Compsci 101: Test 1

October 3, 2012

Name: _____

NetID/Login: _____

Community Standard Acknowledgment (signature) _____

	value	grade
Problem 1	32 pts.	
Problem 2	18 pts.	
Problem 3	16 pts.	
Problem 4	23 pts.	
TOTAL:	89 pts.	

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 : (*Smith [Corona—Wesson] (32 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([5,3,1])
```

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([5,3,1])`	int or integer	3
`a = 27/15`		
`h = 32 % 11`		
`e = sum(range(6))`		
`d = len("dog") < len("catsup")`		
`c = "1,234,567".split(",")`		
`i = "platform"[-2]`		
`g = "beat" + "nick"`		
`b = "snapdragon"[3:8]`		
`f = 0.23*10`		
`k = [8,7,6,7,6,5,4,3,2,1][5:7]`		
`j = 2**4`		

Part B (4 points)

The *Rohrer's Index* is an alternative to body mass index (BMI) as an anthropmetric statistic. It is calculated using the formula below where weight is in pounds and height is in inches. Write the function `rohrer` whose header is provided below the formula. For example, `rohrer(165,73)` is $\frac{165 \times 2768}{73^3} = 1.174$

$$\frac{\text{weight} \times 2768}{\text{height}^3}$$

```
def rohrer(weight,height):
    '''
    returns float value for rohrer's index given
    int parameters weight and height
    '''
```

(continued)

Part C (6 points)

Heron's Formula for the area of a triangle with three sides a, b, c can be calculated in two steps: the first calculates the *semi-perimeter* (s below) and the second calculates the area (A below):

$$s = \frac{a+b+c}{2}$$
$$A = \sqrt{(s-a) \times (s-b) \times (s-c) \times s}$$

For example, if the sides of a triangle are 3,4,5, then $s = (3+4+5)/2 = 6$ and the area is

$$\sqrt{(6-3)(6-4)(6-5)(6)} = \sqrt{3 \times 2 \times 1 \times 6} = \sqrt{36} = 6$$

Complete the function `triangle_area` that returns the area of a triangle with sides whose lengths are given by parameters `a`, `b`, and `c` (To compute a square root use the function `math.sqrt` or raise a number to the 0.5 power). For example, `triangle_area(3,4,5)` should evaluate to 6.0.

```
def triangle_area(a,b,c):
    '''
    return area of triangle given int parameters a,b,c, lengths of sides
    '''
```

PROBLEM 2 : (***Accumulated Wisdom (18 points)***)

Part A (5 points)

Write the function `divisor_sum` that returns the sum of the proper divisors of a number (i.e., all divisors less than number itself). For example:

call	**return value**	**reason**
`divisor_sum(12)`	16	$1 + 2 + 3 + 4 + 6 = 16$
`divisor_sum(284)`	220	$1 + 2 + 4 + 71 + 142 = 220$
`divisor_sum(28)`	28	$1 + 2 + 4 + 7 + 14 = 28$

```
def divisor_sum(n):
    '''
    return int, sum of proper divisor of n, n is an int > 0
    '''
```

Part B (5 points)

A pair of numbers is *amicable* if the sum of the proper divisors of one number is equal to the other, and vice versa. For example, the pair (220, 284) is amicable because the proper divisors of 220 are 1,2,4,5,10,11,20,22,44,55, and 110; the proper divisors of 284 are 1,2,4,71,142; and we have

$$1 + 2 + 4 + 71 + 142 = 220$$
$$1 + 2 + 4 + 5 + 10 + 11 + 20 + 22 + 44 + 55 + 110 = 284$$

call	**return value**
`is_amicable(220,284)`	True
`is_amicable(10,35)`	False
`is_amicable(1184,1210)`	True
`is_amicable(2214,2688)`	False

Write `is_amicable` below. **You should call** `divisor_sum`, assume it works as specified.

```
def is_amicable(x,y):
    '''
    returns True if and only if int parameters x,y are amicable
    '''
```

Part C (8 points)

Cities or towns are specified by a three-element list such as `["San Jose",37.2406,-121.7457]` where the first element is a string and the second two elements are float values specifying the latitude and longitude of the city.

Write the function `nearby` that has three parameters: the first parameter `city` is one three-element list specifying a city, the second parameter `clist` is a list of three-element lists specifying a list of cities, and the third parameter `apart` is a float representing a distance in miles. The function should return a list of strings: the names of the cities in `clist` that are no more than `apart` miles from `city`. For example, the call below would return the list `["Palo Alto", "Big Sur"]`, the only cities in `clist` within 150 miles of San Jose.

```
nearby(["San Jose", 37.2406, -121.7457],
       [ ["Palo Alto", 37.2833, -121.9179], ["Princeton", 40.3436, -74.694],
         ["Big Sur", 35.9348, -121.46894], ["Los Angeles",34.0522, -118.2428]],
       150)
```

In writing `nearby` you must call the function `distance` shown below that takes two coordinate pairs of latitudes and longitudes and correctly returns the distance in miles between the coordinates.

```
def distance(la1,lo1, la2,lo2):
    '''
    return distance in miles between (la1,lo1) and (la2,lo2), latitude and longitude pairs
    '''
    x = 69.1*(la1-la2)
    y = 53.0*(lo1-lo2)
    return math.sqrt(x*x + y*y)

def nearby(city, clist, apart):
```

PROBLEM 3 : (*Big Ugly Gigantic Spiders (16 points)*)

Part A (4 points)

The function `censored` below is supposed to return `"censored"` if the string parameter `headline` contains any of the words in parameter `bad`, a list of strings, and return `"clean"` if `headline` does not contain any words in `bad`.

The table illustrates what `censored` is *supposed* to return.

call	**return value**
`censored(["red","blue","green"], "red tide awesome")`	`"censored"`
`censored(["red","blue","green"], "cardinal rules")`	`"clean"`
`censored(["big","bad","trouble"], "bad news is trouble")`	`"censored"`
`censored(["big","bad","trouble"], "bigfoot badboy  troublesome")`	`"clean"`

The implementation below is **not** correct. However it passes three of the four tests/examples shown above. Explain which one test the implementation below fails and why it fails.

```
def censored(bad,headline):
    words = headline.split()
    for b in bad:
        if b in words:
            return "censored"
        else:
            return "clean"
```

Part B (4 points)

Write a correct version of `censored` below – this should work for all parameters, not just those shown in the table above. You can modify the code above, or copy/change it below.

The *Reverse Name* APT 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's one student's solution that is all green. You'll be asked two questions about this code.

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

Part C (4 points)

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

Part D (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 4 : (*Conjunction Junction (23 points)*)

Part A (15 points)

The spelling idiom of *i before e except after c* can be checked by the Python function `spellwell` below (comments label each line of code).

```
def spellwell(word):
    iei = word.find("ie")                    #1
    eii = word.find("ei")                    #2
    if iei == -1 and eii == -1:              #3
        return True                          #4
    if iei != 0 and word[iei-1] == 'c':      #5
        return False                         #6
    if eii != 0 and word[eii-1] == 'c':      #7
        return True                          #8
    return False                             #9
```

For example, consider this run and the corresponding output:

```
words = ["niece","either","receive", "recieve","neighbor","piece","peice","dog"]

for x in words:
    print x, spellwell(x)

niece True
either False
receive True
recieve False
neighbor False
piece True
peice False
dog True
```

- One of the words shown above is labeled in the output as `True` by the return statement on line 4. Which word and why?

- Which line returns `True` for *receive* and why?

- Which line returns `False` for *recieve* and why?

- Why is the value `False` returned for *either*?

- No words in the English language start with "ie", so "ei" at the beginning of a word should not be considered a violation of this spelling rule Modify the code to reject (return False) any word that starts with "ie" and accept any word that starts with "ei". (Don't worry about multiple occurrences of "ei" or "ie" in a word).

Part B (4 points)

The *Alliteration* APT is attached to the end of this test. Write a very simple implementation of the function `countall` that will almost certainly get at least some green, but definitely not get all green. Explain why you think you'll get at least one green with the solution you write. **Your solution should be the simplest code you can write that will likely get at least one green.**

```
def countall(words):
```

Part C (4 points)

What do you think the hardest part of getting all green will be *for you* in writing code to solve this APT. Write at most four sentences in which you refer to your own experiences in writing code to solve APTs and to this particular APT.

(use back if needed)

(this is blank)