Introduction à la programmation, Leçon 4: Dictionnaires et classes

Réchauffement

1. Dans cet exercice, vous devrez ajouter des nombres et des chaînes aux listes à l'aide de la méthode "append". Ajoutez les numéros 1, 2 et 3 à la liste "numbers", et les chaînes "hello" et "world" à la liste "strings". Vous devrez également renseigner la variable second_name, en utilisant l'opérateur parenthèses [].

```
numbers = []
strings = []
names = ["John", "Eric", "Jessica"]

# write your code here
second_name = None

# this code should write out the filled arrays and the second
name in the names list (Eric).
print(numbers)
print(strings)
print("The second name on the names list is %s" %
second_name)
```

2. Allez sur Moodle, leçon 4, et téléchargez le fichier avec les données #MeToo. Le fichier contient les données relatives au mouvement «Me Too», qui s'est répandu comme un hashtag utilisé sur les médias sociaux pour aider à démontrer la prévalence généralisée des agressions sexuelles et du harcèlement, en particulier sur le lieu de travail. Le fichier a été généré par Google Trends. Chaque ligne du fichier est de forme "semaine, popularité". Popularité est un nombre indiquant la popularité des requêtes liées au mouvement au cours d'une certaine semaine de l'année. Votre tâche consiste à trouver la semaine où le mouvement était le plus populaire. Utilisez la fonction `split` pour diviser une ligne en semaine et popularité.

```
week, popularity = line.split(“,”)
```

Nouveau matériel

5. Dictionnaires

Un dictionnaire est un type de données similaire aux listes, mais fonctionne avec des clés et des valeurs au lieu d'index. Chaque valeur stockée dans un dictionnaire peut être accédée en utilisant une clé, qui est n'importe quel type d'objet (une chaîne, un nombre, une liste, etc.) au lieu d'utiliser son index pour l'adresser. Par exemple, une base de données de numéros de téléphone pourrait être stockée en utilisant un dictionnaire comme celui-ci :

```
phonebook = {}
phonebook["John"] = 938477566
phonebook["Jack"] = 938377264
phonebook["Jill"] = 947662781
print(phonebook)
```

Alternativement, un dictionnaire peut être initialisé avec les mêmes valeurs de la façon suivante :

```
phonebook = {
    "John" : 938477566,
    "Jack" : 938377264,
    "Jill" : 947662781
}
print(phonebook)
```

Les dictionnaires peuvent être itérés, comme une liste. Cependant, un dictionnaire, contrairement à une liste, ne conserve pas l'ordre des valeurs qui y sont stockées. Pour itérer sur des paires de valeurs clés, utilisez la syntaxe suivante:

```
phonebook = {"John" : 938477566,"Jack" : 938377264,"Jill" : 947662781}
for name, number in phonebook.items():
    print("Phone number of %s is %d" % (name, number))
```

Pour supprimer une entrée spécifiée, utilisez :

```
del phonebook['John']
```

6. Ajoutez "Jake" à `phonebook` avec le numéro de téléphone 938273443 et supprimez Jill de `phonebook`.

```
phonebook = {
    "John" : 938477566,
    "Jack" : 938377264,
    "Jill" : 947662781
}

# write your code here

# testing code
if "Jake" in phonebook:
    print("Jake is listed in the phonebook.")
if "Jill" not in phonebook:
    print("Jill is not listed in the phonebook.")
```

7. **Générateur de texte**

- (vous pouvez ignorer cet exercice si vous l'avez implémenté au cours de la classe précédente) Écrivez une fonction qui génère une liste de paires de mots consécutifs dans un texte. Enregistrez cette fonction dans un module. Écrire un script qui importe le module que vous venez de créer, reçoit le nom d'un fichier, lit le fichier et imprime les paires de mots consécutifs dans le fichier.
- Ajoutez à votre module une fonction qui reçoit un texte et génère un dictionnaire. Chaque clé est un mot du texte, et la valeur de cette clé est une liste de tous les mots qui suivent la clé dans le texte.
- Ecrivez une fonction qui reçoit le nom d'un fichier, lit son contenu et génère un nouveau texte au hasard. Le premier mot W^1 du nouveau texte est un mot aléatoire de l'ancien texte. (Utilisez la fonction `random()` pour le choisir.) Le deuxième mot W^2 est un mot aléatoire dans la liste des mots suivant W^1 dans l'ancien texte, et ainsi de suite. Votre nouveau texte doit être de la même longueur ou plus court que l'ancien texte. Une fois que vous avez fini de générer le texte, écrivez-le dans un (nouveau) fichier. Vous pouvez tester votre fichier sur le roman de Poe, qui est disponible sur Moodle.

- (*) Changez votre générateur de sorte que des triplets de mots consécutifs dans le nouveau texte soient des triples de mots consécutifs dans l'ancien texte.

  Vous pouvez en savoir plus sur les générateurs de texte automatiques sur Wikipedia: **https://fr.wikipedia.org/wiki/Génération_automatique_de_textes**

8. **Classes**

   Les objets sont une encapsulation de variables et de fonctions dans une seule entité. Les objets obtiennent leurs variables et fonctions des classes (méthodes). Les classes sont essentiellement un modèle pour créer vos objets. Une classe très basique ressemblerait à ceci:

```
class Student:
    name = "John"
    family_name = "Sierpinski"
    university = "ENS"

    def summary(self):
          print "My name is %s %s, I am a student of %s" %
(name, family_name, university)

new_student = Student()#create a new object of class Student
print new_student.name #access the variable name and print
its value
new_student.summary() #call the function summary
```

   Vous pouvez créer plusieurs objets différents appartenant à la même classe (les mêmes variables et méthodes sont définies). Cependant, chaque objet contient des copies indépendantes des variables définies dans la classe. Par exemple, si nous devions définir un autre objet avec la classe "Student", puis changer le nom de l'étudiant:

```
another_student = Student()#create a new object of class
Student
another_student.name = "Bob"
another_student.summary() #call the method summary
```

9. **Classe Véhicule**

   Nous avons une classe définie pour les véhicules. Créez deux nouveaux véhicules appelés car1 et car2. Définir car1 pour être un cabriolet rouge d'une valeur de 60 000 $ avec un nom de Fer, et car2 pour être une camionnette bleue nommée Sauter valant 10 000 $.

```
# define the Vehicle class
class Vehicle:
    name = ""
    kind = "car"
    color = ""
    value = 100.00
    def description(self):
        desc_str = "%s is a %s %s worth $%.2f." % (self.name,
self.color, self.kind, self.value)
        return desc_str
# your code goes here
```

```
# test code
print(car1.description())
print(car2.description())
```

10. **(*) Classe Point**

Votre première tâche consiste à ajouter plusieurs méthodes à la classe Point.

```
class Point:
  def __init__(self, x, y):
    self.x = x
    self.y = y
  def distance(self, another_point):
   distance_squared = (self.x - another_point.x)**2 + (self.y
- another_point.y)**2
    return distance_squared**0.5
```

- Ajouter une méthode `tuple` qui renvoie un tuple de coordonnées du point.
- Ajouter une méthode `copy` qui renvoie une copie du point (Quand vous modifiez la copie, le point original ne doit pas changer.)
- Ajouter une méthode `translation` qui reçoit un vecteur (u, v) et un point (x, y), et renvoie un point (x+u, y+v).
- Ajouter une méthode `multiply` qui reçoit un nombre z et un point (x, y), et renvoie un point (xz, yz).

Votre deuxième tache consiste à écrire une fonction qui reçoit une liste de points, considère chaque point comme un vecteur, et retourne la position finale si on suivait ces vecteurs les unes à la suite des autres (en commençant à l'origine).