TDLOG – séance n° 2
Premiers pas en Python : TP

Xavier Clerc – xavier.clerc@enseignants.enpc.fr
Cédric Doucet – cedric.doucet@inria.fr
Thierry Martinez – thierry.martinez@inria.fr

30 septembre 2015

À rendre au plus tard le 4 octobre 2015

Cette séance se divise en quatre parties : une partie introductive portant sur l'installation de *Python*, une partie d'exploration de l'interpréteur, une partie composées d'exercices, et enfin une annexe présentant quelques fonctions prédéfinies utiles aux exercices.

1 Installation

Pour effectuer les travaux pratiques, puis ultérieurement le projet de fin de module, vous devez installer *Python* sur votre machine. Pour cela, nous vous suggérons d'utiliser *Miniconda* qui est un gestionnaire de paquets développé spécifiquement pour *Python*. Le fait d'utiliser ce gestionnaire de paquets vous permettra d'installer très facilement des librairies additionnelles pour *Python*, *p. ex.* pour les interfaces graphiques ou le calcul numérique.

Pour installer *Miniconda*, vous pouvez le télécharger à l'adresse suivante :

`http://conda.pydata.org/miniconda.html`

ou le récupérer depuis une clef USB disponible pendant le TP. La page suivante (présente sous la forme d'un fichier pdf sur la clef USB) précise les étapes d'installation pour les différents systèmes d'exploitation :

`http://conda.pydata.org/docs/install/quick.html`

Une fois l'installation terminée, vous pouvez lancer l'interpréteur en mode interactif en tapant `python` depuis un *shell* ou *invite de commande*. Il est possible de quitter l'interpréteur en tapant `exit()`. Attention, l'état de l'interpréteur n'est pas sauvegardé et toutes les valeurs, fonctions et classes définies sont perdues. Il ne faut donc utiliser l'interpréteur de manière interactive qu'à des fins d'expérimentation.

Lorsque vous voulez exécuter un programme *Python* stocké dans un fichier, il faut exécuter `python` *fichier*`.py`. N'oubliez pas d'ajouter au fichier une en-tête spécifiant l'encodage de votre fichier si vous utilisez des caractères accentués (*p. ex.* `#encoding: latin1` ou `#encoding: utf8`).

Pour être en mesure, sous MacOS X ou Linux, d'exécuter directement un fichier contenant un programme *Python* sans taper explicitement `python`, il faut que la première ligne du fichier soit de la forme `#!/path/to/python` où `/path/to` est le répertoire contenant l'interpréteur *Python*[1]. Il faut également que le fichier ait la propriété d'être exécutable, ce que l'on obtient par l'exécution de la commande `chmod +x` *fichier*`.py`.

2 Exploration

Dans cette partie, nous proposons une suite de manipulations de l'interpréteur que nous vous suggérons de répéter pas-à-pas afin de vous familiariser avec *Python*.

Quand l'interpréteur est lancé, il affiche un court message contenant les informations de version, puis une invite qui attend que vous entriez des fragments de code à évaluer. L'invite par défaut de l'interpréteur est `>>>`. Lorsque vous tapez une ligne qui attend (optionnellement) une suite, l'interpréteur change son invite en `...` pour indiquer qu'il attend la suite de la portion de code. Vous devez bien entendu respecter l'indentation, et il vous suffit de taper la touche "entrée" sans aucun caractère à une invite `...` pour indiquer la fin de votre saisie.

Basiquement, l'interpréteur peut être utilisé comme une calculatrice, avec la possibilité de sauvegarder les résultats dans des variables :

```
>>> 3 + 4
7
>>> a = 3
>>> b = a + 5
>>> print(a, b)
3 8
```

On peut définir des fonctions puis les utiliser dans des expressions :

```
>>> def carre(x):
...   return x * x
...
>>> print(carre(5))
25
```

1. Si vous ne le connaissez pas, tapez `which python` pour l'obtenir.

Pour parcourir les éléments d'une liste ou d'une chaîne de caractères, il suffit d'utiliser une construction `for`, en respectant l'indentation :

```
>>> for e in [1, 2, 3]:
...   print(e)
...
1
2
3
>>> for ch in "chaine":
...   print(ch)
...
c
h
a
i
n
e
```

Dans un dictionnaire, l'énumération porte sur ses clefs :

```
>>> for clef in { 0:"zero", 1: "un", 2:"deux" }:
...   print(clef)
...
0
1
2
```

Si l'on souhaite répéter un traitement n fois, le plus simple est d'utiliser la fonction `range(`n`)` :

```
>>> for i in range(5):
...   print("****", i)
...
**** 0
**** 1
**** 2
**** 3
**** 4
```

La concaténation de deux chaînes de caractères se fait en utilisant l'opérateur `+`, et la concaténation d'une liste de chaînes en utilisant un séparateur par la méthode `join` :

```
>>> print("premiere chaine" + " " + "deuxieme chaine")
premiere chaine deuxieme chaine
>>> print(", ".join(["un", "deux", "trois"]))
un, deux, trois
```

Lorsque l'on travaille avec des ensembles ou des dictionnaires, les opérateurs `in` et `not in` permettent de tester l'appartenance d'une valeur à l'ensemble ou aux clefs du dictionnaire :

```
>>> e = {2, 4, 5}
>>> print((2 in e), (5 not in e))
True False
```

Les compréhensions permettent de définir de manière succinte des listes, ensembles ou dictionnaires :

```
>>> [ i * i for i in range(10) ]
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
>>> def mult7(x): return (x % 7) == 0
...
>>> { i for i in range(100) if mult7(i) }
set([0, 98, 35, 70, 7, 42, 77, 14, 49, 84, 21, 56, 91, 28, 63])
>>> { i: i*i for i in range(10) }
{0: 0, 1: 1, 2: 4, 3: 9, 4: 16, 5: 25, 6: 36, 7: 49, 8: 64, 9: 81}
```

Définissez une fonction `est_premier` qui renvoie `True` ssi le paramètre qu'elle reçoit est un nombre premier, puis utilisez cette fonction dans une compréhension d'ensemble afin de définir l'ensemble des nombres premiers inférieurs à 100.

3 Exercices

La société `CleanCode` développe des logiciels à une échelle internationale depuis une dizaine d'années. Elle jouit d'une très bonne réputation dans la communauté informatique grâce à la grande qualité des codes qu'elle écrit. Soucieux de préserver leur valeur ajoutée, les membres de l'équipe de développement ont mis en place des tests permettant d'évaluer l'adéquation des candidats à la philosophie de l'entreprise. Vous êtes l'un de ces candidats et vous allez être évalué sur les trois exercices ci-dessous. Durant toute la durée du test, vous avez un accès libre à internet.

Durant ce test, tous vos codes devront être écrits dans un seul et unique fichier, créé à partir du fichier `test.py` mis à votre disposition (sur educnet [2] et la clef USB). Les réponses aux différentes questions qui vous seront posées devront apparaître dans un fichier texte `test_nom_prenom.txt`, où `nom` et `prenom` correspondent à vos propres nom et prénom. Le fichier devra être déposé sur educnet avant la date limite.

3.1 Écriture d'un Algorithme simple

Dans cet exercice, vous serez évalués à la fois sur la correction de votre algorithme et sur la qualité de son écriture. Le code final doit être le plus lisible possible.

2. `https://educnet.enpc.fr`

Tâche 1 : Écrivez une fonction qui convertit des nombres en chaînes de caractères. L'utilisation des fonctions fournies par le langage Python est proscrite.

Avant de démarrer cet exercice, nous vous invitons à réfléchir au sens que vous donnez à son énoncé. Quelle sera la portée de votre algorithme ? Quels types de nombre allez-vous traiter ? Sous quelle forme pourront-ils être écrits ? Réfléchissez également à l'organisation de votre code, à vos choix de noms de variables et de fonctions, aux commentaires qui vous semblent utiles sans être redondants avec le code, *etc.*

3.2 Prise en main d'un code existant

Le fichier `test_nom_prenom.py` contient deux fonctions nommées `piksrt` et `shell`. Ces fonctions ont été écrites par un stagiaire qui a quitté l'entreprise et qui n'est plus joignable. Le responsable des développements vous demande de reprendre ce code.

Tâche 2 : expliquez brièvement ce que font les fonctions `piksrt` et `shell`, en indiquant vos sources d'information (code, commentaires, documentation, tests, livres, internet, collègues, *etc.*).

Ces fonctions ne répondent pas aux exigences de qualité d'écriture de l'équipe de développement. Celle-ci a adopté des conventions de codage, fortement inspirées du PEP-0008 :

`https://www.python.org/dev/peps/pep-0008`

Ce document fournit des conseils quant à la manière de coder en Python : formatage des fichiers, nommage des variables et des fonctions, *etc.*

Tâche 3 : à l'aide du guide PEP-0008 (section *code layout*), améliorez le formatage du fichier `test_nom_prenom.py`. Quel intérêt voyez-vous à adopter des conventions de formatage ?

Malgré cette première étape de formatage, l'écriture des fonctions `piksrt` et `shell` ne répond toujours pas aux critères d'exigence de l'entreprise.

Tâche 4 : proposez une nouvelle version de `piksrt` qui soit plus lisible. On attend de vous que vous réfléchissiez à des noms de fonction et de variables plus explicites. On s'attend également à ce que vous définissez au moins une sous-fonction bien nommée qui facilite la compréhension du code.

4 Annexe : Fonctions et méthodes usuelles

Cette annexe liste les fonctions et méthodes qu'il est nécessaire de connaître pour faire les exercices et écrire des programmes simples. Cependant, les descriptions sont volontairement

succintes ; pour une description plus détaillée et l'ensemble des fonctions, classes et méthodes, on se reportera à la documentation de référence fournie avec le langage ou à la fonction `help(...)` de l'interpréteur.

4.1 Principales fonctions *built-in*

Quelques fonctions de conversion, chacune transformant son unique paramètre en une instance du type dont elle porte le nom :

- `bool` ;
- `int` ;
- `float` ;
- `complex` ;
- `list` ;
- `dict` ;
- `set` ;
- `frozenset` ;
- `str` (pour *string*, chaîne de caractères).

Quelques fonctions de manipulation de listes :

- `len(`l`)` retourne la longueur de l ;
- `map(`f`,` l`)` retourne `[` $f(e_0), f(e_1), \ldots, f(e_n)$ `]` où $l =$ `[` $e_0, e_1, \ldots, e_n$ `]` ;
- `min(`l`)` retourne le plus petit élément de l ;
- `max(`l`)` retourne le plus grand élément de l ;
- `sum(`l`)` retourne la somme les éléments de l ;
- `reduce(`f`,`l`,`z`)` retourne `[` $f(z, f(e_0, f(e_1, \ldots)))$ `]` où $l =$ `[` $e_0, e_1, \ldots, e_n$ `]`.

La fonction `range` prend de un à trois paramètres et renvoie et un générateur d'entiers :

- un premier paramètre entier optionnel qui indique la borne inférieure (incluse), par défaut 0 ;
- un deuxième paramètre entier qui indique la borne supérieure (exclue) ;
- un troisième paramètre optionnel qui indique l'écart (ou *pas*) entre deux éléments successifs.

La fonction `print` prend un nombre quelconque de paramètres et les affiche successivement en les séparant par des espaces puis effectue un retour à la ligne.

La fonction `input` lit une chaîne tapée sur le clavier. Elle accepte un paramètre optionnel, utilisé comme *prompt*.

Les fonctions `ord` et `chr` permettent respectivement de convertir un caractère en entier et l'inverse.

4.2 Principales méthodes des listes

Dans ce qui suit, l désigne une liste :

- l`.append(`x`)` ajoute x en fin de liste ;
- l`.insert(`i, x`)` insère x à la position i ;
- l`.index(`x`)` retourne l'index de la première occurrence de x dans l ;
- l`.count(`x`)` retourne le nombre d'occurrences de x dans l ;
- l`.sort()` modifie l en la triant ;
- l`.reverse(`x`)` modifie l en inversant l'ordre de ses éléments.

4.3 Principales méthodes des chaînes de caractères

Dans ce qui suit, s désigne une chaîne de caractères :

- s`.upper()` renvoie une copie de s où toutes les lettres sont majuscules ;
- s`.lower()` renvoie une copie de s où toutes les lettres sont minuscules ;
- s`.replace(`o, n`)` renvoie une copie de s où toutes les occurences de o ont été remplacées par n ;
- s`.split(`*sep*`)` renvoie une liste qui contient le découpage de s selon le séparateur *sep* ;
- s`.join(`*seq*`)` renvoie la concaténation des éléments de la séquence *seq*, s étant utilisé comme séparateur ;
- s`.format(...)` prend un nombre quelconque de paramètres (possiblement nommé) et applique une mise en forme semblable à `printf` mais beaucoup plus puissant et flexible. Par exemple `"x = {}, y = {}, z = {}".format(3, 5, 7)` renvoie la chaîne `"x = 3, y = 5, z = 7"`. De plus, si un paramètre est nommé n, il est possible d'y faire référence par `"{n}"`. Une documentation complète des options de formattage est disponible à l'adresse `https://docs.python.org/3.4/library/string.html#formatspec`.

4.4 Principales méthodes des dictionnaires

Dans ce qui suit, d désigne un dictionnaire :

- d`.items()` retourne une liste des couples $\langle$ clef, valeur $\rangle$ de d ;
- d`.keys()` retourne la liste des clefs de d ;
- d`.values()` retourne la liste des valeurs de d.