Initiation interactive à Python 1 : Présentation, instructions

Ce document est interactif. Vous pouvez tester et modifier les programmes directement dans ces pages. Il fonctionne avec l'aide de, Javascript, JQuery, l'éditeur Ace, et surtout Brython (http://www.brython.info). L'emploi de ces techniques ne donne qu'un sous ensemble de ce qu'on peut faire réellement avec Python 3. Néanmoins, aucune instruction ou module ne devrait faire défaut dans ce tutoriel. Si vous utilisez un logiciel de téléchargement adéquat, vous pouvez même tout enregistrer en local sur votre machine, et travailler sur ces document sans connexion internet.

Il n'y a pas de niveau minimum requis pour lire ce qui suit. Néanmoins, certaines explications et certaines mises au point peuvent paraître trop *évidentes* à certains, qui les connaissent déjà. Elles sont indiquées par des blocs de ce type :

Un ordinateur est une machine qui peut effectuer automatiquement les choses pour lesquelles on l'a programmée. Le fait de pouvoir modifier à volonté ce programme est essentiel pour qu'une machine automatique soit appelée ordinateur.

Écrire des instructions

Un *programme informatique* est un *algorithme* écrit dans un langage informatique particulier. Mais qu'est-ce qu'un algorithme ? C'est une *recette*, une *méthode*, une suite d'opérations, qu'on peut effectuer de manière complètement mécanique et automatique, sans avoir recours à l'intelligence, afin de réaliser une tâche particulière. Il existe des algorithmes pour toutes sortes de choses : pour poser et calculer le résultat d'une addition, pour partager des pièces de monnaie entre plusieurs personnes de manière équitable sans pour autant les compter ...

L'algorithme le plus connu est sans doute celui qu'on utilise lorsqu'on pose une addition. Il vous a probablement été enseigné à l'école primaire, et comme nous ne sommes pas des machines, il vous a sans doute été présenté de manière assez peu formelle, avec beaucoup d'explications et d'exemples, ce qui est sûrement la bonne méthode pour enseigner des algorithmes à des êtres humains.

On pourrait toutefois résumer la méthode d'addition de deux nombres entiers de manière un peu formelle :

- écrire les deux nombres entiers l'un sous l'autre en alignant les chiffres de droite
- répéter pour chaque colonne en partant de la droite
    - ajouter les deux chiffres de la colonne avec la retenue éventuelle
    - si le résultat est strictement inférieur à 10 :
        - la retenue passe à 0
        - écrire le chiffre résultat en dessous de la colonne

    - sinon :
        - la retenue passe à 1
        - écrire le chiffre des unités du résultat en dessous de la colonne

Bien sûr on pourrait discuter du degré de précision à utiliser dans la description de cet algorithme, de l'ambiguïté presque inévitable lorsqu'on rédige un texte en français, ou du fait qu'on peut l'écrire d'autres manières... mais l'idée est là : **ce qui précède est un algorithme**. Si on l'écrit dans un langage informatique particulier, ça deviendra un programme. Lorsque le programme sera fait, la machine saura faire la tâche (additionner deux nombres) dont la résolution est décrite par l'algorithme.

Dans ce tutoriel, nous utiliserons le langage Python. Il fait partie des plus simples à apprendre, il est très répandu, et il permet de programmer tout ce qu'on peut vouloir programmer sur une machine. Comme c'est le cas pour beaucoup de langages informatiques, les mots-clés sont en anglais. Mais comme il y en a peu à retenir, ça ne devrait pas être trop gênant.

Savoir programmer, c'est donc : savoir concevoir des algorithmes (des méthodes de résolution de problèmes), *puis* traduire ces méthodes dans un langage particulier, de la manière la plus efficace / concise / élégante possible (sachant qu'on est souvent obligé de sacrifier certains des adjectifs qui précèdent au profit des autres...).

L'interpréteur Python

Écrire un programme dans un langage donné (comme Python) n'est pas suffisant pour que la machine nous obéisse. En effet, celle-ci ne comprend pas le langage Python... mais uniquement un langage de très bas niveau, appelé langage machine, difficilement (mais ce n'est pas impossible) accessible à l'être humain. Les machines modernes disposent donc de programmes qui font l'intermédiaire avec l'homme.

Essentiellement, et pour simplifier, ces intermédiaires sont de deux types : les *compilateurs* et les *interpréteurs*. Dans le premier cas, le compilateur transforme notre programme en langage machine, puis la machine l'exécute. Dans le second cas, l'interpréteur lit notre programme et le fait exécuter au fur et à mesure par la machine. Ce n'est pas une règle absolue, mais généralement, un langage donné est soit interprêté, soit compilé.

Le langage Python est *interprêté*[1)].

Dans ce document, vous pouvez exécuter les programmes proposés et voir ce qu'ils font en cliquant sur le bouton **Exécuter**. Toutes les zones interactives ont la même forme :

- une partie (généralement sur la gauche, sur fond noir) qui permet d'entrer un programme. Les lignes sont numérotées : c'est la *zone d'édition*.
- une partie (généralement sur la droite, sur fond gris), qui permet de voir le résultat de

l'exécution du programme (essentiellement, on voit ce que le programme affiche) : c'est la *zone d'exécution*.
- une zone de boutons, dans la partie supérieure, contenant un bouton **Exécuter** qui permet... d'exécuter le programme entré dans la zone d'édition.

Essayez le programme qui suit en cliquant sur le bouton **Exécuter le programme** :

Si tout se passe bien, voici ce que vous devriez voir apparaître dans la zone d'exécution :

```
Bonjour
Je suis l'interpréteur Python
Je sais faire des opérations :
6 fois 7 égale 42
```

Jetons un oeil au programme dans la zone d'édition... Il contient 4 lignes sur le même modèle :

```
print(QUELQUECHOSE)
```

le mot `print` fait partie du langage Python. Il indique que nous voulons *afficher*[2] quelque chose. Pour indiquer ce que nous souhaitons afficher, nous le mentionnons dans les parenthèses qui suivent immédiatement le mot `print`. Cette notation : *une fonction suivie de choses placées entre parenthèses* est très courante. Elle indique ici qu'on fait *appel* à la *fonction* `print` et qu'on lui passe en *paramètres* ce qui est dans les parenthèses. Les trois premiers `print` contiennent comme paramètre du texte placé entre guillemets. Un tel texte s'appelle une *chaîne de caractères*. Il est simplement affiché tel quel dans la zone d'exécution.

Le dernier `print` est un peu plus compliqué :

```
print("6 fois 7 égale", 6 * 7)
```

Il faut le décomposer ainsi :

```
print( PARAMETRE 1 , PARAMETRE 2 )
```

En effet, il est possible de donner plusieurs paramètres à une fonction en les séparant par des virgules. Dans le cas de la fonction `print`, elle affichera chacun de ses paramètres, séparés par un espace. Le premier paramètre est **entre guillemets** : “6 `fois 7 égale`” ⇒ **c'est une chaîne de caractères, qui est affichée telle qu'elle**. Le second paramètre n'est pas indiqué entre guillemets : `6 * 7`. C'est une *expression arithmétique*. Si nous regardons ce qui s'affiche dans la partie droite de la fenêtre lorsqu'on exécute le programme, nous comprenons ce qui se passe : l'expression arithmétique est *évaluée* (c'est à dire que Python calcule combien elle vaut) et c'est le *résultat* de cette évaluation (42) qui est affiché par la fonction `print`.

Modifier un programme

Il est possible de modifier le programme proposé et de voir quelles sont les conséquences de ces modifications. C'est une **très** bonne pratique, qui permet de comprendre beaucoup de choses en programmation. Malheureusement, *modifier un programme*, surtout au début, c'est parfois difficile : la moindre faute aura pour conséquence que le programme ne marchera plus du tout. C'est parfois... énervant. Mais il faut persévérer.

Il y a plusieurs types d'erreurs possibles dans un programme. Les plus courantes sont les erreurs de *syntaxe* ou assimilées (dans le cadre de ce document). Par exemple, oublier de refermer une parenthèse, ne pas mettre les guillemets où il faut, remplacer la virgule par un point virgule, ou écrire `pront` au lieu de `print` sont des erreurs de syntaxe[3].

Imaginons que vous écriviez un algorithme en français, et qu'il est destiné à un humain et non à une machine. Par exemple :

```
Écrire "Bonjour"
```

Si vous vous trompez et indiquez :

```
Écire "Bonjour"
```

ou encore :

```
Écrire "Bonjour
```

ça n'aurait probablement pas de grandes conséquences (peut être même que vous devez vous y reprendre à deux fois pour trouver ce qui cloche...). La personne qui lit l'algorithme comprendra tout de même. Pour la machine c'est très différent : une toute petite erreur, et elle ne comprend plus.

Lorsqu'il y a une erreur dans le programme entré dans la zone d'édition et qu'on l'exécute, un *message d'erreur* s'affiche dans la zone d'exécution. Ces messages ne sont pas toujours très simples à comprendre, et c'est à force de les voir qu'on corrigera rapidement ses erreurs. Voici un exemple de message d'erreur :

```
IndentationError: unexpected indent
module '__main__',exec' line 2
 print("Je suis l'interpréteur Python")
^
```

Le message qui précède indique que le problème est situé juste avant `print` de la ligne 2. Le type d'erreur est `IndentationError`... nous allons y revenir.

Voici les points à vérifier en priorité pour éviter les erreurs :

- tous les symboles qui marchent par paire (crochets, parenthèses, guillemets) doivent être correctement appariés et imbriqués
- le symbole qui sépare les différents paramètres d'une fonction est la virgule
- du texte *littéral* doit être entre guillemets (“ ou ' c'est pareil en Python, mais si on ouvre les guillemets avec un ”, il faut aussi les refermer avec “)
- le séparateur décimal (comme dans trois virgule cinq : `3.5`) est le point `.` et non la virgule
- les lignes de code, si elles font partie d'un même bloc, ont la même *indentation*

Le dernier point mérite quelques détails, d'autant plus qu'il est *spécifique* au langage Python alors que le reste est assez général. Nous verrons par la suite qu'il est parfois nécessaire de grouper les instructions données à la machine en paquets (on parle de *blocs*). L'interpréteur Python, qui relit notre programme, **doit** disposer d'un moyen pour identifier ces blocs : savoir où ils commencent, où ils finissent et ce qu'ils contiennent. **En Python, les blocs sont délimités par l'indentation** c'est à dire le décalage horizontal de la ligne. C'est assez inhabituel [4)]. Généralement, les blocs sont délimités par des symboles (accolades en C) ou des mots clés (`begin` et `end` en Pascal).

Dans l'exemple qui suit, il n'y a pas de bloc. Donc, toutes les lignes doivent être **collées à la marge de gauche** (pas d'indentation). L'ajout d'un simple espace en début de ligne *indentera* cette ligne, signifiant ainsi à Python qu'elle fait partie d'un autre bloc. Si l'interpréteur n'attend pas de bloc à cet endroit précis, il dira donc qu'il y a une erreur.

Maintenant que vous êtes mis en garde, n'hésitez pas à faire quelques tests dans la zone d'édition suivante :

Ne faites qu'une seule modification à la fois, afin de mieux en mesurer les effets. À tout moment, vous pouvez remettre le programme dans son état d'origine en cliquant sur le bouton *Remettre la page à l'état initial*.

Vous pouvez par exemple essayer les choses suivantes :

- voir ce qu'est un message d'erreur :
  - en ajoutant un espace au début de la ligne 2
  - ou en effaçant la parenthèse fermante de la ligne 2
  - ou en effaçant la virgule de la dernière ligne
  - ou en écrivant `pront` au lieu de `print` sur la ligne 4[5)].
- différencier une expression arithmétique d'un chaîne de caractères en
  - modifiant la dernière ligne en :

```
print("6 fois 7 égale ","6 * 7")
```

- ne pas hésiter à expérimenter :
  - peut être qu'on peut faire aussi des additions ?
  - peut-être qu'on peut afficher à la fois le produit et la somme des deux nombres sur la même ligne ?

Lire la suite : Aller vers Initiation interactive à Python 2 : Calculatrice, variables, IO

1)
encore qu'en réalité, la situation est un peu plus compliquée, car python dispose d'une *machine virtuelle*, un intermédiaire supplémentaire entre l'homme et la machine.

2)
imprimer se dit *print* en anglais

3)

Le dernier exemple d'erreur est un peu différent, mais pour le moment, ce n'est pas trop grave de considérer qu'il s'agit aussi d'une erreur de syntaxe.

4)

Il est très habituel d'indenter correctement du code, mais uniquement pour qu'il soit plus facile à lire par un humain, pas pour faciliter la compréhension par la machine. Cela dit, le créateur de Python, Guido van Rossum, a fait d'une pierre deux coups en utilisant des éléments de présentation comme éléments syntaxiques.

5)

Notez que dans ce cas, les trois premiers print seront quand même exécutés. C'est pour celà que remplacer `print` par `pront` n'est pas une *vraie* erreur de syntaxe ; elle n'est pas détectée en *première lecture* par l'interpréteur, mais plus tard, au moment où il essaie d'exécuter l'ordre donné. Les erreurs de syntaxe sont recherchées *avant même* que l'interpréteur n'exécute la première ligne de code