CMSC 201 Fall 2015

Lab 03 – A Simple Program

Assignment: Lab 03 – A Simple Program
Due Date: During discussion, September 14th through September 17th
Value: 1% of final grade

This week's lab is in-person with your TA. They'll briefly cover some of the material learned in class, including variables, expressions, and user input. (Having concepts explained in a new and different way can often lead to a better understanding, so please pay attention.)

Part 1: Review – Variables

A variable is how we store information in our Python programs. In simple terms, you can think of a variable as a "box" that you can put stuff in. The "stuff" we place in the box is then the value of the variable.

For example, we can create a variable called **`classSize`**, and set its value to be **`20`**. To express this in Python code, we would write

```
classSize = 20
```

We've discussed many different types of variables in class, but today we'll focus on three of them: **strings**, **ints**, and **floats**.

- A **string** is a collection (or "string") of characters. You already used a string in Lab 1, when you wrote
  ```
  print("Hello world!  I am Jane Doe")
  ```
  Strings can be nearly anything, and are specified through the use of quotation marks. The following examples are all valid strings:
  - **`"3.14"`**
  - **`"True"`**
  - **`"201"`**
  - **`"Goodbye cruel world!"`**
  - **`"This is the song that doesn't end..."`**

- The word **int** stands for "integer," which is a number that can be written without a decimal or a fractional part. The numbers **`-8`**, **`16`**, **`50`**, **`-8674`**, and **`1000000000`** are all examples of an integer.

- A **float** is also a type of number – a "floating point number." In contrast with integers, floats are numbers that contain a decimal point. This includes "whole" numbers, where the value following the decimal is zero. The numbers **`-7.2`**, **`6.0`**, **`5.007`**, **`-84.5`**, and **`1.0000000005`** are all examples of a float.

Using variables in Python is easy! There are just two important rules we have to remember:

1. Use meaningful variable names! For example, **`numberOfBooks`** is a much better variable name than **`NOB`** or **`numb`** or **`x`**.
2. Before we can use a variable, it must be *initialized*. In other words, we have to put a value into the "box" before we can start using the variable. We do this using the *assignment operator*, or equal sign (=). For example:

```
booksPerShelf = 50
numberOfShelves = 22
sizeOfLibrary = booksPerShelf * numberOfShelves
```

   We had to *initialize* the value of the variables **`booksPerShelf`** and **`numberOfShelves`** before we could use them to calculate the size of the library.

Here are some more examples of variables:

```
address = "1000 Hilltop Circle"
biggestDinosaur = "Argentinosaurus"
ageOfEarth = 4543000000
minimumWageMD = 8.25
```

Part 2: Review – Expressions

An expression is code that calculates or produces new data and data values. Expressions are what allow us to create interesting Python programs. The word "expression" is really just a fancy name for something that can be evaluated to a single value.

One important thing to remember is that expressions must always be on the right hand side of the assignment operator!

Right now, we will use expressions in Python mostly for mathematical equations. For example, if we want to find the addition of two numbers in Python, we could write code like this:

```
sum1 = 15 + 42
```

Our expression is "`15 + 42`" which evaluates to 57. This result is stored in the variable `sum1` – in other words, the value inside the box labeled "`sum1`" is now 57.

We could have achieved the same thing with the following code:

```
num1 = 15
num2 = 42
sum1 = num1 + num2
```

Even though we used the variables `num1` and `num2` to hold the values 15 and 42, the expression still evaluates to 57, and that value is still stored in `sum1`.

Here are a few more examples of expressions:

```
numStudents = 500
totalPrice = numCookies * priceOfCookie
numHours = numDays * 24
triArea = (1/2) * triBase * triHeight
```

Part 3: Review – User Input (and Casting)

User input is a way to get information from the user after you've finished writing your program. Much like expressions, user input is part of creating Python programs that do interesting things.

The Python code you need to use in order to get input from the user will look something like this:

```
userName = input("Enter your name please: ")
```

When your program is run, this will print out the message **`"Enter your name please: "`** to the screen. After the user enters their answer and presses enter, the text that they entered will be stored as the value of **`userName`**.

Casting

However, even if we ask for an integer and the user enters one, the value will be stored as a string instead. We can't do addition or multiplication like we might want to with a string – Python treats ints and strings differently.

We can fix this by telling the program that the input is actually an integer. Doing this is called *casting*, a process that changes a variable from one type to another. For example, if we want to convert the user's age to an integer, we could write something like this:

```
userAge = int(input("Enter your age please: "))
```

If we wanted their GPA (which would be a float, and not an integer) we could write something like this:

```
userGPA = float(input("Enter your GPA please: "))
```

Part 4A: Exercise: Miles per Gallon Calculator

In this lab, you'll be creating a Python program completely from scratch. The problem you'll be solving today is calculating a car's MPG (miles per gallon).

Before you start programming, think about what you need as input, what you plan to output, and what process you'll use to get from input to output.

Part 4B: Input and Process

To determine a car's MPG you only need two things:

1. The distance the car traveled in one trip (*e.g.*, 250 miles)
2. The amount of gas used for that trip (*e.g.*, 10 gallons)

The formula to calculate the MPG is simple: divide the distance traveled by the number of gallons used. Using the example numbers given above, we would get 25 miles per gallon.

Part 4C: Writing your Program

After logging into GL, navigate to the **Labs** folder inside your **201** folder. Create a folder there called **lab3**, and go inside the newly created **lab3** directory.

```
linux2[1]% cd 201
linux2[2]% cd Labs
linux2[3]% pwd
/afs/umbc.edu/users/k/k/k38/home/201/Labs
linux2[4]% mkdir lab3
linux2[5]% cd lab3
linux2[6]% pwd
/afs/umbc.edu/users/k/k/k38/home/201/Labs/lab3
linux2[7]%
```

Now you are going to write a simple Python program called calculateMPG.py.

To open your file for editing, type

emacs calculateMPG.py &

and hit enter. (The ampersand at the end of the line is important – without it, your terminal will "freeze" until you close the emacs window. **Do not include the ampersand if you are not on a lab computer.**)

The first thing you should do in your new file is create and fill out the comment header block at the top of your file. Here is a template:

```
# File:          calculateMPG.py
# Author:        YOUR NAME
# Date:          TODAY'S DATE
# Section:       YOUR SECTION NUMBER
# E-mail:        USERNAME@umbc.edu
# Description:   YOUR DESCRIPTION GOES HERE AND HERE
#                YOUR DESCRIPTION CONTINUED SOME MORE
```

After this, you can start writing code. First, you need to get input from the user for the distance the car traveled and the amount of gas used. Make sure you:

- Assign the results of the input to variables
- Give the variables meaningful names
- Cast the input to **`int`** (otherwise they'll be strings)

Once you have both of these variables, you can use them to calculate the car's miles per gallon, and assign the results to a third variable. After that, all you need to do is display the MPG and you're done!

To test your program, first enable Python 3, then run it with the **`python`** command:

```
linux2[7]% /usr/bin/scl enable python33 bash
bash-4.1$ python calculateMPG.py
Please enter the distance traveled (in miles): 190
Please enter the gallons used: 8
The MPG is: 23.75
bash-4.1$
```

Part 5: Completing Your Lab

Since this is an in-person lab, you do not need to use the **`submit`** command to complete your lab. Instead, raise your hand to let your TA know that you are finished.

They will come over and check your work – they may ask you to run your program for them, and they may also want to see your code. Once they've checked your work, they'll give you a score for the lab, and you are free to leave.

IMPORTANT: If you leave the lab without the TA checking your work, you will receive a **zero** for this week's lab. Make sure you have been given a grade before you leave!