Justin Calhoun

Professor Summer Rae Elasady

Foundations Of Programming: Python

8 October 2018

Assignment 02

For assignment two, we have been tasked to “Create a new program that asks the user to input 2 numbers then prints out the sum, difference, product, and quotient.” To start, I chose to open with some nice friendly text:

|  |
| --- |
| print("Simple Numerical Operations Script\n")  print(  "This python script takes two numbers, provided by you, and outputs a "  "variety of mathematical comparisons between them.\n"  ) |

This gives the user a quick heads-up, and also gives me some strings that could easily be repurposed if I choose later to bring in argparse and add an -h flag; I like scripts that can be run entirely from the command line if so desired.

Early on, I realized that I was going to want to validate the user’s input; both prior experience and some failed iterations on the code for this assignment reinforced the idea that if a user is allowed to type anything, eventually they’ll type something that breaks it. The method I came up with involves two concepts we haven’t discussed in this course yet: while loops and the try statement:

|  |
| --- |
| while True:  try:  valueOne = float(input("Please provide the first number: "))  except ValueError:  print("That was not recognized as a number. Please try again.\n")  continue  else:  break |

First, I start up a while loop that is always True. Inside this loop, I use try to attempt to collect input from the user and store it as a float. If they provide input that cannot be typed as a float, we catch the ValueError with the except clause, inform them to try again, and continue the loop from the top. This will continue until we can successfully store a float value into valueOne, at which point the else clause will break the loop.

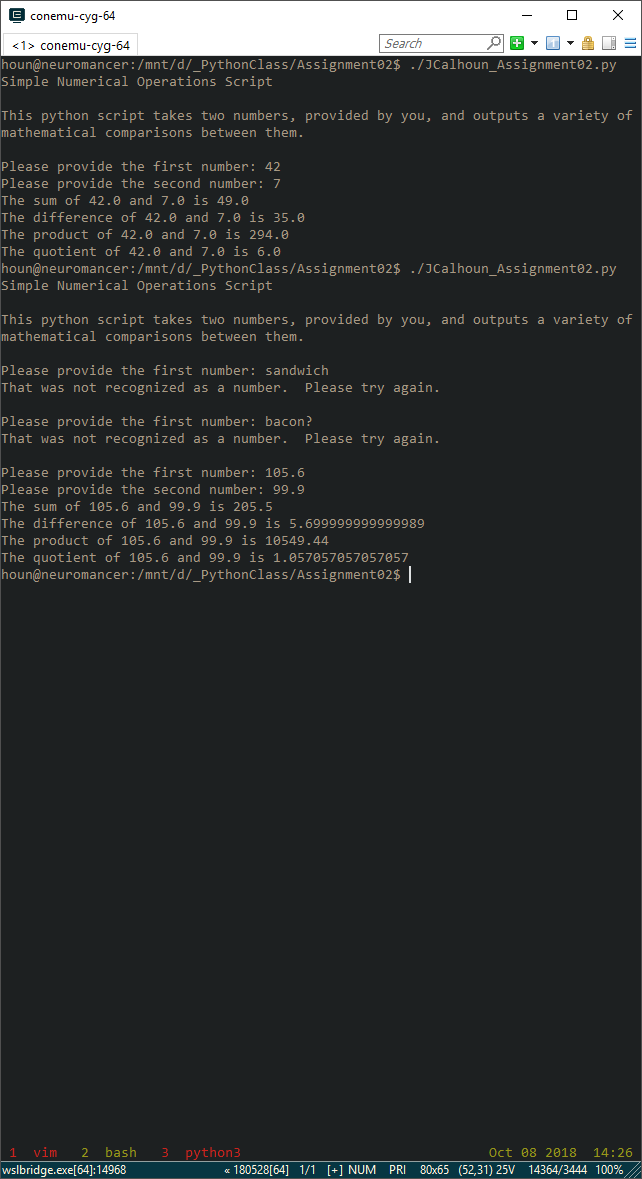


Figure 1: Input validation

Now that we have two values from the user, we simply have to apply math and print the results:

|  |
| --- |
| # First, the sum  print("The sum of {0} and {1} is {2}".format(  valueOne,  valueTwo,  valueOne + valueTwo  )  )  # Second, the difference  print("The difference of {0} and {1} is {2}".format(  valueOne,  valueTwo,  (valueOne - valueTwo)  )  )  # Third, the product  print("The product of {0} and {1} is {2}".format(  valueOne,  valueTwo,  (valueOne \* valueTwo)  )  )  # Finally, the quotient  print("The quotient of {0} and {1} is {2}".format(  valueOne,  valueTwo,  (valueOne / valueTwo)  )  ) |

As seen in Figure 1, this would appear to work fine. Until you put a zero in the quotient.

Figure 2: Division by zero

So, I used an if statement for the final operation in the special case of valueTwo being 0:

|  |
| --- |
| # Finally, the quotient  if valueTwo == 0:  print("Sadly, {0} cannot be divided by {1}...".format(  valueOne,  valueTwo  )  )  else:  print("The quotient of {0} and {1} is {2}".format(  valueOne,  valueTwo,  (valueOne / valueTwo)  )  ) |

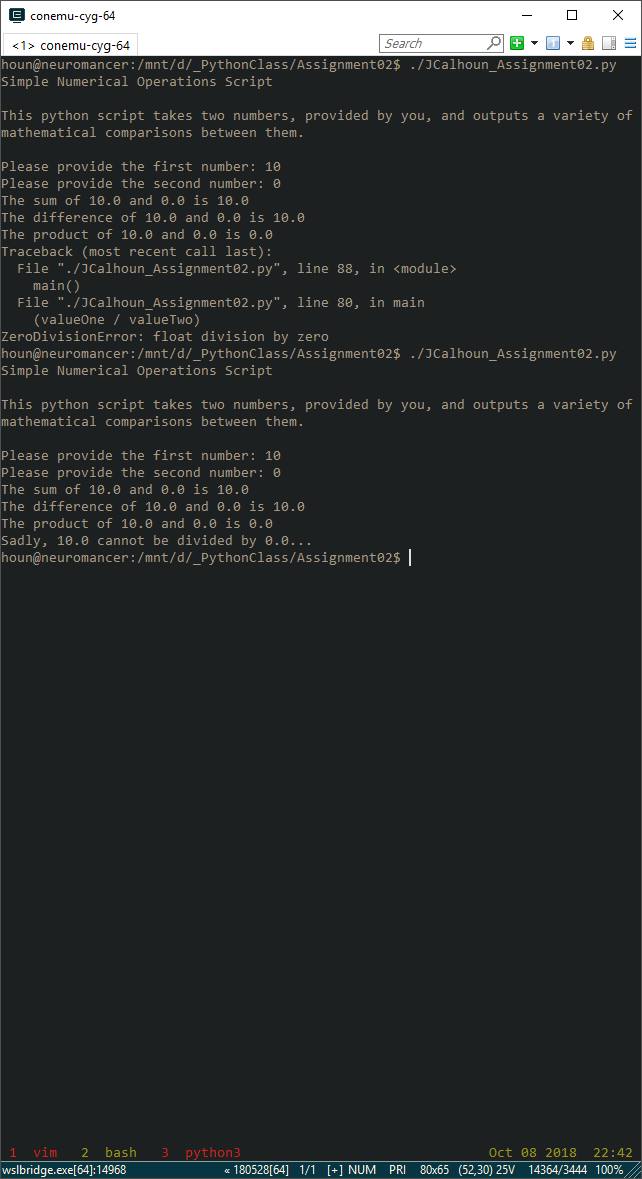
Finally, I have a complete run:

Figure 3: Division by zero avoided

This overview comprises the final version of this script, but there were a few earlier iterations that missed things like the user putting in a zero, or the division by zero. While I’m sure there’s still some ways to break it, it feels like it can handle real-world users at this point with little fear of breaking in normal use.

The full script, with comments, can be found in JCalhoun\_Assignment02.py. Additionally, I decided to challenge myself to remove some of the code redundancy, so I created a second version that moves the user input validation to a separate function, and imports operator and collections.OrderedDict to allow the math and output to be done via iteration instead of copy/paste text blocks. It can be found in JCalhoun\_Assignment\_extra.py.