Feuille de cours 1 (informatique) : Variables et fonctions

On peut utiliser Python comme une calculatrice standard, et obtenir le résultat d'une opération comme $(1+2)\times 3$ en utilisant les symboles appropriés pour la somme, la multiplication, etc. Mais la puissance du langage de programmation est d'autoriser à utiliser la mémoire de l'ordinateur d'une façon plus complexe en utilisant des **variables**. Une variable est un emplacement mémoire auquel on attribue un nom et dans lequel est stockée une valeur.

1 Affectation de variables

1.1 Affectation d'une seule variable

L'affectation d'une variable à une valeur se fait avec le signe =. Il prend alors le sens du = en mathématique lorsque celui-ci exprime une définition comme lorsqu'on dit : "soit $\Delta = b^2 - 4ac$ le discriminant du polynôme $P(X) = aX^2 + bX + c$".

Pour définir une variable `ma_variable` et lui attribuer la valeur `ma_valeur` on écrit :

```
1 ma_variable = ma_valeur
```

On pourra par exemple écrire :

```
1 x = 1
```

Attention, la variable à définir est donc placée systématiquement **à gauche** du signe égal. Contrairement à ce dont on a l'habitude en mathématique, le symbole = de l'affectation n'est pas symétrique. Par exemple, l'instruction

```
1 1 = y
```

renvoie un message d'erreur car `1` est une constante à laquelle on ne peut pas affecter de valeur.

Une fois qu'une variable est définie, on peut la réutiliser au sein d'autres calculs. Par exemple, le calcul suivant renvoie la valeur

```
1 z = 1
2 z + 2
```

Les affectations successives **écrasent** les valeurs des variables, c'est-à-dire que si on réaffecte une variable déjà affectée à une nouvelle valeur alors l'ancienne valeur est perdue. Par exemple, une fois le code ci-dessous exécuté, Python "ne se souvient pas" que `a` a eu la valeur `1` avant d'avoir la valeur `2`.

```
1 a = 1
2 a = 2
3 a
4 >>> 2
```

Enfin, une fois qu'une variable a été définie, on peut modifier sa valeur en utilisant sa valeur actuelle. Le nom de la variable apparaît alors des deux côtés du signe = : à gauche car on affecte une nouvelle valeur à cette variable, et à droite en tant que référence à l'ancienne valeur. Par exemple, à la fin de l'exécution du script suivant, la variable `b` a la valeur

```
1 b = 1
2 b = b + 1
```

Attention, dès qu'une expression fait référence à une variable, celle-ci doit avoir un sens pour Python. Il faut donc que cette variable soit apparue dans le code **avant** de l'utiliser : soit car on lui affecté une valeur, soit car c'est une variable prise en argument par une fonction (voir ci-dessous).

Par exemple, si je réinitialise le shell (pour cela, utiliser l'icône à la flèche verte dans le menu du shell) et que je demande à Python la valeur de `b+2` alors

En cas de doute, on peut afficher la liste des variables actuellement en mémoire : dans l'onglet "Outils", sélectionner "Environnement".

Pour conclure, voici quelques règles et conseils à suivre concernant les noms de variables. Les noms que vous choisissez pour vos variables sont libres, mais vous devez respecter les règles suivantes :

- Le nom d'une variable est constitué de lettres, de chiffres et de tirets du 8 (underscore).
- Il ne contient ni espace, ni parenthèses, ni point, ni tiret du 6, ni accents.
- Il ne commence pas par un chiffre.
- Pyhton distingue les majuscules des minuscules.
- Un certain nombre de mots sont réservés et ne peuvent pas devenir le nom d'une variable, vous ne pouvez par exemple pas utiliser (cette liste est loin d'être exhaustive) : les constantes (comme `1`), `def`, `for`, `global`, `import`, `lambda`, `return`...

Enfin, il est très important d'accompagner la lecture du code en choisissant des noms de variables appropriés : ils doivent être *courts* (pour ne pas se tromper en les recopiant) et *évocateurs*. Utilisez par exemple `compt` pour un compteur, `som` pour une somme, `res` pour le résultat, etc.

Exercice 1
Que renvoie Python à l'exécution des codes suivants ?

```
1 x = 2
2 x = x - 1
3 x = x + 1
4 x
```

```
1 x = 3
2 y = 2
3 x = y
4 x + y
```

```
1 x = 1
2 y = 2
3 1 + x = y
4 x
```

1.2 Affectations multiples, échanges de valeurs

Il est possible de réaliser en parallèle l'affectation de plusieurs variables. Pour cela, on sépare les variables et les valeurs qu'elles doivent prendre par des virgules :

```
var1, var2 = valeur1, valeur2
```

Par exemple, les deux codes suivants sont équivalents :

```
x = 1
y = 2
```

En fait, on peut séparer plusieurs variables par des virgules pour obtenir une variable d'un nouveau type appelé *tuple* et qui correspond au n-uplet mathématique. Là où on écrirait (x_1, x_2, x_3) en mathématique, on écrit `x_1,x_2,x_3` en Python.

Exercice 2
Un exercice classique consiste à échanger les valeurs de deux variables `A` et `B`. Que se passe-t-il si on fait le code ci-dessous ? Proposer deux méthodes pour résoudre le problème.

```
A = 1
B = 2
A = B
B = A
```

2 Fonctions

2.1 Opérations usuelles

Lorsqu'on a défini des variables Python (par exemple `x` et `y`) on peut les combiner pour former des expressions mathématiques (comme par exemple $\frac{x^2}{2x+y}$). On utilise pour cela les opérations de base suivantes :

- `+` : somme
- `-` : différence
- `*` : multiplication
- `**` : puissance
- `//` : quotient de la division euclidienne
- `%` : reste de la division euclidienne
- `/` : division

Exercice 3
Écrire des expressions en Python correspondant au expressions mathématiques suivantes :

1. $2xy + z^2$
2. $\frac{3x}{5y^2}$
3. $\frac{x}{x^y + 1}$
4. $\frac{\sqrt{x}+1}{\frac{3}{x}}$

Les opérations usuelles que nous venons de citer sont des fonctions de base, déjà présentes dans Python par défaut. Mais le langage nous permet aussi de définir nos propres fonctions. On peut alors effectuer toute une séquence d'instructions choisies en appelant seulement le nom de cette fonction.

2.2 Syntaxe

Pour définir une fonction voici comment procéder :

```
def nom_fonction(arguments):
    instruction_1
    instruction_2
    return resultat
```

Faisons quelques remarques :

- Le nom de la fonction, ici `nom_fonction`, doit respecter les mêmes règles que les noms de variables.
- Les arguments sont mis entre parenthèses, mais il n'y a pas de parenthèses après `return`.
- Les **deux points** à la fin de la première ligne ne sont pas optionnels.
- **L'identation**, c'est-à-dire le retrait par rapport au bord gauche, n'est pas optionnel non plus. C'est lui qui permet à Python de distinguer ce qui fait partie de la fonction de ce qui n'en fait pas partie. Il n'y a pas de séparateur pour signaler la fin de la fonction, c'est la fin de l'indentation qui joue ce rôle.

Voici un premier exemple de fonction simple :

```
1 def double(x):
2     return 2*x
```

La variable x a-t-elle besoin d'être définie préalablement pour que la fonction soit correctement définie ?

Comment faire pour tester si cette fonction renvoie bien le résultat attendu ?

Une fonction peut avoir plusieurs arguments, c'est-à-dire plusieurs paramètres (ou variables) dont elle dépend. On les sépare alors par des virgules. En fait une fonction prenant plusieurs arguments n'est rien d'autre qu'une fonction prenant un seul argument qui est un

Contrairement aux fonctions mathématiques, les fonctions Python peuvent aussi n'avoir aucun argument, il faut alors placer des parenthèses vides. Enfin, une fonction peut renvoyer plusieurs résultats, il faut alors les séparer par des virgules. Par exemple :

```
1 def somme(x,y):
2     return x+y
3
4 def constante():
5     return 1
6
7 def simple_et_double(x):
8     return x,2*x
```

Exercice 4
Écrire des fonctions Python correspondant aux fonctions mathématiques suivantes :

1. $\begin{array}{rcccc} f_1 & : & \mathbb{R} & \longrightarrow & \mathbb{R} \\ & : & x & \longmapsto & x^2 \end{array}$

2. $\begin{array}{rcccc} f_2 & : & \mathbb{Z}\times\mathbb{R} & \longrightarrow & \mathbb{R} \\ & : & (n,x) & \longmapsto & (x+1)^{n+2} \end{array}$

3. $\begin{array}{rcccc} f_3 & : & \mathbb{R}\times\mathbb{N} & \longrightarrow & \mathbb{R}\times\mathbb{R} \\ & : & (x,n) & \longmapsto & \left(\dfrac{x}{n+1}, n^{x+1}\right) \end{array}$

4. $\begin{array}{rcccc} f_4 & : & \mathbb{R}\times\mathbb{R} & \longrightarrow & \mathbb{R}\times\mathbb{R}\times\mathbb{R} \\ & : & (x,y) & \longmapsto & (1+x^2, 1+2x^2, y) \end{array}$

Terminons avec quelques remarques et conseils pour les fonctions :

- pour bien se fixer les idées dans un calcul compliqué, il peut être utile de définir une variable `res` qui contiendra le résultat de la fonction. Par exemple, pour définir la fonction $f : x \longmapsto -x^3 + 36x^2 + \sqrt{2}\dfrac{x}{3}$ on peut écrire

```
def f(x):
    res1 = -x**3
    res2 = 36*x**2
    res3 =
    res =
    return res
```

- sauf cas particulier, une fonction ne modifie pas les variables qu'elle prend en arguments. Ainsi pour écrire la fonction $f : x \longmapsto x + 1$ on écrira :

```
def f(x):
    res = x+1
    return res
```

et non

```
def f(x):
    x = x+1
    return x
```

De plus, si on veut évaluer cette fonction en 3 (c'est-à-dire calculer $f(3)$) alors on n'inscrira pas la valeur 3 dans l'éditeur. Il faut au contraire **aller dans la console** pour tester `f(3)`. Ainsi, il est hors de question d'écrire :

```
def f(x):
    x = 3
    return x+1
```

À savoir pour le prochain TP :

1. connaître la différence entre l'éditeur (script) et la console (shell)
2. savoir affecter une valeur à une variable
3. savoir échanger les valeurs de deux variables
4. savoir définir une fonction