Refresher

February 10, 2019

1 Refresher Assignment

Programming assignment on Python,Numpy and Pandas

1.1 1. Python

1.1 Functions A python function follows the following syntax `def <function_name>(arguments): our code`
It should be noted that a space or tab (intendation) is necessary for the syntax while definig a fuction.

```
In [1]: #we define a fuction to say hello. The fuction takes one argument(name we need to gree
        def greetings(name):
            #we use python print function to display outputs of the function.
            print("Hello, "+name)
            print("Welcome to cmpe 257 refresher assignment")
```

Now that we have our function definition,we can call it using the following syntax `function_name(pass necessary arguments)`

```
In [2]: greetings('shabari') #here we call greetings functions with a string value(user name)
```

```
Hello, shabari
Welcome to cmpe 257 refresher assignment
```

Another import part of python function to keep in mind is that a function can return values.

```
In [3]: #we define a function to check if users is registered for the class.
        def user_status(name):
            #list of students registered for class
            cmpe257_names = ['bob','foo','baz','alice','shabari']
            return (name in cmpe257_names) #retuns True if name is in the list else False.
```

```
In [4]: status = user_status('shabari')
        print('shabari is in class? : {}'.format(status))
        status = user_status('girish')
        print('girish is in class? : {}'.format(status))
```

```
shabari is in class? : True
girish is in class? : False
```

1.2 Classes A class in python is defined using the following syntax `class <class_name>: class variables and functions` Like functions the intendation is used to check what chunk of code belongs to the class and which don't.

```
In [5]: #We define a class to represent the cmpe257 course
        class cmpe257:
            # the init function in python class is like a constructor. It is the first functio
            # when a object of a class is created.
            def __init__(self):
                # we use the keyword self before using variables of functions that belong to t
                # of the class end up using them,the referece to the variable and functions re
                # here we have defined the list of students registered for the class as seen i
                self.cmpe257_names = ['bob','foo','baz','alice','eve']
                self.hw0_marks = [10,3,6,9,10] #marks for each of the above students in homewo

            def print_record(self):
                # function to print the list of students and their respective marks
                print('HW0 Marks')
                # the zip function is usefull when we have more than one lists/arrays or simil
                # to iterate them both simultaneously.
                for name,mark in zip(self.cmpe257_names,self.hw0_marks):
                    print('{} : {}'.format(name,mark))
```

Now that we have the class let's create a object for it and use the object to print the homework marks. object creation has the following syntax `<object_name> = <class_name>()` It should be noted that inital arguments may be passed to the class constructor using `<class_name>(arguments)`. The `.` operator is used to access the variables and functions that belong to the object.Example `<object_name>.<vairable_name>` or `<object_name>.<function_name>`

```
In [6]: #object declaration
        cmpe257_03 = cmpe257()
        #acess function to print records
        cmpe257_03.print_record()
```

```
HW0 Marks
bob : 10
foo : 3
baz : 6
alice : 9
eve : 10
```

1.3 Inheritance A class can inherit another class using the following syntax `class classB(classA): class variables and functions`

In the above syntax the classB inherits a class named classA

```
In [15]: # We define a class that holds the family name.
         class Family:
             def __init__(self):
                 self.family_name = 'Ganapathy'

         #Every member of the family will end up with the
         #same family name initially. Though they can end
         #up choosing a new one later.
         class Member(Family):
             def __init__(self,first_name):
                 #we now initialise our parent class
                 Family.__init__(self)
                 self.first_name = first_name

             def print_full_name(self):
                 print("{} {}".format(self.first_name,self.family_name))
```

```
In [16]: member = Member('Shabari')
         member.print_full_name()
```

```
shabari Ganapathy
```

```
In [17]: #Now let's say we need to change the family name for
         #a new member
         member = Member('Ben')
         member.family_name = 'Adams'
         member.print_full_name()
```

```
Ben Adams
```

1.4 Tuples A tuple is represented as follows `a = (1,2,3,4)` It is similar to a python list with the only difference being a tuple's value cannot change(immutable)

```
In [18]: #we have a list of students with their enrollment
         #numbers.Since these numbers cannot and should not change
         #we use a tuple to store the values.
         enrollment_numbers = [
             ('bob',101),
             ('foo',102),
             ('baz',103),
             ('alice',104),
             ('eve',105)
         ]

         print('{} : {}'.format(enrollment_numbers[1][0],
                                 enrollment_numbers[1][1]))
```

```
foo : 102
```

```
In [19]: #now lets try and update the a value
         enrollment_numbers[1][1] = 106
```

```
        ---------------------------------------------------------------------------

        TypeError                                 Traceback (most recent call last)

        <ipython-input-19-4c581d4dd821> in <module>
          1 #now lets try and update the a value
    ----> 2 enrollment_numbers[1][1] = 106


        TypeError: 'tuple' object does not support item assignment
```

1.5 Dictionary A disctionary is a set of key value pairs represented as follows d = {key1 : value1, key2 : value2, key_n : value_n}

```
In [20]: class cmpe257:
             def __init__(self):
                 self.cmpe257_names = ['bob','foo','baz','alice','eve']
                 #marks for homework zero represented with student name
                 self.hw0_marks = {'bob' : 10,
                                   'foo' : 3,
                                   'baz' : 6,
                                   'alice' : 9,
                                   'eve' : 10}

             def print_record(self):
                 print('HW0 Marks')
                 for student in self.hw0_marks:
                     print('{} : {}'.format(student,
                                            self.hw0_marks[student]))

         cmpe257_03 = cmpe257()
         cmpe257_03.print_record()
```

```
HW0 Marks
bob : 10
foo : 3
baz : 6
alice : 9
eve : 10
```

1.6 Sets

```
In [26]: #courses taken in fall
         courses_eve = {'cmpe202','cmpe273','cmpe255'}
         #adding courses for spring
         courses_eve.add('cmpe275')
         courses_eve.add('cmpe255')
         courses_eve.add('cmpe272')
```

```
In [29]: print(courses_eve)
         #total credits student will complete by the semester
         print(len(courses_eve)*3)
```

```
{'cmpe202', 'cmpe272', 'cmpe275', 'cmpe273', 'cmpe255'}
15
```

1.2 2. Numpy

2.1 Array Indexing 1-D array indexing is similar to array indexing in any programming language.we use `array[num]` to access element at position `num`. 2-D array is also similar with the only difference being both column and row are represented together inside the same box bracket. To get element at (2,3) use `array[2,3]`

```
In [30]: #Since Numpy is a separate python package which we installed
         #We first need to import it into our program env.
         import numpy as np
```

```
In [33]: #We define an array using a list to represent each row.
         a = np.array([1,2,3,4,5,6])
         for i in range(len(a)//2):
             tmp = a[i]
             #it could be noticed here that 0-i-1 is a negative number
             #Numpy arrays negative index means element count from the
             #end of the array
             a[i] = a[0-i-1]
             a[0-i-1] = tmp

         print(a) # we have now reversed the array using negative index.
```

```
[6 5 4 3 2 1]
```

2.1 Array Comparison Just like any two numbers, numpy arrays have a ton of comaprisons that could be used between them.Say we need `= <= >= !=` all of the logical operations can be performed on two numpy arrays

```
In [42]: #student-1 score in 4 assignments
         student_1 = np.array([10,9,8,10])
         #student-2 score in 4 assignments
         student_2 = np.array([10,8,5,9])
         #variable to count truth values
         t_count = 0

         #np.greater does an elementwise comparison to return
         #a numpy array that has the truth values
         for item in np.greater(student_1,student_2):
             if item == True:
                 t_count+=1
         #If student-1 has greater score in more than 2 subjects
         # he/she will have better grades else student-2 will have
         #better grades
         if t_count > len(student_1)//2:
             print("student 1 has better grades")
         else:
             print("student 2 has better grades")
```

```
student 1 has better grades
```

2.3 Multi Array Math Just like any two numbers, numpy arrays have arithmetic that could be used between them. Say we need + - * all of the arithmetic operations can be performed on two numpy arrays.

```
In [62]: #array of student's first semester grades per course
         student_grades = np.array([3.0,4.0,3.3])
         #array of each course's credit point
         course_credits = np.array([3.0,3.0,3.0])
         #np.tranpose() is used so that the two matrices dimentions are such
         #that matrix multiplication can be performed
         total = student_grades.dot(np.transpose(course_credits))
         print("Total : {}".format(total))
         #final GPA for the semester is calculated by dividing the total by
         #the total credits the student took last semester
         print("GPA : {0:.2f}".format(total/np.sum(course_credits)))
```

```
Total : 30.9
GPA : 3.43
```

2.4 Subsetting Slicing an array can be done using : for either a row or column

```
In [54]: #2-D matrix representing:
         # course-id,grade point, course credits
         record = np.array([[202,3.0,3.0],
                           [272,3.3,3.0],
                           [273,4.0,3.0],
                           [255,4.0,3.0],
                           [257,3.7,3.0],
                           [275,4.0,3.0]])

         print(record)
```

```
[[202.    3.    3. ]
 [272.    3.3   3. ]
 [273.    4.    3. ]
 [255.    4.    3. ]
 [257.    3.7   3. ]
 [275.    4.    3. ]]
```

```
In [55]: # we slice the 2-D matrix
         # for rows 1 to 3 we change credit to 1.0
         record[0:3,2] = 1.0
         print(record)
```

```
[[202.    3.    1. ]
 [272.    3.3   1. ]
 [273.    4.    1. ]
 [255.    4.    3. ]
 [257.    3.7   3. ]
 [275.    4.    3. ]]
```

2.5 Ravel

```
In [61]: letter_grades = np.array([[3.0,3.3,3.0],
                                   [2.7,3.3,3.0],
                                   [4.0,4.0,4.0],
                                   [3.0,4.0,3.7]])
         letter_grades = np.ravel(letter_grades) * 3.0
         np.sum(letter_grades)/(len(letter_grades)*3.0)

Out[61]: 3.4166666666666665

In [ ]:
```