TP 1.2 (3h): Gestion des processus

Partie 1 : Passage à Python

L'objectif de cette partie est de se familiariser avec le langage Python (v3) utilisé pour la suite des TP.

***Environnement** : Vous avez le choix entre lancer les scripts dans un terminal (avec la commande python3) ou utiliser un IDE comme PyCharm (attention à créer un nouveau "projet" par exercice et à configurer l'interprète python3 dans les options du projet.*

***A savoir** : La plupart des appels système Unix/POSIX sont implémentés également dans ce langage, principalement dans le module **os**. Il convient donc de démarrer vos programmes avec la clause suivante :*

from os import *

***Documentation** : https://docs.python.org/3.5/*

Ex. 1. Premiers pas : arguments du programme, saisie clavier et affichage, manipulation de chaînes

Dans cet exercice on utilisera les fonctions fournies par Python pour manipuler des chaînes de caractères. Implémenter un programme permettant de :

- Récupérer un argument lors de l'appel du programme. Cet argument est une chaîne de caractères. Affichez-le.

  Attention :
  - *il faudra importer le module **sys**. Consulter la doc, en particulier **sys.argv***
  - *un seul argument sera permis; il peut cependant comporter des espaces !*
- Demander à l'utilisateur de saisir au clavier une deuxième chaîne
- Concaténer les deux chaînes
- Afficher le résultat
- Afficher la longueur de la chaîne concaténée.
- Découper la chaîne mot par mot *(avec la méthode **split**)*.
  A l'aide d'une boucle, afficher les mots en ajoutant le symbole « & » entre chaque mot. Tous les mots doivent être sur la même ligne.

```python
from os import *
import sys

def main():
    if len(sys.argv) != 2:
        print("Un seul argument possible")
        exit(1)

    args = sys.argv[1]
    ch = str(input("Saisissez une chaine :"))
    concat = args + " " + ch
    print("La chaine concat :" + concat)
    print("La longueur de la chaine : " + str(len(concat)))
    spl = concat.split(sep=" ")
    for elm in spl:
        sys.stdout.write(elm + " & ")

if __name__ == '__main__':
    main()
```

Ex. 2. Manipulation de liste, de fonction et importation de module.
Dans cet exercice on manipule des listes et on définit des sous-programmes.

(a) Implémenter un programme permettant de remplir une liste de valeurs tirées aléatoirement entre 0 et 32. Pour cela, utilisez le module random et sa fonction randint (cf. **random.randint**). Le nombre de valeurs aléatoires générées doit être préalablement demandé à l'utilisateur. Afficher la liste.

```python
def main():
    nbVal = int(input("Saisissez un nombre e valeurs à tirer :"))
    li = tirerRandom(nbVal)
    print(li)

def tirerRandom(n):
    li = []
    for i in range(n):
        li.append(randint(0, 32))
    return li
```

(b) Ecrire un sous-programme pour parcourir la liste (qu'il reçoit en paramètre) et tester pour chaque élément si sa valeur est supérieure à un seuil. Le sous-programme renvoie une liste des éléments supérieurs au seuil. Le seuil est passé en paramètre du sous-programme et saisi au clavier dans le programme principal.

```python
def main():
    nbVal = int(input("Saisissez un nombre de valeurs à tirer :"))
    li = tirerRandom(nbVal)
    print(li)
    seuil = int(input("Saisissez un seuil :"))
    liSeuil = chercherValSeuil(li, seuil)
    print(liSeuil)

def tirerRandom(n):
    li = []
    for i in range(n):
        li.append(randint(0, 32))
    return li

def chercherValSeuil(li, seuil):
    tmp = []
    for elm in li:
        if int(elm) > seuil:
            tmp.append(elm)
    return tmp
```

(c) Ecrire un programme utilisant différentes fonctions de manipulation de listes python (**append(), insert(), remove(), pop(), count(), sort()**) pour modifier une liste

```python
li.pop(2)
li.remove(2) if 2 in li else 0
li.sort()
print("Nombre de valeurs dans la liste : " + str(li.count()))
```

(d) Tester les affichages produits par les syntaxes utilisant les *slices* python pour accéder au contenu d'un tableau/liste.

Exemples :

- ***a[start:end]*** *désigne les items de* ***start*** *à* ***end-1***
- ***a[start:]*** *désigne les items de* ***start*** *à la fin du tableau*
- ***a[:end]*** *désigne les items du début à* ***end-1***
- *les indices négatifs sont possibles et sont considérés relatifs à la fin du tableau*

Un ***slice*** *peut être utilisé aussi bien du côté droit que du côté gauche d'une affectation (pour lire ou pour écrire dans le tableau).*

```python
print(li[:2])
tmp = []
tmp[:-2] = li
print(tmp[1:5])
```

Ex. 3. : Exceptions et saisies contrôlées

Ecrire une fonction **input_int()** qui effectue une saisie contrôlée d'un entier (reboucle tant que la valeur saisie n'est pas numérique). Pour cela, la fonction devra utiliser le traitement des exceptions (**try...except**) lors de la conversion de la chaîne saisie en entier (voir l'exception **ValueError** dans la documentation Python).

Utiliser la fonction dans un programme qui permet de saisir la taille et le contenu d'une liste d'entiers et affiche la liste triée en ordre croissant.

```
def main():
    print("Saisissez la taille de la liste")
    taille = input_Int()
    li = []
    for i in range(taille):
        print("Saisissez une valeur")
        li.append(input_Int())
    li.sort()
    print(li)

def input_Int():
    val = input("Saisissez une valeur numérique : ")
    try:
        int(val)
    except ValueError:
        val = input_Int()
        return int(val)
    return int(val)
```

Partie 2 : Gestion des processus en Python

Ex. 4. Ecrire un shell simplifié en Python

Ecrire un programme Python (*myshell.py*) qui lit des lignes de commande à l'entrée standard et les lancent en exécution, à l'instar du *shell* standard Linux. Le programme ne doit pas utiliser la fonction *system,* l'exécution est lancée avec une fonction de la famille *os.exec*.
Pour simplifier le travail, nous allons faire les hypothèses suivantes :

- Une commande est formée d'une seule ligne. Seules les commandes *externes* (i.e., programmes exécutables avec arguments) seront traitées.
- Les arguments sont séparés par des espaces. Les caractères d'échappement (\), guillemets, etc. ne sont pas traités.
- Les opérateurs de combinaison de commandes (|, ||, &&, etc.) ou de redirection (<, >, etc.) ne sont pas pris en compte.
- En revanche, la ligne **peut se terminer par &** (séparé du dernier argument par un espace) ce qui produit un effet similaire au lancement en arrière-plan en *shell* standard.
- A la fin d'un processus lancé en premier plan (sans &), le shell affiche son PID et son code de retour

A noter : il convient de traiter les erreurs potentielles à chaque appel système (exception **OSError***) et afficher une description de la cause de l'erreur (cf.* **OSError.strerror** *)*

`

```python
def main():
    while(True):
        afficherMenu()
        cmd = str(input("Saisissez une commande : "))
        cmdSplit = cmd.split(sep=" ")
        background = True if cmdSplit[len(cmdSplit)-1] == "&" else False
        if background:
            cmdSplit.remove("&")
        try:
            pid = fork()
            if pid == 0:
                execvp(cmdSplit[0], cmdSplit[0:])
            else:
                if not background:
                    codeR = wait()
                    print ("Fin de " + str(codeR[0]) + " avec le code de retour : " + str(codeR[1]))
        except OSError:
            print(OSError.strerror)
```

A l'exécution, on obtient :

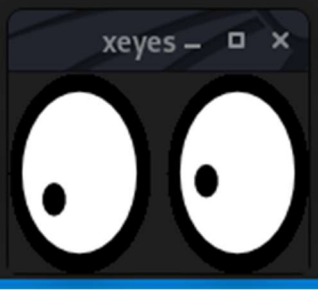
```
└─$ /bin/python3 /home/pack/PSE/TP1.2/myshell.py
Bienvenue dans mon Shell !
Saisissez une commande :
```

On saisit ls -l :

```
└─$ /bin/python3 /home/pack/PSE/TP1.2/myshell.py
Bienvenue dans mon Shell !
Saisissez une commande : ls -l
total 8
-rw-r--r-- 1 pack pack  873 Oct 20 11:41 myshell.py
-rw-r--r-- 1 pack pack 1587 Oct 18 11:02 TP1.2.py
Fin de 2826 avec le code de retour : 0
Bienvenue dans mon Shell !
Saisissez une commande :
```

On va maintenant saisir une commande en arrière-plan : xeyes &

```
└─$ /bin/python3 /home/pack/PSE/TP1.2/myshell.py
Bienvenue dans mon Shell !
Saisissez une commande : ls -l
total 8
-rw-r--r-- 1 pack pack  873 Oct 20 11:41 myshell.py
-rw-r--r-- 1 pack pack 1587 Oct 18 11:02 TP1.2.py
Fin de 2826 avec le code de retour : 0
Bienvenue dans mon Shell !
Saisissez une commande : xeyes &
Bienvenue dans mon Shell !
Saisissez une commande :
```



Le shell propose de saisir une autre commande alors que la dernière est en cours d'exécution. On va saisir xeyes :

Les deux xeyes sont lancer mais le dernier bloque la saisie de commande.

Après la fermeture des xeyes, on peut de nouveau écrire une commande :