Introduction to programming, Lesson 4: functions, files, modules

Warm-up

1. In this exercise, you will need to add numbers and strings to correct lists using the “append” method. Add numbers 1, 2, and 3 to the list “numbers”, and strings 'hello' and 'world' to the list “strings”.You will also have to fill in the variable second_name, using the brackets operator [].

```
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. Go to Moodle, Lesson 4, and download the file with #MeToo data. The file contains the data related to the “Me Too” movement, which spread virally as a hashtag used on social media to help demonstrate the widespread prevalence of sexual assault and harassment, especially in the workplace. The file was generated by Google Trends. Each line of the file is of form “week,popularity”. Popularity is a number indicating the popularity of queries related to the movement during a certain week of the year. Your task is to find the week when the movement was most popular. Use function split to break a line into week and popularity.

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

New material

5. **Dictionaries**

A dictionary is a data type similar to lists, but works with keys and values instead of indexes. Each value stored in a dictionary can be accessed using a key, which is any type of object (a string, a number, a list, etc.) instead of using its index to address it. For example, a database of phone numbers could be stored using a dictionary like this:

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

Alternatively, a dictionary can be initialized with the same values in the following notation:

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

Dictionaries can be iterated over, just like a list. However, a dictionary, unlike a list, does not keep the order of the values stored in it. To iterate over key value pairs, use the following syntax:

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

To remove a specified entry, use

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

6. Add "Jake" to the phonebook with the phone number 938273443, and remove Jill from the 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. **Text generator**

- (you can skip this exercise if you have implemented it during previous class) Write a function that generates a list of pairs of consecutive words in a text. Save this function in a module. Write a script that imports the module you have just created, receives a name of a file, reads the file, and prints out the pairs of consecutive words in the file.
- Add to your module a Python function that receives a text and generates a dictionary. Each key is a word of the text, and the value for this key is a list of all the words that follow the key in the text.
- Write a function that receives a name of a file, reads its content, and generates a new text at random. The first word W^1 of the new text is a random word of the old text. (Use function random() to choose it.) The second word W^2 is a random word in the list of words following W^1 in the old text, and so on. Your new text must be of the same length or shorter than the old text. Once you have finished generating the text, write it into a (new) file. You can test your file on The Gold-Bug novel by Poe, which is available on Moodle.
- (*) Change your generator so that triples of consecutive words in the new text are triples of consecutive words in the old text.

You can read more about automatic text generators on Wikipedia: https://en.wikipedia.org/wiki/Natural_language_generation

8. **Classes**

   Objects are an encapsulation of variables and functions into a single entity. Objects get their variables and functions from classes (methods). Classes are essentially a template to create your objects. A very basic class would look something like this:

```
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
```

   You can create multiple different objects that are of the same class (have the same variables and methods defined). However, each object contains independent copies of the variables defined in the class. For instance, if we were to define another object with the "Student" class and then change the name of the student:

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

9. **Class Vehicle**

   We have a class defined for vehicles. Create two new vehicles called car1 and car2. Set car1 to be a red convertible worth $60,000.00 with a name of Fer, and car2 to be a blue van named Jump worth $10,000.00.

```
# 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. **(*) Class Point**

Introduction to programming, Lesson 4: functions, files, modules

Your first task is to add several methods to the class 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
```

- Add a method `tuple` that returns a tuple of coordinates of the point.
- Add a method `copy` that returns a copy of the point. (When we modify the copy, the original point should not change.)
- Add a method `translation` that receives a vector (u, v) and a point (x, y), and returns a point (x+u, y+v).
- Add a method `multiply` that takes a number z and a point (x, y) and returns a point (xz, yz)

Your second task is to write a function that receives a list of points, considers them as vectors, and returns a point that we reach by following these vectors one by one from the origin. Add a second function that returns a list of points we visit on the way.