TP PYTHON, N°3, NON NOTÉ

Durée : 3 heures

Ce TP a pour objectif de vous familiariser avec les bases de `Python 3.#`. La version 2 de Python est encore très largement utilisée par certains logiciels, et présente quelques différences minimes par rapport à la version 3.

`Python` est installé par défaut sur les machines de l'IUT. Pour utiliser le langage on utilise soit le terminal, soit un EDI (environnement de développement informatique, ou IDE en Anglais) tel que `IDLE` (que l'on utilisera à l'IUT) ou bien `Spyder` qui est plus adapté aux usages mathématiques et numériques. Pour installer chez vous `Python`, deux solutions à conseiller :

— le site officiel `https://www.python.org/downloads/`, le package installe également `IDLE` par défaut,

— l'installateur `Anaconda` sur `http://continuum.io/downloads` qui livre `Spyder` comme `IDE` de base et est plus destiné à l'usage Mathématique.

Pour démarrer une interface Python, rechercher `IDLE` à l'aide de l'explorateur `Windows`, taper par exemple `Python` et valider avec `Entrer` afin de tester la bonne installation du programme.

1 Types et opérateurs de base

Dans la première partie nous nous intéressons aux principaux types d'objets. Un exemple : `1` est un objet du type `int(eger)` avec les fonctions et opérations qui lui sont propres, mais `1.0` est du type `float`. On dit que `Python` est un langage *typé*. Pour connaitre le type d'un objet on utilisera la commande `type(nomobjet)`.

1.1 Variables

Dans une ligne, tout ce qui est à droite du dièse # est en commentaire et n'est donc pas compilé. Ainsi dans les exemples suivants, on ne demande pas de taper ces commentaires. Exécuter les lignes suivantes :

```
>>> 2+3
>>> 2+3-5
>>> 2*(1-(3+5))
>>> 2/3
>>> 2.0/3
>>> a=3
>>> a
>>> print(a)
>>> a+=5
>>> a
>>> type(a)
>>> a/3
>>> float(a)
>>> float(a)/3
```

Comprendre ce que chaque ligne représente et les noter si besoin à droite. Les principaux types sont `int, string, list, dict`, et dans chaque type les variables ne peuvent prendre qu'une plage de valeurs. Par exemple `int` contient les entiers entre -2^{31} et $2^{31}-1$ pour des questions de mémoire. On peut passer, pour des types proches, de l'un à l'autre. Tester par exemple ceci à la suite des lignes *supra*.

```
>>> b=float(a)  # conversion de type
>>> print b
>>> type(b)
>>>
>>> c=str(b)  # conversion de type
>>> print(c)
>>> type(c)
>>> del c    # destruction de c
>>> c # ERREUR: c n'existe plus
```

La commande `print` sert à **afficher uniquement** une variable, par contre on ne peut pas transférer une variable vers l'autre avec `print`, seule la commande d'affectation via le signe = peut le faire. Tester :

```
>>> c=2
>>> print(c)
>>> c
>>> d=print(c)
>>> d
>>> print(d)
>>> d=c
>>> d
```

Les variables peuvent donc contenir des types, et leur typage se fait automatiquement par `Python`. Passons-les en revue.

1.2 Chaines de caractères, type `str`

Une chaîne de caractères (`string`) correspond à un tableau de caractères, lesquels sont accessibles par leur indice. Il s'agit d'un moyen très économique en mémoire pour stocker des données, mais qui ne peut être modifié simplement. Certains types sont dits *non immuables* (qui peut être changé) et d'autres *immuables* (qui ne peut être changé).

Le type `str` appartient au second cas, une fois la chaine de caractère crée on ne peut la modifier, à moins d'en réaliser une copie puis de garder ce que l'on souhaite.

```
>>> s='acgt'
>>> s  #la valeur
>>> print(s)
>>> len(s)              #la longueur
>>> s[0]
>>> s[3]
>>> s[4]                #erreur ?
>>> s[-1]
>>> s[-4]
>>> s[1:3]
>>> s[2:]
>>> s[:3]
>>> s[:4:2]
>>> s[-2::-1]
>>> s[0]='b'
>>> print(s+'att')  # concaténation
>>> s+='b'          # crée une NOUVELLE ch
>>> print(s)
>>> print(2*s)
>>> a='rac2 vaut '+str(1.41421)
>>> print(a)
```

Une chaîne de caractères peut aussi être délimitée par `"` et `"""`. Le dernier délimiteur autorise les *retours chariot* (mot pour désigner un retour à la ligne dans un programme) dans les `strings`.

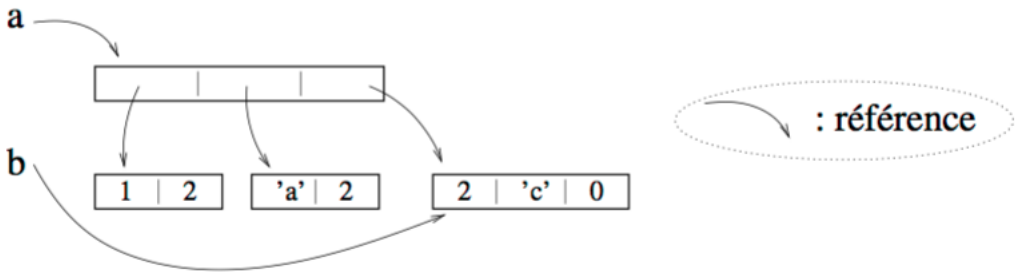
Enfin, puisque notre cerveau n'est pas extensible, il n'est aucunement nécessaire de retenir toutes les opérations envisageables sur ce type. Il suffit de taper `help(str)` pour les faire afficher.

1.3 Listes, type `list`

Les listes sont matérialisées par des crochets `[]`. Elles ont l'avantage d'être non immuables et possèdent, par exemple, ces types d'opérateurs : `len, [], +, +=`. Essayer :

```
>>> range(10)    # commande qui crée une liste
>>> range(8,1,-1)
>>> range(0,19,4)  #attention à la numérotation à partir de 0 pour range
>>> x=[1.2,4,'a',6]  # liste
>>> type(x)
>>> x[:3]
>>> x[1:4:2]
>>> x.append(2)      # ajout à la fin de la liste
>>> print(x)
>>> a=[[1,2],['a',4],[2,'c',0]]  # liste de listes, des matrices
>>> print(a[1][1])
>>> a[1]='all'
```

```
>>> print(a)
>>> b=a[2]
>>> b[0]='tt'
>>> print(a)
>>> b=[9]
>>> print(a)              #ne pas oublier d'annoter toutes ces actions à côté
#des lignes de codes pour vous en souvenir au prochain TP
```



Il peut être utile de représenter les affectations de variables au moyen d'un tel diagramme, notamment lorsque certaines sont déclarées à l'intérieur ou à l'extérieur de boucles.

2 Structures de Contrôle : conditions, boucles et fonctions

Jusqu'à présent, toutes les commandes ont été tapées dans l'interpréteur basique. Mais comment sauvegarder un programme ? Il faut pour cela utiliser ce qu'on appelle l'éditeur : ouvrir une fenêtre avec `New File`, enregistrer sous un fichier `.py` où vous le souhaitez. Attention cependant, en exécutant le fichier avec la commande `CTRL F5` on exécute tout le fichier en question, donc deux solutions : créer plusieurs fichiers, ou bien mettre en commentaire ce qu'il ne faut pas exécuter.

La conception d'un programme passe par l'exécution de conditions, boucles, fonctions que l'on appelle des *structures de contrôle*. Pour délimiter correctement les structures, les indentations sont **fondamentales** ; autant sur une feuille elles n'ont d'intérêt que pour la clarté de lecture, mais en `Python` elles sont indispensables pour que l'ordinateur interprète correctement votre programme. Tester par exemple les deux programmes suivants et observer :

```
a=2
for a in range(0,2):
        a=a+1
print(a)
```

```
a=2
for a in range(0,2):
a=a+1
print(a)
```

⊙**Attention.** `range` n'est pas un objet de type `list` comme on pourrait le croire, mais est ce qu'on appelle un *ittérateur*. Pour regarder à quoi il correspond, il faut d'abord le convertir via `list(range(10))` par exemple.

L'indentation est naturelle, elle concerne notamment les instructions qui doivent être réalisées dans une boucle (`for, if, while` peu importe). Si une autre boucle est crée dans une boucle, on indente alors deux fois. L'indentation standard est réalisée avec la touche `Tab`.

Afin de finir cette introduction, une `fonction` sert uniquement à regrouper une suite d'instructions afin de la réutiliser plus tard. Elles sont utiles pour donner de la clarté aux programmes.

2.1 Conditions

Il s'agit d'effectuer des ordres sous une condition. Pour cela, il faut effectuer un test, et si le résultat de ce test est vrai, les ordres voulus sont effectués. Éventuellement, dans le cas contraire, d'autres ordres sont effectués. Pour effectuer des tests, on dispose de nombreux opérateurs qui re-

tournent `True` ou `False`.

```
>>> a=2
>>> a==2
>>> a!=2
>>> 2>3
>>> 2>=2
>>> 3<=4
>>> not(a>=4)
>>> a in [3,2,4]
>>> (a in [0,2]) and (2<5)
>>> (3>5) or (2<5)
```

Et voici le premier exemple de *boucle*, le test conditionnel.

```
>>> a=5
>>> b=3
>>> if a<b:
...     print('b est > à a')
```

On peut aussi ajouter un scénario dans le cas où le test renvoie `Faux` au moyen d' `else` :

```
>>> a=5
>>> b=3
>>> if a<b:
...     c=0
...     print(b)
... else:
...     c=1
...     print(a)
>>> print(c)
```

ou même plusieurs scénarios :

```
>>> x='a'
>>> if x=='c':
...     print('cyt')
... elif x=='g':
...     print('gua')
... elif x=='t':
...     print('thy')
... else:
...     print('ade')
```

Exercice 1

Une année est dite bissextile si c'est un multiple de 4, sauf si c'est un multiple de 100. Toutefois, elle est considérée comme bissextile si c'est un multiple de 400. Écrire un programme qui détermine si une année donnée est bissextile ou non.

Remarque. La commande `input('message d'info qui peut être vide')` permet d'intéragir avec l'utilisateur. `a=input()` va ainsi demander d'entrer une valeur et la placer ensuite dans `a`. En pratique, on utilisera plutôt la notion de `fonction`, définie *infra*.

2.2 Boucle `For`

Il s'agit d'exécuter plusieurs fois la même succession de commandes. Par le mot réservé `for`, on peut procéder à des itérations finies. Regarder et comprendre les deux codes suivants, et bien sûr il faut bien contrôler l'indentation.

```
>>> s='accgcacgaagt'
>>> for i in s:
     print i
>>> l=len(s); print(l)

>>> for i in range(l):
     if s[i]=='a':
         print('ade position'+str(i))
#concaténation de str
```

Exercice 2

Ecrire un programme qui calcule et affiche la somme $S = 1+5+9+13+\ldots+1001$.

2.3 Boucle `While`

On pourrait se passer d'une telle boucle, en utilisant les deux structures vues précédemment, mais `while` permet alors de rendre le code plus clair en combinant une boucle `for` qui doit s'executer non pas sur un certain domaine, mais tant qu'une certaine condition est vérifiée. Autrement

dit, si cette dernière l'est toujours votre boucle `while` ne s'arrêtera jamais. La structure est

```
while condition logique :
        Actions à effectuer
        ...
```

Exercice 3 *Rebonds*

Une balle chute de 400 mètres. A chaque rebond, la hauteur diminue de 10%. Écrire un programme qui affiche la hauteur de rebond tant que celle-ci est supérieure à 5 mètres. Afficher alors le nombre de rebonds.

2.4 Les fonctions

Un exemple de définition d'une fonction qui affiche les types de deux variables rentrées, et retourne la somme :

```
def fonc(a,b):
        print(type(a),type(b))
        return a+b
#appel de fonction fonc
fonc(2,3)
```

Il faut noter que `print` affiche seulement, mais ne permet pas de récupérer la valeur, dans un futur programme par exemple. Pour ce faire, il faut utiliser la commande `return`. La commande `def` permet seulement de définir la fonction, mais ne la compile pas : ainsi si des erreurs sont présentes en son corps, vous ne les constaterez qu'à partir de la première exécution.

Exercice 4

Créer une fonction `Euclide` qui renvoie, étant donnés deux nombres, leur reste et quotient de division Euclidienne.