Pre-lab exercises - this section is required, but nothing is submitted for grading. BEFORE coming to lab, read Chapters 3-6 of the textbook.

Functions

Open IDLE, then open a new Python script window. In the script window, type in:

```
print max(1, 19, -2, 7)    # max is a built-in function that returns the maximum value
print len("Hello")         # len is a built-in function that returns length of a string
```

Max and len are examples of functions that are built-in (i.e., they are already defined for programmers to use). In this lab you will learn to write a function.

This is a Pair Programming lab. Watch the video about pair programming that is provided in the assignment. Then find a peer in your lab section to work with for this lab. Include BOTH student names at the start of your lab2.py file. Only ONE of the students in the pair should submit the file lab2.py in Sakai.

Each function you write must contain a docstring within the function must use good, meaningful variable names.

Converting Python script to a function:

What if we wanted to calculate a student named Herbert's grade. Herbert had a lab score of 47.2 (out of 50) and combined test scores of 122.4 (out of 150). To calculate Herbert's grade, you'd add the lab score and the test scores and divide by the total number of points, or something like this: (47.2 + 122.4) / 200.0 Now suppose we want to calculate Bertha's grade. Bertha got a 42.6 on her labs and a total of 142.5 on her tests, so her grade would be (42.6 + 142.5) / 200.0

Now we've got Herbert and Bertha's grade. Unfortunately, we have 200 more students! Wouldn't it be nice if we had a way of calculating a generic grade by programming something like

```
grade = (labscore + testscore) / 200.0
```

We can by using functions with parameters and variables. A function is a set of instructions (code) identified with a name. The instructions inside a function are only executed when you call the name of the function. In this case, let's call the function CalculateGrade. We'll define it in Python as follows:

```
def CalculateGrade(labscore, testscore):
    grade = (labscore + testscore) / 200.0
    return (grade)
```

Note that in Python, indenting matters! Python knows where the end of the function CalculateGrade is by when you stop indenting. When you call CalculateGrade, only those instructions that are indented will be executed.

That's how you define the function CalculateGrade in Python. The instructions (code) inside this function do not execute until the function is "called" from somewhere else in your program.

Function parameters are places in memory that hold nothing until a value is put in them. To call the function CalculateGrade with Herbert's scores, you type:

CalculateGrade(47.2, 122.4)

The "argument" 47.2 is passed into the "parameter" labscore, and the argument 122.4 is passed into the parameter testscore. To calculate Bertha's score, you'd call the function CalculateGrade as:

CalculateGrade(42.6, 142.5)

In this case, 42.6 is passed into the parameter labscore and 142.5 is passed into the parameter testscore.

Recall: variables name memory locations that are "undefined" (i.e., they contain nothing) until the program assigns a value into them. grade is a variable in the function. grade has no value in it until the program assigns a value, in this case in the line grade = (labscore + testscore) / 200.0.

Once grade has a value, that value can be "returned" by the function, passed back to function call.

In lab2.py, program the function CalculateGrade as follows:

```
def CalculateGrade(labscore, testscore):
    """ Simple example to demonstrate a function.
    Parameters:
        labscore is the sum of a student's lab scores
        testscore is the sum of a student's two test scores
    Variables:
        grade is the student's returned course grade
    """
    grade = (labscore + testscore) / 200.0
    return(grade)

CalculateGrade(47.2, 122.4)
CalculateGrade(42.6, 142.5)
```

Save the script file and run it (Run->Run Module). Since there are no input or print statements, nothing happens. (If you typed something incorrectly, go back and correct what you typed in.)

Even though there is no output, grades are being calculated by CalculateGrade. Your program can now use that returned value for later functions, or whatever else you might want to use it for. Now we want to test to see if the function CalculateGrade is "semantically" correct, i.e., it's doing what we want it to do.

Testing a Function:

First, download a copy of the attachment file cisc106.py and save the file in the SAME directory (or folder) in which you are keeping your lab2.py file.

Import this file into your program by typing the following line at the top of the lab2.py just after your header comments:

```
from cisc106 import *
```

(Note that the file cisc106.py may change during the semester. When it does, I shall let you know and you will have to download it again and save it in the directory where your .py files are located.)

Now you can test to make sure CalculateGrade is doing what we want. We'll test CalculateGrade using a special function assertEqual (which has already been written and saved in cisc106.py, which you imported into this file).

In your lab2.py file, change the following lines:

```
CalculateGrade(47.2, 122.4)
CalculateGrade(42.6, 142.5)
```

to

```
assertEqual(CalculateGrade(47.2, 122.4), 0.848)
assertEqual(CalculateGrade(42.6, 142.5), 0.9255)
```

Your script file should now look like:

```
# <Student name(s)>

# <Student lab section>

from  cisc106  import  *

def CalculateGrade(labscore, testscore):
    """ Simple example to demonstrate a function.
    Parameters:
        labscore is the sum of a student's lab scores
        testscore is the sum of a student's two test scores
    Variables:
        grade is the student's returned course grade
    """

    grade = (labscore + testscore) / 200.0
    return(grade)

assertEqual(CalculateGrade(47.2, 122.4), 0.848)
assertEqual(CalculateGrade(42.6, 142.5), 0.9255)
```

What is assertEqual doing? It is testing to see if the function CalculateGrade returns the answer we think it should. When we run the function CalculateGrade with 47.2 as the lab score and 122.4 as the test score, the grade calculated should be 0.848 (We made up this test by hand.)

If we run CalculateGrade with 42.6 and 142.5 as the lab and test score, respectively, the returned grade should be .9255. If it isn't, the function CalculateGrade is not working properly.

THIS REPRESENTS ONE OF THE MOST IMPORTANT CONCEPTS OF COMPUTER PROGRAMMING - YOU CANNOT WRITE A PROGRAM WITHOUT KNOWING WHAT THE CORRECT ANSWERS ARE FOR AT LEAST SOME SET OF TEST INPUTS. YOU SHOULD KNOW THESE CORRECT ANSWERS BEFORE YOU EVEN RUN THE PROGRAM THE FIRST TIME!

Save the script lab2.py and run it. Your output should be something like:

[line 16] assertEqual(CalculateGrade(47.2, 122.4), 0.848) SUCCESS

[line 17] assertEqual(CalculateGrade(42.6, 142.5), 0.9255) SUCCESS

This output means that both assertEqual lines were successful. CalculateGrade(47.2,122.4) returned 0.848. CalculateGrade(42.6,142.5) returned 0.9255.

Problems to submit (5 points each):

Problem 1:
Add 3 other assertEqual test conditions to test the CalculateGrade function. In other words, pick a lab score and a test score, then calculate (with a calculator if you like) what you think CalculateGrade should be with those lab scores and test scores. Now add an assertEqual(CalculateGrade(…) for each of your 3 test conditions. (Important: For all numbers in this lab, use floating point numbers, not integers. For example, use 42.0 not 42) Run the file. If you got failures, figure out why and correct your errors.

Problem 2:
Write a function that calculates the area of a right triangle. This function should take two input parameters: the triangle height and the triangle width, return the area of the triangle. The function header should look like this:

```
def CalculateTriangleArea(height, base):
    """ Compute the area of a triangle given its height and base lengths
    Parameters:
        height is the length of the triangle's height
        base is the length of the triangle's base
    Variables:
        <comment here every variable that you use>
    """
```

Test CalculateTriangleArea using the following 3 tests, and also make up 3 of your own tests:

```
assertEqual (CalculateTriangleArea (32.0, 24.5), 392.0)
assertEqual (CalculateTriangleArea (1.0, 1.0), 0.5)
assertEqual (CalculateTriangleArea (.001, 3000.32),1.50016)
```

Problem 3:
Write a function called CalculateBoxVolume. It takes 3 parameters: length, width, height. CalculateBoxVolume should calculate the volume of a box. Write at least three tests to make sure CalculateBoxVolume is working correctly.

Problem 4
Given the following tests, deduce what the function MakeStringSandwich does and write the function:

```
assertEqual (MakeStringSandwich ("bbb", "a"), "bbbabbb")
```

assertEqual (MakeStringSandwich ("12","3"), "12312")
assertEqual (MakeStringSandwich ("Z",""), "ZZ")

Include at least 3 more assertEqual tests to help verify your function is working correctly.

Problem 5:
An Internet service provider charges a base rate per megabyte (MB) transferred depending on market conditions. Transfers up to 50 MB are charged the base rate only. Transfers between 50 and 400 MB are charged the base, plus an additional 33% of the base, plus an additional $0.05/MB for each MB over 50MB. Transfers between 400 MB and 1000 MB (i.e., 1 GB) are charged the base, plus an additional 44% of the base, plus an additional $0.08/MB for each MB over 400MB. Transfers above 1000 MB are charged simply twice the base.

Program the function, BillAmount, which takes an amount of data transferred (megabytes) and a base rate (dollars) and computes the total charge. Write 4 assertEqual tests for 45, 120, 600 and 2000 MB, and a base rate of $50.00. Write 3 other assertEqual tests for values of your choosing.

Problem 6:
A manufacturing company open from 7am to 7pm measured the productivity of its workers and found that during the morning hours starting at 7,8,9,10, they could produce 25 pieces/hour/worker; the hours starting at 11,12,13,14 (i.e., 13 is 1-1:59pm, 14 is 2-2:59pm, etc.) they could produce 35 pieces/hour/worker; and starting at 15,16,17,18, they could produce 30 pieces/hour/worker. The factory is closed from 7pm to 7am.

Develop a function, PiecesProduced, which takes the start of an hour of the day in the range [0 – 23], along with the number of workers, and computes the total number of pieces produced during that hour. Write assertEqual tests for hours 8am, 2pm, 6pm and 9pm, and 50 workers. Write 3 more assertEqual tests for values of your choice.

Problem 7:
Write a function, ReverseThreeDigitNumber, which takes an integer value in the range [0-999] and returns a value with the ‘hundreds’ and ‘ones’ digits reversed. As tests, ReverseThreeDigitNumber(826) should return 628, ReverseThreeDigitNumber(930) should return 39, ReverseThreeDigitNumber(20) should return 20, and ReverseThreeDigitNumber(7) should return 700. Write 3 additional assertEqual tests for your function. [Important: The argument to ReverseThreeDigitNumber should not begin with a zero. ReverseThreeDigitNumber(025) will not return 520. Remember what you learned in Lab 1.]

Problem 8 – Extra Credit (Note – if you find this problem reasonable to solve correctly and make up useful assertEqual tests, you have the logical ability needed to be a computer scientist):

You are tasked with writing a function, MortgageApproval, to decide on a mortgage loan (“yes”, “no” or “maybe”). You are given 6 pieces of information about a mortgage applicant: the loan amount s/he is applying for, the current salary, the current cash in accounts, the estimated non-cash assets, a numerical credit score, and a last name. You are given the following business rules to guide the process:

1. No mortgage will be approved if the applicant has less than 10% of the loan amount as cash in accounts (ie, the mortgage decision is “no”).
2. No mortgage will be approved if the applicant has less than a credit score of 590.
3. For applicants with credit scores in the range [590 – 700] (“[ ]” means the values 590 and 700 are included), current cash in accounts must be greater than or equal to 20% of the loan amount.
4. All applicants must have a current salary greater than one-third of the balance of the loan, which is considered to be the loan amount minus the cash in accounts.
5. Any applicant with the last name Smith must have at least $600,000 in non-cash assets or the loan is declined.
6. Any applicant that made it past the rules 1-5 and has more cash in accounts than the loan amount is automatically approved (the mortgage decision is “yes”, otherwise the applicant must come to the bank for a personal interview (ie, the mortgage decision is “maybe”.)

Write a mininum of 9 assertEqual tests that test these rules. Your tests should follow the business rules in the *order* they are given.

What to turn in:

Using Sakai, electronically submit your lab2.py script file which contains:

- CalculateGrade and tests
- CalculateTriangleArea and tests
- CalculateBoxVolume and tests
- MakeStringSandwich and tests
- BillAmount and tests
- PiecesApproved and tests
- ReverseThreeDigitNumber and tests
- (extra credit) MortgageApproval and tests