Algorithmes et Pensée Computationnelle

Structure de données, Itération et récursivité - Avancés

Le but de cette séance est de mettre en pratique les connaissances apprises en cours. Elle portera essentiellement sur les structures de données qui seront utilisées lors des prochaines séances de cours/Travaux Pratiques. Les structures de données abordées lors de cette séance sont les tuples, les listes et les dictionnaires. Lors de cette séance, nous aborderons aussi les notions d'itération et de récursivité.
Au terme de cette séance, l'étudiant sera capable de distinguer une structure de données immuable et non immuable, écrire un programme de façon simplifiée en utilisant les notions d'itération et de récursivité.
Le code présenté dans les énoncés se trouve sur Moodle, dans le dossier "Exercices > Code". Le temps mentionné (🕒) est à titre indicatif.

1 Structures de données

Dans la première partie de cette section, les exercices devront être traités en Python. Dans la seconde partie, les exercices seront traités en Java.

1.1 Tuples

Pour rappel, les tuples sont des listes d'éléments immuables, ce qui signifie que ces listes ne peuvent pas être modifiées. L'une des particularités des tuples est la possibilité d'y inclure des données de types différents. Les tuples sont utiles pour stocker des données que l'on va réutiliser plus tard.
En Python, pour créer un tuple, il suffit de définir une variable et de lui assigner des valeurs entre parenthèses et séparer les valeurs entre elles par des virgules :

Exemple

```
1    mon_tuple = (1,"une chaîne de caractères",3,4)
```

Question 1: (🕒 *5 minutes*) **Manipulation d'un tuple**
Créez un tuple nommé **mon_tuple** contenant les chiffres 1,2,3,4 et 5. Obtenez le nombre d'éléments contenus dans votre tuple et stockez le résultat dans une nouvelle variable nommée **taille_tuple**. Pour finir, affichez le contenu de **taille_tuple**.

Conseil

Pour calculer la taille d'un tuple, ou d'une liste, vous pouvez utiliser la fonction **len**().

Solution

```
1  mon_tuple = (1,2,3,4,5)
2  taille_tuple = len(mon_tuple)
3  print(taille_tuple)
```

1.2 Listes

La différence entre une liste et un tuple est que l'on peut modifier les éléments d'une liste. Par exemple, il est possible de remplacer un élément d'une liste par un autre tandis qu'il est impossible de le faire avec un tuple.
Pour créer une liste, il suffit de définir une variable avec des valeurs entre crochets :

Exemple

```
1    ma_liste = [1,2,3,4,5]
```

Question 2: (*5 minutes*) **Manipulation d'une liste**
Créez une liste nommée **ma_liste** contenant les nombres 1,2,3,4 et 5. Vérifiez si le chiffre 6 est dans votre liste. S'il y est, affichez "OK", s'il n'y est pas, rajoutez le à votre liste.

Utilisez le code ci-dessous pour contrôler que tout a bien été ajouté :

```
for c in ma_liste :
    print(c)
```

Conseil

Comme pour les tuples, vous pouvez utiliser l'opérateur **in** pour contrôler si un élément est présent dans votre liste.

Solution

```
ma_liste = [1,2,3,4,5]

if 6 in ma_liste :
    print("OK")
else :
    ma_liste.append(6)
```

1.3 Dictionnaires

Les dictionnaires sont des listes associatives, c'est-à-dire des listes qui lient une valeur à une autre. Dans un dictionnaire en papier, les mots sont liés à leur définition.

Dans un dictionnaire Python, les éléments sont liés par une relation dite **clé, valeur**. La clé étant le moyen de "retrouver" notre valeur dans notre dictionnaire. Par exemple, dans un dictionnaire en papier, nous pouvons retrouver une définition en cherchant le mot qui lui correspond, alternativement, nous pouvons retrouver le contenu d'une page d'un livre en utilisant le numéro de celle-ci, dans ce cas le numéro est la clé et le texte de la page, la valeur.

Pour créer un dictionnaire, il suffit de lister les couples **clé, valeurs** entre 2 accolades :

```
elon_musk = {
    "prénom": "Elon",
    "nom": "Musk",
    "age": 48,
    "talents": ["programmation", "entreprenariat", "aéronautique"]
}
```

Question 3: (*10 minutes*) **Manipulation d'un dictionnaire**
Gardez le dictionnaire de la question 7 des exercices basiques. La traduction du mot "oiseau" est mal orthographiée, modifiez la valeur associée à "oiseau" pour qu'elle devienne "bird". Ajoutez un nouvel élément au dictionnaire en associant le mot "horse" au mot "cheval".

Utilisez ce code pour avoir un aperçu des changements :

```
for x in fr_eng :
        print(x + " : " + fr_eng[x])
```

Conseil

Comme avec les listes, vous pouvez accéder aux éléments d'un dictionnaire en utilisant des crochets []. Seulement cette fois-ci, au lieu d'y mettre l'index, mettez-y la clé associée à l'élément auquel vous voulez accéder.

Solution

```
fr_eng = {
            "chat": "cat",
            "chien": "dog",
            "oiseau": "bir",
            "poule": "chicken",
            "papillon": "butterfly",
            "souris": "mouse",
            "ours": "bear",
            "mouton": "sheep",
            "cochon": "pig"
}
fr_eng["oiseau"] = "bird"
fr_eng["cheval"] = "horse"
```

1.4 Structures de données (Java)

Dans cette partie, les exercices devront être effectués en Java.
Les syntaxes des principales structures de données abordées dans cette partie sont les suivantes :

Déclaration d'un tuple en Java :

```
Object[] mon_tuple = {1,2,3,4};
```

Déclaration d'une liste en Java :

```
List ma_liste = List.of(1,2,3,4);
```

Déclaration d'un dictionnaire en Java :

```
HashMap mon_dictionnaire = new HashMap(Map.of("un", 1, "deux", 2));
```

Informations utiles

— En Java, les listes sont immuables, comme les tuples en Python. Au cas où vous devriez modifier une liste, on vous recommande d'utiliser une liste liée (**LinkedList**). Une liste liée se déclare comme suit :

```
List liste = List.of(1,2,3,4);
LinkedList ma_liste = new LinkedList(liste);
```

— En Java, les listes ne peuvent contenir que des éléments homogènes c'est-à-dire de même type.

Question 4: (*5 minutes*) **Lecture de code**
Qu'affichera le code suivant ?

```
import java.util.List;
import java.util.HashMap;
import java.util.Map;

public class Main {
   public static void main(String[] args) {
      List<Integer> ma_liste = List.of(3,2,6,4,1,5);
      HashMap<Integer, String> mon_dictionnaire = new HashMap<Integer, String>(Map.of(1,"un",2,"deux",3,"trois",
        4,"quatre",5,"cinq",6,"six"));
      System.out.println(ma_liste.get(3));
      System.out.println(ma_liste.get(2));
      System.out.println(mon_dictionnaire.get(ma_liste.get(4)));
      System.out.println(mon_dictionnaire.get(ma_liste.get(0)));
   }
}
```

Choisissez parmi les possibilités suivantes :

— 3
2
quatre
zero

— 4
6
un
trois

— 3
2
un
trois

— 4
6
quatre
zero

Conseil

Commencez par voir quel est l'élément de la liste qui sera retourné. Ensuite, regardez dans le dictionnaire quelle est la valeur associée à cet élément qui vient de nous être retourné.

Solution

4
6
un
trois

2 Itération

Quelques rappels de concepts théoriques :

— L'itération désigne l'action de répéter un processus (généralement à l'aide d'une boucle) jusqu'à ce qu'une condition particulière soit remplie.
— Il y a deux types de boucles qui sont majoritairement utilisées :

Boucle for : Une boucle `for` permet d'itérer sur un ensemble. Cet ensemble peut être une liste, un dictionnaire, une collection, ... La syntaxe d'une boucle en Python est la suivante `for` **nom de variable** `in` suivi du nom de l'élément sur lequel vous voulez itérer. La variable prendra la valeur de chaque élément dans la liste, un par un.

La syntaxe en Java est la suivante : `for` **(**`type` **nom_de_variable; condition de fin de boucle; incrémentation à chaque itération)** suivi d'une accolade.

Boucle while : Les boucles `while` sont des boucles qui s'exécutent de façon continue jusqu'à ce qu'une condition soit remplie. La syntaxe est la suivante : En Python, on écrit d'abord `while`, suivi de la condition à atteindre.

En Java, il n'y a pas de différence majeure à part le fait qu'il faille mettre entre parenthèses la condition et les deux points sont remplacés par des accolades.

— La fonction **range(n)** permet de créer une liste de nombres de 0 à la valeur passée en argument moins 1 (n-1). Lorsqu'elle est combinée à une boucle **for**, on peut itérer sur une liste de nombres de 0 à n-1.

— Si vous avez fait une erreur dans votre code, il se peut que la boucle **while** ne remplisse jamais la condition lui permettant de sortir. On parle dans ce cas d'une **boucle infinie**. Ceci peut entraîner un crash de votre programme, voire même de votre ordinateur. Il faut toujours s'assurer qu'il y' ait une condition valide dans une boucle **while**.

Question 5: (*10 minutes*) **Création d'une liste à partir d'un tuple (Python)**
Transformer le tuple (1,4,5,8) en une liste à l'aide d'une boucle **for**.

Conseil

— Déclarer une liste vide à l'extérieur de la boucle.
— La boucle permet d'itérer sur le tuple.
— Rajouter un à un les éléments du tuple dans la liste.

Solution

```
1  liste_finale = []
2  tuple_initial = (1, 4, 5, 8)
3  for valeur in tuple_initial:        # A chaque iteration valeur devient une copie d'un element du tuple. La boucle se
       termine au dernier élément de tuple_initial.
4      liste_finale.append(valeur)   # On ajoute valeur à la liste créée en dehors de la boucle.
```

3 Récursivité

La récursivité est le fait d'appeler une fonction de façon récursive c'est-à-dire une fonction qui s'appelle elle-même de façon directe ou indirecte.

Question 6: (*15 minutes*) **Fibonacci - Java**
Ecrivez deux fonctions permettant de calculer un nombre donné de la suite de Fibonacci. L'une doit utiliser la récursivité, et l'autre l'itération. Pour rappel, chaque élément de la suite de Fibonacci est la somme des deux derniers éléments. L'élément 0 vaut 0, l'élément 1 vaut 1 et l'élément 2 vaut 1.

Début de la suite : [0,1,1,2,3,5,8,13,...]

Conseil

L'algorithme permettant de faire une suite de Fibonacci a été présenté dans les diapositives du cours. En cas de difficulté, n'hésitez pas à vous y référer.

Solution

Via Itération :

```
1  public static int fibonacci_i(int n){
2      if (n==0 || n==1){
3          return n;
4      }
5      else{
6          int old_fib = 1;
7          int new_fib = 1;
8          int tmp;
9          for (int i=2; i<n; i++){
10             tmp = new_fib;
11             new_fib = new_fib+old_fib;
12             old_fib = tmp;
13         }
14         return new_fib;
15     }
16 }
```

Via récursivité :

```
1  public static int fibonacci_r(int n){
2      if (n==0 || n==1){
3          return n;
4      }
5      else{
6          return fibonacci_r(n-1) + fibonacci_r(n-2);
7      }
8  }
```