TP1 bis : more python

AP3 Algorithmique et programmation — L2 mathématiques

22 septembre 2016

1 Programmes indépendants en python

La semaine dernière on a utilisé la *boucle interactive d'interprétation* de `python` : on tapait une **unique** commande que `python` interprétait avant de nous retourner le résultat et une nouvelle invite de commande. C'est en quelque sorte le mode d'*essai* de `python`, parfait pour tester une commande ou obtenir de l'aide sur une fonction, mais pas du tout adapté au développement d'un logiciel complet.

Aujourd'hui, on va utiliser *python* pour faire de vrais programmes indépendants. C'est sur ce schéma qu'on codera nos programmes pendant le reste du semestre en TP :

(1) on écrit notre code `python` dans un fichier à l'aide d'un éditeur de texte tel qu'`emacs` ou `gedit` ; par exemple imaginons qu'on tape le code suivant dans un fichier nommé `main.py` :

```
"""
table takes a integer n and a optional integer stop.
It prints the table of n from 1*n to stop*n.
"""
def table(n,stop=10):
    for i in [j+1 for j in range(stop)]:
        print str(i)+"*"+str(n)+"="+str(i*n)
    return None

# This is the main part of the program
n = 7
print "Multiplication table of "+str(n)
table(n,10)
```

(2) on navigue ensuite dans un terminal jusqu'à l'emplacement de `main.py` puis on lance le programme en exécutant la commande suivante :

```
python main.py
```

Ce qui nous rend dans notre cas :

```
Multiplication table of 7
1*7=7
2*7=14
3*7=21
4*7=28
5*7=35
```

```
6*7=42
7*7=49
8*7=56
9*7=63
10*7=70
```

Exercice 1. Écrire un programme indépendant qui affiche les 100 premiers nombres premiers. On écrira en particulier une fonction `prime` prenant un entier en paramètres et retournant `True` si l'entier est premier et `False` sinon.

2 Fonctions d'input

On a déjà vu l'utilisation de la fonction `print` qui permet d'afficher une chaîne de caractères à l'écran :

```
print "Hello, World!"
```

Il existe son pendant pour récupérer des caractères tapés au clavier par l'utilisateur : c'est la fonction `raw_input`. Elle prend en paramètres une chaîne de caractères optionnelle et renvoie la chaîne de caractères entrée par l'utilisateur. Le mieux est encore de tester avec des petits exemples comme le suivant :

```
name = raw_input("What is your name? ")
print "Hello, "+name+"!"
```

Exercice 2. Écrire un programme qui demande à l'utilisateur deux nombres `n` et `m` et qui affiche le quotient et le reste de la division euclidienne de `n` par `m`.

Exercice 3. Écrire un programme qui demande à l'utilisateur une durée en heures-minutes-secondes au format `hh:mm:ss` et qui renvoie la même durée convertie en secondes. Indice : on utilisera la **méthode** `split` des `string`.

3 Mini-jeux

Exercice 4. Écrire un jeu du *plus ou moins*. Le jeu se déroule comme suit :

(1) un nombre `bound` est fixé à l'avance,

(2) un nombre de coups `chances` est fixé à l'avance,

(3) le programme génère un nombre aléatoirement entre `0` et `bound`,

(4) l'utilisateur est invité à entrer un nombre et le programme indique si le nombre cherché est plus grand ou plus petit (ou égal),

(5) on répète cette instruction jusqu'à ce que le joueur trouve le nombre (et dans ce cas il gagne) ou que le nombre de coups soit écoulé (et dans ce cas il perd).

Exercice 5. Améliorer le programme en proposant différent niveau à l'utilisateur. On pourra par exemple afficher un menu en début de programme de la forme :

```
1. Level 1 -- number between 0 and 10 -- 5 chances
2. Level 2 -- number between 0 and 100 -- 20 chances
3. Level 3 -- number between 0 and 100 -- 5 chances
4. Level 4 -- number between 0 and 1000 -- 10 chances
Level (enter corresponding number):
```

Exercice 6. Améliorer encore un peu le jeu en ajoutant une dernière option au menu où on laisse le joueur libre de rentrer lui même les nombres `bound` et `chances`.

Pour les exercices suivants, on aura besoin de fonctions `python` qui permettent d'ouvrir un fichier et de le lire ligne par ligne. La fonction `open('filepath','r')` permet d'ouvrir un fichier en *lecture seule* (remplacer `r` par `w` pour l'ouvrir en mode écriture) : c'est-à-dire qu'il retourne un objet `python` représentant le fichier dont le chemin d'accès est `filepath`. On peut ensuite utiliser la méthode `readline()` qui renvoie la ligne courante du fichier et passe à la suivante. Quand il n'y a plus de ligne à lire, la méthode renvoie simplement une chaîne de caratères vide. Une fois le contenu du fichier lu, il faut utiliser la méthode `close` pour fermer le fichier.

Pour éclaircir tout cela, considérons l'exemple suivant :

```
f = open("words.txt", 'r')

line = f.readline() # read the first line
while line != "":
    print line
    line = f.readline() # and all the other ones

f.close()
```

Ce programme lit ligne par ligne (et les affiche à l'écran) le fichier `words.txt` situé dans le même dossier que le programme lui-même, puis ferme le fichier.

En fait, `python` nous autorise une syntaxe plus légère. Le programme suivant fait exactement la même chose :

```
f = open("words.txt", 'r')

for line in f: # f is behaving as an iterable
    print line

f.close()
```

Exercice 7. Écrire une fonction de la forme

```
def number_of_lines(filepath):
```

qui compte le nombre de ligne du fichier dont le chemin d'accès est la chaîne de cractères `filepath`.

Pour l'exercice qui suit, nous aurons besoin de tirer un nombre au hasard. Cela se fait avec la fonction `randint(start,stop)` de la bibliothèque `random` qui renvoie un entier aléatoire situé (au sens large) entre `start` et `stop`. Exemple :

```
import random

r = random.randint(19,42)
print str(r)+" is so random!"
```

Remarquez qu'il faut importer la bibliothèque tout au début du fichier `python` avec la commande `import`. Puis on utilise une fonction `fun(...)` de `random` en l'appelant par `random.fun(...)`.

Exercice 8. Écrire un jeu du pendu. Le fichier de mots situé à l'adresse suivante contient un mot (anglais) par ligne :
`http://www.normalesup.org/~cagne/teaching/20162017/algo_proj_m1/words.txt`
Le programme devra ouvrir le fichier, y tirer au sort un mot (i.e. une ligne) et réaliser le classique jeu du pendu avec ce mot.

Attention : dans le jeu du pendu, on peut à chaque tour, soit poposer une lettre, soit directement le mot entier. Il faudra donc trouver un moyen de savoir ce que l'utiliateur choisit de proposer à chaque tour.

Exercice 9. On peut améliorer le jeu du pendu de bien des façons : proposer au joueur de rejouer après une partie, ne sélectionner que des mots de longueur raisonnable (disons entre 4 et 10 lettres), proposer à l'utilisateur de fixer un nombre de chances au début (plutôt que les 6 coups arbitraire), dessiner le pendu en ASCII au fur et à mesure, etc. N'hésitez pas à en faire un vrai mini-jeu.

4 Arguments de la ligne de commande

Sous système UNIX-like (GNU/Linux, BSD, Solaris, etc.), on peut appeler un programme avec des arguments. Par exemple `firefox "www.google.fr"` ouvre firefox directement à l'adresse demandée. On peut faire pareil avec nos programmes `python` !

```
python prog.py arg1 arg2 arg3
```

Encore faut-il savoir comment exploiter ces arguments...

C'est là qu'interviennent la variable `argv` de la bibliothèque `sys` : c'est une liste des arguments passés au programme. L'élément d'index 0 de `argv` existe toujours car `python`[1] considère que le nom du programme exécuté est le premier argument. Dans l'exemple ci-dessus, on a donc `"prog.py"` dans `argv[0]`. Puis ceux d'index 1, 2, 3, etc. sont les arguments passés au programme dans l'ordre où l'utilisateur les a entrés (`arg1` puis `arg2` puis `arg3` dans l'exemple ci-dessus). Rendons les choses un peu plus claires avec le programme suivant :

```
import sys

i=0
for arg in sys.argv:
    print "arg "+str(i)+": "+arg+" "+str(type(arg))
    i +=1
```

En entrant la commande

```
  python argvexample.py foo bar 42 3.14159
```

on ontient alors

```
arg 0: argvexample.py <type 'str'>
arg 1: foo <type 'str'>
arg 2: bar <type 'str'>
```

1. En fait, c'est l'OS que `python` appelle via `sys` qui considère cela.

```
arg 3: 42 <type 'str'>
arg 4: 3.14159 <type 'str'>
```

Remarque. Remarquons en particulier que **tous** les arguments passés en ligne de commande sont de type `str`, y compris les arguments numériques. Dans l'exemple précédent, l'argument numéro 3 est `"42"` et non `42`. Si on veut l'utiliser en tant qu'entier, il faudra le convertir avec la fonction `int()` de `python`.

Exercice 10. Modifier votre jeu du pendu pour donner à l'utilisateur la possibilité de passer en paramètre le chemin d'accès vers un dictionnaire de son choix (par exemple un dictionnaire français). Le jeu se lancera alors avec une commande de la forme :

```
python hangman.py /home/username/mondicoperso
```

Exercice 11. Si l'utilisateur utilise un dictionnaire d'une langue accentuée (comme le français), le jeu du pendu peut vite devenir impossible. Modifier votre programme pour que celui-ci ne propose que des mots non accentués (ou qu'il désaccentue le mot tiré au hasard).

Exercice 12. Aller encore plus loin en proposant plusieurs paramètres en ligne de commande. Votre programme pourra alors se lancer avec un commande de la forme :

```
python hangman.py --dict /home/username/mondico --chances 9
```