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

Xavier Clerc – xavier.clerc@enseignants.enpc.fr
Thierry Martinez – thierry.martinez@inria.fr
Olivier Cros – olivier.cros0@gmail.com

3 octobre 2016

À rendre au plus tard le 8 octobre 2016 sur educnet

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 composée 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 un 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]. Alternativement, il est possible d'utiliser `env` afin d'éviter d'avoir à spécifier le chemin absolu ; dans ce cas, la première ligne est `#!/usr/bin/env python`. 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

L'objectif est de manipuler des structures de données représentant les résultats d'une compétition opposant des équipes réparties en groupes. Nous utiliserons une partie des résultats du Championnat d'Europe de football de 1992 [2].

Les données de la phase de groupes sont représentées de la manière suivante :

- liste des équipes engagées, par un dictionnaire dont les clefs sont les identifiants des groupes et les valeurs des ensembles d'équipes ;
- liste des matches joués, par une liste de quadruplets (nom de la première équipe, score de la première équipe, score de la seconde équipe, nom de la seconde équipe).

Soit en *Python* :

```
groups = {
    "1": { "Denmark", "England", "France",      "Sweden" },
    "2": { "CIS",     "Germany", "Netherlands", "Scotland" }
}
```

2. https ://fr.wikipedia.org/wiki/Championnat_d%27Europe_de_football_1992

```
matches = [
    ("Sweden",      1, 1, "France"),
    ("Denmark",     0, 0, "England"),
    ("Netherlands", 1, 0, "Scotland"),
    ("CIS",         1, 1, "Germany"),
    ("France",      0, 0, "England"),
    ("Sweden",      1, 0, "Denmark"),
]
```

note : les données de la compétition de 1992 peuvent être récupérées sur educnet (fichier data.py).

3.1 Affichage du classement

La première étape consiste à calculer et afficher le classement d'un groupe, sous la forme suivante :

```
Sweden .... 3 pts +1
France .... 2 pts +0
England ... 2 pts +0
Denmark ... 1 pts -1
```

Le classement est calculé à partir de la liste des matches déjà joués, en attribuant 2 points pour une victoire, 1 point pour un match nul, et 0 point pour un défaite. Le nombre relatif qui suit le nombre de point est la différence de buts, *i. e.* le nombre de buts marqués moins le nombre de buts concédés.

On classe d'abord par points. En cas d'égalité de points, on classe par différence de buts. En cas de double égalité, on classe par nombre de buts marqués. En cas de triple égalité, l'ordre n'est pas spécifié ; dans la réalité, il repose sur des informations non présentes dans les données fournies (*p. ex.* le nombre de cartons jaunes et rouges reçus).

L'affichage doit respecter l'alignement montré par l'exemple ci-dessus, le nombre minimal de caractères "point" après le nom de l'équipe étant de 3. Le jeu de données proposé est tel que le nombre de points est toujours sur un caractère, mais on peut chercher à écrire un code qui alignerait le nombre de points à droite :

```
Sweden .... 13 pts +1
France ....  2 pts +0
England ...  2 pts +0
Denmark ...  1 pts -1
```

3.2 Affichage de la liste des matches joués

La deuxième étape consiste à afficher la liste des matches déjà joués au sein d'un groupe, sous la forme suivante :

```
Sweden  1 - 1 France
Denmark 0 - 0 England
France  0 - 0 England
Sweden  1 - 0 Denmark
```

L'affichage doit respecter l'alignement montré par l'exemple ci-dessus.

3.3 (optionnel) Affichage de la liste des matches restants

La troisième étape consiste à afficher la liste des matches encore à jouer au sein d'un groupe, sous la forme suivante :

```
CIS     vs Netherlands
CIS     vs Scotland
Germany vs Netherlands
Germany vs Scotland
```

L'affichage doit respecter l'alignement montré par l'exemple ci-dessus. L'ordre d'affichage des matches n'est pas spécifié.

3.4 Affichage complet

La dernière étape consiste à organiser le travail déjà effectué afin d'afficher l'état de la compétition à partir des variables `groups` et `matches`, sous la forme suivante :

```
Group 1
-------
Sweden .... 3 pts +1
France .... 2 pts +0
England ... 2 pts +0
Denmark ... 1 pts -1

Sweden  1 - 1 France
Denmark 0 - 0 England
France  0 - 0 England
Sweden  1 - 0 Denmark

Denmark vs France
England vs Sweden
```

```
Group 2
-------
Netherlands ... 2 pts +1
Germany ....... 1 pts +0
CIS ........... 1 pts +0
Scotland ...... 0 pts -1

Netherlands 1 - 0 Scotland
CIS         1 - 1 Germany

CIS     vs Netherlands
CIS     vs Scotland
Germany vs Netherlands
Germany vs Scotland
```

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 `sorted` prend de un à trois paramètres et renvoie une liste triée (de manière croissante) :
- le premier paramètre est l'*itérable* (*p. ex.* une liste) à trier ;
- un paramètre optionnel, `key`, permet de spécifier la clef de tri ;
- un paramètre optionnel, `reverse`, permet de demander de renverser la liste (et donc d'obtenir un tri décroissant).

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`.reverse()` modifie l en inversant l'ordre de ses éléments ;
- l`.sort()` modifie l en la triant, la fonction accepte deux paramètres optionnels, `key` et `reverse` qui permettent respectivement de choisir comment effectuer le tri et s'il faut le renverser. Par exemple, `l.sort(reverse=True)` trie la liste en ordre décroissant.

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 puissante 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.