Chapitre 1

Programmation en Python

Pourquoi choisir le langage Python dans l'enseignement ? Les raisons en sont nombreuses : c'est un langage facile à apprendre, avec des lignes de commande claires et concises. En outre, sa portée est générale et on l'utilise pour la composition de nombreux logiciels en *open source*. Aussi, le connaître, c'est un atout non seulement dans son cursus scolaire mais aussi pour plus tard.

Un scientifique

Guido van Rossum, le créateur du langage de programmation Python, est né en 1956 à Amsterdam. Il effectue des études de mathématiques dans sa ville natale et obtient un master en 1982. Il travaille alors en tant que développeur dans la conception du langage ABC dont il s'inspire pour élaborer le langage Python en 1991. Il reçoit en 2002 le prix du logiciel libre.

LE SAVIEZ-VOUS ?

Un groupe d'humoristes britanniques animait une émission télévisuelle à la BBC sous le titre de *Monty Python's Flying Circus* entre 1969 et 1974. Celle-ci recueillit un tel succès qu'elle fut reprise par de nombreuses chaînes étrangères.
Le mathématicien hollandais Guido van Rossum s'est enthousiasmé pour cette émission. Son autre passion était l'informatique et lorsqu'il développa un langage, il le nomma par humour PYTHON, du nom de la troupe de comiques.

Ce chapitre rappelle les bases de la programmation en Python et présente quelques compléments importants concernant les fonctions.

OBJECTIFS

- Maîtriser les instructions élémentaires :
  - connaître les notions de variable et de fonction ;
  - distinguer une expression et une instruction ;
  - utiliser une affectation, une instruction conditionnelle, une boucle ;
  - savoir définir et utiliser une fonction.
- Approfondir la notion de fonction :
  - décrire des conditions et écrire une spécification ;
  - mesurer l'importance des tests.
- Comprendre la modularisation :
  - créer un programme et l'utiliser dans un autre programme ;
  - savoir importer une bibliothèque existante ;
  - être capable de trouver des informations dans une documentation.

Cours

Introduction

Un programme est la description d'un **algorithme** dans un **langage** compréhensible par un humain et par une **machine** qui l'exécute afin de traiter des **données**. Les concepts antérieurs à l'informatique et qui en sont les fondements sont tous présents ici.

Les programmes sont écrits par des humains et lus par des machines. Les nombreux langages de programmation sont donc pour certains plus proches du langage naturel, pour d'autres plus proches de celui de la machine.

Le langage de programmation Python, retenu dans le cadre de cet enseignement, est déjà utilisé au lycée en Mathématiques. Il est également présent dans d'autres disciplines et dans le nouvel enseignement de Sciences numériques et Technologie. Ce langage est de plus en plus répandu dans l'enseignement secondaire et supérieur.

Le langage Python a été créé par un ingénieur informaticien néerlandais, Guido Van Rossum. La première version publique date de 1991. Van Rossum a continué ensuite à s'investir sur le projet Python et a travaillé entre autres pour Google puis Dropbox. C'est la version 3 de Python qui est utilisée dans cet enseignement. Cette version est disponible depuis 2008 avec des mises à jour régulières. Nous en sommes actuellement à la version 3.7 et la 3.8 est en développement.

Le langage Python est multiplateforme. Il fonctionne aussi bien sur des ordinateurs avec les systèmes d'exploitation Linux, MacOs, Windows ou des smartphones avec les systèmes Android ou iOS. Il est gratuit et placé sous licence libre.

Les constructions élémentaires présentées en langage Python sont communes à de nombreux autres langages de programmation. Un programme est composé de séquences, (des instructions exécutées l'une après l'autre dans l'ordre où elles sont écrites), de définitions de variables et de fonctions, d'affectations, d'instructions conditionnelles, de boucles, utilisant des expressions, en particulier des appels de fonctions.

Éléments de base

Variables et affectation

Les données utilisées par un programme sont stockées dans des variables. Ceci se fait par une affectation qui associe une donnée, représentée par une valeur ou une expression, avec un nom. Quand on dit "variable", on peut penser à une boîte sur laquelle est écrit un nom et dans laquelle on place des informations diverses. Une même boîte peut avoir plusieurs noms. Le nom peut être n'importe quelle chaîne alphanumérique, exceptés certains mots réservés, sans commencer par un chiffre. L'opérateur d'affectation est noté "`=`". L'instruction `x=3` associe la valeur `3` au nom `x`. L'instruction `y=3+5` associe la valeur de l'expression à droite du signe `=`, ici `8`, au nom `y`. L'instruction `z=x+y` associe la valeur de l'expression à droite du signe `=`, ici `11` (la somme des valeurs de `x` et de `y`), au nom `z`.

Python permet de procéder à des affectations multiples comme avec l'instruction `x,y,z=1,3,5` qui associe les valeurs `1`, `3` et `5` respectivement aux noms `x`, `y` et `z`. Ceci permet de remplacer une séquence d'instructions, les trois affectations `x=1; y=3; z=5` écrites sur une seule ligne en les séparant par un point-virgule ou sur trois lignes successives, par une instruction unique.

Attention : le vocabulaire est important et la signification de chacun des quatre termes "variable", "expression", "instruction" et "affectation" doit être parfaitement connue.

- Une **variable** est composée d'un nom (ou identificateur), d'une adresse en mémoire où est enregistrée une valeur (ou un ensemble de valeurs), d'un type qui définit ses propriétés.
- Une **expression** a une valeur qui est le résultat d'une combinaison de variables ou d'objets, de constantes et d'opérateurs.
- Une **instruction** est une commande qui doit être exécutée par la machine.
- Une **affectation** est une instruction qui commande à la machine de créer une variable en lui précisant son nom et sa valeur.

Ce n'est pas toujours facile, mais il faut bien distinguer une expression d'une instruction. Une expression se calcule, elle a une valeur, alors qu'une instruction s'exécute et n'a pas de valeur. L'écriture `x=0.5*x**2+1` est une instruction, soit : affecter la valeur de l'expression `0.5*x**2+1` à la variable qui a pour nom `x`. En particulier, un appel de fonction qui renvoie un résultat est une expression.

Note : en pratique les lettres capitales ne sont pas utilisées dans les noms de variables ; nous écrivons `variable` ou `ma_variable` mais pas `Variable` ni `MaVariable`.

Types

- Le **type** d'une variable définit l'ensemble des valeurs qui peuvent lui être affectées ainsi que les opérations et les fonctions utilisables.

Types simples

Les types de base sont les types numériques `int`, `bool`, `float` et le type `str`. Ces types sont détaillés plus précisément au chapitre 3.

Le type `int` est utilisé pour représenter les nombres entiers. Leur taille n'est limitée que par la capacité de la machine et le temps nécessaire à leur utilisation. Un calcul avec de très grands entiers est possible en Python mais peut prendre un temps important.

Le type `bool` permet de représenter les valeurs booléennes `True` (vrai) et `False` (faux). Ces valeurs peuvent être considérées comme les entiers 1 pour `True` et 0 pour `False`.

Le type `float` est utilisé pour représenter les nombres réels. Attention, le point remplace la virgule utilisée en mathématiques en France. On écrit par exemple 8.3 ou -6.17. Les nombres du type `float`, forcément en nombre limité, sont des décimaux qui servent à approximer les réels. Ils sont stockés dans la machine sous la forme de "nombres en virgule flottante" avec des valeurs comprises entre environ $-1,7 \times 10^{308}$ et $1,7 \times 10^{308}$.

Citons aussi le type `None` qui n'a qu'une seule valeur, la valeur `None`. C'est par exemple la valeur que renvoie une fonction si rien n'est précisé dans le code.

Opérations sur les types numériques

Nous avons les opérations mathématiques habituelles : addition, soustraction, multiplication, exponentiation et division notées respectivement `a+b`, `a-b`, `a*b`, `a**b` et `a/b`.

Le résultat est de type `float` si `a` ou `b` est de type `float` et sinon de type `int`, sauf pour la division avec laquelle le résultat est toujours de type `float`.

La division entière `//` et l'opération modulo `%` utilisés avec des entiers naturels, donnent respectivement le quotient et le reste, de type `int`, dans la division euclidienne. Si $a = bq + r$ avec $0 \leq r < b$ et $b \neq 0$ alors `a//b` donne q et `a%b` donne r.

Des raccourcis d'écriture existent pour toutes ces opérations. Par exemple, l'écriture `x+=y` est équivalente à l'écriture `x=x+y`, l'écriture `x*=2` est équivalente à l'écriture `x=x*2`.

Comparaison et opérateurs booléens

Les opérateurs mathématiques classiques de comparaisons $=$, $\neq$, $<$, $\leq$, $>$, $\geq$ s'écrivent en Python respectivement `==`, `!=`, `<`, `<=`, `>` et `>=`.

`x==y` prend la valeur `True` si x et y sont égaux, sinon prend la valeur `False`.

`x!=y` prend la valeur `True` si x et y ne sont pas égaux, sinon prend la valeur `False`.

Opérations logiques :

`a and b` prend la valeur `True` si a et b sont `True` et sinon prend la valeur `False` ;

`a or b` prend la valeur `False` si a et b sont `False` et sinon prend la valeur `True` ;

`not a` prend la valeur `True` si a est `False` et prend la valeur `False` si a est `True`.

Le type str

Le type `str`, abréviation de string, est utilisé pour représenter des chaînes de caractères, par exemple ce qui est obtenu dans la machine lorsqu'on appuie sur les touches du clavier. On utilise des guillemets ou des apostrophes comme `ch="bonjour"` ou `ch='bonjour'`. Attention, si on écrit `ch=bonjour`, alors `bonjour` est considéré comme le nom d'une variable et une erreur est signalée si cette variable n'existe pas.

On obtient la longueur d'une chaîne, le nombre de caractères contenus dans la chaîne, avec la fonction `len`. Par exemple, `len("bonjour")` a pour valeur l'entier 7.

Après l'instruction `ch=au revoir!`, l'expression `len(ch)` a pour valeur l'entier 10. En effet, l'espace " ", ou le caractère blanc, et le point d'exclamation "!" sont aussi des caractères.

Chaque caractère d'une chaîne a un indice qui va de 0 à $n-1$ si n est la longueur de la chaîne et il est possible d'accéder au caractère d'indice `i` d'une chaîne `ch` avec la notation `ch[i]`.

Nous pouvons aussi accéder à une suite de caractères d'une chaîne `ch` avec la notation `ch[i:j]` qui est une chaîne de caractères contenant dans l'ordre les caractères de `ch` d'indices `i` inclus à `j` exclu.

Types composés

Ce sont par exemple les types `tuple`, `list`, `dict`. Nous y reviendrons plus longuement au chapitre 4 mais voici quelques points importants concernant le type `list`.

Un objet de type `list`, nous dirons une "liste", représente un ensemble ordonné d'objets éventuellement de types différents. Les éléments sont indexés par un entier commençant à 0 comme pour les chaînes de caractères.

Les instructions suivantes permettent de définir une liste :

`liste1=[]`, une liste vide ;

`liste2=[5]`, une liste à un élément ;

`liste3=[5,'bonjour',25.4,['a','b']]`, une liste avec des éléments de différents types.

La fonction `len` renvoie la longueur d'une liste, le nombre de ses éléments. Donc l'expression `len(liste3)` a pour valeur l'entier 4.

L'accès à un élément particulier ou à une suite d'éléments s'obtient comme pour les chaînes de caractères. Par exemple `liste3[2]` est l'élément 25.4, `liste3[0:3]` est la liste des éléments d'indices 0, 1 et 2 de `liste3`, soit `[5,'bonjour',25.4]`.

Un point très important est qu'il est possible de modifier un élément avec une instruction d'affectation. Nous verrons au chapitre 4 que ce n'est pas le cas avec le type `tuple`.

L'instruction `liste3[1]="au revoir"` modifie la liste `liste3` qui prend alors pour valeur `[5,'au revoir',25.4,['a','b']]`

De nombreuses méthodes permettent de travailler avec des listes. Nous emploierons très souvent la méthode `append`. La syntaxe doit être bien connue : le nom de la méthode s'écrit après le nom de l'objet auquel elle s'applique, les deux noms étant séparés par un point sans espace.

L'instruction `liste.append(x)` ajoute l'objet `x` à la fin de la liste `liste`. Cette méthode sert en particulier à construire une liste en ajoutant les éléments un par un à la fin de la liste.

Opérations sur les types str et list

L'expression `x in c` vaut `True` si `x` est un élément de `c` et sinon `False`.

`c[i]` est l'élément de `c` d'indice i, donc le (i+1)-ème, `c[0]` est le premier élément.

Il est possible aussi d'indexer à partir du dernier élément qui est `c[-1]`, et avec `c[-2]`, on obtient donc l'avant-dernier élément de `c`.

`c[début:fin]` est une tranche de `c` qui commence à l'index `début` et se termine à l'index `fin-1`.

`len(c)` est la longueur de `c`.

`c1+c2` est la concaténation de `c1` et `c2`.

Par exemple `'bon'+'jour'` donne `'bonjour'`, `[1,2]+[3,4,5]` donne `[1,2,3,4,5]`.

`n*c` est la concaténation de n copies de `c` si n est un entier naturel.

Par exemple `2*"bon"` donne `"bonbon"`, `3*[1,2]` donne `[1,2,1,2,1,2]`.

La fonction `type` permet de déterminer le type d'un objet. Les fonctions `int`, `float`, `str` permettent de convertir si c'est possible un objet d'un autre type respectivement en type `int`, `float`, `str`. Par exemple, avec les instructions `ch="12.5"` puis `x=float(ch)`, la variable `ch` est une chaîne de caractères et la variable `x` est un nombre flottant. De manière analogue, avec les instructions `x=12.5` puis `ch=str(x)`, la variable `x` est un nombre flottant et la variable `ch` est une chaîne de caractères.

La fonction `list` permet de convertir une chaîne de caractères en une liste dont les éléments sont les différents caractères de la chaîne. Avec les instructions `ch='12.5'` puis `liste=list(ch)`, la variable `liste` a pour valeur la liste `['1','2','.','5']`.

■ Instructions conditionnelles et boucles

L'indentation, décalage vers la droite du début de ligne, est un élément de syntaxe important en Python. Elle délimite des blocs de code et elle aide à la lisibilité en permettant d'identifier facilement ces blocs. La ligne précédant l'indentation se termine par le signe deux-points.

■ Instructions conditionnelles

La structure la plus simple est :

```
if condition:
    instructions
```

Le mot `condition` désigne une expression et le mot `instructions` désigne une instruction ou un bloc d'instructions écrites sur plusieurs lignes. Voici un exemple :

```
if n % 2 == 0:
    n = n // 2
```

La structure `if-else` présente une alternative :

```
if condition:
    instructions1
else:
    instructions2
```

Par exemple :

```
if n % 2 == 0:
    n = n // 2
else:
    n = 3 * n + 1
```

La structure `if-elif-else` présente plusieurs alternatives :

```
if condition1:
    instructions1
elif condition2:
    instructions2
elif condition3:
    instructions3
...
else:
    instructions
```

Par exemple :

```
if n % 4 == 0:
    n = n // 4
elif n % 4 == 1:
    n = (3 * n + 1) // 4
elif n % 4 == 2:
    n = n // 2
else:
    n = (3 * n + 1) // 2
```

Remarques :

- les mots `if` et `elif` sont suivis d'une expression qui a la valeur `True` ou `False`, et la ligne se termine par le signe deux-points ;
- le mot `else` est suivi immédiatement par le signe deux-points ;

- c'est l'indentation qui permet de délimiter les blocs d'instructions à exécuter si la condition est vérifiée.

Note : les mots `True` et `False` sont avec `None` les rares mots dont l'écriture en Python commence par une lettre capitale.

■ Boucles conditionnelles

La structure est :

```
while condition:
    instructions
```

La structure est identique à celle de l'instruction conditionnelle `if`. La condition qui suit le mot `while` est une expression dont la valeur interprétée est `True` ou `False`. Le bloc d'instructions qui suit, indenté, est exécuté tant que la condition est vérifiée. C'est ce bloc d'instructions qui doit modifier la valeur de l'expression à tester afin que la condition ne soit plus vérifiée après un nombre fini d'exécutions du bloc. Si le programme a une suite, elle pourra alors être exécutée.

Voici un exemple :

```
while a >= b:
    a = a - b
    q = q + 1
```

Tant que la condition `a>=b` est vérifiée, les instructions indentées sont exécutées, ici `a=a-b` puis `q=q+1`. Il faut absolument toujours s'assurer qu'après un nombre fini de passages dans la boucle, la condition n'est plus vérifiée, ce qui est bien le cas ici grâce à la répétition de l'instruction `a=a-b`.

■ Boucles non conditionnelles

Une boucle non conditionnelle permet de répéter `n` fois, `n` étant connu, une instruction ou un bloc d'instructions. La syntaxe est :

```
for i in range(n):
    instructions
```

Ce code peut évidemment être remplacé par le code :

```
i = 0
while i < n:
    instructions
    i = i + 1
```