Algorithmique PEIP2

TP1 : Prise en main de Python

Le contenu de ce TP est à implémenter en utilisant le langage Python. La syntaxe Python est très proche de celle du pseudo-code. Chaque instruction est renseignée sur une ligne différente, et lorsque l'on rentre dans le corps d'une instruction, on utilise des tabulations. Par exemple, le programme suivant définit une fonction `somme` qui calcule la somme des éléments d'une liste, puis l'exécute sur un exemple.

```
def somme(lst):
      res = 0
      for x in lst:
            res += x
      return res

lst = [1,2,3,4]
print(somme(lst))
>> 10
```

Afin de structurer le code produit au cours du TP, on séparera les définitions de fonction et les exécutions sur des valeurs test. Toutes les définitions de fonction se feront dans la première partie du code, et tous les tests se feront dans la seconde partie, au sein d'une instruction spéciale.

```
''' Definitions de fonctions '''

# Exercice 1
def fun1(arg1):
      res = 0
      # corps de la fonction
      return res

# Exercice 2
def fun2(arg1,arg2):
      # corps de la fonction


''' Tests '''
if __name__ == '__main__':
      arg1 = val1
      print("Le resultat de fun1 sur l'entree", arg1, "est", fun1(arg1))

      arg2 = val2
      print("Le resultat de fun2 sur l'entree", arg1, arg2, "est", fun2(arg1,arg2))
```

Le code produit pour ce TP est à enregistrer dans un fichier texte `td1.py`. Pour l'exécution, on rentre l'expression `python3 td1.py` dans un terminal. On peut également lancer un interpréteur Python dans un terminal, via l'instruction `Python3`, afin de tester la valeur de diverses expressions Python.

1 Expressions

Le langage Python définit un certain nombre d'opérateurs sur les nombres, qui peuvent être indifféremment entiers ou flottants. D'autres opérateurs pouvant être utilisés sur des variables aux types variés permettent de faire des tests, en renvoient une valeur booléenne.

1. Quelles sont les priorités entre opérateurs sur les nombres (tester plusieurs expressions directement dans l'interpréteur pour répondre) ?

Opérateur	Opération
**	puissance
+ (unaire)	nombre positif
- (unaire)	changement de signe
*	multiplication
/	division réelle (sans reste)
//	quotient de la division euclidienne
%	*modulo* ou reste de la division euclidienne
+ (binaire)	addition
- (binaire)	soustraction

(a) Les opérateurs sur les nombres (entiers et flottants)

Opérateur	Sens
>	strictement supérieur à
<	strictement inférieur à
>=	supérieur ou égal à
<=	inférieur ou égal à
==	égal à
!=	différent de
in	appartenant à
not in	n'appartenant pas à

(b) Les opérateurs renvoyant un booléen

2. *On accède à une entrée utilisateur via un appel à* `input()`*, qui renvoie la saisie dans une chaîne de caractères. On affiche un résultat via la fonction* `print`*. On peut lui donner autant d'arguments que ce que l'on souhaite afficher, peu importe leur type (chaque type a accès à une procédure d'affichage qui est automatiquement invoquée par la fonction* `print`*) ; par défaut ils seront séparés d'une espace et l'affichage se termine par un saut de ligne (on peut changer ça en ajoutant* `sep=...` *et/ou* `end=...` *en argument).*

   Écrire un programme qui fait les présentations avec l'utilisateur : il lui demande son nom, son âge, et réagit à ces réponses.

2 Boucles `for`

En Python, les boucles `for` (équivalentes aux boucles `Pour` du cours) fonctionnent en énumérant les éléments d'une structure de données. Si on cherche à itérer sur les entiers entre 0 et $n-1$, on utilisera la structure donnée par `range(n)`. Pour itérer entre i_0 et $n-1$, on fera appel à `range(`i_0`,n)`. Enfin, si on souhaite en plus avoir un pas de r, on fera appel à `range(`i_0`,n,r)`.

```
for i in range(3,17,2):
    print(i,end=" ")
>> 3 5 7 9 11 13 15
```

1. Définir une fonction `SommeCarrés(n)` qui utilise une boucle `for` afin de renvoyer la somme des n premiers carrés parfaits $1^2 + 2^2 + \cdots + n^2$.

2. *En Python, la fonction* `len` *prend en entrée une structure de donnée non élémentaire (liste, chaîne de caractères, ...) et renvoie sa taille. On accède au i-ème élément d'une chaîne de caractères* `s` *par un appel à* `s[i]`*, et on peut accéder au dernier élément par un appel à* `s[-1]`*.*

   Définir une fonction `EstPalindrome(s)` qui prend en entrée une chaîne de caractères `s`, et qui renvoie `Vrai` s'il s'agit d'un palindrome (`s` peut se lire à l'endroit et à l'envers) et `Faux` sinon.

3. *En Python, la liste vide s'écrit* `[]`*, et pour ajouter un élément* `x` *à une liste* `lst`*, on fait un appel à* `lst.append(x)`*. Une liste* `lst` *en Python est en fait un* tableau dynamique*, une structure de donnée qui permet d'insérer ou supprimer des éléments (comme pour une liste), tout en ayant un accès direct à chacune de ses cases* `lst[i]` *(comme pour un tableau).*

   Définir la fonction `occurences(x,tab)` qui renvoie la liste des positions donc la valeur est `x` dans le tableau `tab`.

3 Jeu du nombre mystère

Le langage Python a accès à un certain nombre de modules qui fournissent des fonctions pré-implémentées. Par exemple, le module `random` nous fournit la fonction `random()` qui renvoie un flottant aléatoire compris entre 0 et 1. Afin de pouvoir utiliser cette fonction, il faut l'importer en préambule du fichier de code.

```
from random import random
```

1. *En Python, on peut transformer la valeur d'une variable `x` en entier via un appel à `int(x)`. Si `x` a une valeur flottante, cela tronque sa valeur à l'entier inférieur.*

   En utilisant la fonction `random`, définir une fonction `EntierAleatoire(n)` qui renvoie un entier aléatoire compris entre 0 et n.

2. *La boucle `Tq` est implémentée en Python de la sorte.*

   ```
   i=0
   while i<100:
         print(i)
         i=i*2
   ```

   Il n'existe pas de boucle `Répéter` en Python, il faudra donc trouver des solutions alternatives pour les émuler.

   Définir une fonction `EntierMystère(n)` qui tire aléatoirement un entier mystère `x` compris entre 0 et n à faire deviner à un utilisateur. Cet algorithme demande de façon répétée à l'utilisateur de rentrer une valeur jusqu'à ce qu'il ait deviné la valeur `x` auquel cas il le félicite et arrête le jeu, et en précisant si la valeur à deviner est supérieure ou inférieure à celle proposée en cas d'échec.

4 Opérations sur les matrices

Pour initialiser un tableau `tab` de taille n à 0, on peut faire appel à deux instructions équivalentes.

```
tab = n * [0]
tab = [0 for i in range(n)]
```

Les choses se corsent pour initialiser une matrice $n \times n$, car si on la définit comme un tableau contenant n copies de `tab` avec l'instruction `mat = n*[tab]`, alors les lignes de la matrices sont liées entre elles : modifier une case d'une liste change la valeur de la même case de toutes les lignes. Pour éviter cela, on peut utiliser l'instruction suivante.

```
mat = [n*[0] for i in range(n)]
```

1. Définir une fonction `Add`, qui calcule la somme de deux matrices (tableaux bidimentionnels) A et B qu'on suppose de même taille. Quelle est la complexité de cette fonction, en fonction de la taille de A et B?

2. Définir une fonction `Mult`, qui calcule le produit de deux matrices carrées A et B. Quelle est la complexité de votre fonction ?

3. Définir une fonction récursive `Puissance`, qui prend en entrée une matrice A supposée carrée et un entier n, et retourne la matrice A^n. Pour cela, on fera plusieurs appels à la fonction `Mult`, et on utilisera une exponentiation rapide, qui repose sur le principe suivant :

$$A^n = \begin{cases} \left(A^{n/2}\right)^2 & \text{si } n \text{ est pair ;} \\ A * \left(A^{\lfloor n/2 \rfloor}\right)^2 & \text{si } n \text{ est impair.} \end{cases}$$

   Combien d'appels à `Mult` fait-on dans le pire cas ? Quelle est la complexité de `Puissance` dans le pire cas ?

5 Recherche dichotomique

1. Définir une fonction `EstPrésent` qui prend comme paramètre une valeur `x` et un tableau `tab` trié dans l'ordre croissant, et qui renvoie `Vrai` si `x` est présent dans `tab`, et `Faux` sinon, en utilisant une recherche dichotomique.

   Le principe de la recherche dichotomique consiste à comparer l'élément recherché `x` avec l'élément central `y`. S'ils sont égaux, on renvoie `Vrai`. Si $x < y$ on continue la recherche dichotomique sur la moitié gauche de `tab`, et si $x > y$ on continue la recherche dichotomique sur la moitié droite de `tab`.

2. Quelle est la complexité dans le pire cas de la recherche dichotomique ?