Introduction au langage Python

I) Environnement de programmation

Nous utiliserons EduPython, une solution qui intègre un éditeur de programme (zone de saisie de texte, pour écrire votre programme) et un compilateur (qui va traduire votre programme en langage machine) : en une seule étape, on peut donc écrire son programme et en tester le fonctionnement.
Le logiciel est téléchargeable gratuitement à l'adresse http://edupython.tuxfamily.org/download.php.
Il utilise la version 3 de Python.

Le langage Python est placé sous une licence libre et fonctionne sur la plupart des plates-formes informatiques, des supercalculateurs aux ordinateurs centraux, de Windows à Unix en passant par Linux et Mac OS.
Il est caractérisé par langage où la syntaxe, clairement séparée des mécanismes de bas niveau, permet une initiation plus aisée aux concepts de base de la programmation.

C'est le cinquième langage le plus utilisé en programmation (en septembre 2016, d'après http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html), derrière Java, C, C++ et C#.

D'autres sites intéressants :
http://python.org
http://python.lycee.free.fr

II) Le langage Python

1) Notions de base sur les langages de programmation

Un langage de programmation est comme une langue étrangère, il faut donc respecter une syntaxe, qui précise de quelle façon les éléments doivent être agencés pour être compris par le compilateur.
Le langage de programmation obéit aussi à un vocabulaire précis, des symboles d'opération et de ponctuation.
L'alphabet était basé sur la norme ASCII. Depuis la version 3 de Python, il est basé sur la norme Unicode (ce qui autorise notamment les caractères accentués).

1. Les mots clé sont des mots dont la signification est fixée dans le langage (if, for…)
2. Les commentaires sont des explications, rédigés (pour vous, ou pour un relecteur) en français courant, pour expliciter la raison d'une ligne de programme, pour dire ce que va représenter une variable ou autre. En python, les commentaires sont précédés de # (pour faire comprendre au compilateur que ce qui suit ce symbole n'est pas à tenter d'interpréter).
3. Les identifiants : ce sont des éléments constitutifs du programme : les variables, les procédures et les types.
   a. Une variable : c'est un nom utilisé dans un programme pour faire référence à une donnée manipulée par le programme. C'est la personne qui écrit son programme qui décide du nom de chaque variable.
      Dans python, les variables peuvent être constituées d'une ou plusieurs lettres, avec divers symboles (−_ ou autre), le premier caractère du nom de variable doit être une lettre
      Attention, python est sensible à la casse : A et a sont deux variables distinctes.
   b. Les types : Chaque donnée a une classification, celle-ci influe sur la plage de valeurs possibles, les opérations qui peuvent être effectuées, et la représentation de la donnée sous forme de bits. Chaque langage de programmation offre une gamme de types primitifs, incorporés dans le langage.

2) Les types de base (ou « primitifs ») dans le langage Python

Il y a notamment, les types suivants, que l'on utilisera cette année :

1. Les booléens (`bool`) : ne comporte que deux constantes : `True` (Vrai, ou bien `1`) et `False` (Faux, ou bien `0`). On a donc `True > False` (c'est très moral).

2. Les entiers (`int`) : le type « entier » de python concerne tous les entiers (positifs et négatifs), la taille de l'entier n'est limitée que par les capacités de stockage de la machine.
3. Les réels (`float`) : ce type est constitué d'une suite de chiffres, avec le symbole « `.` » comme séparateur décimal, la lettre « `e` », pour marquer la multiplication par une puissance de 10 et éventuellement le signe « + » ou « − ».

   Par exemple `-.9e-2` signifie $-0{,}9 \times 10^{-2} = -0{,}009$.
4. Les complexes (`complex`) : que l'on n'utilisera pas tout de suite, et probablement pas cette année dans le cadre de la spécialité. Toutefois, au cas où cela vous serait utile dans une démarche personnelle : le nombre complexe i (en mathématiques) est noté `j` (en python), donc une variable de type complexe est sous la forme `a + bj`, où `a` et `b` sont des variables de type `float` (ou `int`, éventuellement).
5. Les chaînes de caractère (`str`) : Une suite de caractères encadrée soit par des apostrophes `'`, soit par des guillemets anglo saxons `"`. (Les deux syntaxes sont équivalentes).

   La chaîne de caractère vide est donc `""` ou bien `''`.

   Si votre chaîne de caractères doit contenir une apostrophe, vous encadrerez la chaîne par des guillemets, et vice versa.
6. Les listes (`list`) : une liste est une suite de valeurs, éventuellement de types différents, chaque élément de la liste étant repéré par son rang. Attention, en python le premier élément de la liste est numéroté `0`.

   Concrètement, la liste se présente comme la succession des différentes valeurs, dans l'ordre et séparées par des virgules, le tout entre crochet. La liste vide est `[]`.
7. Les itérateurs (range) : un itérateur permet de désigner une suite de nombres entiers avec un début, une fin et éventuellement un pas. Depuis la version 3 de Python, l'objet créé décrit la suite, avant, c'était une liste contenant lesdits nombres.

3) Les opérations de base dans le langage Python

1. L'affectation se fait à l'aide du signe =.

   `a = b` signifie que la valeur b est désormais stockée sous le nom de variable a.

2. Les opérateurs mathématiques classiques : `+` pour l'addition, `-` pour la soustraction, `*` pour la multiplication et `/` pour la division ont leur fonction usuelle sur les nombres entiers, réels ou complexes.

   Pour calculer une puissance, on utilise l'opérateur `**`. `a**b` est donc a^b.

   `//` est l'opérateur qui donne le quotient dans la division euclidienne et `%` est l'opérateur qui donne le reste dans la division euclidienne.

   `25//8` donnera donc 3 et `25%8` donnera 1, car $25 = 3 \times 8 + 1$.
3. Pour les comparaisons, on utilise <, > pour les comparaisons strictes, <= et >= pour les comparaisons larges, == pour tester si deux variables sont égales, et `!=` pour tester si elles sont différentes.

   `a = b` et `a == b` sont donc deux choses distinctes.

   Quand on effectue une comparaison, le résultat est un booléen : soit la comparaison est vraie, soit elle est fausse.
4. Sur les booléens : On a l'opérateur de négation, pour donner le contraire d'un booléen (`not`), d'intersection, pour avoir une condition et une autre (`and`), et l'opérateur d'union, pour une condition ou une autre (`or`).
5. Sur les chaines de caractères.

   La concaténation se fait en utilisant +.

   `'Roch' + 'ambeau'` va renvoyer `'Rochambeau'`

   `in` va chercher si un caractère ou une chaîne de caractère est présent dans une chaîne de caractères.

   `'au' in 'Rochambeau'` va renvoyer `True`.

   Pour extraire un caractère ou une chaîne de caractères d'une autre chaîne, on utilise `[]`, en mettant entre crochet la position ou les positions des caractères à extraire. Ceci s'écrit après la chaîne de caractères

`a = 'Rochambeau'`
`a[1] = 'o'` (la première lettre est numérotée 0).
`a[2:5] = 'cha'` (de la position 2 incluse à la position 5 exclue)
`a[5:] = 'mbeau'` (de la position 5 incluse à la fin de la chaîne `a`).
`a[:4] = 'Roc'` (du début de la chaîne à la position 4 exclue).
Remarque : on peut aussi utiliser cela avec des entiers négatifs : `−1` étant la dernière lettre de la chaîne, `−2`, celle d'avant etc.
Sur une chaîne de longueur *n*, on peut donc invoquer des positions allant de –*n* jusqu'à *n* – 1. Tout appel en dehors de cet intervalle renverra une erreur "index out of range"

6. Sur les listes.
L'ajout d'une valeur en fin de liste est faite par : `L.append(valeur)`.
Pour extraire une valeur d'une liste, on va la chercher comme on irait chercher un caractère dans une chaîne de caractères. On peut aussi extraire une sous-liste de la même façon.
On peut aussi concaténer deux listes (avec +) ou rechercher si une valeur est présente dans une liste (avec `in`).

7. Sur les itérateurs.
L'instruction `range` crée un itérateur (plus ou moins une liste de nombres entiers).
`range(5)` est équivalent à la liste `[0, 1, 2, 3, 4]` (les nombres entiers de 0 inclus à 5 exclu)
`range(4,15)` est équivalent à la liste `[4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14]` (les entiers de 4 inclus à 15 exclu)
`range(4, 15, 2)` est équivalent à la liste `[4, 6, 8, 10, 12, 14]` (les nombres de 4 inclus à 15 exclu, par pas de 2).
Les arguments de la fonction range doivent être entiers (si vous y mettez un réel, il sera arrondi à l'entier par défaut).

4) Instructions d'entrée / sortie

Elles sont là pour faire interagir le programme et son utilisateur.
Pour lire une donnée, c'est-à-dire faire demander à l'utilisateur une valeur, on utilise :
`variable = input('texte à afficher').`
variable est le nom sous lequel on veut utiliser la valeur entrée par l'utilisateur.
texte à afficher est ce qui s'affichera dans la fenêtre de saisie de l'utilisateur.
La fonction « `input` » considère que ce que va saisir l'utilisateur est une chaîne de caractères (type `str`). Si on souhaite que la saisie soit un nombre, on va utiliser une fonction de conversion de type :
`int(input('saisir un nombre entier'))`
ou bien `float(input('saisir un nombre réel')).`

Exemple :
```
n = int(input('entrer le nombre n').
```

Pour afficher une donnée, on utilise `print`, en séparant les différents éléments à afficher par des virgules.

Exemple :
```
print('hello world',5)
```

5) Les instructions conditionnelles

Elles ne sont effectuées que si une condition est vérifiée. C'est là qu'on va utiliser des booléens.
L'instruction de base est `if`, suivi de la condition (qui doit donc renvoyer un booléen, un `0` ou un `1`) La ligne doit impérativement se terminer par un deux-points « **:** ».
Si le booléen est `True` ou si on a un `1`, alors une série d'instruction sera effectuée.

Python est un langage indenté, donc le « bloc » d'instruction à n'effectuer que si la condition est vérifié doit être décalé vers la gauche.
La fin de ce bloc est marquée par le retour à l'alignement vertical avec le « `if` ».
On peut aussi intégrer un « sinon » et même un (ou plusieurs) sinon, si :

```
n = int(input('n='))
if n > 0 :
    n = n+1
print (n)
```

```
n = int(input('n='))
if n > 0 :
    n = n +1
else :
    n = n - 1
print (n)
```

```
n = int(input('n='))
if n > 0 :
    n = n + 1
elif n==0 :
    n = n - 1
else :
   n = 'Rochambeau'
print (n)
```

6) Les instructions itératives

Une instruction itérative est une instruction qui indique qu'un « bloc » d'instructions sera répété un certain nombre de fois, ou éventuellement tant qu'une condition est avérée.
Le bloc d'instruction est délimité par l'indentation, à nouveau.

a) La boucle "pour" (`for`)

L'instruction est assez simple. Là encore, on remarque que la fin de la ligne contenant `for` doit être identifiée par un deux-points.

```
for i in L :
```

`L` doit être une liste ou un itérateur. Le « bloc » d'instructions sera exécuté autant de fois qu'il y a d'éléments dans la liste `L`. La variable `i` sera modifiée à chaque exécution de la boucle pour correspondre chacune des valeurs de la liste `L`.
Exemple : Le programme suivant renvoie…

```
s = 0
for i in range(8) :
  s = s + i
print s
```

```
s = ''
for i in 'Rochambeau' :
  s = s + i
print s
```

Pour bien comprendre l'importance de l'indentation, vous pouvez essayer de saisir le programme ci-dessus, en décalant le `print(n)` final pour l'aligner verticalement avec le `s=s+i`.

b) La boucle "tant que" (`while`)

Même principe :

```
while b :
```

`b` doit être un booléen, et donc peut être un test (le plus souvent), ou bien `0` ou `1`.
Le bloc d'instructions sera exécuté tant que le booléen `b` correspond à `True`. Le risque d'une boucle « `while` » est d'avoir mal préparé son algorithme, et du coup que le booléen `b` reste toujours vrai, auquel cas la boucle s'exécute indéfiniment : l'ordinateur n'est plus utilisable, il passera le reste de l'éternité à effectuer le bloc d'instructions. Il est parfois prudent d'intégrer une sécurité : intégrer un compteur d'itérations et donner en condition de ne pas dépasser un certain nombre d'itérations.

III) Exercices

Pour prendre en main python, vous allez implanter sous python quelques algorithmes simples. (implanter un algorithme, c'est le transformer en programme, en utilisant un langage de programmation).

1. Un algorithme qui demande un nombre entier n à l'utilisateur, et qui affiche $n^2 + 2n - 1$.

2. Un algorithme qui demande un nombre entier n à l'utilisateur, et qui affiche le terme d'indice n dans la suite définie par récurrence par : $u_0 = 2$ et, pour tout entier naturel n : $u_{n+1} = \frac{1}{2}\left(u_n + \frac{2}{u_n}\right)$

3. Un algorithme qui demande une chaîne de caractères à l'utilisateur et qui affiche le troisième caractère de la chaîne (si la chaîne est assez longue), ou le dernier caractère de la chaîne sinon.

4. Un algorithme qui demande deux chaînes de caractères et qui affiche la concaténation de ces deux chaînes.

5. Une fonction qui demande une chaîne de caractères et qui renvoie une chaîne avec l'ordre des caractères inversé.

6. Un algorithme qui demande une chaîne de caractères et qui vérifie s'il s'agit d'un palindrome (lisible de gauche à droite et de droite à gauche, comme « laval » ou « radar »).

7. Un algorithme qui fusionne deux chaînes de caractères, en alternant un caractère pris dans la chaîne n°1 et un dans la chaîne n°2. On veut que l'algorithme choisisse comme chaîne n°1 la chaîne la plus longue, et que quand la chaîne n°2 est « épuisée », car tous ses caractères ont été utilisés, on reprend cette chaîne au début. L'algorithme doit s'arrêter dès que tous les caractères de la chaîne n°1 ont été utilisés. En bonus, l'algorithme affichera la chaîne créée, sa longueur et la longueur des deux chaînes de départ.

8. Écrire un programme qui, étant données deux bornes entières a et b, additionne les nombres multiples de 3 et de 5 compris entre ces bornes.
Prendre par exemple $a = 0$, $b = 32$ le résultat devrait être alors $0 + 15 + 30 = 45$.

9. Modifier légèrement ce programme pour qu'il additionne les nombres multiples de 3 ou de 5 compris entre les bornes a et b. Avec les bornes 0 et 32, le résultat devrait donc être :
$0 + 3 + 5 + 6 + 9 + 10 + 12 + 15 + 18 + 20 + 21 + 24 + 25 + 27 + 30 = 225$.

10. Déterminer si une année (dont le millésime est introduit par l'utilisateur) est bissextile ou non. (Une année A est bissextile si A est divisible par 4. Elle ne l'est cependant pas si A est un multiple de 100, à moins que A ne soit multiple de 400).

11. Écrire un programme qui, étant donnés trois nombres a ; b et c, entrés par l'utilisateur va afficher le plus grand des trois nombres.

12. Écrire un programme qui, étant donnés cinq nombres entrés par l'utilisateur va afficher la médiane de cette série statistique.

Vous aurez peut être besoin (en particulier pour les derniers exemples) d'autres fonctions du programme python : n'hésitez pas à demander, ou à rechercher une solution sur un moteur de recherche (encore mieux).
Notez les fonctions ainsi découvertes et leur syntaxe.