Welcome to Jupyter Notebook

Welcome to Jupyter Notebook, one of the many ways that you can use Python. Jupyter notebooks run lines of code in what are called **cells**. This makes it convenient to run certain lines of code and see the output of those lines of code in real time. To run a cell of code, simply press Shift + Enter.

Python Functions

Python is a dynamically typed language, meaning you don't have to declare the types of variables. These are determined at run time. Also, Python uses spacing to determine scopes and groupings. For example, when you write a function, it will look like this:

```
In [ ]: #Function definition
        def helloWorld():
            print "Hello World"

        #Calling the function
        helloWorld()
```

Note that the **def** means we are defining a function. The colon is necessary for defining what is part of the function. Also, everything that is part of the function needs to be spaced separately (convention is 4 spaces or a tab). Spacing is the number one reason for code not working.

Tuples

One of Python's useful capabilities is its ability to pass and return tuples. For example, if you need to return two pieces of information from a function, you simply can name the two pieces of information and treat them as a tuple.

```
In [ ]: def rectangleInfo(width, height):
            perimeter = 2*width + 2*height
            area = width * height
            return perimeter, area

        myPerimeter, myArea = rectangleInfo(10,10)
        print "My perimeter is", myPerimeter, "and my area is", myArea, "."
```

Default Parameters

Lastly, Python has a very simple default parameter value system. To state a default parameter, simply state what that parameter would be equal to if no parameter is given in the function description.

In [ ]:
```
def countByNum(startingNumber=1,stepSize=1,numIters=10):
    text = ""
    count = startingNumber
    for i in range(0,numIters):
        text += str(count)
        if i < numIters-1:
            text += ","
        count += stepSize
    print text

#The different ways to use default parameters
countByNum()
countByNum(2, 2)
countByNum(stepSize=2)
```

Function 1: General Fibonaci Sequence

Write a function that takes in the number **n** and it returns the **(n-1)th** and **nth** Fibonacci numbers. Also, allow for the user to specify the **0th** and **1st** Fibonacci numbers that start off the sequence, but they should default to the standard (n_0=1 and n_1=1).

In [ ]:
```
#Your code here
def fibonacci(num, n0, n1):
    return None
```

To test that your code is working, try finding the 6th and 5th Fibonacci numbers for $n_0 = 2$ and $n_1 = 2$. You should get 16 and 26 because the sequence would be (2, 2, 4, 6, 10, 16, 26). The second print statement is how we will grade that you completed the task. **Do not delete any given test cases**.

In [ ]:
```
#Test Cases
print fibonacci(6, 2, 2)
print fibonacci(24, 3, 5)
```

--

Data Structures - Lists

Python has two primary types of data structures built in: lists and dictionaries. These data types are the most commonly used for Python methods and programming. Other types of data structures can be imported as needed. Let's start with lists.

In [ ]:
```
myList = [] #Square brackets initialize a list

#Add elements to the list
myList.append(3)
myList.append(4)
myList.append(7)
#Insert the number 10 at the 0th index
myList.insert(0, 10)

print myList
```

In [ ]:
```
#List concatenation is very easy
print [1,2,3,4,5] + [6,7,8,9,10]
```

In [ ]:
```
#Accessing an element in a list
print myList[1]
#Grab the last element in a list
print myList[-1]
```

In [ ]:
```
#Grab all elements in a list
print myList[:]
#Grap a certain section of the list
print myList[0:2]
```

Warning: Python excludes the last number in a range, as shown above. This is different from most common programming languages. To see this, run the following code.

In [ ]:
```
print range(0,10)
```

Going back to lists, Python has a very powerful list accessing system known as slice notation. It can be summarized as

[first element to include : first element to exclude : step]

Some common examples are given below.

In [ ]:
```
myList2 = range(0,10)
print myList2
```

In [ ]:
```
#Grab every other element in the list
print myList2[::2]

#Grab every other element for the first 5 numbers
print myList2[0:5:2]

#Reverse the list
print myList2[::-1]
```

In Python, for loops are designed with lists in mind. You can see some examples below.

In [ ]:
```
myList3 = [1, 5, 8, 19, 7, 6]
total = 0

# You may be tempted to do this
length = len(myList3)

for i in range(0, length):
    num = myList3[i]
    total += num

print total
```

That is a very Java type way of doing list iteration, but there are better ways in Python. For example:

In [ ]:
```
#It is much faster to do this
total = 0

for num in myList3:
    total += num

print total
```

Note that Python can give you elements of a list as the iterator through a for loop. This is very helpful in many problems.

For more help with lists, you can visit https://docs.python.org/3/tutorial/datastructures.html (https://docs.python.org/3/tutorial/datastructures.html)

Function 2: List Manipulation

Write a function called listShift. This function should take the second half of a list, reverse it, and append it to first. For example,

[1, 2, 3, 4, 5, 6, 7, 8, 9] would become [9,8,7,6,1,2,3,4,5]

Hint: This is a lot easier if you don't use a for loop.

In [ ]:
```
#Your code here
def listShift(myList):
    return None
```

In [ ]:
```
#Test Case
print listShift([1,2,3,4,5,6,7,8,9])
print listShift([10,9,8,7,6,5,4,3,2,1,10,9,8,7,6,5,4,3,2,1])
```

Data Structures - Dictionaries

We won't be using dictionaries as much in this class, but would be an injustice if you didn't get some exposure to them. They are one of the most powerful constructs in Python. The dictionary data structure is also sometimes known as a map data structure in other languages.

In [ ]:
```
myDictionary = {} #Curly brackets initialize a dictionary

#Add elements to dictionary using key/value system
myDictionary["University"] = "BYU"
myDictionary["City"] = "Provo"

print myDictionary
```

In [ ]:
```
#Another way to instatiate a dictionary
myDictionary2 = {'Name': 'Zara', 'Age': 7, 'Grade': '1'}
print myDictionary2

#Go through data in a dictionary
for key in myDictionary2:
    print myDictionary2[key]

#Remove data
del myDictionary2["Grade"]
print myDictionary2

myDictionary2.clear()
print myDictionary2
```

Python Classes

Python can use object oriented programming through the use of classes like most languages. However, the syntax for classes is very unique to Python.

Below is an example of a class in Python.

In [ ]:

```
from math import pi

class Circle():

    def __init__(self, radius, center_x=0.0, center_y=0.0):
        self.radius = radius
        self.x = center_x
        self.y = center_y

    def getCircumference(self):
        return 2*pi*self.radius

    def getArea(self):
        return pi*self.radius**2

    def getArc(self, angle):
        return self.radius * angle

myCirc = Circle(25)
print myCirc.getCircumference()
print myCirc.getArea()
print myCirc.getArc(1.5)
```

Note the keyword **class** is used to start the definition. The constructor for the class is always defined within the **init** function. Any other functions can be named as needed.

Also, note that the keyword **self** must be passed into each function. This is where the variables for the class are stored. You can think of it similar to the **this** keyword in Java. However, note that this variable does not need to be passed in on call, only definition.

Inheritance

Classes can also inherit methods and properties from other classes. This can be done by placing the name of the super class in the parenthesis of the class defintion. These super classes can also have abstract methods by using the keyword **pass**.

In [ ]:
```
class Shape():

    def getPerimeter(self):
        pass

    def getArea(self):
        pass

class Square(Shape):

    def __init__(self, size):
        self.size = size

    def getPerimeter(self):
        return 4*self.size

    def getArea(self):
        return self.size**2


mySquare = Square(10)
if isinstance(mySquare, Shape):
    print mySquare.getPerimeter()
```

For more information on classes in Python, see https://docs.python.org/3/tutorial/classes.html (https://docs.python.org/3/tutorial/classes.html)

Functions 3: Push and Pop

Write your own class called MyPriorityQueue. This class should have two methods: **push** and **pop**. Push takes in a value and priority number and puts it in a dictionary. Pop returns the value with highest priority number and removes that value from the dictionary.

In [ ]:
```
class MyPriorityQueue():

    #Your Code Here
```

In [ ]:
```
#Test Case
myQueue = MyPriorityQueue()
myQueue.push("were toiling upward in the night.", 2)
myQueue.push("The heights by great men reached and kept", 5)
myQueue.push("-Longfellow", 1)
myQueue.push("were not attained by sudden flight,", 4)
myQueue.push("but they, while their companions slept,", 3)
print myQueue.pop()
print myQueue.pop()
print myQueue.pop()
print myQueue.pop()
print myQueue.pop()
```

Importing Packages in Python

Python has multiple ways to import packages to use in the system. The keywords you can use are **import**, **from**, and **as**.

In [ ]:
```
import math
print math.pi

from math import pi
print pi

from math import pi as taco
print taco

from math import * #Import everything from the package
print e
```

One of the main packages we will use is called Numpy. It is a huge matrix processing library that is very helpful for image processing and graphics. The conventional import method looks like this.

In [ ]:
```
import numpy as np
print np.version.version
```

Make sure that you have at least version 1.08 running. If you can't get the package to work, make sure that you run *pip install numpy* or *conda install numpy* in the command line according to the setup instructions.

Numpy allows you store matrices natively and do matrix operations in a single step.

Let's look at a simple matrix.

In [ ]:
```
import numpy as np
a = np.matrix([[1, 2],[3,4]])

print a
```

In [ ]:
```
print a[0,1] #Row, Column zero-based indexing
```

In [ ]:
```
print a[:,1] #Grab all rows and the second column
print
print a[0,:] #Grab everything in the first row
```

Note that all splice notation works with Numpy arrays.

You can also do operations on all elements of a Numpy array at the same time. This is shown below and is called vectorization (we will talk about this more later).

In [ ]:
```
print a + 1
print
print 3*a
```

In [ ]:
```
print np.multiply(a,a) #Element-wise multiply
print
print a*a #True matrix multiply
```

There are plenty of Numpy operations that we are not covering, but we will see more of these as we go throughout the class. For more details, go to https://docs.scipy.org/doc/numpy/reference/ (https://docs.scipy.org/doc/numpy/reference/) .

Now let's look at how we can import images as multidimensional arrays.

In [ ]:
```
from scipy.ndimage import imread
geese = imread('geese.jpg')

import matplotlib.pyplot as plt
plt.imshow(geese)
plt.title("The Geese")
plt.show()
```

In [ ]:
```
#Notice geese is a 3-dimensional Numpy array (row, col, rgb color).
#In this case, it is 256 x 256 x 3
#This means we can access it like any other list in Python.
print geese[0,0,:]
```

Functions 4: Working with Numpy

Write the following two functions:

1. **flip**: Takes in a numpy array, flips it upside down and returns it. (Hint: You don't need a for loop)
2. **greenify**: Takes in a 3-dimensional array and returns only the green channel of the array.

In [ ]:
```
#Your Code Here
def flip(image):
    return None

def greenify(image):
    return None
```

In [ ]:
```
#Test Cases
from scipy.ndimage import imread
geese = imread('geese.jpg')

test1 = flip(geese)
plt.imshow(test1); plt.title("Flipped"); plt.show()

test2 = greenify(geese)
plt.imshow(test1, cmap="Greys_r"); plt.title("The Grey Value of the Green Chan
nel"); plt.show()
```

In [ ]:
```
```