Test 1: Compsci 06

Owen Astrachan

February 23, 2011

Name: ____

NetID/Login: _____________

Honor code acknowledgment (signature) _______________________________________

	value	grade
Problem 1	27 pts.	
Problem 2	20 pts.	
Problem 3	16 pts.	
Problem 4	12 pts.	
TOTAL:	75 pts.	

This test has 10 pages, be sure your test has them all. Do NOT spend too much time on one question — remember that this class lasts 75 minutes.

In writing code you do not need to worry about specifying the proper `import statements`. Don't worry about getting function or method names exactly right. Assume that all libraries and packages we've discussed are imported in any code you write.

PROBLEM 1 : (***Stereo, Geno, and Proto (27 points)***)

Part A (22 points)

Each of the variables below has a *type* and a *value*. The type is one of: list, boolean, int, string, float. For example, consider the assignment to variable `x` below:

```
x = len([1,2,3,4,5])
```

The type and value are shown in the first row of the table below. Fill in the other type and value entries based on the variable/expression in the first column.

variable/expression	**type**	**value**
`x = len([1,2,3,4,5])`	int or integer	5
`a = 35/9`		
`b = "concatenate"[3:6]`		
`c = "one two three".split()`		
`d = 32 < 6`		
`e = sum(range(0,5))`		
`f = 4.0*2.5`		
`g = "apple" + "pie"`		
`h = 37 % 6`		
`i = "tortoise"[2]`		
`j = 4**3`		
`k = [2,3,5,7,11,13,15][2:4]`		

(continued)

Part B (2 points)

In the python code shown below an assignment is made to `b[0]` (the value `'tiny'` is assigned to `b[0]`) and as shown the value of `a[0]` changes as well. In a sentence or two explain why the value of the list `a` changes when the assignment is to `b[0]`.

```
>>> a = ['big', 'small', 'medium', 'tired']
>>> b = a
>>> a
['big', 'small', 'medium', 'tired']
>>> b
['big', 'small', 'medium', 'tired']
>>> b[0] = 'tiny'
>>> b
['tiny', 'small', 'medium', 'tired']
>>> a
['tiny', 'small', 'medium', 'tired']
>>>
```

Part C (3 points)

The average word length of the words in the list

```
   words = ["dog", "cat", "eagle", "vulture", "lion", "salamander"]
```

is 5.333. Complete function `avglen` below so that the expression `avglen(words)` evaluates to 5.333 by adding to the one missing line that uses a list comprehension. Complete the missing line by filling in the list comprehension so that the function returns the average length of the strings in parameter `words`. You should only modify the line that assigns a value to `num`.

```
   def avglen(words):
       """
       words is a list of strings, returns average length of strings
       """
       denom = float(len(words))
       if denom == 0.0:
           return denom


       num = sum([                       for w in words])

       return num/denom
```

PROBLEM 2 : (***Disfunctional, Malfunctioning, Perfunctory (20 points)***)

Part A (5 points)

Write the function `cone_volume` that returns the volume of a cone whose radius is given by the parameter `r` and whose height is given by the parameter `h`. The formula for the volume of a cone is

$$\frac{1}{3} \times \pi \times r^2 \times h$$

You can use either 3.1415 or `math.pi` for π.

call	**return value**
`cone_volume(2,3)`	12.55368
`cone_volume(1,1)`	1.04614
`cone_volume(3,2)`	18.83052

```
def cone_volume(r,h):
    """
    returns volume of cone whose radius is float r, height is float h
    """
```

Part B (5 points)

Complete the function `big_words` to return a list of the strings in parameter `words` whose length is greater than eight. The strings in the returned list should be in the same relative order as they appear in `words`.

For example, consider the lists below:

```
w1 = ["hello", "crabgrass", "teakettle", "darkness", "facelessness"]
w2 = ["goodbye", "fruitless", "small", "deadline", "ragamuffin", "east"]
```

call	**return value**
`big_words(w1)`	`["crabgrass", "teakettle", "facelessness"]`
`big_words(w2)`	`["fruitless", "ragamuffin"]`

```
def big_words(words):
    """
    return list of strings in words that are more than 8 characters long
    """
```

For parts C and D you *may* call the function `divisors` below. The function returns a list containing the proper divisors of parameter `num`, that is numbers that divide `num` evenly, but not `num` itself. For example:

call	**return value**
`divisors(24)`	`[1,2,3,4,6,8,12]`
`divisors(28)`	`[1,2,4,7,14]`
`divisiors(18)`	`[1,2,3,6,9]`
`divisiors(44)`	`[1,2,4,11,22]`

```
def divisors(num):
    return [x for x in range(1,num) if num % x == 0]
```

Part C (5 points)

An *abundant* number is a number whose proper divisors total more than the number itself. For example 24 is abundant because $1+2+3+4+6+8+12 = 36$ but 44 is not abundant because $1+2+4+11+22 = 40$. The number 28 is not abundant because its proper divisors add up to exactly $28 = 1+2+4+7+14$.

Write the boolean function `is_abundant` that returns `True` if its parameter `num` is abundant, and returns `False` otherwise. For example `is_abundant(44)` should evaluate to `False` while `is_abundant(24)` should evaluate to `True`.

```
def is_abundant(num):
    """
    returns True if num is abundant, False otherwise
    """
```

Part D (5 points)

Write the function `abundant_count` that returns the number of abundant numbers between, and including, parameters `low` and `high`. For example, the call `abundant_count(40,60)` should evaluate to 6 since the following numbers are the only abundant numbers between, and including, 40, and 60: $40, 42, 48, 54, 56, 60$. Assume `is_abundant` works correctly, you should call `is_abundant` in writing `abundant_count`.

```
def abundant_count(low,high):
    """
    return number of abundant numbers between and including int parameters low and high
    """
```

PROBLEM 3 : (***Alea iacta est (16 points)***)

The program shown on the next page prints the pattern below when it's run. You'll be asked some questions about the program and you'll be asked to make some changes to the program.

```
|############|
|~~~~~~~~~~~~|
|++++++++++++|
|------------|
|//\\//\\//\\|
|++++++++++++|
|~~~~~~~~~~~~|
|============|
|\\//\\//\\//|
|############|
```

The program prints ten lines when it is run, each line is generated by calling one of the different functions whose names starts with `stripe_`. The function is chosen at random and called in the function `random_stripe`.

Part A (4 points)

If the line below in `random_stripe`

```
    index = random.randint(0,len(stripe_funcs)-1)
```

is changed to

```
    index = random.randint(0,len(stripe_funcs))
```

Then *sometimes* when the program is run this error message is printed:

```
   func = stripe_funcs[index]
IndexError: list index out of range
```

Explain two things: why the error message is generated when the code is changed and why the error only happens *sometimes* and not every time the program is run.

```
'''
@author: ola
'''
import random

def stripe_wave_up():
    a = r"123456789012"
    s = r"/\/\/\/\/\/\"
    return s

def stripe_wave_down():
    a = r"123456789012"
    s = r"\/\/\/\/\/\/"
    return s

def stripe_sharp():
    a = r"123456789012"
    s = r"############"
    return s

def stripe_pattern():
    a = r"123456789012"
    s = r"^**^^**^^**^"
    return s

def stripe_less():
    a = r"123456789012"
    s = r"<><><><><><>"
    return s

def stripe_horizontal():
    a = r"123456789012"
    s = r"------------"
    return s

def stripe_plus():
    a = r"123456789012"
    s = r"++++++++++++"
    return s

def stripe_equal():
    a = r"123456789012"
    s = r"============"
    return s

def stripe_tilde():
    a = r"123456789012"
    s = r"~~~~~~~~~~~~"
    return s

def random_stripe():
    stripe_funcs = \
        [stripe_sharp,stripe_pattern,stripe_tilde,stripe_equal, \
         stripe_plus,stripe_wave_up,stripe_wave_down, \
         stripe_horizontal,stripe_less]
    index = random.randint(0,len(stripe_funcs)-1)
    func = stripe_funcs[index]
    a = "|"+ func() + "|"
    return a

def design(n):
    for i in range(0,n):
        s = random_stripe()
        print s

if __name__ == "__main__":
    design(10)
```

(another copy on the next page)

Feb 23, 11 8:44 **Designs.py** Page 1/1

```python
'''
@author: ola
'''
import random

def stripe_wave_up():
    a = r"123456789012"
    s = r"//\\//\\//\\"
    return s

def stripe_wave_down():
    a = r"123456789012"
    s = r"\\//\\//\\//"
    return s

def stripe_sharp():
    a = r"123456789012"
    s = r"############"
    return s

def stripe_pattern():
    a = r"123456789012"
    s = r"^**^^**^^**^"
    return s

def stripe_less():
    a = r"123456789012"
    s = r"<><><><><><>"
    return s

def stripe_horizontal():
    a = r"123456789012"
    s = r"------------"
    return s

def stripe_plus():
    a = r"123456789012"
    s = r"++++++++++++"
    return s

def stripe_equal():
    a = r"123456789012"
    s = r"============"
    return s

def stripe_tilde():
    a = r"123456789012"
    s = r"~~~~~~~~~~~~"
    return s

def random_stripe():
    stripe_funcs = \
        [stripe_sharp,stripe_pattern,stripe_tilde,stripe_equal, \
         stripe_plus,stripe_wave_up,stripe_wave_down, \
         stripe_horizontal,stripe_less]
    index = random.randint(0,len(stripe_funcs)-1)
    func = stripe_funcs[index]
    a = "|"+ func() + "|"
    return a

def design(n):
    for i in range(0,n):
        s = random_stripe()
        print s

if __name__ == "__main__":
    design(10)
```

Wednesday February 23, 2011 Designs.py

(continued)

Part B (4 points)

Suppose the function `stripe_star` below is added to the program. What changes are needed in the program so that this function is one of those called to generate a line of the design printed? Indicate in words or in the printed code on page 7.

```
def stripe_star()
    a = r"123456789012"
    s = r"************"
    return s
```

Part C (4 points)

Add or modify a small amount of code in the program so that two different patterns are printed side-by-side as shown in the new output below. Indicate the changes needed in the code so that the output below is generated. The two side-by-side patterns must be different. Write the changes below or make them on the code printout on page 7.

```
|\\//\\//\\//|   |//\\//\\//\\|
|//\\//\\//\\|   |++++++++++++|
|++++++++++++|   |++++++++++++|
|~~~~~~~~~~~~|   |^**^^**^^**^|
|############|   |^**^^**^^**^|
|++++++++++++|   |~~~~~~~~~~~~|
|<><><><><><>|   |############|
|------------|   |//\\//\\//\\|
|############|   |++++++++++++|
|~~~~~~~~~~~~|   |<><><><><><>|
```

Part D (4 points)

The output below shows a wider pattern where each line is 24 characters wide instead of 12 characters wide as in the original program (not including the two vertical bars on each side of the patterns). Indicate how to modify the original program so that a wider pattern is printed. You cannot modify any of the functions that begin with `stripe_`, you should modify only the functions `random_stripe` and/or `design`. Indicate your modifications below or on the code printout on page 8.

```
|~~~~~~~~~~~~~~~~~~~~~~~~|
|^**^^**^^**^^**^^**^^**^|
|========================|
|^**^^**^^**^^**^^**^^**^|
|++++++++++++++++++++++++|
|~~~~~~~~~~~~~~~~~~~~~~~~|
|<><><><><><><><><><><><>|
|========================|
|~~~~~~~~~~~~~~~~~~~~~~~~|
|++++++++++++++++++++++++|
```

PROBLEM 4 : (***In the Style of Apple and Blueberry Pie (12 points)***)

The mode of a list of numbers is the number that occurs the most often. The mode of the list `[12, 3, 3, 12, 4, 12]` is 12 because it occurs three times and no other element of the list occurs more than three times.

Part A: 8 points

Write a function `find_mode` that returns the mode of its parameter `nums`, a list of integer values. You can assume the mode is unique — only one element of the list occurs the maximal number of times.

```
def find_mode(nums):
    """
    return mode of nums, a list of integers
    """
```

Part B: 4 points

Does the code you wrote work correctly if `nums` is a list of strings, i.e., does it return the string that occurs more often than any other string in `nums` if it contains strings? If so, give a brief description why; if not, briefly describe why the code doesn't work.