Compsci 101: Test 1 Practice

September 25, 2013

Name: ____

NetID/Login: _____________

Community Standard Acknowledgment (signature) _______________________________________

	value	grade
Problem 1	12 pts.	
Problem 2	12 pts.	
Problem 3	12 pts.	

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

PROBLEM 1 : (***Big Ugly Gigantic Spiders***)

The *Reverse Name* problem is attached at the end of this test. The function `change` is intended to reverse the last and first names when in the format *last, first*, so that the call below returns `"bob jones"`.

```
  change("jones, bob")
```

Here is one student's solution that is all green. You will be asked two questions about this code.

```
def change(name):
    index = name.find(",")
    return name[index+2:]+" "+name[:index]
```

Part A (4 points)

Explain in words why the first slice used in the return uses `index+2` and why the second slice uses `index`.

Part B (4 points)

Pat looks at the code and says it will generate an error message if it's called with a string without commas (not allowed in the APT) so that `change("bob jones")` will generate an error. Ryan says no, it won't generate an error, runs the code, and the call `change("bob jones")` returns the string below (no error is generated).

```
  ob jones bob jone
```

Explain why the function generates this return value and does not result in an error.

PROBLEM 2 : (***Play that Funky Music***)

Part A

A number is *abundant* if it is greater than the sum of its proper divisors, that is its divisors other than itself. For example 12 is abundant because $1 + 2 + 4 + 6 = 13 > 12$. The first 10 abundant numbers are 12, 18, 20, 24, 30, 36, 40, 42, 48, 54.

Write a boolean function `is_abundant` to return True if its parameter is abundant and False otherwise.

call	**return value**
`is_abundant(4)`	`False`
`is_abundant(12)`	`True`
`is_abundant(24)`	`True`
`is_abundant(28)`	`False`

```
def is_abundant (num):
    """
    return True if int parameter num is abundant and
    returns False otherwise
    """
```

Part B

Write a function `abundant_count` that returns the number of abundant numbers between (and including) parameters first and last. You should call `is_abundant` and assume it works correctly.

call	**return value**
`abundant_count(1,11)`	0
`abundant_count(1,20)`	3
`abundant_count(20,30)`	3
`abundant_count(70,80)`	4

```
def abundant_count (start, end):
    """
    return how many numbers between start and end (inclusive)
    are abundant
    """
```

PROBLEM 3 : (***Genus, Order, Class, . . .***)

Data is stored in a file in the format shown below. Each line contains data for one animal giving the animal's name (string), gestation period in days (int), and estimated longevity in years (int). The information on a line is delimited by commas as shown, for example the file below shows information for eight animals in the format used in this problem.

```
bear,180,15
cat,52,10
dog,53,10
hamster,15,2
elephant,510,30
hippopotamus,220,30
human,253,65
lion,106,10
```

Write the function `getAgeList` that returns a list of those animals whose estimated longevity is between the values given by its two int parameters: `low` and `high`. The name of the file holding the data to be read and processed is given by parameter `filename`.

For example, if `"data.txt"` is the name of the sample data file above, then the call `getAgeList("data.txt",15,30)` should return the list `["bear","elephant","hippopotamus"]`, the call `getAgeList("data.txt",1,8)` should return the list `["hamster"]` and the call `getAgeList("data.txt",70,100)` should return the empty list `[]`

```
def getAgeList(filename, low, high):
    file = open(filename);




    file.close()
```

APT: Reverse Names

Problem Statement

Names are often stored in the format 'last, first' so they can be easily ordered by last name. However, often it is more convnenient to view them in the format 'first last' since that is how people are used to reading them. Write a function that converts a given string from the first format to the second.

Specification

```
filename: Reverse.py

def change (name):
    """
    return a String in the format 'first last' when
    given a String parameter in the format 'last, first'
    """
    # you write code here
```

Constraints

- the given string will always contain one and only one comma to separate first and last names
- both last and first names will contain at least one character

Examples

1. `word = "Astrachan, Owen"`

   `returns "Owen Astrachan`

2. `word = "l, f"`

   `returns f l`

3. `word = "Doe Jr., Bubba-John"`

   `returns Bubba-John Doe Jr.`