Additional Practice Problems For Topic 1: Python Basics  
CISC 121, Fall 2014

General Note: You don’t have to do any type-checking of parameter types in these exercises. Assume that the functions will be called with parameters of the required types.

Practice Problems Without Strings or Lists:
This set starts with very simple problems and increases in complexity.

1. Write a simple function that computes the sum of 15, 7.1 and 2.8 and prints the answer. Don’t add up the numbers yourself and tell Python to print the sum; make Python add up the numbers.

2. Write a function that takes three parameters, assumes they are all numbers, and prints the sum of the numbers.

3. Write a function that takes three parameters, assumes they are all numbers, and returns the sum of the numbers. Make very sure you understand the difference between this problem and the previous one. See the below to illustrate this difference:

   Examples using my solutions for problems 2&3:
   >>> problem2(1,2,3)
   6
   >>> problem3(1,2,3)
   6
   >>> x=problem2(1,2,3)
   6
   >>> print x
   None

   Technically, a function which does not return a value returns a special value called None which conveys no information at all.
   >>> x=problem3(1,2,3)
   >>> print x
   6

   problem3 returns an actual numerical value.
   >>> problem2(1,2,3)+4
   6

   Traceback (most recent call last):
     File "<pyshell#27>" , line 1, in <module>
       problem2(1,2,3)+4
   TypeError: unsupported operand type(s) for +: 'NoneType' and 'int'

   Here I’m trying to add the special value None to 4. Since None is not a number, you can’t do arithmetic with it.
   >>> problem3(1,2,3)+4
   10

   The result of problem3 is the number 6, which can easily be added to 4
4. Use your solution to Problem 3 to write a function which takes 6 numbers as parameters and returns their sum. Do this by calling your problem 3 solution once to add up the first 3 parameters and again to add up the last 3, then return the sum of those two results.

5. Write a function called `triangle` which takes three numbers as parameters and prints a string describing the kind of triangle three lines with these lengths could make. If any of the lengths are negative or zero, or if one of the lengths is equal to or greater than the sum of the other two, return “not a triangle”. Otherwise, print either “right triangle”, “obtuse triangle”, or “acute triangle”.

Quick geometry review: If a triangle has sides of length a, b and c, where a is the longest length, the triangle is a right triangle if $a^2 = b^2 + c^2$. It is an obtuse triangle (meaning it has an angle greater than 90 degrees) if $a^2 > b^2 + c^2$. Otherwise, it is an acute triangle (meaning all of its angles are less than 90 degrees).

Examples using my `triangle` function:
>>> triangle(1,3,5)
'not a triangle'
>>> triangle(4,11,7)
'not a triangle'
>>> triangle(6,1,4)
'not a triangle'
>>> triangle(3,4,5)
'right triangle'
>>> triangle(5,13,12)
'right triangle'
>>> triangle(3,4,6)
'obtuse triangle'
>>> triangle(5,13,10)
'obtuse triangle'
>>> triangle(6,8,9)
'acute triangle'
>>> triangle(6,13,12)
'acute triangle'
>>> triangle(3,4,4.5)
'acute triangle'

6. Write a function that takes a number n as a parameter and prints all of the odd numbers between 1 and n, including n if it is an odd integer. If the parameter is less than 1, write an error message instead. Your function should work even if n is a floating-point number; that means you can't use a for loop with a range. Use a while loop.
**Practice Problems Using Strings:**

7. Write a function with no parameters that asks the user to enter their name and returns their first initial. Don't worry about whether the user actually enters something that looks like a name; just return the first character of their name. If the user just types return to the prompt, their name will be an empty string. In this case, make sure that your function returns an empty string ("") instead of aborting with an error message.

   Example:
   ```python
   >>> problem7()
   What is your name? Margaret
   'M'
   >>> problem7()
   What is your name? (user didn't type anything before hitting Enter)
   ''
   ```

8. Write a function that takes a string as a parameter. Assume that the string is in the format of a name in the format “Jane Smith” -- i.e. containing exactly one space and that space is not at the beginning or end. Write a message stating the person's first and last name.

   Example for Problem 8:
   ```python
   >>> problem8('Harry Potter')
   Your first name is Harry
   Your last name is Potter
   ```

9. Write a function called `nameCheck` that performs the check missing from Problem 8: take a string as a parameter and return True if it is in the required format and False otherwise. Just check for spaces; there should be exactly one and it should not be at the beginning or end of the list. It's OK if the name is something silly like “a57,##$ 8!!!”. Just check the spaces in the string.

   Example:
   ```python
   >>> nameCheck('Harry Potter')
   True
   >>> nameCheck(' Harry Potter')
   False
   >>> nameCheck('Harry Potter')
   False
   >>> nameCheck('Harry Potter ')
   False
   >>> nameCheck('Harry Potter')
   False
   >>> nameCheck('Harry Potter')
   False
   ```

   Once you've written this function you can improve your solution to problem 8 by making it call `nameCheck` and writing an error message if the name is not in the right format.

   You can solve this problem with nothing but a loop and string indexing and character comparisons. There are also some built-in string functions and methods that can save you a few steps. I recommend
that you first write a solution without any built-ins; you need to know how to go back to the basics when you've got a problem that a build-in won't help you solve. But afterwards, take a look and see what you can find to make the task simpler; it's also helpful to know how to find shortcuts when they exist.

To look at the basic built-in operators, functions and methods for strings, go to the “Help” menu in Python and choose “Python Docs”, then choose “Library Reference”. In the list you will see, you may be tempted to go to item 7 ("String Services"). That's actually a list of more advanced stuff that might be helpful in later programming but isn't what you need now. Instead, go to 5.6, which is “Sequence Types”. This section starts with operators, functions and methods that can be used with both strings and lists and then sub-section 5.6.1. lists more useful stuff that is specific to strings. Familiarize yourself with what's there and see if you can spot anything that could make nameCheck a bit simpler.

Additional Problems Using Lists:

10. Write a function called squares that takes a list of numbers as a parameter and returns a list containing the squares of each of the numbers. Do not change the original list.

Examples:
   >>> numbers = [3,5,2,-7]
   >>> newList = squares(numbers)
   >>> newList
   [9, 25, 4, 49]
   >>> numbers
   [3, 5, 2, -7]
   >>> squares([1.5,2.4])
   [2.25, 5.76]
   >>> squares([])
   []

11. Write a function called squares2 that takes a list of numbers as a parameter and does not return a value, but changes the list so that it now contains the squares of the original numbers. Make very sure that you understand the difference between how to write and use squares and squares2! Ask for help if you don't this; it's something you will trip up against frequently in this course and in programming in general.

Example:
   >>> numbers = [3,5,2,-7]
   >>> squares2(numbers)
   >>> numbers
   [9, 25, 4, 49]
12. Create a function called `initials` that takes a list of strings as a parameter and returns a string consisting of the first character of each string in the list. Your function should not change its parameter.

Examples:
```python
>>> initials(['Peter', 'Susan', 'Edmond', 'Lucy'])
'PSEL'
>>> fruit = ['apple', 'grape', 'banana', 'orange', 'kiwi']
>>> initials(fruit)
'agbok'
>>> fruit
['apple', 'grape', 'banana', 'orange', 'kiwi']
>>> initials([])
''
```

13. Create a function called `extract` that takes two parameters:
   - a list of values (doesn't matter what kind of value)
   - a list of indexes (integers which are all legal indexes into the list of values)

Return a list consisting of the elements of the list of values which have the indexes given in the second parameter. To start with, don't worry about what happens if an index is out of bounds.

Examples:
```python
>>> myList = [42, 81, 3.14159, 'Saturday', [5], 'theEnd']
>>> extract(myList, [1, 3, 4])
[81, 'Saturday', [5]]
>>> extract(myList, [5])
[42]
>>> extract(myList, [])
[]
>>> myList
[42, 81, 3.14159, 'Saturday', [5], 'theEnd']
```

Once you've got that working, fix your function so that if one of the indexes in the second parameter is not a legal index into the first parameter you print an error message and return an empty list.

Examples:
```python
>>> myList = ['Elementary', 'my', 'dear', 'Watson']
>>> extract(myList, [0, 2, 4])
error: index 4 is out of bounds
[]
```
14. Write a function called `mash` that “mashes” together a list of lists into a single list, containing all of the elements in the list of lists. The elements of the lists can be of any type.

Examples:
```python
given_sentence=[[42,'bottles'],['of','beer','on','the'],['wall']]
mash(given_sentence)
[42, 'bottles', 'of', 'beer', 'on', 'the', 'wall']
given_sentence_mashed=[]
mash(given_sentence_mashed)
[]
mash([[3.14],[],['pi',[4,5,6]])
[3.14, 'pi', [4, 5, 6]]
```

Notice in the last example above that if the lists are nested at a depth greater than two you don't have to flatten them out completely.