>> assert resultat == (2, 3) # Un tuple
>>> resultat1, resultat2 = returnMultiple()
>>> assert resultat1 == 2 # "unpack" du tuple retourné
>>> assert resultat2 == 3
```

- Les paramètres sont transmis par référence

```
def ajouteCleValeur(un_dico, cle, valeur):
    un_dico[cle] = valeur
    return
>>> dico = {}
>>> ajouteCleValeur(dico, 'un', 1)
>>> assert dico['un'] == 1
```

- Il y a un dictionnaire de toutes les variables et leurs valeurs

```
def voirLocales():
    dummy = 0
    assert 'dummy' in locals()
    print(locals())
>>> voirLocales()
```

11.3 Précisions sur les arguments de fonction

- Plusieurs arguments pour une fonction. On doit fournir tous les arguments positionnels

```
def foo(arg1, arg2):
    """Arguments positionnels"""
    return arg1 + arg2
>>> assert foo(2, 3) == 5
```

- Arguments nommés (qui ont une valeur par défaut) sont optionnels. Ils sont toujours après les positionnels

```
def foo(arg1, arg2=2, arg3=3):
    """Argument nommé **"""
```

```
    return arg1 + arg2 + arg3
>>> assert foo(2)== 7 #Tous les arguments nommés = par défaut
>>> assert foo(2,3) == 8 #2 pour le second argument
>>> assert foo(2,arg2=3) == 8 #Equivalent ci-dessus
>>> assert foo(2,arg3=4) == 8 #On utilise arg2 par défaut, on doit nommer arg3
```

11.4 Fonctions imbriquées

Il peut y avoir une fonction à l'intérieur d'une fonction.

```
def outer(param_o):
    locale_o = 0
    def inner(param_i):
        param_o = 4
        locale_o = 1
        return
    inner(param_o)
    return
>>> outer(6)
```

11.5 La fonction anonyme « lambda » (hors programme)

- La fonction classique add2, parce qu'elle est très simple, peut se traduire par la fonction lambda add1

```
def add2(x, y):
return x + y
>>> add1 = lambda x, y: x + y # Pour les fonctions en une ligne
>>> assert add1(2, 3) == 5 # Sans surprise
```

11.6 Fonctions « builtin »

- 80 fonctions « builtin »
- Ne réinventez pas l'eau chaude
- Opérations les plus courantes sur les types standard
- `len(objet)` : nombre de sous-éléments d'un objet multivalué
- `globals(), locals()` : dictionnaires des variables globales et locales
- `int(objet), float(objet), hash(objet)` : conversions
- `help(objet)` : doc d'API de l'objet dans la console interactive
- `list(objet), tuple(objet)` : conversions d'objets multivalués
- etc...
- Toutes les fonctions « builtin »:

http://docs.python.org/library/functions.html

A faire vous même 20. !!! MINI-PROJET INDIVIDUEL NOTE !!!

Programmez différentes fonctions :

- Fonction `maximum(n1,n2,n3)` qui renvoie le plus grand de 3 nombres n1,n2, n3
- Fonction `aireDisque` avec un seul argument : rayon
- Fonction `volumeCylindre` avec 2 arguments : rayon et hauteur
- Fonction `volumeSphere`
- Fonctions `aireRectangle`
- Fonction `volumePrismeDroit` que l'on puisse appelé avec 1, 2 ou 3 arguments

Ajoutez dans le corps de programme un menu qui :

- demande quelle fonction est à utiliser
- demande quelle(s) paramètre(s) est(sont) à utiliser
- appelle la fonction correspondante
- boucle tant que l' utilisateur n' a pas décidé de quitter.

BONUS : Une fonction peut faire appel à une autre fonction.