Week4_Exercise1_Corrigé

October 15, 2019

1 Algorithmique et Pensée Computationnelle

2 Algorithmique et Pensée Computationnelle

3 ACT Week 4 (Python)

3.1 Exercice 1

3.1.1 Structures de données

3.2 Tuples

3.2.1 Rôle

Les `tuples` sont des `listes immutables` (dont les élèments ne peuvent pas être modifiés.)
Les `tuples` sont utiles pour grouper plusieurs valeurs ou objets entre eux et les réutiliser collectivement plus tard.

3.2.2 Création

Pour créer (instantier) un `tuple`, on définit ses valeurs entre parenthèses, par exemple:

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

3.2.3 Accéder à un élément

Pour accéder à un élément dans un `tuple`, il suffit d'écrire le nom de celui-ci suivi de crochets contenant la position de l'élément que nous voulons. Nous commençons à compter à partir de `0`, le premier élément est donc l'élément 0.
Pour accéder au deuxième élément de mon tuple, je fais donc:

```
deuxieme_element = mon_tuple[1]
print(deuxieme_element)
# 2
```

3.2.4 Taille

Pour afficher la taille d'un `tuple`, utilisez la fonction `len()`:

```
taille = len(mon_tuple)
print(taille)
# 4
```

3.2.5 Vérifier si un élément fait partie de la liste

Pour vérifier si un élément est dans un `tuple`, utilisez l'opérateur `in`:

```
membre = 10 in mon_tuple
print(membre)
# False
```

[ ]:
```
 En programmation informatique, n# EXEMPLE

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

deuxieme_element = mon_tuple[1]
print(deuxieme_element)

taille = len(mon_tuple)
print(taille)

membre = 10 in mon_tuple
print(membre)
```

Tuples

- *Créez* un `tuple` Qui contient votre nom et votre âge.
- *Print* les informations dans le `tuple`.

[ ]:
```
mon_tuple = ("Tanja", 23)
print("My name is {0} and I am {1} years old.".format(mon_tuple[0],␣
↪mon_tuple[1]))
```

3.3 Listes

3.3.1 Rôle

Les `listes` fonctionnent comment des `tuples` à la différence qu'ils ne sont pas `immutables`, donc nous pouvons modifier leur contenu ou retirer et rajouter dynamiquement des élèments.

3.3.2 Création

Pour créer (instantier) une `liste`, on définit ses valeurs entre crochets (parenthèses carrées), par exemple:

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

3.3.3 Modification

Pour modifier un élément, nous pouvons modifier sa valeur en utilisant son index comme ceci:

```
ma_liste[2] = 0
print(ma_liste)
# [1, 2, 0, 4]
```

3.3.4 Ajouter un élément

Pour ajouter un élément dans une `liste`, on utilise la `méthode append()`:

```
ma_liste.append(10)
print(ma_liste)
# [1, 2, 0, 4, 10]
```

[ ]:
```
ma_liste = [1, 2, 3, 4]
print(ma_liste)

ma_liste[2] = 0
print(ma_liste)

ma_liste.append(10)
print(ma_liste)
```

Lists

- `Créez` une `liste` qui contient des noms de villes.
- `Print` la deuxième ville de votre liste.
- `Ajoutez` la ville `Hong Kong` à la liste avec la `méthode append()`

[ ]:
```
cities = ["Montreux","Vevey","Aigle", "Ollon", "Rougemenont"]
print(cities[1])
```

3.4 Dans la cellule suivante, nous avons créé une liste de pays, exécutez (run) cette cellule pour pouvoir utiliser la liste pays dans les exercices suivants.

[ ]:
```
pays = ["Switzerland", "Germany", "France", "Italy", "Spain", "Netherlands",␣
 ↪"Norway", "Austria", "USA", "UK"]
```

3.4.1 Combien y a-t-il de pays?

[ ]:
```
from assertion import nb_pays

# VOTRE REPONSE ICI
nombre_pays = len(pays)
# FIN DE VOTRE REPONSE

nb_pays(pays, nombre_pays)
```

3.4.2 Ajoutez la Grèce à la liste

[ ]:
```
from assertion import add_greece

# VOTRE REPONSE ICI
pays.insert(0,"Grèce")
#pays.append("Grèce")

# FIN DE VOTRE REPONSE

add_greece(pays)
```

3.4.3 Vérifiez si la Grèce est dans la liste

[ ]:
```
from assertion import est_membre

# VOTRE REPONSE ICI
membre = "Grèce" in pays
print(membre)
# FIN DE VOTRE REPONSE

est_membre(pays, membre)
```

3.4.4 Changez la valeur du 5ème pays vers “Canada”

[ ]:
```
from assertion import Canada

# VOTRE REPONSE ICI
pays[4]= "Canada"
print(pays)
# FIN DE VOTRE REPONSE

Canada(pays)
```

3.5 Dictionnaires

3.5.1 Rôle

Les `dictionnaires` sont des listes associatives, c’est-à-dire des listes qui relient une valeur à une autre. Dans un dictionnaire en papier, les mots sont reliés à leur définition. Dans les `dictionnaires` Python, nous pouvons relier n’importe quelle valeur à n’importe quelle autre valeur (même à elle-même.)

Dans un `dictionnaire` Python, on parle d’une relation `clef`, `valeur`. La `clef` 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 `clef` et le texte de la page, la `valeur`.

3.5.2 Création

Pour créer (instantier) un `dictionnaire`, on écrit entre des accolades (`{}`) les `clefs`, suivies d’un `:` et de leur `valeur`, par exemple:

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

3.5.3 Accéder à un élément

Pour accéder à un élément dans un `dictionnaire`, il suffit d’écrire le nom de celui-ci suivi de crochets contenant la `clef` de l’élément que nous voulons. Pour accéder à l’âge d’Elon Musk dans mon `dictionnaire`, je fais donc:

```
age_elon = elon_musk["age"]
print(age_elon)
```

```
# 48
```

3.5.4 Ajouter un élément

Pour ajouter un élément dans un `dictionnaire`, il suffit de faire comme si nous souhaitions accèder à cet élément (la `clef` entre crochets `[]`), et d'y assigner une valeur.

```
elon_musk["richesse"] = 20000000000
print(elon_musk["richesse"])
# 20000000000
```

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

age_elon = elon_musk["age"]
print(age_elon)

elon_musk["richesse"] = 20000000000
print(elon_musk["richesse"])
```

3.6 Dans la cellule suivante, nous avons créé un `dictionnaire` qui contient des mots français et les traduit vers l'anglais, exécutez (run) cette cellule pour pouvoir utiliser le `dictionnaire pays` dans les exercices suivants.

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

3.6.1 Combien y a-t-il de mots dans le dictionnaire?

[ ]:
```
from assertion import nb_pays

# VOTRE REPONSE ICI
nombre_mots =len(fr_eng)
# FIN DE VOTRE REPONSE

nb_pays(fr_eng, nombre_mots)
```

3.6.2 Comment le mot “papillon” est-il traduit en anglais selon le dictionnaire?

[ ]:
```
from assertion import papillon

# VOTRE REPONSE ICI
papillon_traduction =fr_eng["papillon"]
# FIN DE VOTRE REPONSE

papillon(fr_eng, papillon_traduction)
```

3.6.3 Ajoutez la traduction pour le mot “vache” -> “cow” dans le dictionnaire

[ ]:
```
from assertion import add_cow

# VOTRE REPONSE ICI
fr_eng["vache"]= "cow"
# FIN DE VOTRE REPONSE

add_cow(fr_eng)
```

3.6.4 “oiseau” se traduit par “bird”, mais il a mal été écrit dans le dictionnaire, pouvez-vous corriger cette erreur?

[ ]:
```
from assertion import correct

# VOTRE REPONSE ICI
fr_eng["oiseau"]="bird"
print(fr_eng)
# FIN DE VOTRE REPONSE

correct(fr_eng)
```

[ ]:
```

```

Week4_Exercise2_Corrigé

October 15, 2019

1 Algorithmique et Pensée Computationnelle

2 Algorithmique et Pensée Computationnelle

3 ACT / Week 4 (Python)

3.1 Exercice 2

3.2 Boucles

3.2.1 Rôle

Dans l'exercice 1, nous avons vu les structures de données `tuple`, `liste` et `dictionnaire`. Ces structures de données permettent de grouper différents éléments les uns avec les autres. Ces structures sont très pratiques car grâce à elles, nous n'avons besoin que d'une seule variable pour accèder à une grande quantité de donnée.

Cependant, comme nous l'avons vu, l'accès à un élément d'une liste se fait de cette manière:

```
ma_liste = [1, 2, 3]
mon_element = ma_liste[2]
# mon_element = 3
```

Intuitivement, pour `print()` tous les éléments de `ma_liste` ligne par ligne, il faudrait faire ceci:

```
print(ma_liste[0])
print(ma_liste[1])
print(ma_liste[2])
```

Ce procédé est long, ennuyant et peut entraîner des erreurs de la part du programmeur. De plus, il suppose que l'on connait la taille de la liste à l'avance, ce qui est rarement le cas. Et si je voulais ajouter un élément à cette liste avec la `méthode append()`, mon code ne marcherait plus car le nouvel élément ne serait plus pris en compte.

Heureusement, nous avons ce que l'on appelle des `boucles`. Celles-ci nous permettent d'écrire du code une seule fois et de l'exécuter plein de fois d'affilée jusqu'à ce qu'une condition soit atteinte. C'est très pratique pour les `listes`, `tuples`, etc. puisque nous pouvons écrire du code une fois et l'appliquer à tous les éléments d'une liste.

3.2.2 Il y a deux types de `boucles`, les boucles `for` et les boucles `while`.

3.3 Boucles `for`

Les boucles `for` en python permettent de passer sur tous les éléments d'une liste pour leur appliquer du code de votre conception. La syntax est la suivante:

On écrit `for` suivi d'un nom de variable de votre choix, suivi du mot-clef `in` et enfin le nom de la `liste` sur laquelle vous voulez appliquer la `boucle`. La variable prendra la valeur de chaque élément dans la liste, un par un. L'exemple précédent pourrait être réécrit de cette manière avec une boucle:

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

[ ]:
```
ma_liste = [1, 2, 3]
print(ma_liste)

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

3.3.1 La fonction `range()`

Il existe aussi une fonction très pratique à utiliser avec les boucles `for`, c'est la fonction `range()`. Cette fonction 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`.

Par exemple:

```
for i in range(4):
    print(i)
# 0
# 1
# 2
# 3
```

[ ]:
```
for i in range(4):
    print(i)
```

3.3.2 `range()` + `len()`

La vraie force de `range()` est lorsqu'elle est combinée avec la fonction `len()`.
Pour rappel, `len()` nous donne la taille d'une `liste/tuple/dictionnaire/string`, etc. Donc combinée avec `range()`, nous pouvons obtenir tous les `indexes` d'une liste, ce qui nous permet de modifier les éléments de celle-ci.

Par exemple, pour rajouter `+1` à chaque élément dans une `liste` de `int`, je ferais comme ceci:

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

for i in range(len(ma_liste)):
    ma_liste[i] = ma_liste[i] + 1
```

On ne peut pas modifier une liste avec une simple boucle `for i in` car la variable `i` est une copie et pas l'élément lui-même.

[ ]:
```
print("Avec range + len")

ma_liste = [1, 2, 3]
print(ma_liste)

for i in range(len(ma_liste)):
    ma_liste[i] = ma_liste[i] + 1
print(ma_liste)



print("------------")
print("Avec for i in ma_liste")

ma_liste = [1, 2, 3]
print(ma_liste)

for i in ma_liste:
    i = i + 1
print(ma_liste)
```

3.3.3 Dans la cellule suivante, écrivez deux boucles qui affichent tous les mots de la liste `countries` ligne par ligne, d'abord avec une boucle `for ... in` puis avec un boucle `for ... in + range + len`

[ ]:
```
countries = ["Switzerland", "Germany", "France", "Italy", "Spain",␣
↪"Netherlands", "Norway", "Austria", "USA", "UK"]
for i in countries: #Switzerland à UK
    print(i)

for i in range(len(countries)): #  0 à n
    print(countries[i])
```

3.3.4 Dans la cellule suivante, il y a une très grande liste de `int`, écrivez une boucle qui mutliplie chaque élément par 2

[ ]:
```
from assertion import save_liste, liste_times_two

liste = [3, 2, 5, 10, 2, 3, 12, 100, 56, 23, 80, 73, 23, 22, 19, 55, 78 ,20,␣
 ↪30, 39, 31, 60, 152, 1212, 14324]
save_liste(liste)

# VOTRE CODE ICI
for i in range(len(liste)):
    liste[i]= liste[i]*2
# FIN DE VOTRE CODE

liste_times_two(liste)
```

3.4 Boucles `while`

Les boucles `while` sont des boucles qui s'exécutent plein de fois d'affilée jusqu'à ce qu'une `condition` soit accomplie (une `booléenne` `True` ou `False`). La syntax est la suivante:

On écrit d'abord `while`, suivi de la `condition` à atteindre. Par exemple pour afficher tous les nombre de 0 à 10:

```
i = 0
while i < 10:
    print(i)
    i = i + 1
```

[ ]:
```
i = 0
while i < 10:
    print(i)
    i = i + 1
```

3.4.1 /! Attention, les boucles `while` sont plus dangereuses que les boucles `for`

Si vous avez fait une erreur dans votre code, il se peut que la boucle n'arrive jamais à sa condition et qu'elle ne se termine donc jamais. Ceci peut entraîner un `crash` de votre programme, voire même de votre ordinateur.

Exemple à ne pas reproduire:

```
while True:
    print("Boucle infinie")
```

3.4.2 Comme dans l'exercice précédent, écrivez une boucle `while` qui multiplie chaque nombre d'une `liste` par 2

[ ]:
```
from assertion import save_liste, liste_times_two

liste = [3, 2, 5, 10, 2, 3, 12, 100, 56, 23, 80, 73, 23, 22, 19, 55, 78 ,20,␣
↪30, 39, 31, 60, 152, 1212, 14324]
save_liste(liste)

# VOTRE CODE ICI
i=0
while i <= len(liste)-1:
    liste[i]=liste[i]*2
    i=i+1

print(liste)

# FIN DE VOTRE CODE

liste_times_two(liste)
```

[ ]:
```

```