PS2_Chowdhury

September 11, 2017

0.1 The Standard Library

Problem 1

```
In [12]: Input_list=[4,5,8,4.5,6.2,9,12,4,2.2]

In [16]: New_list=[min(Input_list),max(Input_list), round(float (sum(Input_list)/len(Input_lis
         New_list

Out[16]: [2.2, 12, 6.1]

In [10]: len(Input_list)

Out[10]: 9
```

Problem 2
Numbers

```
In [31]: num_1 = 5

In [32]: num_2 = num_1

In [40]: num_2 = num_2 + 2

In [42]: num_1 == num_2

Out[42]: False
```

Since alterning num_2 did not alter num_1, we conclude that a copy has been created and that therefore numbers are immutable.

Strings

```
In [43]: word_1 = 'a'

In [45]: type(word_1)

Out[45]: str

In [46]: word_2 = word_1
```

```
In [49]: word_2='b'

In [50]: word_1 == word_2

Out[50]: False
```

Since alterning word_2 did not alter word_1, we conclude that a copy has been created and that therefore numbers are immutable.

lists

```
In [5]: list_1 = [1,2,3,4,5]

In [6]: list_2 = list_1

In [8]: list_2.append(6)

In [10]: list_1 == list_2

Out[10]: True
```

Since alterning list_2 altered list_1, we conclude that no copy has been created and that therefore numbers are mutable.

Tuples

```
In [23]: tuple_1 = (1,2,3,4,5)

In [24]: tuple_2 = tuple_1

In [25]: tuple_2 = tuple_2 + (6,7)

In [27]: tuple_1 == tuple_2

Out[27]: False
```

Since alterning tuple_2 did not alter tuple_1, we conclude that a copy has been created and that therefore numbers are immutable.

Dictionaries

```
In [30]: dict_1={1:'A',2:'B'}

In [31]: dict_2=dict_1

In [33]: dict_2[1]='T'

In [34]: dict_1 == dict_2

Out[34]: True
```

Since alterning dict_2 altered dict_1, we conclude that no copy has been created and that therefore numbers are mutable.

Problem 3

```
In [28]: import calculator as cal

In [43]: def hypotenuse(val1,val2):
             hypote = cal.squareroot((cal.summation((cal.product(val1,val1)),(cal.product(val2
             return hypote

In [45]: hypotenuse(4,3)

Out[45]: 5.0
```

0.2 Introduction to NumPy

Problem 1

```
In [2]: import numpy as np

In [3]: A= np.array([[3,-1,4],[1,5,-9]])

In [4]: print(A)

[[ 3 -1  4]
 [ 1  5 -9]]


In [5]: B = np.array([[2,4,-5,3],[5,-8,9,7],[9,-3,-2,-3]])

In [6]: print(B)

[[ 2  4 -5  3]
 [ 5 -8  9  7]
 [ 9 -3 -2 -3]]


In [7]: AB = np.dot(A,B)

In [8]: print(AB)

[[ 37   8 -32 -10]
 [-54  -9  58  65]]
```

Problem 2

```
In [9]: A = np.array([[3,1,4],[1,5,9],[-5,3,1]])

        print(A)

[[ 3  1  4]
 [ 1  5  9]
 [-5  3  1]]


In [10]: AA=np.dot(A,A)

In [11]: print(AA)

[[-10  20  25]
 [-37  53  58]
 [-17  13   8]]
```

```
In [13]: AAA=np.dot(AA,A)
         z33 = np.zeros((3,3), dtype=np.int)
         z33AAA

Out[13]: array([[-135,  165,  165],
                [-348,  402,  387],
                [ -78,   72,   57]])

In [14]: B = 9*AA
         B

Out[14]: array([[ -90,  180,  225],
                [-333,  477,  522],
                [-153,  117,   72]])

In [15]: C = 15*A
         C

Out[15]: array([[ 45,  15,  60],
                [ 15,  75, 135],
                [-75,  45,  15]])

In [16]: D = -AAA + B - C
         D

Out[16]: array([[0, 0, 0],
                [0, 0, 0],
                [0, 0, 0]])
```

Problem 5

```
In [3]: A = np.array([[0,2,4],[1,3,5]])
        A

Out[3]: array([[0, 2, 4],
               [1, 3, 5]])

In [4]: B = np.array([[3,0,0],[3,3,0],[3,3,3]])
        B

Out[4]: array([[3, 0, 0],
               [3, 3, 0],
               [3, 3, 3]])

In [5]: C = np.array([[-2,0,0],[0,-2,0],[0,0,-2]])
        C

Out[5]: array([[-2,  0,  0],
               [ 0, -2,  0],
               [ 0,  0, -2]])
```

```
In [6]: Atrans = A.T
        Atrans

Out[6]: array([[0, 1],
               [2, 3],
               [4, 5]])

In [7]: z23 = np.zeros((2,3), dtype=np.int)
        z23

Out[7]: array([[0, 0, 0],
               [0, 0, 0]])

In [8]: z22 = np.zeros((2,2), dtype=np.int)
        z22

Out[8]: array([[0, 0],
               [0, 0]])

In [9]: z33 = np.zeros((3,3), dtype=np.int)
        z33

Out[9]: array([[0, 0, 0],
               [0, 0, 0],
               [0, 0, 0]])

In [10]: z32 = np.zeros((3,2), dtype=np.int)
         z32

Out[10]: array([[0, 0],
                [0, 0],
                [0, 0]])

In [11]: I = np.eye((3), dtype=np.int)
         I

Out[11]: array([[1, 0, 0],
                [0, 1, 0],
                [0, 0, 1]])

In [12]: X1 = np.hstack((z33,Atrans,I))
         X1

Out[12]: array([[0, 0, 0, 0, 1, 1, 0, 0],
                [0, 0, 0, 2, 3, 0, 1, 0],
                [0, 0, 0, 4, 5, 0, 0, 1]])

In [13]: X2 = np.hstack((A,z22,z23))
         X2
```

```
Out[13]: array([[0, 2, 4, 0, 0, 0, 0, 0],
                [1, 3, 5, 0, 0, 0, 0, 0]])

In [14]: X3 = np.hstack((B,z32,C))
         X3

Out[14]: array([[ 3,  0,  0,  0,  0, -2,  0,  0],
                [ 3,  3,  0,  0,  0,  0, -2,  0],
                [ 3,  3,  3,  0,  0,  0,  0, -2]])

In [15]: X = np.vstack((X1,X2,X3))
         X

Out[15]: array([[ 0,  0,  0,  0,  1,  1,  0,  0],
                [ 0,  0,  0,  2,  3,  0,  1,  0],
                [ 0,  0,  0,  4,  5,  0,  0,  1],
                [ 0,  2,  4,  0,  0,  0,  0,  0],
                [ 1,  3,  5,  0,  0,  0,  0,  0],
                [ 3,  0,  0,  0,  0, -2,  0,  0],
                [ 3,  3,  0,  0,  0,  0, -2,  0],
                [ 3,  3,  3,  0,  0,  0,  0, -2]])
```

0.3 Object-Oriented Programming

Problem 1

```
In [1]: class Backpack(object):
            """A Backpack object class. Has a name and a list of contents.

            Attributes:
                name (str): the name of the backpack's owner.
                contents (list): the contents of the backpack.
                color(str): the color of the backpack.
                max_size(int) : the maximum number of items that can be put in the backsack.

            """
            def __init__(self,name, color, max_size):
                """Set the name and initialize an empty contents list.

                Inputs:
                    name (str): the name of the backpack's owner.
                    color(str): the color of the backpack.
                    max_size(int): the size of the backpack.

                Returns:
                    A Backpack object wth no contents.

                """
                self.name = name
```

```
        self.color = color
        self.max_size = 5
        self.contents = []

    def put(self,item):
        """Add 'item' to the backpack's list of contents
        'max_size' limits the backpack's list of contents"""
        if len(self.contents)>=self.max_size:
            print("No Room!")
        else:
            self.contents.append(item)


    def take(self, item):
        """Remove 'item' from the backpack's list of contents"""
        self.contents.remove(item)

    def dump(self):
        """If the Backpack to be emptied, use Backpack.dump() method"""
        self.contents = []
```

```
In [2]: my_backpack = Backpack("Maidul","Black",5)

In [3]: my_backpack.max_size

Out[3]: 5

In [4]: my_backpack.put("notebook")
        my_backpack.put("pencils")
        my_backpack.contents

Out[4]: ['notebook', 'pencils']

In [5]: my_backpack.put("Eraser")
        my_backpack.put("pen")
        my_backpack.put("stapler")
        my_backpack.put("watch")
        my_backpack.contents

No Room!


Out[5]: ['notebook', 'pencils', 'Eraser', 'pen', 'stapler']

In [6]: my_backpack.dump()

In [7]: my_backpack.contents

Out[7]: []
```

Problem 2

```
In [10]: class Jetpack(Backpack):
             """A Jetpack object class. Inherits from the Backpack class.
             A jetpack is smaller than a backpack and can accept an amount of fuel.

             Attributes:
                 name (str): the name of the Jetpack's owner.
                 contents (list): the contents of the Jetpack.
                 color(str): the color of the Jetpack.
                 max_size(int) : the maximum number of items that can be fit in the Jetsack.
                 Fuel(int) : Amount of fuel.
             """
             def __init__(self,name,color,max_size):
                 """Use the Backpack constructor to initialize the name, color, and max_size a

                 Inputs:
                     name (str): the name of the Jetpack's owner.
                     color(str): the color of the Jetkpack.
                     max_size(int): the maximum number of items that can be fit in the Jetpack

                 Returns:
                     A Jetpack object with no contents.
                 """
                 Backpack.__init__(self,name,color,max_size)
                 self.max_size=2
                 self.limit=0
                 self.fuel=10
                 self.usage = []
             def fly(self,amt):
                 """If add fuel to Jetpack, use the Jetpack.fly() method."""
                 if (self.fuel-sum(self.usage))<=self.limit:
                     print("Not enough fuel!")
                 else:
                     self.usage.append(amt)
             def dump(self):
                 """If the Jetpack to be emptied, use Jetpack.dump() method"""
                 self.contents = []
                 self.usage = []

In [11]: my_jetpack = Jetpack("Maidul","Black",2)

In [12]: my_jetpack.put("pencil")
         my_jetpack.put("book")
         my_jetpack.contents

Out[12]: ['pencil', 'book']

In [13]: my_jetpack.put("pen")
         my_jetpack.contents
```

```
No Room!


Out[13]: ['pencil', 'book']

In [14]: my_jetpack.fly(2)

In [15]: my_jetpack.fly(7.5)

In [16]: my_jetpack.fly(0.5)

In [17]: my_jetpack.fly(1)

Not enough fuel!


In [18]: my_jetpack.dump()

In [19]: my_jetpack.usage

Out[19]: []

In [20]: my_jetpack.contents

Out[20]: []

In [ ]:
```