ITI 1120

Lab # 5

lists , more for loops, "pointer/ reference" variables, aliasing

Starting Lab 5

- Open a browser and log into Blackboard Learn
- On the left hand side under Labs tab, find lab5 material contained in lab5-students.zip file
- Download that file to the Desktop and unzip it.

Before starting, always make sure you are running Python 3

This slide is applicable to all labs, exercises, assignments … etc

ALWAYS MAKE SURE FIRST that you are running Python 3.4 (3.5 is fine too)

That is, when you click on IDLE (or start python any other way) look at the first line that the Python shell displays. It should say Python 3.4 or 3.5 (and then some extra digits)

If you do not know how to do this, read the material provided with Lab 1. It explains it step by step

Do all the exercises labeled as
Task in your head i.e. on a paper

Later on if you wish, you can type them into a computer (or copy/paste from the solutions once I poste them)

Task 1: What does this print?

```
s1="this is a string"
s2="this is a string"
s3=s1
print(s1==s3)
s1="aha"
print(s3==s1)
print(s3==s2)
```

Task 2: What does this print?

```
s1="this is a string"
print("A variable s1, referes to an object of what type?")
print( type(s1) )

s4=["this is a string"]
print("A variable s4, referes to an object of what type?")
print( type(s4) )

print("A variable s4[0], referes to an object of what type?")
print( type(s4[0]) )
```

Task 3: What does this print?

```
s5=[2, 4, "oaxaca", True]
s6=[2, 4, "oaxaca", True]
print(s5==s6)
print(s5 is s6)
s7=s6
print(s6 is s7)
s7[3]=False
print(s5)
print(s6)
print(s7)
```

Study the following four elementary functions:

Returns sum of positive odd integers smaller than **n**

```
def sum_odd(n):
    odd_sum = 0
    for num in range(n):
        if num % 2 == 1:
            odd_sum = odd_sum + num
    return odd_sum
```

Returns a product of positive odd integers smaller than **n**

```
def product_odd(n):
      prod = 1
      for num in range(n):
            if num % 2 == 1:
                  prod = prod * num
      return prod
```

Sum of odd numbers in a given list **a** A solution with loop over elements

```
def sum_list_v1(a):
    ''' (list)->num '''
    list_sum = 0
    for item in a:
        if item % 2 == 1:
            list_sum = list_sum + item
    return list_sum
```

Sum of odd numbers in a given list **a** A solution with a loop over indices

```
def sum_list_v2(a):
    ''' (list)->num '''
    list_sum = 0
    i = 0
    for i in range(len(a)):
        if a[i] % 2 == 1:
            list_sum = list_sum + a[i]
    return list_sum
```

Task 4 and Programming Exercise 1

After studying the previous four functions

1. Open the file called **four_functions.py** Copy/paste, one by one, Example 1 to 4 into Python visualizer. Run through each example and understand how the solutions work and how the variables change in the loops. As always, you can find python visualizer here (make sure you choose Python 3)
   http://www.pythontutor.com/visualize.html#mode=edit

2. **Programming exercise:** Write a function called **ah(l,x,y)** that given a list **l**, and integers **x** and **y** such that **x <=y**, returns two numbers. The first is the number of elements of l that are between x and y (including x and y). The second number is the minimum element of l that is between x and y (including x and y). Example test:

```
>>> t=[5, 1, -2.5, 10, 13, 8]
>>> ah(t, 2,11)
(3, 5)
```

Recall that you can return two numbers referred by variables (a,b) by just doing return (a,b)

Task 5

How many starts does the following program print?

```
for i in range(-2,14):
    print("*")
    print("**")
```

a) 0 **b) 15** **c) 45** **d) 48** **e) 68**

Intermission: print function revisited

Built-in function print, when its completes printing, enters a new line. For example:

```
print("This is")
print("Lab 5”)
```

Prints:

```
This is
Lab 5
```

As mentioned in the last lab, this default behavior of the print function can be changed by specifying what we want print function to end with. For example:

```
print("This is", end='***')
print("Lab 5")
print("This is", end='')
print("Lab 5", end='\n')
print("This is", end=' ')
print("Lab 5")
```

Would Print =>

```
This is***Lab 5
This isLab 5
This is Lab 5
```

In other words, unless specified otherwise, the default end for the print function is `end='\n'`

More examples of use of print function

Function `print` prints, by default,
a newline character after printing its arguments

```
>>> pets = ['boa', 'cat', 'dog']
>>> for pet in pets:
        print(pet)


boa
cat
dog
>>> for pet in pets:
        print(pet, end=', ')


boa, cat, dog,
>>> for pet in pets:
        print(pet, end='!!! ')


boa!!! cat!!! dog!!!
>>>
```

The **`end`** argument allows for customized end characters

Task 6

1. What does this print?

```
fruits = ["apple", "orange", "banana", "cherry"]

for afruit in fruits:      # by item
    print(afruit, end=" ")
```

2. What does this print?

```
fruits = ["apple", "orange", "banana", "cherry"]

for position in range(len(fruits)):      # by index
    print(fruits[position], end=" ")
```

3. What does this print?

```
for name in ["Joe", "Amy", "Brad", "Angelina", "Zuki", "Thandi", "Paris"]:
    print("Hi ", name, "  Please come to my party on Saturday!")
```

Task 7

What does the following program print?

```
for i in range(5,0,-1):
    num=1
    for j in range (1,i+1):
        print(num, end="| ")
        num=num*2
    print()
```

Task 8

A programmer who wrote a function below thinks that the function tests if a given positive integer n>=2 is a prime. However there is a logical mistake.

1. Where is the mistake?
2. For what number does the function return a wrong answer? Can you think of the smallest such number?
3. Fix the mistake.

```
def is_prime(n):
    '''(number)->bool
    Returns True if a given positive number n>=2  is prime and false otherwise
    Precondition n>=2
    '''
    for divisor in range(2, n//2+1):
        if n % divisor == 0:
            return False
        else:
            return True
    return True
```

Programming Exercise 2: Experiment with Perfect Numbers

- A positive integer is called a perfect number if it is equal to the sum of **all** of its positive divisors, **excluding itself**. For example, 6 is perfect number since 6=1+2+3. The next is 28=1+2+4+7+12. There are four perfect numbers less than 10,000. Write a program that prints all these four numbers.
- Your program should have a function called is_perfect that takes as input a positive integers and returns True if it is perfect and False otherwise.

- Once you are done. Modify your program so that it looks for all perfect numbers smaller than 35 million. What do you notice? Assuming that you computer can do a billion instructions in a sec, can you figure out how long, roughly, will it take your computer to find 5th perfect number (it is 33,550,336). Is the answer roughly: couple of minutes, couple of hours, couple of days, … weeks, months, years ?
- What if you wanted to wait until it prints 6th perfect number, which is 8,589,869,056?

Programming Exercise 3a: Arithmetic progression

Recall that a sequence of numbers forms an arithmetic progression if the difference between every pair of consecutive numbers is the same. For example: -5, -1, 3, 7, 11 forms an arithmetic progression since the difference between every pair of consecutive numbers is 4. On the contrary 5, 10, 15, 24, 29 is not an arithmetic progression since the difference between some consecutive pairs is 5 and some 4. A sequence that has exactly one number is considered arithmetic, too.

- Write a function called arithmetic that takes as input a list of numbers and returns True if the numbers of the list form arithmetic progression. And False otherwise

```
Testing:

>>> arithmetic( [-5, -1, 3, 7, 11] )
True
>>> arithmetic([0, -1, 3, 7, 11])
False
>>> a = [5, 10, 15, 24, 29]
>>> arithmetic(a)
False
>>> arithmetic(a[:3])
True
```

Programming Exercise 3b: and now … is it sorted?

Now modify your method arithmetic slightly so that instead it tests if the numbers in the give lists are ordered for smallest to largest. Call the new function is_sorted

```
Testing:

>>> is_sorted([1, 1, 1, 7, 7])
True
>>> is_sorted([-10, -1, 3, 7, 100])
True
>>> is_sorted([0, 3, 1, 7, 11])
False
>>> a = [5, -10, 15, 24, 29]
>>> is_sorted(a)
False
>>> is_sorted(a[1:4])
True
```

Task 6: Flow of execution

The the following two multiple choice questions:

http://interactivepython.org/courselib/static/thinkcspy/Functions/FlowofExecutionSummary.html