Travaux pratiques n°1

Répertoires et fichiers sous Windows
Présentation générale de Python 3

F. Cuvellier, J. Tanoh

M.P.S.I. 1, 2018–2019

Usage de Windows

1 Répertoires et fichiers sous Windows

1.1 Répertoires

Un *fichier* est une suite de caractères.
Un *répertoire* (ou *dossier*, dans les versions françaises de Windows) est un fichier qui contient des fichiers. En pratique, pour organiser le contenu d'un ordinateur, on crée des répertoires pour regrouper des fichiers conçus suivant un même principe ou relevant d'un même thème.

1.2 Commandes

Pour gérer un grand nombre de fichiers, par exemple ceux d'un même répertoire qui ont été créés à une date donnée, on peut utiliser des commandes d'utilisateurs, qui permettent d'exécuter une même opération (une commande) sur tous les fichiers. C'est ainsi qu'on peut renuméroter les milliers de photographies prises pendant de passionnantes vacances avec trois téléphones portables, deux tablettes numériques et quatre appareils photographiques numériques, de sorte qu'elles se succèdent dans l'ordre chronologique. (Renuméroter à la main chaque fichier de photographie pourrait s'avérer fastidieux.) Un ordinateur doit disposer de fonctionnalités qui facilitent ce travail, c'est-à-dire les effectuent en quelques secondes.
Les systèmes d'exploitation de type Unix proposent des langages de programmation efficaces, appelés *shells de commandes*, qui effectuent de telles tâches et bien d'autres, par exemple, les shells *sh*, *bash*, *tcsh*, *zsh*, etc. Les systèmes d'exploitation de type Windows proposent aussi une fonctionnalité qui s'apparente à un shell. Dans la version française, elle s'appelle *Invite de commandes*.

Exercice 1. Pour visualiser l'arborescence d'un répertoire, appliquer la fonction `tree`.

Exercice 2. Créer cent fichiers, appelés `x00.txt`, `x01.txt`, ..., `x99.txt`, qui contiennent chacun uniquement leur nom sans l'extension. (Par exemple, `x47.txt` contient la chaîne de trois caractères `x47`.)

Python est un langage de programmation orienté objet. Le sens de cette expression se précisera peu à peu.
Le livre *Programming in Python 3. A Complete Introduction to the Python Language*, de Mark Summerfield (Addison-Wesley Publishing Company, 2010, seconde édition), décrit soigneusement Python 3. Les exemples qui suivent proviennent de la section *Python's "Beautiful Heart"* du premier chapitre ou s'inspirent de ceux que propose l'auteur.

2 Types

Les objets employés par Python sont *typés* : leur nature est connue et détermine en partie le comportement des fonctions qui s'appliquent à eux.
Par exemple, `'python'` est une chaîne de caractères (*string*), de type `str`, tandis que `17` est un nombre entier (*integer*), de type `int`.
Aucun de ces objets ne peut être modifié. De plus, une fois créés, ils restent à la même adresse-mémoire. On dit qu'ils sont *non mutables*. Les autres sont dits *mutables*.
La fonction `type()` permet de connaître le type d'un objet.
Les booléens sont `True` et `False`.

Exercice 3. Déterminer les types de `13`, de `'MPSI1'`, de `'MPSI1'[2]`, de `.1` et de `5**6`.

Pour convertir un objet au type *datatype*, on lui applique la fonction *datatype* correspondante. (Python 3 se chargera alors d'interpréter ce qu'il faut.) Par exemple, `str(27)` fait du nombre 27 une chaîne de caractères, et `int(27.)` fait du nombre flottant 27 un nombre entier.

Exercice 4. Déterminer le type de `str(27)`.

3 Variables

L'affectation d'une variable se fait à l'aide de l'opérateur `=`, comme en C. Par exemple, `x = 17` affecte l'objet `x` de la valeur 17. (La réalité est un peu plus subtile : les variables de Python sont des références d'objet, et l'on en verra une conséquence plus loin, avec l'usage d'une *méthode*.)
Pour afficher la valeur d'une variable, par exemple lors de l'exécution d'une fonction ou d'une boucle, on se sert de la fonction `print()`. Pour la variable `x`, `print(x)`. (En Python 2, la syntaxe est différente : `print x`.) En mode interactif, il suffit d'entrer le nom de la variable pour obtenir sa valeur : `x`.

Exercice 5.

1. Affecter les variables `x`, `y`, `z` des valeurs `'rouge'`, `'vert'`, `'jaune'`.
2. Afficher ces trois variables.
3. Affecter `z` de la valeur `x`, et afficher ces trois variables.
4. Affecter `x` de la valeur 0 et afficher `z`. Expliquer le résultat.

4 Calculs numériques

Les principaux opérateurs arithmétiques sont `+` (addition), `-` (soustraction), `*` (multiplication), `/` (division) et `**` (exponentiation). Tous sont binaires, mais `-` permet aussi d'obtenir l'opposé. Ils s'appliquent aussi bien à des nombres entiers qu'à des nombres flottants. La division retourne un nombre flottant.
L'opérateur binaire `//` a pour paramètres des nombres entiers et donne le quotient de la division euclidienne du premier par le second, et l'opérateur `%` le reste.
L'addition sert aussi à concaténer des chaînes de caractères.

Chacun des opérateurs précédents peut être combiné à `=`, pour affecter une variable déjà définie d'une nouvelle valeur. Par exemple, après `a = 35`, `a %= 4` donne à `a` la valeur `35 % 4`.

5 Structures de données de collection

Il est souvent utile de regrouper des objets de type quelconque. À cet effet, Python dispose de plusieurs types de structure, par exemple les tableaux (`list`) et les n-uplets (`tuple`). Certains peuvent être modifiés (`list`), d'autres non (`tuple`).
Pour former un n-uplet, il suffit d'écrire des objets, qu'on sépare seulement par une virgule : `4, 'pair', 5, 'premier'`. Pour un 1-uplet, on fait suivre l'objet d'une virgule : `'un objet',`. Pour créer un 0-uplet, on se contente d'écrire des parenthèses : `()`.
Un tableau peut se former de la même façon, à ceci près que les objets *doivent* être entourés de crochets : `[4, 'pair', 5, 'premier']`.
Les objets d'une telle structure sont indexés par des nombres entiers naturels consécutifs. Il faut noter que l'indice du premier est toujours 0. Si l'objet est de longueur n, l'indice du dernier est donc $n - 1$.
La longueur de l'objet *obj* s'obtient à l'aide de la fonction `len()`. Chaque élément est accessible : celui d'indice k s'obtient en écrivant k entre crochets derrière l'objet *obj*, et le dernier à l'aide de `[-1]`, le précédent de `[-2]`, etc.
Par exemple,

```
>>> len(('un seul',))
1
>>> ('un seul')[-1]
'l'
>>> ('un seul',)[-1]
'un seul'
>>> 'un seul'[4]
'e'
```

Exercice 6.

1. Déterminer la longueur des objets `[4, 4., -.1, .1, 'a']`, `"Que c'est long!"`, `range(7)`, `range(5,9)` et `4, 'pair', 5, 'premier'`.

2. Déterminer, si possible, l'avant-dernier élément de chacun d'eux.

Les variables affectées d'un tableau sont modifiables. Par exemple, si `L` est un tableau, l'instruction `L[3] = 9` donne à l'élément d'indice 3 (c'est-à-dire le quatrième) la valeur 9.

On peut aussi se servir d'une *méthode* adaptée à la structure de données. Par exemple, la méthode `list.append()` est adaptée à la structure de tableau (`list`) et l'instruction `L.append('fin')` place la chaîne `'fin'` au bout du tableau `L`.

6 Comparaison et opérateurs logiques

L'opérateur d'identité des valeurs de deux variables est `is`.
Par exemple, après `a=5`, `b=6`, la commande `a is b` renvoie `False`. Après `a=b`, elle renvoie `True`.
Les opérateurs de comparaison sont `==` (égal), `<` (strictement plus petit), `<=` (inférieur ou égal), `>` (strictement plus grand), `>=` (supérieur ou égal) et `!=` (différent).
L'opérateur d'appartenance est `in`, celui de non-appartenance `not in`.
Les opérateurs logiques sont `and`, `or` et `not`. Les deux premiers ne retournent un booléen que si leurs opérandes sont booléens. Le dernier retourne toujours un booléen.

Exercice 7. Étudier (et deviner) le comportement des opérateurs logiques `and` et `or` avec des opérandes non booléens.

7 Structures de contrôle

Les opérateurs précédents servent notamment dans les structures de contrôle, pour les tests.

7.1 Instruction if

L'instruction `if` sert à effectuer les tests conditionnels. La structure est la suivante.

```
if test_1:
    instruction_1
elif test_2:
    instruction_2
...
else:
    instruction_finale
```

Les deux-points sont obligatoires, ainsi que les retraits. Les lignes `elif` et `else` sont optionnelles. Si un ou plusieurs tests retournent `True`, c'est l'instruction qui suit le premier d'entre eux qui est exécutée. Sinon, c'est l'instruction qui suit `else`, ou à défaut, rien n'est fait.

7.2 Instruction while

La version simplifiée de cette instruction est la suivante.

```
while test:
    instructions
```

Là encore, le deux-points et le retrait sont obligatoires. Tant que le test retourne `True`, les instructions sont exécutées.

Exercice 8. Former le tableau des cubes parfaits (c'est-à-dire des cubes de nombres entiers naturels) strictement plus petits que 2^{20}.

7.3 Instruction for

La version simplifiée de cette instruction est la suivante.

```
for variable in structure_de_collection:
    instructions
```

Là encore, le deux-points et le retrait sont obligatoires. La variable prend successivement les valeurs des éléments de la structure de donnée de collection et, à chaque fois, les instructions sont exécutées.

Exercice 9. Afficher les caractères de la chaîne `"Que c'est long!"`.

Exercice 10. Dans ce qui suit, les chaînes de caractères sont formées de 0 et de 1.

1. Une telle chaîne de caractères, s, représente en base 2 un nombre entier relatif. Si elle est de longueur l, ce nombre entier est le nombre n tel que s représente le nombre entier naturel $n + 2^{l-1}$ en base 2.

   Donner une suite d'instructions qui transforme s en n.

2. Donner une suite d'instructions qui transforme un couple (n, l) de nombres entiers tels que $-2^{l-1} \leqslant n < 2^{l-1}$ en la chaîne de caractères de longueur l qui représente n en base 2.

3. On fixe le nombre entier naturel l. Afficher les chaînes de caractères qui représentent en base 2 les nombres entiers relatifs n tels que $-2^{l-1} \leqslant n < 2^{l-1}$.

4. Donner une suite d'instructions qui transforme une chaîne de caractères de longueur l en la chaîne de caractères qui représente le nombre entier relatif opposé à celui qu'elle représente, puis une suite d'instructions qui transforme deux chaînes de caractères de longueur l en la chaîne de caractères qui représente la somme des nombres entiers relatifs qu'elles représentent.

   En réalité, la représentation usuelle d'un nombre entier relatif n'est pas la précédente, mais la représentation dite *en complément à deux*. Un mot de longueur l dont le premier bit est un 0 représente un nombre entier naturel strictement plus petit que 2^{l-1} : les bits suivants sont les chiffres de la représentation en base 2 du nombre entier naturel n. Un mot de longueur l dont le premier bit est un 1 représente un nombre entier relatif strictement négatif supérieur ou égal à -2^{l-1} : les bits suivants sont les chiffres de la représentation en base 2 du nombre entier naturel $n + 2^{l-1}$.

5. Reprendre les quatre questions précédentes. Conclure.