Séance 1b: L'APPRENTI-(SORCIER|PROGRAMMEUR)

Université Paris-Diderot

Objectifs:

- Exécuter un programme.
- Apprendre à lire des programmes.
- Identifier les mécanismes de la programmation impérative.
- Comprendre ce que fait un programme.
- Modifier un programme existant.

Rappel : Tous les fichiers que vous sauvegarderez ou créerez pour ce TP devront être mis dans le répertoire `TP1b` sous-répertoire du répertoire `IP1-Python` que vous avez créé au TP précédent.

Avant d'apprendre à écrire des programmes, nous allons commencer par en étudier quelques-uns déjà écrits. Pour chacun des programmes, vous allez successivement : les lire, imaginer ce qu'ils font, les exécuter et enfin, les modifier légèrement. En étudiant ces programmes, vous allez découvrir la plupart des mécanismes de programmation impérative étudiés ce semestre : les affectations, les instructions conditionnelles, les boucles, les fonctions et les procédures. Ces travaux pratiques ne sont qu'une première expérimentation : nous reviendrons en détails sur chacun des mécanismes au fur et à mesure du semestre. Il est donc tout à fait normal que vous n'en compreniez pas encore tous les détails.

Exercice 1 (Hello World !, ⋆)

Le fichier `helloworld.py` *contient le code source d'un programme qui est traditionnellement le premier programme que l'on écrit lorsque l'on apprend un nouveau langage de programmation*[1].

1. *Lisez le code source du programme, que fait-il selon vous ? Vérifiez votre hypothèse en exécutant la commande suivante dans un terminal :*

```
python3 helloworld.py
```

2. *Quelles sont les parties du code source qui correspondent à des commentaires ? Quelles sont les parties du code source qui correspondent à du code* PYTHON *? De combien d'instructions est composé ce programme ? Quel est le rôle de l'instruction* `print` *?*
3. *Remplacez l'expression « `"Hello World!"` » par « `"Hello" + " " + "World!"` » puis exécutez le programme, est-ce que le comportement du programme a changé ? Que fait l'opérateur + selon vous ?*
4. *Rajoutez les deux lignes suivantes tout au début du programme :*

```
print ("What is your name?")
name = input ()
```

1. D'où vient cette tradition ? La réponse ici : http://blog.hackerrank.com/the-history-of-hello-world/

et modifiez l'expression « `"Hello" + " " + "World!"` » en « `"Hello" + " " + name + "!"` ». Que fait cette nouvelle version du programme selon vous ? Que fait la fonction `input` *? À quoi sert la variable* `name` *? L'instruction « `name = input ()` » affiche-t-elle quelque chose sur le terminal ? Si non, quel est l'effet de cette instruction ? Vérifiez vos hypothèses en exécutant de nouveau le programme. Écrivez des commentaires avant chacune des lignes que nous venons de rajouter pour la paraphraser avec vos mots.*

5. *Modifiez le programme pour qu'il demande le nom et le prénom de l'utilisateur puis affiche :*

```
Hello prenom nom!
```

6. *Modifiez maintenant votre dernière instruction en la remplaçant par deux instructions comme suit :*

```
print (...)
```

devient :

```
myfile = open("helloyou.txt", mode="w")
print (..., file = myfile)
```

Quel est maintenant l'effet de l'instruction `print` *?*

□

Exercice 2 (Le devin, ⋆⋆)

Le fichier `devin.py` *contient le code source d'un programme qui devine un nombre entre 0 et 100 choisi par l'utilisateur en lui posant des questions.*

1. *Lisez attentivement le code source du programme (son code* PYTHON *mais surtout ses commentaires). Une fois que vous l'avez lu, exécutez-le plusieurs fois grâce à la commande :*

```
python3 devin.py
```

en essayant de saisir des séquences de réponses sollicitant toutes les instructions du programme.

2. *Quelles valeurs successives prennent les variables* `min_number`, `max_number` *et* `candidate` *si le nombre à chercher est 73 ? Pour vérifier votre réponse, insérez au bon endroit dans le code source, les instructions suivantes qui affichent la valeur de ces trois variables :*

```
print ("min = " + str (min_number))
print ("candidate = " + str (candidate))
print ("max = " + str (max_number))
```

3. *Modifiez le programme pour que le Devin cherche un nombre entre 0 et 1000 plutôt qu'entre 0 et 100. Exécutez le programme pour tester ce changement.* [2]
4. *Modifiez le programme pour que le Devin ne demande la validation du nombre que s'il l'a trouvé avec certitude (Indice : L'utilisateur ne demandera que le si le nombre est correct si* `max_number` *et* `min_number` *ont la même valeur, dans les autres cas, il demandera seulement si le nombre est plus grand que* `candidate`*) .*
5. *Remplacez l'instruction*

```
candidate = min_number + ((max_number - min_number) // 2)
```

par :

```
candidate = min_number
```

puis exécutez le programme. Est-ce que le programme fonctionne toujours ? Que pensez-vous de ce changement ?

2. Est-ce que le Devin met dix fois plus de temps pour trouver le nombre ?

6. *Que se passe-t-il si l'utilisateur répond mal à une question en confondant un "O" et un "N" ?*

□

Exercice 3 (L'artiste, ⋆⋆)

Le fichier `city.py` *contient un programme qui dessine les rues d'une ville et quelques maisons. Pour cela, il utilise une bibliothèque de fonctions de dessin inspirées du langage LOGO*[3]*, un des premiers langages conçus pour apprendre la programmation.*

1. *Lisez attentivement le code source du programme (son code* Python *mais surtout ses commentaires). Une fois que vous l'avez lu, exécutez-le grâce à la commande :*

```
python3 city.py
```

   Au fur et à mesure de l'affichage, mettez en correspondance les actions de la tortue et les instructions du programme. Avez-vous remarqué que vous pouvez modifier la vitesse de la tortue ?

2. *Modifiez la ligne de code qui permet de répartir horizontalement les maisons dans une rue de façon à y insérer une nouvelle maison.*

3. *Remplacez l'instruction :*

```
draw_house (100, xpositions[i], y)
```

   par

```
draw_house (10 + 100 * i / len(xpositions), xpositions[i], y)
```

   Que se passe-t-il ? Pourquoi ?

4. *En vous inspirant de la procédure* `walls`*, écrivez une procédure* `door` *qui dessine une porte dans la maison, et une procédure* `windows` *qui dessinent des fenêtres.*

□

Exercice 4 (Morpion, ⋆⋆⋆)

Le fichier `tictactoe.py` *contient un programme qui permet à deux utilisateurs de faire une partie de morpion.*

1. *Lisez attentivement le code source du programme (son code* Python *mais surtout ses commentaires). Une fois que vous l'avez lu, exécutez-le grâce à la commande :*

```
python3 tictactoe.py
```

   et faites plusieurs parties de façon à tester les cas où `X` *gagne,* `O` *gagne ou bien le jeu se termine sur une égalité.*

2. *Dans ce programme, c'est toujours le joueur des croix qui commence à jouer. Modifiez le programme pour que le joueur des cercles débute.*

3. *Pour mettre un peu de piment dans le jeu, on souhaite qu'à chaque tour on puisse tirer à pile ou face celui qui commence. Pour cela, il suffit d'utiliser l'expression* `random.randint (0, 1)` *qui tire un nombre entier au hasard dans l'intervalle* $[0, 1]$*. (Il faut insérer une instruction "*`import random`*" pour l'utiliser.)*

4. *(Bonus) Modifiez le programme pour que la grille contienne 4 lignes de 4 cases. Il faudra maintenant aligner 4 cases pour gagner...*

5. *(Bonus) Modifiez le programme qu'il affiche des suggestions de coup au joueur qui s'apprête à jouer.*

□

3. http://el.media.mit.edu/logo-foundation/what_is_logo/history.html