HW0 (Score: 14.5 / 15.0)

1. Comment
2. Test cell (Score: 1.5 / 2.0)
3. Comment
4. Test cell (Score: 2.0 / 2.0)
5. Comment
6. Test cell (Score: 3.0 / 3.0)
7. Comment
8. Test cell (Score: 3.0 / 3.0)
9. Test cell (Score: 3.0 / 3.0)
10. Comment
11. Test cell (Score: 2.0 / 2.0)

DATA 601: Fall 2019¶

HW0¶

Due: Wed. Sep. 11 at 23:55¶

Learning Objectives

- Explore built-in data types in Python.
- Review fundamental programming and problem solving concepts with Python.
- Implement functions based on mathematical concepts and definitions.
- Gain experience working with the Jupyter notebook environment.

This is an individual homework assignment.

Please complete this homework assignment within the Jupypter notebook environment.

Submission¶

Your submission will be auto-graded. In order to ensure that everything goes smoothly, please follow these instructions:

- Please provide your solutions where asked; please do not alter any other parts of this notebook.
- Submit via the HW0 dropbox on D2L. Please ensure that your submitted file is named 'HW0.ipynb'.
- Do not submit any other files.

In [1]:

```
# Check that we are using a recent version of Jupyter.
import IPython
assert IPython.version_info[0] >= 3, "Your version of IPython is too old, please update it."
```

Part A¶

This part focuses on scalar types. You should be able to complete the following questions without using any collection types.

Question 1¶

The n-th triangular number T_n ($n \ge 1$) is defined as: $$ T_n := \sum_{k=1}^n k = 1 + 2 + \cdots + n = \frac{n(n+1)}{2} $$

In the cell below, please provide the code for the function `tri(n)` that returns the n-th triangular number as an integer.

In [2]:
Student's answer(Top)

```
def tri(n):
    '''Returns the n-th triangular number for an input
    integer n.
    '''
    return (n * (n + 1)) / 2
    raise NotImplementedError()
```

Comments:

This is almost correct, but there's a subtle bug here. Hint: all triangular numbers should be integers.

Also, `raise NotImplementedError()` is redundant. No marks lost for that, though.

In [3]:
Grade cell: `tri_test` Score: 1.5 / 2.0 (Top)

```
'''Check that tri returns the correct output'''
assert tri(1) == 1
assert tri(5) == 15
assert tri(36) == 666
assert tri(10 ** 5) == 5000050000
### BEGIN HIDDEN TESTS
assert tri(10 ** 10) == 50000000005000000000
### END HIDDEN TESTS
```

```
---------------------------------------------------------------------------
AssertionError                            Traceback (most recent call last)
<ipython-input-3-000d8485dc7f> in <module>
      5 assert tri(10 ** 5) == 5000050000
      6 ### BEGIN HIDDEN TESTS
----> 7 assert tri(10 ** 10) == 50000000005000000000
      8 ### END HIDDEN TESTS

AssertionError:
```

Question 2¶

In the cell below, provide code for the function `powerOfTwo(n)` that takes a positive integer `n` and returns `True` if `n` is a power of 2, and `False` otherwise.

In [4]:
Student's answer(Top)

```
def powerOfTwo(n):
    '''For a positive integer n, returns True if n is a
    power of two, False otherwise.
    '''

    x = 0
```

```
    y = 0
    while y < n:
        y = 2**x
        x += 1
    return n == y
    raise NotImplementedError()
```

Comments:

Interesting approach, though it could be simplified.

In [5]:
Grade cell: `powerOfTwo_test` Score: 2.0 / 2.0 (Top)

```
'''Check that powerOfTwo returns the correct output'''

assert powerOfTwo(1) is True
assert powerOfTwo(2) is True
assert powerOfTwo(5) is False
assert powerOfTwo(64) is True
assert powerOfTwo(96) is False
### BEGIN HIDDEN TESTS
assert powerOfTwo(128) is True
assert powerOfTwo(192) is False
### END HIDDEN TESTS
```

Question 3¶

The n-th Fibonacci number F_n can be calculated using the following closed-form formula: $$ F_n = \frac{\varphi^n - \psi^n}{\sqrt{5}}, $$ where $\varphi = \frac{1+\sqrt{5}}{2}$ and $\psi = \frac{1-\sqrt{5}}{2}$. Use this formula to compute F_n in the function `fib(n)` below.

In [6]:
Student's answer(Top)

```
import math
def fib(n):
    '''Computes the n-th Fibonacci number for a positive
    integer n and returns the answer as an integer.
    '''

    phi = (1 + 5**(1/2)) / 2
    psi = (1 - 5**(1/2)) / 2
    return int((phi**n - psi**n) / (5**(1/2)))
    raise NotImplementedError()
```

Comments:

Kudos for avoiding math.sqrt().

In [7]:
Grade cell: `fib_test` Score: 3.0 / 3.0 (Top)

```
'''Check that fib returns the correct output'''
from nose.tools import assert_equal
assert fib(1) == 1
assert fib(2) == 1
assert fib(5) == 5
assert fib(11) == 89
### BEGIN HIDDEN TESTS
assert fib(20) == 6765
### END HIDDEN TESTS
```

Part B¶

Questions in this part rely on strings, lists and tuples.

Question 1¶

Write a function called `isPalindrome` which takes a `string` as an input and returns `True` if the string is a palindrome, and `False` otherwise.

In [8]:
Student's answer(Top)

```
def isPalindrome( str ):
    '''
    Returns True if str contains a string that is a palindrome,
    False otherwise. Does not ignore whitespace characters.
    '''
    new_str = []
    old_str = []
    for i in range(len(str)):
        new_str.append(str[-(i+1)])
        old_str.append(str[i])
    return old_str == new_str
    raise NotImplementedError()
```

Comments:

Interesting approach, but `old_str` has the same contents as `str`. This could be shortened.

In [9]:
Grade cell: `isPalindrome_test` Score: 3.0 / 3.0 (Top)

```
'''Check that isPalindrome returns the correct output'''
assert isPalindrome("") is True
assert isPalindrome("a") is True
assert isPalindrome("ab") is False
assert isPalindrome("aa") is True
assert isPalindrome("racecar") is True
assert isPalindrome("race car") is False
assert isPalindrome("step on no pets") is True
### BEGIN HIDDEN TESTS
assert isPalindrome("rats live on no evil star") is True
### END HIDDEN TESTS
```

Question 2¶

Write a function called `transpose` that returns the transpose of an input matrix. You can assume that the input matrix has integer entries and is represented as a list of rows. Furthermore, you can assume that the input is a matrix and not a vector.

In [10]:
Student's answer(Top)

```
def transpose(A):
    '''Returns the transpose of A as a list of lists. Each
    column of A becomes a row in the returned matrix.
    '''
    new_list = [[] for k in range(len(A[0]))]
    for i in range(len(A)):
```

```
        for j in range(len(A[i])):
            new_list[j].append(A[i][j])
    return new_list
    raise NotImplementedError()
```

In [11]:
Grade cell: `transpose_test` Score: 3.0 / 3.0 (Top)

```
'''Check that transpose returns the correct output'''
assert transpose([[1,2],[3,4]]) == [[1,3],[2,4]]
assert transpose(transpose([[1,2],[3,4]])) == [[1,2],[3,4]]
assert transpose([[1,2,3],[4,5,6]]) == [[1,4],[2,5],[3,6]]
assert transpose([[1,4],[2,5],[3,6]]) == [[1,2,3],[4,5,6]]
### BEGIN HIDDEN TESTS
I = [[1,0,0],[0,1,0],[0,0,1]]
assert transpose(I) == I
T4 = [[1,2,3,4],[5,6,7,8],[9,10,11,12],[13,14,15,16]]
assert transpose(transpose(T4)) == T4
### END HIDDEN TESTS
```

Question 3¶

Suppose you are given a list of floating point values in the range $[0.0,1.0)$. Write a function to compute a histogram of these values. Your function should take two input parameters, the list `L` and the number of equal-width bins `N` to split the range into. It should return a list containing the counts as integers.

For each bin, the left end point is included but the right end-point is not. For example, if we have five bins, the boundaries would be as follows: $[0.0,0.2), [0.2, 0.4), [0.4, 0.6), [0.6, 0.8), [0.1, 1.0)$

In [12]:
Student's answer(Top)

```
def hist(L, N=10):
    '''For a list of values L, returns a histogram
       consisting of N equal-width bins. The histogram is returned as
       a list consisting of the counts in each bin. The values are
       assumed to be in floating point format in the range [0.0,1.0).
    '''
    hist = [0] * N
    for i in L:
        hist[int(i * N)] += 1
    return hist
    raise NotImplementedError()
```

Comments:

Clean and efficient. But could you do it in one line?

In [13]:
Grade cell: `hist_test` Score: 2.0 / 2.0 (Top)

```
'''Check that hist returns the correct output'''
import random
SIZE = 10000
# Do not change the random number seed or tests will fail.
random.seed(a=601, version=2)
l1 = []
for i in range(SIZE):
    l1.append(random.random())
assert hist(l1) == [979,1011,991,1019,1004,1000,1050,971,1015,960]
assert hist(l1, N=5) == [1990, 2010, 2004, 2021, 1975]
```

```
### BEGIN HIDDEN TESTS
assert hist(l1, N = 8) == [1208,1344,1204,1248,1273,1247,1263,1213]
### END HIDDEN TESTS
```

In [ ]: