Pablo Rauzy
pr@up8.edu
https://pablo.rauzy.name/teaching/mdlp

UNIVERSITÉ PARIS 8 VINCENNES-SAINT-DENIS

Méthodologie de la programmation

Université Paris 8 – Vincennes à Saint-Denis
UFR MITSIC / L1 informatique

Séance 1 (TP) : Terminal, shell, éditeur de texte ; Premiers pas avec Python

N'oubliez pas :
- Les TPs doivent être rendus par courriel au plus tard le lendemain du jour où ils ont lieu avec "[mdlp]" suivi du numéro de la séance et de votre nom dans le sujet du mail, par exemple "[mdlp] TP1 Rauzy".
- Quand un exercice demande des réponses qui ne sont pas du code, vous les mettrez dans un fichier texte `reponses.txt` à rendre avec le code.
- Le TP doit être rendu dans une archive, par exemple un tar gzippé obtenu avec la commande `tar czvf NOM.tgz NOM`, où `NOM` est le nom du répertoire dans lequel il y a votre code (idéalement, votre nom de famille et le numéro de la séance, par exemple "rauzy-tp1").
- Si l'archive est lourde (> 1 Mo), merci d'utiliser `https://bigfiles.univ-paris8.fr/`.
- Les fichiers temporaires (si il y en a) doivent être supprimés avant de créer l'archive.
- Le code doit être proprement indenté et les variables, fonctions, constantes, etc. correctement nommées, en respectant des conventions cohérentes.
- Le code est de préférence en anglais, les commentaires (si besoin) en français ou anglais, en restant cohérent.
- **N'hésitez jamais à chercher de la documentation par vous-même sur le net !**

Dans ce TP :
- Se familiariser avec l'environnement informatique des salles de TP (terminal, shell, éditeur de texte).
- Écrire et exécuter du code Python.
  Documentation de Python : `https://docs.python.org/3/`.

Exercice 0.

Un *terminal* (ou *émulateur de terminal*, ou *console*) est un logiciel qui sert de point d'accès de communication entre l'humain et la machine. Le plus souvent, on entend par terminal une fenêtre dans laquelle est lancé un *shell*.

Un *shell* est un interpréteur de commande. On lui envoie des commandes au clavier, le plus souvent un programme à exécuter et les arguments à lui donner en entrée, et le shell exécute la commande et sert au programme à la fois d'*entrée standard* et de *sortie standard* (c'est à dire que par défaut le programme va lire ce qu'on tape dans le shell et ce qu'il va écrire sera affiché dans le shell).

Dans cet exercice on va voir comment utiliser quelques commandes de base pour explorer l'*arborescence* de *répertoires* et de *fichiers*.

1. → Ouvrir un terminal.
2. Le programme `echo` affiche simplement sur sa sortie standard ce qu'on lui a donné en argument.
   → Exécuter la commande `echo "coucou"`.
3. Par défaut, vous êtes dans votre *répertoire maison*, c'est la *racine* de la partie du système de fichier qui appartient à votre compte.
   On appelle le répertoire courant le *répertoire de travail*, ou *working directory* en anglais. La commande `pwd` (pour "Print Working Directory") affiche le *chemin* de ce répertoire.
   Un *chemin* contient des noms de répertoire séparer par des `/`, chaque `/` veut dire que l'on entre dans un sous-répertoire. Dans chaque répertoire il y a deux répertoires cachés :
   - `.` est le répertoire lui-même, et
   - `..` est le répertoire parent, c'est à dire celui qui est juste au dessus dans l'arborescence.

   Le dernier élément d'un chemin peut être un fichier qui n'est pas un répertoire (comme par exemple un fichier texte, un fichier audio, etc.).
   Si un chemin commence par un `/`, on dit qu'il est *absolu*, c'est à dire qu'il commence à la racine du système de fichier.
   Sinon, on dit qu'il est *relatif*, et il commence dans le répertoire de travail.

   (a) → Afficher le chemin de votre répertoire de travail avec la commande `pwd`.
   (b) → Quel est le chemin absolu du répertoire parent de votre répertoire maison ?
4. Il existe un raccourci pour l'adresse de votre répertoire maison, c'est `~` (tilde). C'est ce que vous voyez dans votre *invite de commande* (ou *prompt*).
   Par défaut, le prompt affiche votre login, puis un `@`, puis le nom de la machine sur laquelle vous êtes connecté, puis un `:`, puis le répertoire de travail, et enfin un `$` qui signifie que vous pouvez entrer une commande.
   → Vérifier que la commande `echo ~` donne bien la même chose que `pwd`.

5. Pour afficher la liste des fichiers présent dans un répertoire, la commande est `ls` *`chemin-du-répertoire`* (pour "List"). Par défaut, `ls` liste les fichiers du répertoire de travail.
   La commande `ls` accepte en argument des *options*, par exemple :
   — `--all` ou `-a` permet de lister aussi les fichiers cachés (ceux qui commencent par un `.`),
   — `-l` donne plein d'informations en plus sur chaque fichier (on ne s'attardera pas sur leur signification).
   → Afficher la liste des fichiers présent dans votre répertoire maison, puis afficher cette même liste mais avec les fichiers cachés.
6. Pour créer un répertoire, on utilise la commande `mkdir` *`chemin-du-répertoire`* (pour "MAKe DIRectory").
   → Créer un répertoire `mdlp` dans lequel on mettra tout ce qui est lié à ce cours. Vérifier avec `ls` que le répertoire est bien là.
7. On peut "aller" dans un répertoire avec la commande `cd` *`chemin-du-répertoire`* (pour "change Directory").
   (a) → Aller dans le répertoire `mdlp`.
   (b) → Afficher tout le contenu (y compris caché) du répertoire `mdlp`. Quelle commande permet de revenir dans le répertoire parent ?
   (c) → Dans le répertoire `mdlp`, créer un répertoire pour le travail de ce TP (vous pouvez l'appeler comme vous voulez, par exemple `tp1`) puis aller dedans.
8. La commande `rmdir` *`chemin-du-répertoire`* (pour "REMove DIRectory") permet de supprimer un dossier, mais refuse par sécurité de le faire si celui-ci n'est pas vide.
   → Créer un dossier `foo` et dedans un dossier `bar`, puis tenter de supprimer `foo` directement, puis faire ce qu'il faut pour supprimer `foo`.
9. Pour créer un fichier, il suffit d'ouvrir un fichier qui n'existe pas. Par exemple, on peut lancer l'éditeur de texte Mousepad et lui demander d'ouvrir le fichier `essai.txt` avec la commande `mousepad essai.txt`.
   Avant de poursuivre, merci de quitter complètement Mousepad (fermer toutes les instances ouvertes du logiciel).
   (a) → Lancer Mousepad sur le fichier `essai.txt`, et écrire par exemple "coucou" dans le fichier, et l'enregistrer.
   (b) → Revenir sur le terminal sans quitter Mousepad, que constatez-vous ?
   (c) → Quitter Mousepad et revenir sur le terminal.
10. Pour lancer une commande en *arrière plan* on peut la suffixer avec `&`.
    (a) → Relancer Mousepad sur le même fichier mais cette fois-ci en arrière plan.
    (b) → Revenir sur le terminal sans quitter Mousepad, que constatez-vous ?
11. On peut afficher le contenu d'un fichier sur la sortie standard avec la commande `cat` *`chemin-du-fichier`* (pour "CATenate" (parce qu'elle peut prendre plusieurs fichiers d'un coup)).
    → Afficher le contenu de `essai.txt`, modifier le dans mousepad, puis le réafficher.
12. On peut supprimer des fichiers avec la commande `rm` *`chemin-du-fichier`* (pour "REMove").
    Attention la suppression est définitive, ça ne va pas dans une "corbeille".
    → Supprimer le fichier `essai.txt`.
13. Pour avoir de l'aide sur une commande, il est possible de consulter son manuel d'utilisation avec la commande `man` *`commande`* (pour "MANual").
    → Consulter rapidement les manuels de `echo`, `pwd`, `ls`, `mkdir`, `cd`, `mousepad`, `cat`, `rm`, et `man`.

Exercice 1.

Jouer avec Python.

1. Pour exécuter un programme écrit en Python, on fait appel à l'*interpréteur* Python. L'interpréteur Python est la commande `python3` (parce qu'une vieille version 2 du langage pas totalement compatible avec la 3 s'appelle encore `python`).
   Pour l'utiliser il suffit de lui passer en argument le fichier Python à exécuter.
   (a) → Recopier le code suivant dans un fichier `hello.py` :

```python
print("Hello, world!\n")
```

   (b) → Lancer le programme Python `hello.py`.
2. Écrire un programme qui compte de 0 à 100, mais remplace les nombres divisibles par 3 par "Fizz", ceux divisibles par 5 par "Buzz", et ceux qui le sont à la fois par 3 et par 5 par "FizzBuzz". Ne recopiez pas celui du cours, essayez de le refaire par vous même.

(a) → Écrire le script Python `fizzbuzz.py`.

(b) → Tester le code pour vous assurer qu'il fonctionne comme attendu.

Exercice 2.

Un petit jeu.

1. Écrire une fonction qui prend un entier en argument, choisi un nombre aléatoire entre 0 et cet argument, et le fait deviner au joueur ou à la joueuse en lui donnant un indice après chaque tentative.

   Pour générer un nombre aléatoire entre a et b, vous avez besoin de la fonction `random.randint(`a`, `b`)`. Pour utiliser cette fonction, vous devez importer le module `random` de Python dans votre programme. Vous pouvez faire cela avec la directive `import random` en début de fichier.

   Pour lire une entrée clavier, pour pouvez utiliser la fonction `input()`, qui prend en argument une chaîne de caractère qui sera affichée, et qui retournera ce que la personne a tapé. Par exemple `nom = input("Nom : ")` affichera "Nom : " et stockera dans la variable `nom` le texte entré au clavier.

   Pour lire un entier, il faut convertir l'entrée clavier avec la fonction `int()`, cela peut être fait directement en sorti de `input()`, c'est à dire qu'on peut faire `int(input("Âge : "))`.

   Pour au contraire manipuler un nombre comme une chaîne de caractères (par exemple pour le concaténer avec l'opérateur `+` a une autre chaîne de caractères), on peut utiliser la fonction `str()`

   → Créer un programme Python `guessing_game.py` avec dedans la fonction `guess_the_number`.

2. Quand le nombre est correctement deviné par le joueur ou la joueuse, on voudrait lui afficher le nombre de coups que ce·tte dernier·e a dû utiliser.

   → Modifier la fonction `guess_the_number` en conséquence, et lui faire retourner cette valeur.

3. On voudrait proposer au joueur ou à la joueuse de faire plusieurs parties.

   En dehors de la fonction, on veut que le programme demande le nombre maximum à faire deviner, si celui ci est 0 alors il s'arrête, sinon il lance le jeu (c'est à dire appelle la fonction `guess_the_number`) avec ce nombre comme maximum.

   → Modifier le programme `guessing_game.py` pour implémenter ce comportement.

4. Maintenant, on voudrait afficher un résumé des parties quand le jeu est quitté : pour chaque partie jouée, on veut afficher le nombre maximum à deviner et combien de coups il a fallu pour trouver le bon nombre.

   → En utilisant une liste (un élément par partie) de dictionnaire (avec une entrée pour le maximum et une pour le nombre de coups), implémenter le comportement souhaité dans le programme `guessing_game.py`.