chapitre 1

Premiers pas en Python

Table des matières

I Notion d'algorithme . . . 2
II Windows et scribe . . . 2
2.1 Votre environnement . . . 2
2.2 Quelques raccourcis clavier . . . 3
III Début avec Python . . . 4
3.1 Quelques caractéristiques du langage . . . 4
3.2 Environnement de travail . . . 4
3.3 La console . . . 5
3.4 L'éditeur . . . 6
3.5 Importation de modules . . . 7
IV Notion de variables et de types . . . 7
4.1 Variables et expressions . . . 7
4.1.1 Variables . . . 7
4.1.2 Notion d'expression . . . 8
4.1.3 Affectation . . . 8
4.2 Types de variables . . . 9
V Entrées sorties . . . 11
5.1 Affichage non graphique . . . 11
5.2 Entrée au clavier . . . 12
VI D'autres exercices . . . 12

Ce document est un TP/cours. Certaines choses seront évoquées en classe, vous en découvrirez d'autres durant votre premier TP. Nous avons ici plusieurs objectifs :

- ⋆ Prendre en main les environnements de travail (Windows et Python) qui vont vous accompagner pendant au moins deux ans.
- ⋆ Découvrir Python : en mode « interprété ligne à ligne » et en mode « exécution d'un fichier ».
- ⋆ Commencer à comprendre les notions de variable et d'affectation.

I Notion d'algorithme

Le mot **algorithme** vient du nom latinisé du mathématicien Al-Khawarizmi, il s'agit d'une suite finie et non ambiguë d'opérations ou d'instructions permettant de résoudre un problème.

Nous verrons cette année différents algorithmes : recherche du maximum dans une liste de nombres, algorithmes de tri d'un tableau, ...

Regardons d'ores et déjà un algorithme pour la résolution des équations du second degré à coefficients réels.

Variables utilisées. 4 réels : a,b,c,delta
Description de l'algorithme.
Entrer a,b,c
Calculer delta=$b^2 - 4*a*c$
Si delta>0
    Afficher : il y a deux racines réelles qui sont $(-b + \sqrt{delta})/(2a)$ et $(-b - \sqrt{delta})/(2a)$
Sinon si delta=0
    Afficher : il y a une racine réelle qui est $-b/(2a)$
Sinon
    Afficher : il n'y a pas de racine réelle

Les ordinateurs sont des machines capables d'exécuter des algorithmes pour peu qu'il soit écrit en langage machine : une suite de 0 et de 1. En pratique, on écrit le code source d'un programme [1] dans un **langage de programmation** (C, Python, Java, ...) sous la forme d'un fichier texte puis un **compilateur** ou un **interpréteur** le transforme en une suite d'instructions compréhensibles par la machine.

Le langage avec lequel nous allons travailler cette année est **Python**. Comme la plupart des langages de programmation, Python comporte des constructions fondamentales : la déclaration et l'affectation de variables, le test, la boucle, la fonction, ... Nous découvririons tout cela au cours de l'année.

II Windows et scribe

2.1 Votre environnement

Les ordinateurs du lycée fonctionnent avec le système d'exploitation Windows (32 bits ou 64 bits, XP ou Seven selon les machines). Ils sont interconnectés en réseau : un ordinateur central joue le rôle de serveur pour les autres machines appelées stations de travail. L'architecture logicielle du réseau est Scribe : le serveur est sous Linux et gère un réseau de stations sous Windows.

Exercice 1.1 *Se loguer au réseau Scribe.*

Par défaut le login est de la forme `prenom.nom` ou `p.nom` ou `prenom.n` pour les noms ou prénoms composés ou trop long.

Le mot de passe est la date de naissance au format jjmmaaaa. Lors de la première connexion changer immédiatement de mot de passe si cela n'a pas déjà été fait.

L'interface graphique utilisateur est celle de la version de Windows utilisée avec quelques spécificités dans les actions permises à un utilisateur élève. Les différents raccourcis du bureau ou du menu Démarrer permettent de lancer les applications disponibles.

Le lecteur local `C :` n'est pas visible pour les élèves. Les espaces offerts à l'utilisateur élève sur les disques durs du serveur pour stocker et gérer ses fichiers de données sont dénommés *lecteurs réseaux* :

- ⋆ Lecteur Commun `T :`

  Il contient un dossier [2] `Logiciels` avec les petits logiciels portables et un dossier `Travail` en lecture seule

1. Un programme est un texte qui décrit un algorithme que l'on veut faire exécuter par un ordinateur.
2. Selon les écoles, on parle de répertoire / dossier et sous-répertoire/sous-dossier

pour les élèves. Ce dernier contient des documents destinés à tous les utilisateurs comme la plaquette de présentation de Scribe.

- ⋆ Lecteur Personnel `U :`

  La racine de ce dossier est accessible depuis le raccourci « Mes Documents » (dans le bureau ou l'explorateur de fichier). L'élève et son professeur peuvent lire et écrire dans ce dossier à une exception près, son sous-dossier `prive` n'est visible que de l'élève. Le professeur peut déposer/récolter un devoir dans le sous-dossier `devoirs`.

- ⋆ Lecteur Groupes `S :`

  Celui-ci contient au moins le dossier de la classe qui contient deux sous-dossier : `donnees` en lecture seule pour l'élève et lecture/écriture pour le professeur ; `travail` en lecture/écriture pour tous.

  Le professeur peut par exemple déposer un énoncé de devoir dans le dossier `donnees` et les élèves peuvent rendre leur travail dans `travail` même s'il est préférable d'utiliser le sous-dossier devoirs de leur lecteur personnel `U:`.

Exercice 1.2 *Création du dossier informatique*

Dans un dossier personnel de documents, créer un sous-dossier consacré à l'informatique. Dans ce dernier, créer un sous-dossier consacré aux TP de première année, et dans ce sous-dossier, créer un sous-dossier consacré à ce TP.

Remarque : Si cela n'a pas été le cas, pour nommer vos dossiers faites le sans accent ni espace ou caractère spécial (du type `@` ou `&`). Par contre, le tiret `-` (« moins ») et le tiret-bas `_` (« underscore ») sont autorisés [3].

Vous pouvez accéder à vos différents dossiers et fichiers depuis chez vous. Pour ce la il suffit de vous connecter au portail du lycée :

http://portail.lyc-du-parc.ac-lyon.fr/

Cliquez alors sur *« Mes documents au lycée »* et connecté vous. Les identifiants de connexion sont les mêmes que sur le réseau, c'est à dire ceux que vous venez d'utiliser.

2.2 Quelques raccourcis clavier

- ⋆ Ctrl-C : pour copier un fichier sous Windows, ou un morceau de texte préalablement sélectionné (grisé) sous un éditeur.
- ⋆ Ctrl-X : pour couper (un fichier ou un morceau de texte préalablement sélectionné). Il est mis dans le « presse-papiers », donc peut être ensuite collé.
- ⋆ Ctrl-V : pour coller (un fichier ou un morceau de texte) qui aura préalablement été mis de côté via un *copier* ou un *couper*.
- ⋆ Ctrl-S : pour sauver le fichier sur lequel on est en train de travailler (sur quasiment tous les éditeurs, dont Word, libre/open-office, Pyzo, le bloc-notes Windows...)

Je profite de ce paragraphe pour vous signaler à quoi servent certaines touches de votre clavier. Remontons depuis la touche Ctrl à gauche du clavier.

- ⋆ Shift : sert à faire des majuscules, ou bien obtenir un chiffre sur la clavier principal.
- ⋆ Caps Lock : ne sert pas à obtenir une majuscule, contrairement à ce que pensent certains. Il est utile pour faire n majuscules, avec n assez grand. En deçà de ce certain nombre, qui varie suivant les individus, on laisse un doigt appuyé sur Shift.
- ⋆ La touche au-dessus de Caps Lock est la touche de tabulation, qui permet d'écrire grosso modo un bloc d'espaces. En Python, les tabulations sont pour la plupart mises automatiquement par l'éditeur, mais certaines sont parfois à ajouter. On préférera alors cette touche à l'utilisation des espaces
- ⋆ La touche bizarre avec un 2 : elle n'existe pas, en fait ! Oubliez-la (on ne l'utilisera pas pour calculer le carré de quelque chose).
- ⋆ La touche Alt Gr à droite de la touche d'espace est celle qui permet d'obtenir les symboles en bas à droite des touches de chiffres : par exemple, on obtient les crochets via Alt Gr-5 (ouvrant) et Alt Gr-$^\circ$ (fermant).

3. De manière générales quand vous créez des dossiers ou des fichiers essayez d'appliquer ces consignes cela évitera quelques ennuis je vous le garantis. Néanmoins sachez que vous pouvez vous en passer pour choisir le nom d'un fichier Word... ou même (pour être honnête) d'un programme Python destiné à ne plus être lu une fois le TP terminé. Mais prenez quand même tout de suite de bonnes habitudes.

III Début avec Python

3.1 Quelques caractéristiques du langage

Python est un langage de programmation, sous licence libre GPL et gratuit, créé à la toute fin des années 80 par Guido van Rossum et qui est devenu très populaire à partir des années 2000. C'est un langage bien adapté à l'apprentissage de la programmation, qui est également l'un des plus utilisés dans « l'industrie ». Il existe de très nombreux outils et environnements de développement pour Python, dont la majorité sont libres et gratuits. Ceci en fait un langage d'une très grande richesse : on peut en faire à peu près ce que l'on veut. il fonctionne sur toutes les plateformes (Windows, Linux, OSX ...).

Vous trouverez sur le web de nombreuses ressources sur Python, documentation, tutoriels ... La documentation complète est disponible sur le site officiel, à l'adresse ci-dessous :

http://www.python.org/doc/

Python est très lisible. Le programmeur doit d'ailleurs faire un effort de présentation, non seulement pour le lecteur, mais aussi pour l'interpréteur, comme nous le verrons plus tard.

Python est semi-interprété, c'est-à-dire que les instructions sont retraduites en langage machine à chaque exécution d'un programme.

Remarque : Il existe aussi des langages compilés (par exemple C) : les programmes sont traduits une fois pour toutes en langage machine dans un fichier exécutable. Il y a aussi les langages interprétés (par exemple Perl) : un interpréteur transcrit les instructions du code source en langage machine à la volée, au fur et à mesure de son exécution. Les langages compilés sont plus rapides et les langages interprétés ont une meilleure portabilité sur les différentes architectures de machines.

Une petite particularité de Python est que deux versions majeures coexistent actuellement : Python 2 (qui en est à la version 2.7) et Python 3 (version actuelle 3.6). Python 2 et Python 3 sont très similaires mais pas totalement compatibles. Vu ce que nous allons faire cette année les différences entre les deux versions sont minimes. Retenez néanmoins que certaines commandes sont différentes. Je vous donne les principales différences :

- **La division**. voilà la différence de l'utilisation du symbole de division /

Sous Python 2

```
>>>7/2
3
```

Sous Python 3

```
>>>7/2
3.5
```

- Sous Python 3, `print` est désormais une fonction [4]. Pour l'utiliser il faut donc obligatoirement utiliser des parenthèses.

Nous travaillerons exclusivement avec Python 3 cette année, mais il n'est pas impossible que vous ayez à travailler avec Python 2 ultérieurement ce qui demandera un (petit) effort d'adaptation.

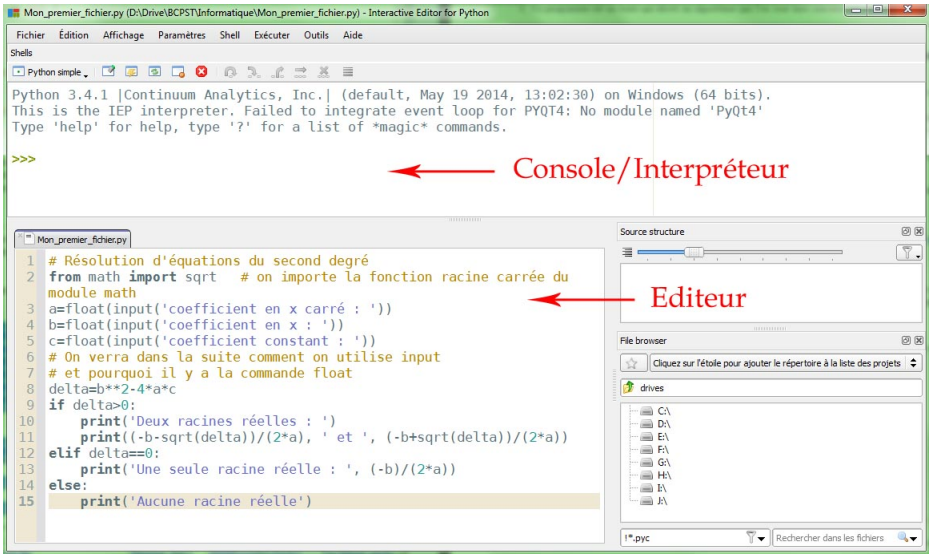
3.2 Environnement de travail

Il existe plusieurs environnements de travail pour travailler en Python. Cette année nous travaillerons avec l'IDE (Integrated Development Environments) Pyzo. Que vous pouvez librement télécharger sur vos ordinateurs à l'adresse suivante :

http://www.pyzo.org

Il vous suffit de suivre les instructions données sur le site de Pyzo. Vous téléchargez et installez la version de Pyzo qui correspond à votre système d'exploitation (Windows, Linux, OSX). Vous téléchargez ensuite et installez la distribution Python Anaconda. Installez bien Anaconda et non pas Miniconda (sauf éventuellement sur votre clef USB) car Anaconda inclut tous les modules nécessaires pour les deux années qui viennent. Et choisissez bien Python 3.

4. On verra dans un prochain chapitre de quoi il s'agit.



Remarque : Vous pourrez voir sur les ordinateurs du lycée d'autres interfaces pour travailler avec Python. On peut travailler avec la console Python et l'éditeur IDLE fourni lorsqu'on télécharge Python directement depuis l'adresse suivante : http://www.python.org. On trouvera aussi un IDE spécialement conçus pour travailler avec Python, Spyder qu'on peut trouver à l'adresse suivante http://code.google.com/p/winpython/

3.3 La console

La console permet de créer des variables, de faire des calculs, d'appeler et d'exécuter des programmes. On saisit une instruction après les trois >>> de l'invite de commande (ou `In[1]`) et on appui sur « Entrée » puis l'interpréteur l'exécute.

Exemple 1.1. Ici on se sert de Python comme d'une calculatrice.

```
>>> 3*5+7-(10+3)
9
>>> 2**3 # pour l'exponentiation (et non pas 2^3)
8
>>> 3*2**3+2 # ** est prioritaire sur *
26
>>> 2,5/3; 2.5/3
(2, 1.6666666666666667)
0.8333333333333334
>>> 5/2, 5//2, 5%2 # Division exacte, Quotient et reste de la division
    euclidienne
(2.5, 2, 1)
>>> 1.18+2.01
3.1899999999999995 # les calculs portant sur les réels ne sont pas tout le temps
    parfaits
```

Opérations algébriques	
`+ * -`	addition, multiplication, soustraction
`**`	exponentiation
`/`	division « classique »
`//`	quotient de la division euclidienne
`%`	reste de la division euclidienne

Remarques : Le séparateur décimal est le point « . ». La « , » permet de créer des n-uplets et le « ; » d'écrire plusieurs instructions sur une même ligne.

Tout ce qui se situe à droite d'un symbole « dièse » # jusqu'au changement de ligne, est ignoré par l'interpréteur. On peut ainsi insérer des **commentaires**.

Exercice 1.3 Taper dans l'interpréteur les expressions suivantes :

```
2+5;  2*5;  2**5;  2.1/3.5;  2.2/3.5;  17.0/5.0;  17 // 5;  17 / 5;  17 % 5 ;
    -3//2; 3//-2; -(3//2); (-3)%2; 3%(-2); -(3%2)
```

Il faut taper ces 15 *expressions sur* 15 *lignes différentes, et sans point-virgule. La fenêtre aura donc l'un des aspects suivants :*

```
>>> 2+5
7
>>> 2*5
10
```

```
In [1]: 2+5
Out[1]: 7
In [2]: 2*5
Out[2]: 10
```

3.4 L'éditeur

Bien sûr pour des instructions simples, comme celles qui viennent d'être faites, on utilisera directement l'interpréteur. Mais dès que nous aurons quelques lignes de commandes, on écrira ces dernières dans un éditeur puis on exécutera ensuite le tout dans l'interpréteur.

La fenêtre en bas à gauche (ou à gauche suivant les versions de Pyzo) de l'interpréteur est une fenêtre d'**édition**. Ce qu'on tape dedans est appelé à être sauvé puis exécuté. Si cette fenêtre n'est pas présente il suffit au choix de faire Ctrl+N, ou Fichier/Nouveau.

Exemple 1.2. Voici un premier exemple de script qui peut être exécuté.

```
# Résolution d'équations du second degré
from math import sqrt # on importe la fonction racine carrée du module math
a=float(input('coefficient en x carré : '))
b=float(input('coefficient en x : '))
c=float(input('coefficient constant : '))
# On verra dans la suite comment on utilise input
# et pourquoi il y a la commande float
delta=b**2-4*a*c
if delta>0:
    print('Deux racines réelles : ')
    print((-b-sqrt(delta))/(2*a), ' et ', (-b+sqrt(delta))/(2*a))
elif delta==0:
    print('Une seule racine réelle : ', (-b)/(2*a))
else:
    print('Aucune racine réelle')
```

Il est très important que vous preniez la peine de largement commenter vos propres programmes, grâce au caractère #. Sans ces remarques, un programme non trivial est très difficilement compréhensible.

Exercice 1.4 *Premières commandes*

Taper les lignes suivantes dans l'éditeur :

```
2 * 5
print(3 * 5)
4 * 5
```

Sauver immédiatement le fichier dans un endroit opportun (un des dossiers créés il y a quelques minutes par exemple), avec un nom raisonnable (qui peut être par exemple : `premieres_commandes.py`), via Ctrl-S, ou via le menu Fichier. Exécuter ensuite le fichier via F5, ou en faisant Exécuter/Exécuter le fichier. Observer et comprendre.

Lorsqu'on « quitte » Pyzo, ce qui est dans l'interpréteur est perdu à tout jamais. Par contre, ce qui a été édité et sauvé… est sauvé ! Et est donc récupérable plus tard.

Exercice 1.5

1. Dans l'éditeur, taper les lignes suivantes :

```
for i in range(2,5):
    print(i**2)
```

   Sauver, exécuter, comprendre.

2. Essayer de taper et exécuter cette boucle en restant dans l'interpréteur.

3.5 Importation de modules

Il arrive, comme on l'a vu plus haut, que l'on ait besoin d'utiliser des fonctions de Python qui ne sont pas chargées pas défaut. Il existe heureusement des **modules** ou **bibliothèques** qui répondent à la plupart des besoins habituels du programmeur. Un module est, en ce qui nous concerne, généralement un ensemble de constantes (c'est-à-dire de variables ayant une certaine valeur) et de fonctions.

Par exemple le module `math` contient les fonctions mathématiques usuelles ainsi que la constante `pi` et `e` qui ne sont pas prédéfinie en Python. Attention, vous pouvez modifier (par accident probablement) leur valeur. Le module `random` quant à lui contient plusieurs types de générateurs de nombres pseudo-aléatoires. Enfin on rencontrera cette année le module `matplotlib` qui sert notamment aux représentations graphiques, et le module `PIL` dont on se servira pour le traitement d'images.

Voici les lignes de code que nous rencontrerons le plus souvent pour importer des fonctions des différents modules.

```
>>> from math import * # On importe tout ce que contient le module math
>>> from math import sqrt # On importe la fonction racine
>>> sqrt(4)
2.0
>>> from math import exp, log # On peut importer plusieurs fonctions à la fois
>>> exp(0)
1.0
>>> from random import random
>>> random() # la function random génère un réel aléatoire dans [0,1[
0.10141832104831405
```

Pour compléter voici d'autres façons de procéder.

```
>>> import math # import du module de mathématique (création d'un point d'accès)
# pour utiliser la fonction sqrt, on la préfixe du nom du module et d'un point
>>> math.sqrt(4) # un peu lourd d'utilisation
2.0
>>> import math as m # est une autre solution on écrit alors
>>> m.sqrt(4)
2.0
>>> # Enfin on peut importer une fonction en la renommant
>>> from math import sqrt as rac
```

IV Notion de variables et de types

4.1 Variables et expressions

4.1.1 Variables

Une **variable** est une « boîte » qui porte un nom et permettant de stocker des données dans la mémoire vive de l'ordinateur. (elle disparait si on éteint l'ordinateur)

Une variable se crée par une **affectation**, par exemple `age=25` affecte la valeur 25 à la variable `age`.

Remarque : Le symbole `=` en informatique n'a pas le même sens qu'en mathématiques. Il n'est par exemple pas symétrique ($a = b$ n'a pas le même sens que $b = a$). C'est pour cette raison que l'on écrit souvent dans du pseudo-code $a \leftarrow 5$ au lieu de $a = 5$.

Choix du nom d'une variable

Il est très important de donner des noms de variables qui évoquent leur contenu, comme ci-dessus. Un nom de variable commence typiquement par une lettre, ne contient pas d'espace, mais peut comporter des chiffres et le caractère `_` (underscore) pour séparer deux mots. Il y a des conventions largement acceptées concernant le nom des variables, par exemple toujours commencer par une minuscule, comme ci-dessus, pour des raisons de lisibilité. Par exemple `age_capitaine` ou `ageCapitaine` sont des noms de variables bien choisis (pour désigner l'âge du capitaine en tout cas). Python est sensible à la casse, `age` n'est pas `Age` ou `AGE` par exemple. Évitez quand même les majuscules.

Remarque : Certains noms de variables sont interdits, car ce sont des mots réservés à Python, comme : `and, as, break, continue, del, def, else, from, global, of, return` ...

Enfin il est possible de supprimer une variable avec la primitive `del`. On le fait de la manière suivante :

```
>>> del a  # si a est la variable que l'on veut supprimer
```

4.1.2 Notion d'expression

En manipulant des valeurs ou des variables à l'aide d'opérateurs, on obtient des **expressions**, qui ont une valeur, qui peut être entière, réelle, booléenne ... Une expression est toujours évaluée par l'interpréteur, à des fins d'affectation, de test ou de simple calcul.

Exercice 1.6 Quels sont les résultats obtenus ?

```
>>> a=3 ; b=4
>>> a+1
.......
>>> (2**a+12)/b
.......
```

4.1.3 Affectation

Une variable peut changer de valeur par exemple :

```
>>> a=851
>>> a=a+1
>>> a
852
```

 Il faut comprendre $a = a + 1$ comme $a \leftarrow a + 1$. Bien entendu, il n'y a aucun rapport avec l'équation $a = a + 1$ en mathématiques qui n'a aucune solution !

Lors d'une affectation `var = expr` il y a **évaluation** de l'expression `expr` puis **affectation** du résultat à la variable `var`.

 À gauche du symbole = on doit donc toujours avoir le nom d'une variable et à droite une expression. Par exemple, les écritures suivantes n'ont pas de sens : `a-1=3` `x**2=4`.

```
>>> 852 = a
SyntaxError: can't assign to literal
```

Exercice 1.7

1. Dans l'éditeur importer depuis le module `math` la variable `pi` et la fonction `floor`
2. Toujours dans l'éditeur, affecter aux trois variables `a`, `b` et `c` les valeurs 5, 3^4 et 2π, et exécuter les deux lignes que vous venez d'écrire.
3. Dans l'interpréteur maintenant, effectuer les calculs suivants : `a+b, a*c/b, (a-b)/c`.
4. Prévoir le résultat obtenu si l'on exécute 4 fois la commande `a=a+2`. Vérifier en tapant quatre fois `a=a+2` dans l'éditeur en allant à la ligne à chaque étape, puis an tapant simplement `a`.
5. Dans l'interpréteur
   a. Affecter à la variable `d`, la valeur `c/4`. Combien vaut `d` ?
   b. Taper alors les commandes suivantes : `int(d),round(d),floor(d)`
   c. Taper les commandes suivantes : `int(-d),round(-d),floor(-d)`
   d. Que font les fonctions `int()`, `round()` et `floor()` ?

Exercice 1.8 *Dans l'éditeur*

Taper les lignes suivantes. À l'exécution, que va-t-il se passer ? Prédire, puis vérifier.

```
1 x = 10
2 y = 15
3 z = x + y
4 x = y
5 y = z
6 print(x + y + z)
```

Quelques raccourcis Python

```
>>> x = 12
>>> x += 3 # à la place de x=x+3
>>> x
15
```

```
>>> x *= 2 # à la place de x=2*x
>>> x
30
```

On a les mêmes possibilités avec les opérations : `-, \, \\`.

Exemple 1.3. Affectations simultanées et identiques :

```
>>> a = b = 5
>>> a, b
(5, 5)
# il s'agit d'un couple (parenthè
    ses facultatives)
```

Exemple 1.4. Affectations simultanées :

```
>>> (u,v)= (9,7)
>>> u
9
>>> v
7
>>> u,v = 7,1
```

Exercice 1.9 Prévoir le résultat des instructions suivantes :

```
>>> a = 100
>>> b = 17
>>> c = a - b
>>> a = 2
>>> c = a + b
>>> (a, b, c)
```

```
>>> a = 3
>>> b = 4
>>> c = a
>>> a = b
>>> b = c
>>> (a, b, c)
```

```
>>> x = 19
>>> x = x + 2; y = x * 2
>>> (x, y)
```

```
>>> x = 19
>>> (x, y) = (x + 2, x * 2)
>>> (x, y)
```

4.2 Types de variables

Python est typé, toutes les variables ont un type (ou classe). Il s'agit d'un **typage dynamique**, dans la mesure où le type d'une variable est déterminé par Python seulement quand cette variable reçoit une valeur et peut changer en cas de réaffectation. Autrement dit les variables ne sont pas déclarées a priori comme dans d'autres langages. Cependant c'est un typage fort, par exemple on ne peut additionner une variable de type `float` et une variable de type `string`.

Remarque : Dans certains langages (C, Java ...), le type d'une variable doit être déclaré avant son affectation et il ne peut pas changer, on parle de **typage statique**.

Voici une liste de quelques types utilisés par Python

type Python	« traduction »	exemple
`int`	entier	-852
`float`	flottant (décimal)	3.1415
`bool`	booléen (`True` ou `False`)	`True`
`str`	chaîne de caractères (string)	`"Lycée du Parc"`
`list`	liste	`[9, 8, 3.14, "Pi"]`
`function`	fonction	`cos`
`complex`	nombre complexe	`1+2j` ($i.e.\ 1+2i$)
`tuple`	n-uplet	`(8, 3, 3)`

Remarque :

- ⋆ le **type entier** `int`, réservé à une variable dont la valeur est un entier relatif stocké en valeur exacte. Il n'y a pas de limite de taille.
- ⋆ le **type flottant** `float` pour un nombre à virgule, comme `3.14`, stocké en valeur approchée sous la forme d'un triplet (s, m, e) sur 64 bits : 1 bit de signe s, 11 bits d'exposant e, 52 bits de mantisse m.

- ⋆ le type `string` pour les **chaînes de caractères** notées entre quotes simples (apostrophes) ou doubles (guillemets). Dans les chaînes entre triples quotes, on peut faire des sauts de ligne.
- ⋆ le **type liste** `list` : une liste peut être vue comme un tableau linéaire, chaque case étant indexée à partir de 0 et pouvant contenir une valeur de n'importe quel type, par exemple une liste ! Une liste est mutable, c'est-à-dire modifiable.

Nous reparlerons spécifiquement des chaînes de caractères et des listes dans de prochains chapitres.

Les primitives (ou fonctions) `int(), float(), bool(), str()` permettent de modifier le type d'une variable.

```
>>> a=4.5
>>> int(a)
4
>>> float(12)
12.
>>> str(4*5)
'20'
```

Exemple 1.5.

```
>>> a = 12
>>> a, type(a)    # Valeur de a et Type de a
(12, <class 'int'>)
>>> reel = 1/3; mot = 'BCPST'
>>> type(reel), type(mot)
(<class 'float'>, <class 'str'>)
>>> a>5           # cette expression est évaluée comme vraie ou fausse
True
>>> test = 1>0;
>>> type(test),
<class 'bool'>
>>> reel+mot
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unsupported operand type(s) for +: 'float' and 'str'
# on ne peut pas faire n'importe quoi avec des variables de types différents
>>> lis=['poires',4,'pommes',5]
>>> type(lis)
<class 'list'>
>>> lis[2],type(lis[2])
('pommes', <class 'str'>)
```

De nombreux opérateurs sont **polymorphes**, c'est-à-dire qu'ils peuvent être appliqués à des objets de différents types, par exemple et cela vous semblera naturel on peut appliquer `+, *, -, /` aussi bien a des flottants que des entiers.

Exemple 1.6. Addition ou concaténation

```
>>> 5 + 8
13
>>> 5.1 + 8.2
13.299999999999999
>>> "BCPST" + " 852"      # concaténation de deux chaînes
'BCPST 852'
>>> [8,5,2] + [1,9,9,8]   # concaténation de deux listes
[8,5,2,1,9,9,8]
```

A propos du type `bool`

Il ne contient que deux valeurs, `True` et `False` et on peut leur appliquer les opérateurs usuels de la logique : `and, or, not`.

Principaux opérateurs sur des expressions booléennes	
`E and F`	Vraie si E est Vraie ET F est Vraie
`E or F`	Vraie si E est Vraie OU F est Vraie
`not E`	Vraie si E est Fausse

Les symboles de comparaison (polymorphes) sont les suivants :

Principaux opérateurs de comparaison de variables	
`x == y`	x est égal à y
`x != y`	x est différent de y
`x > y`	x est strictement supérieur à y
`x < y`	x est strictement inférieur à y
`x >= y`	x est supérieur ou égal à y
`x <= y`	x est inférieur ou égal à y
`x is y`	id(x) == id(y)

et ils sont évalués en booléens. Notons que l'on dispose en Python d'un raccourci agréable : `x < y < z` au lieu de `x < y and y < z`.

Exemple 1.7.

```
>>> x = 9
>>> x > 8
True
>>> 10 > 8 > 9
False
>>> "BCPST" > "HEC"
False
```

Ne pas confondre la comparaison `==` et l'affectation `=`. L'affectation est une instruction qui est exécutée, la comparaison est une expression qui est évaluée en une valeur égale à `True` ou `False`.

Exemple 1.8.

```
>>> (a, b) = (1, 2)
>>> a == 1 or b == 3
True
>>> (not a > 1) and (b != 4)
True
```

Exercice 1.10 Prévoir pour chacune des lignes suivantes la valeur et les types des expressions données

```
>>> a,b=3,4
>>> a+1
>>> (2**a+12)/b
>>> (a==2) or (a!=2 and b>6)
>>> "j'ai "+str(b**a)+" ans"
```

V Entrées sorties

5.1 Affichage non graphique

La fonction `print` est souple et suffira dans la plupart des cas. On peut afficher plusieurs valeurs successivement, même si elles sont de types différents, comme dans l'exemple déjà vu :

```
print('Une seule racine réelle : ', (-b)/(2*a))
```

À l'affichage, par défaut les variables sont séparées par des espaces et `print` passe automatiquement à la ligne, autrement dit lors du prochain `print`, l'affichage se fera au-dessous du précédent. Mais tout ceci est modifiable. En tapant `help(print)` dans l'interpréteur vous verrez une description de la fonction.

Exemple 1.9.

```
>>> a,b,c=1,2,3
>>> print(a,b,c,sep=":") # affiche
1:2:3
>>> print(a,b,sep="+",end="=");print(c)
1+2=3
```

```
>>> print(a,b,c,sep="\n") # le \n signifie new line
1
2
3
>>> print("Il a {} euros dans sa poche".format(b)) # le .format(b) permet d'insé
    rer la valeur de la variable b dans la chaîne de caractères à la place {}.
Il a 2 euros dans sa poche
```

5.2 Entrée au clavier

Comme on l'a vu dans l'exemple 1.2., la primitive `input` permet de capturer une entrée clavier. `input` renvoie une chaîne de caractère (`string`). C'est pour cela que dans l'exemple on a :

```
b=float(input('coefficient en x: '))
```

la fonction float transformera la chaine de caractère en un flottant. Par exemple En tapant 3, l'utilisateur affecte la valeur 3 à la variable `b` qui sera de type flottant (`float`).

VI D'autres exercices

Exercice 1.11

1. En important, si cela n'a pas déjà été fait, les fonctions nécessaires du module `math` écrire un programme demandant un réel x et renvoyant la valeur de :
$$x^5 e^{\pi \lfloor x \rfloor}$$
2. Écrire un programme qui renvoie le quotient et le reste de la division d'un nombre a par un nombre b.
3. Écrire un programme qui demande à l'utilisateur de donner trois notes et qui renvoie la moyenne de ces trois notes.

Exercice 1.12

1. Écrire un programme qui demande à l'utilisateur de donner un prix TTC et qui renvoie le montant de la TVA ainsi que le montant HT, dans le cas d'une TVA à 19.6%.
2. Écrire un programme qui prend comme paramètres un prix HT et une TVA (5.5%, 19.6%, ...) et qui renvoie le montant TTC.

Exercice 1.13 Écrire un programme qui :

- demande à l'utilisateur de rentrer une valeur que l'on stocke dans une variable notée `a`.
- demande à l'utilisateur de rentrer une autre valeur que l'on stocke dans une variable notée `b`.
- échange le contenu des variables `a` et `b`.

Exercice 1.14 En utilisant l'aide en ligne [5], écrire un programme prenant en argument un nombre complexe écrit sous forme algébrique et qui renvoie son module et son argument.
Il vous faudra donc voir comment on écrit un nombre complexe (qui est un autre type de Python) et essayez de trouver des fonctions permettant de calculer module et argument.

5. http://www.python.org/doc/ ou encore http://docs.python.org/3/