

# The Basic Math Program

## Intro

This week, I learned the fundamentals of collecting user input and manipulating it to generate dynamic results based on that input. With a focus on numeric type conversion and mathematical operators in Python, I created a program to calculate user input accurately and to display it back to the user. The following information is a breakdown on how I wrote this program step-by-step.

## Creating the Program

When coding in Python, there are many ways to write a program and achieve the same result. To write the Basic Math program, I decide that I want to first greet the user and give them a brief description of what is being asked of them and how their input will be used.

I start my program off by creating a print function that includes the following instructions. Since the information will span multiple lines and I want to have more freedom than what the standard print command allows, I will be using a triple-quoted print string. (Figure 1.1)

```
print(  
    """  
        Welcome to the Basic Math program!  
        Please enter two numbers below and the program will  
        calculate them for you. Please enter numbers only.  
        Thank you!  
    """)
```

**Figure 1.1: Triple-quoted print string including the Basic Math program introduction.**

Notice that I have kept my beginning and ending parenthesis and quotes in line with each other. This will help keep track of where the function begins and ends and allows me to identify any possible mistakes should they occur when running this program.

Next, I want to ask the user to input any two numbers of their choosing. I'm going to separate these two numbers by writing two variables named "firstNumber" and "secondNumber". The variable is going to capture the user's data with the input function and then use that same variable to convert the provided string value into a numeric value through the float function. (Figure 1.2)

```
firstNumber = input("Enter a first number: ")
firstNumber = float(firstNumber)

secondNumber = input("Enter a second number: ")
secondNumber = float(secondNumber)
```

**Figure 1.2: Using a variable to store input and using float function to convert the string value.**

### Float vs. Integer

Note that I am using the float function in my program. I could have used the integer (int) function to convert the string into a number but I prefer the flexibility that the float function allows the user and my program. The integer function will return the numeric value of the string so long as it is a whole number. On the other hand, floating-point values, or floats, allow a number to include fractional parts.

For example, if a user input the number "1.5" in to the program but my code is written with an integer function, the program will immediately close on the user. This does not make for a pleasant experience.

If I run my code in the Python Shell with the same number of "1.5", I will receive a ValueError informing me that the number is invalid. (Figure 1.3)

```
Enter a first number: 1.5
Traceback (most recent call last):
  File "C:\Users\phann_000\Desktop\BasicMath.py", line 21, in <module>
    firstNumber = int(firstNumber)
ValueError: invalid literal for int() with base 10: '1.5'
```

**Figure 1.3: ValueError stating that '1.5' was not a valid number for the int() function**

As it is completely acceptable to use fractional numbers in the Basic Math program, I will be using the float function to avoid this potential problem altogether.

### Creating the Equations

Now that variables "firstNumber" and "secondNumber" are ready to be used in mathematical equations, I want to use them to add, subtract, multiply, and divide the two numbers that the user provides.

I'm going to do this by creating a variable for each mathematical operator that I plan to use. I will use the operator's name for the variable that the operator will be used in. (Figure 1.4)

```
addition = firstNumber + secondNumber
```

**Figure 1.4: An example of the addition variable that is adding the first and second number together.**

Next, I will use the print function to display the answer to the user. For this, I will need to write the text of the sentence, enclosed in double quotes, with the variables that contain the user's input in the appropriate spots of the print function. (Figure 1.5)

```
print("\nThe sum of", firstNumber, "and", secondNumber, "is:", addition)
```

**Figure 1.5: The print function containing the two user captured variables and for the calculation.**

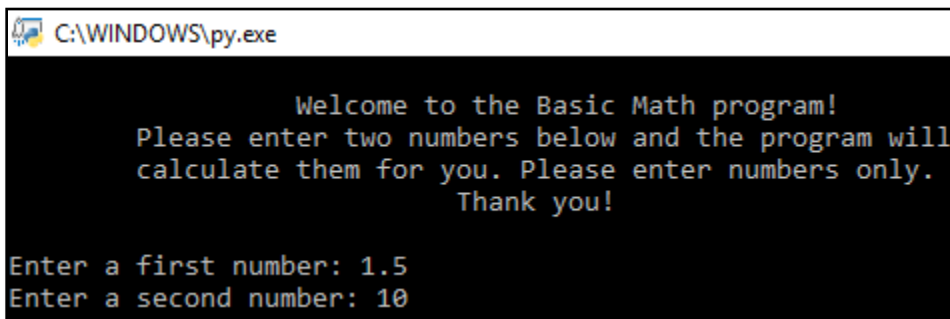
Last, I like to give the user the opportunity to close out of my programs by using the input function, letting them know that they can exit by pressing the Enter key. (Figure 1.6)

```
input("\n\nPlease press the Enter key to exit...")
```

**Figure 1.6: Input function to prevent the program from closing without user's permission.**

## Testing the Program

Now that the code for the program is complete, it's time to run it to ensure that it works. First, I save the script and run it in the command shell. I am given a brief intro and minor instructions from the program to enter a number, press Enter, and I am then prompted to enter a second number. (Figure 2.1)



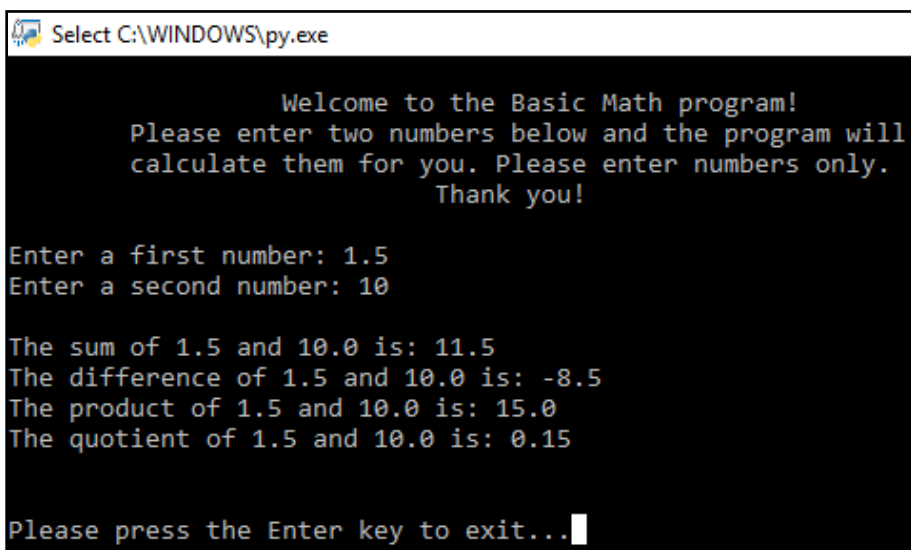
```
C:\WINDOWS\py.exe

Welcome to the Basic Math program!
Please enter two numbers below and the program will
calculate them for you. Please enter numbers only.
Thank you!

Enter a first number: 1.5
Enter a second number: 10
```

**Figure 2.1: The beginning of the program runs as intended in command shell.**

I press Enter again to see if the program calculates my numbers properly and is returning fractional numbers back to me. (Figure 2.2)



```
Select C:\WINDOWS\py.exe

Welcome to the Basic Math program!
Please enter two numbers below and the program will
calculate them for you. Please enter numbers only.
Thank you!

Enter a first number: 1.5
Enter a second number: 10

The sum of 1.5 and 10.0 is: 11.5
The difference of 1.5 and 10.0 is: -8.5
The product of 1.5 and 10.0 is: 15.0
The quotient of 1.5 and 10.0 is: 0.15

Please press the Enter key to exit...
```

**Figure 2.2: The program successfully calculates and displays equations to the user.**

## Nesting the Script

Now that my code runs correctly, I want to clean it up. Although it is organized, I created variables for each calculation when their only purpose was to display the result of that equation to the user. Since the variables are never used after that, I decided that I don't necessarily need to store the results in variables. If I nest the function calls that I already have I will be able to lessen the amount of code in the program while retaining the same result. For the "firstNumber" variable, this is made possible by placing the input function into the float function. (Figure 3.1)

```
firstNumber = float(input("Enter a first number: "))
```

**Figure 3.1: The float function with input nested into it.**

When doing this for the mathematical equations, I can nest "firstNumber" and "secondNumber" together using an operator to manipulate the numbers that they contain. This is done within the print function, calculating the entire equation and showing it to the user in one line of code. (Figure 3.2)

```
print("\nThe sum of", firstNumber, "and", secondNumber, "is:", (firstNumber + secondNumber))
```

**Figure 3.2: Nesting the two variables, added together, within the print function.**

A program that used to be 28 lines of code has now been reduced to 22 lines. It may seem minimal but this preference is always up to the coder and nesting is another way to organize your code.

I always include notes at the top of my code to better understand what I have done. To finish the script, I include the title of my program, its creation date, a description of what its function is, as well as if and when changes have been made. I save my changes and close the program. (Figure 3.3)

```
#.....#  
#Title: BasicMath.py  
#Desc: This script asks the user to input two variables  
#and those two numbers will be be calculated and printed  
#to the user with varies mathematical results  
#Change Log: (PMayner, 2019-04-07, Nested variables)  
#PMayner,2019-04-07, Created File  
#.....#
```

**Figure 3.3: Notes for the Basic Math program.**

## Summary

The Basic Math program shows us that there are many ways to write a program on the back-end while presenting the same front-end to the user. As both versions of the program I created worked properly, I was able to choose how I wanted to write the program by refactoring what I already had written into single statements. There are advantages to nesting the code you have but it is not always required that you do it. Lastly, it is best practice to keep the longevity of your program in mind by ensuring that it is concise and documented well for yourself and others who may review it.