Série d'exercice 4: Structure de données, Itération et Récursivité

Le but de cette séance est de s'exercer avec des structures de données qui seront utilisés lors des prochaines séances de cours/Travaux Pratiques. Les structures de données abordées lors de cette séance sont les tuples, les listes et les collections. 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.

1. Les structures de données

1.1 Les tuples

Pour rappel, les tuples sont des listes d'éléments immuables, ce qui signifie que ces listes ne peuvent pas être modifiées. 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 séparées entre elles par des virgules:

Exemple:

```
mon_tuple = (1,2,3,4)
```

1.1.1

Créez un tuple nommé `mon_tuple` contenant les chiffres 1,2,3,4 et 5. Stockez le 4ème élément dans une variable `element_4`, puis affichez la. (5 minutes)

Conseils

Pour accéder à un élément d'un tuple, ou d'une liste, vous pouvez utiliser l'indexation. Comme pour accéder aux caractères des chaînes de caractères, utilisez [].

Solutions

```
mon_tuple = (1,2,3,4,5)
print(mon_tuple[3])
```

1.1.2

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 dans une nouvelle variable nommée `taille_tuple`. Pour finir, affichez `taille_tuple`. (3 minutes)

Conseils

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

Solutions

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

1.1.3

Vous pouvez contrôler si un élément est contenu dans un tuple, ou une liste, via l'opérateur `in`. Si l'élément est dans la liste, la valeur booléenne `True` vous sera retourné. S'il n'y est pas, `False` sera retourné. (2 minutes)

Qu'affichera le programme suivant?

```
mon_tuple = (1,2,3,4,5,6)

if 6 in mon_tuple :
    print("6 est contenu dans le tuple")
else :
    print("6 n'est pas contenu dans le tuple")
```

Solutions 6 est élément de notre tuple, la condition sera donc remplie et c'est "6 est contenu dans le tuple" qui sera affiché.

1.2 Les 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 :

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

1.2.1

Créez une liste nommée `ma_liste`contenant les nombres 1,2,3,4 et 5. Stockez le deuxième élément de la liste dans une variable nommée `element_2`, puis stockez la taille de la liste dans une variable nommée `taille_liste`. Affichez ces deux variables. (5 minutes)

Conseils Comme pour les tuples, vous pouvez utiliser [] pour accéder à un élément, et la fonction `len()` pour obtenir le nombre d'éléments contenus dans la liste.

Solutions

```
ma_liste = [1,2,3,4,5]
element_2 = ma_liste[1]
l_ma_liste = len(ma_liste)
print(element_2)
print(l_ma_liste)
```

1.2.2

Créez une liste nommée `ma_liste` contenant les nombres 1,2,2,4 et 5. Modifiez le 3ème élément de la liste pour qu'il devienne 2 au lieu de 3. Ajoutez le chiffre 6 à la fin de la liste, et le chiffre 0 au début de cette dernière. (5 minutes)

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

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

Conseils En Python, vous pouvez accéder à chaque élément de la liste en utilisant [] et changer sa valeur. Pour ajouter un élément à la fin de la liste, utilisez la fonction append() et pour choisir l'endroit ou vous désirez insérer votre nouvel élément, utilisez la fonction insert().

Solutions

```
ma_liste = [1,2,2,4,5]
ma_liste[2] = 3
ma_liste.append(6)
ma_liste.append(0,0)
```

1.2.3

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. (10 minutes)

Utilisez ce morceau de code pour vérifier que tout a bien été ajouté :

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

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

Solutions

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

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

1.3 Les 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"]
}
```

1.3.1

Créez un dictionnaire nommé `fr_eng` contenant les éléments suivants :

"chat": "cat", "chien": "dog", "oiseau": "bir", "poule": "chicken", "papillon": "butterfly", "souris": "mouse", "ours": "bear", "mouton": "sheep", "cochon": "pig"

Ce dictionnaire contient des mots en français (les clés) associés à leur traduction en anglais (les valeurs).

Accédez à la traduction du mot "souris", et stockez la dans une variable nommée `souris_traduite`, puis calculez le nombre d'éléments contenus dans le dictionnaire et stockez le dans une variable nommée `taille_fr_eng`. Affichez le contenu des deux variables que vous venez de créer. (10 minutes)

Conseils Comme pour les listes, vous pouvez accéder aux éléments de dictionnaires 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. Comme pour les listes, la fonction `len()` vous aidera à obtenir le nombre d'éléments dans le dictionnaire.

Solutions

```
fr_eng = {
                "chat": "cat",
                "chien": "dog",
                "oiseau": "bir",
                "poule": "chicken",
                "papillon": "butterfly",
                "souris": "mouse",
                "ours": "bear",
                "mouton": "sheep",
                "cochon": "pig"
}
souris_traduite = fr_eng["souris"]
taille_fr_eng = len(fr_eng)
print(souris_traduite)
print(taille_fr_eng)
```

1.3.2

Gardez le même dictionnaire qu'auparavant. 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. Associez le mot "horse" au mot "cheval". (10 minutes)

Utilisez le code suivant pour avoir un aperçu de vos changements.

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

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

Solutions

```
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"
```

1.3.3

Qu'affiche le programme suivant? (5 minutes)

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

print(elon_musk["talents"][2])
print("aéronautique" in elon_musk["talents"])
```

Conseils L'élément associé à la valeur "talents" du dictionnaire `elon_musk` est une liste.

Solutions aéronautique True

En effet, via elon_musk["talents"], on accède à la liste ["programmation", "entreprenariat", "aéronautique"]. Il est donc possible de manipuler cette liste comme une liste classique !

1.4 Les Datastructures en Java

Déclarer un Tuple en Java :

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

Déclarer une liste en Java :

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

Déclarer un dictionnaire en Java :

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

Attention, en **Java**, les listes sont immuables, comme les tuples. Si vous voulez la modifier, il faut créer une liste immuable comme suit :

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

1.4.1

Créez une liste nomée `ma_liste` contenant les nombres 1,2,3,4 et 5. Stockez le deuxième élément de la liste dans une variable nommée `element_2`, puis stockez la taille de la liste dans une variable nomée `l_ma_liste`. Imprimez ces deux variables. Puis créez une liste `ma_liste_m` liée à la liste `ma_liste`. Ajoutez le chiffre 6 à la fin de la liste, et le chiffre 0 au début de cette dernière. (5min)

Ajoutez ceci au début de votre code :

```
import java.util.List;
import java.util.LinkedList;
```

Ajoutez ceci à la fin de votre code :

```
for(int i=0;i<ma_liste_m.size();i++){
            System.out.println(ma_liste_m.get(i));
}
```

Conseils Pour obtenir un élément d'une liste, il faut utiliser la fonction get(index). Pour obtenir la taille de la liste, utilisez la fonction size().

Pour ajouter un élément au début d'une LinkedList, vous devez utiliser addFirst(value) et pour l'ajouter à la fin, vous devez utiliser addLast(value).

Solutions

```
import java.util.List;
import java.util.LinkedList;

public class Main {

    public static void main(String[] args) {
        var ma_liste = List.of(1,2,3,4);
        int element_2 = ma_liste.get(1);
        int l_ma_liste = ma_liste.size();
        System.out.println(element_2);
        System.out.println(l_ma_liste);
        var ma_liste_m = new LinkedList(ma_liste);
        ma_liste_m.addFirst(0);
        ma_liste_m.addLast(6);
        for(int i=0;i<ma_liste_m.size();i++){
            System.out.println(ma_liste_m.get(i));
        }
    }
}
```

1.4.2

Créez un dictionnaire `mon_dictionnaire` contenant les éléments suivants :

("étudiants", 14000, "enseignants", 2300, "collaborateurs", 0)

Stockez le nombre d'étudiants dans une variable nommée `nb_etudiants`. Obtenez la taille du dictionnaire et stockez la dans une variable `l_mon_dictionnaire`. Imprimmez ces deux variables. Pour finir, corrigez le nombre de collaborateurs qui sont en réalité 950, et ajoutez le nombre de pays desquels proviennent les étudiants (86). (5min)

Ajoutez ceci au début de votre code :

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

Ajoutez ceci à la fin de votre code :

```
for (var keys : mon_dictionnaire.keySet())
        {
            System.out.println(keys + " : "+ mon_dictionnaire.get(keys));
        }
```

Conseils Comme pour les listes, accédez aux valeurs via la fonction get(key) et à la taille via la fonction size().

Pour insérer / corriger un élément, utilisez la fonction put(key,value).

Solutions

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

public class Main {

    public static void main(String[] args) {
        var mon_dictionnaire = new HashMap(Map.of("étudiants",14000,"enseignants",2300,"coll
        var nb_etudiants = mon_dictionnaire.get("étudiants");
        var l_mon_dictionnaire = mon_dictionnaire.size();
        System.out.println(nb_etudiants);
        System.out.println(l_mon_dictionnaire);
        mon_dictionnaire.put("collaborateurs",950);
        mon_dictionnaire.put("pays",86);

        for (var keys : mon_dictionnaire.keySet())
        {
            System.out.println(keys + " : "+ mon_dictionnaire.get(keys));
        }
```

```
    }
}
```

2. L'itération

Quelques rappels de concepts théoriques:

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

```java
for (int i=0; i<5;i++) {
    System.out.println("Clap your hands!");
}
```

  1. **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. Voici un exemple:

```python
i = 0
while i<10:
    i += 1
    print("i)
## Le code va afficher:
##  0 1 2 3 4 5 6 7 8 9 10 avec chaque espace qui représente un retour à la ligne.
```

  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 comme dans l'exemple précédent.

Note:

- La fonction `range(n)` permet de créer une liste de nombres de 0 à la valeur passée en argument (n). Lorsqu'elle est combinée à une boucle for, on peut itérer sur une liste de nombres de 0 à n-1.
  - Exemple en python: `Python l = ["J'aime", "le", " chocolat"] for valeur in range(len(l)): print(l[valeur]) ## Le code va affichier:`

```
J'aime ## le ##
chocolat
```

- Si vous avez fait une erreur dans votre code, il se peut que la boucle `while` n'arrive jamais à sa condition et qu'elle ne se termine donc jamais. 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`

Attention: Toutes les réponses devront être écrites en python.

2.1 Utilisation de la fonction range() dans une boucle for

En utilisant la fonction `range()` et une boucle `for`, calculez la somme des entiers de 0 à 20 et affichez le résultat. (5 minutes)

Conseils

- Lorsque `range(n)` est combinée à une boucle `for`, on peut itérer sur une liste de nombres de 0 à n-1.
- Déclarer une variable en dehors de la boucle. Au terme de l'itération, cette variable contiendra la somme des entiers de 0 à 20.

Solutions

```python
somme = 0
for i in range(21):
    somme += i   # À chaque itération on rajoute i à somme.
print(somme)
```

2.2 Création d'une liste à partir d'un tuple

Transformer le tuple (1,4,5,8) en une liste à l'aide d'une boucle `for`. (5 minutes)

Conseils

- 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.

Solutions

```python
liste_finale = []
tuple_initial = (1, 4, 5, 8)
for valeur in tuple_initial:      # A chaque itération valeur est une copie d'un élément
    liste_finale.append(valeur)   # On ajoute valeur à la liste crée en dehors de la bou
```

2.3 Boucle while et input()

A l'aide d'une boucle `while`, demander à l'utilisateur de rentrer une valeur. Tant que cette valeur ne correspond pas à 10, le programme redemande à l'utilisateur une nouvelle valeur. (15 minutes)

Conseils

1. Définir à l'extérieur de la boucle une variable de type booléen utilisée pour le test dans while.
2. La fonction `input()` retourne un `str` ou `String` (chaîne de caractères), pensez à changer son type.

Solutions

```
bool_test = True
while bool_test:                                          # Tant que bool_test est Tru
    test_value = int(input("Veuillez entrer un entier"))  # Pour que le test effectué
                                                          #  ( comparaison entre un si
    bool_test = not(test_value == 10)                     # On veut sortir de la bouc
    if bool_test == True:                                 # False. D'où l'utilisation
        print("Ce n'est pas le bon entier.")
print("Bravo !")
```

Question 2.4 Boucle while et des étudiants

Considérons une liste d'étudiant · e · s contenant le nom de l'étudiant suivi d'une valeur booléenne indiquant si l'étudiant · e est présent · e en classe ou non.

l = [“Schmitt”, True, “Irma”, False, “Khalif”, True, “Yasser”, False, “Wang”, True]

À partir de cette liste et du type de boucle de votre choix, créez un dictionnaire ayant pour clés les noms et comme valeur la présence au cours.

Conseils

- i % 2: renvoie le reste de i par la division euclidienne de 2. (i % 2 == 0) Indique si i est pair.
- Boucle for: Utiliser les indices pour accéder aux éléments de la liste et donc ne pas itérer directement sur les éléments de la liste.
- Boucle while: La condition de sortie devrait être en rapport avec la longueur de la liste.

Solutions Solution avec boucle for:

```
l = ["Schmitt", True, "Irma", False, "Khalif", True, "Yasser", False, "Wang", True]
dict_final = {}
```

```
for i in range(len(l)):                # i va de 0 à 9
    if i % 2 == 0:                     # Test si i est pair. Si oui, on crée une nouvelle
        dict_final[l[i]] = l[i + 1]
```

Solution avec boucle while:

```
l = ["Schmitt", True, "Irma", False, "Khalif", True, "Yasser", False, "Wang", True]
dict_final = {}
i = 0                                  # Variable qui permettra d'itérer et de gérer la b
while i < len(l):                      # La boucle s'arrêtera donc au bout de 10 itérati
    if i % 2 == 0:
        dict_final[l[i]] = l[i + 1]
    i +=1                              # On incrémente i
```

- Pour ce problème la boucle `for` est plus simple à mettre en place et évite le danger d'une boucle infinie. La boucle `while` est utilisée pour des problèmes plus précis qui ne peuvent pas être résolus par une boucle `for`. Donc lorsque on peut résoudre un problème avec les deux types, utilisez de préférence la boucle for.

3. La récursivité

La récursivité est le fait d'appeler une fonction au sein même de cette même fonction. Ce système est souvent utilisé en algorithmique.

3.1

Qu'affichent les programmes suivants? (5 minutes)

3.1.1

```
def fonction_p(x) :
    print(x)
    if x == 0 :
        pass
    else :
        fonction_p(x-1)

fonction_p(5)
```

3.1.2

```
def fonction_p(x) :
    if x == 0 :
        pass
    else :
        fonction_p(x-1)
        print(x)

fonction_p(5)
```

Conseils La principale différence entre ces 2 programmes est l'endroit où est placé le `print()`. Réfléchissez bien à l'ordre d'exécution de la fonction, écrire le développement sur un papier pourrait aider.

Solutions 3.1.1

5 4 3 2 1 0

3.1.2

0 1 2 3 4 5

3.2

Qu'affiche le programme suivant? (5 minutes)

A quoi sert cette fonction ?

```
def fct_a(mot) :

    if len(mot) == 1 :
        if mot[0] == "a" :
            return 1
        else :
            return 0
    else :
        if mot[0] == "a" :
           return 1 + fct_a(mot[1:])
        else :
            return 0 + fct_a(mot[1:])

print(fct_a("blablabla"))
```

Conseils Aidez vous d'un papier et effectuez les étapes une à une afin de bien comprendre comment le code est articulé.

Solutions Cette fonction compte le nombre de "a" contenus dans un mot que vous lui passez.

3.3

Ecrivez la fonction `factoriel()` suivant une approche récursive. (5 minutes)

Pour rappel, cette fonction prend un entier et retourne le factoriel de ce dernier.

factoriel(1) = 1, factoriel(2) = $1\mathit{2}$ = *2, factoriel(3)* = $1\mathit{23}$ = *6, factoriel(4)* = $1\mathit{234}$ = 24,...

Conseils On peut écrire cette fonction comme suit :

f(n) = f(n) * f(n-1) si n > 1 f(n) = 0 si n = 0

Solutions

```
def factoriel(x) :
        if x == 0:
            return 1
        else :
            return x * factoriel(x-1)
```

3.4

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. Le premier élément vaut 0, puis le deuxième et le troisième élément valent tous deux 1. (15 minutes)

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

Conseils 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.

Solutions Itération :

```
def fibonacci_i(n) :
    if n==0 or n==1 :
        return n
    else :
        old_fib = 1
        new_fib = 1
        for i in range(n-2) :
            temp = new_fib
            new_fib = new_fib + old_fib
            old_fib = temp
        return new_fib
```

récursivité :

```
def fibonacci_r(n) :
    if n==0 or n==1 :
        return n
    else :
        return fibonacci_r(n-1) + fibonacci_r(n-2)
```