How To Make a Simple Calculator Program in Python 3

Posted November 16, 2016 172.6k PYTHON DEVELOPMENT PROGRAMMING PROJECT

By: Lisa Tagliaferri

Introduction

The Python programming language is a great tool to use when working with numbers and evaluating mathematical expressions. This quality can be utilized to make useful programs.

This tutorial presents a learning exercise to help you make a simple command-line calculator program in Python 3. While we'll go through one possibile way to make this program, there are many opportunities to improve the code and create a more robust calculator.

We'll be using math operators, variables, conditional statements, functions, and handle user input to make our calculator.

Prerequisites

For this tutorial, you should have Python 3 installed on your local computer and have a programming environment set up on the machine. If you need to either install Python or set up the environment, you can do so by following the appropriate guide for your operating system.

Step 1 — Prompt users for input

Calculators work best when a human provides equations for the computer to solve. We'll start writing our program at the point where the human enters the numbers that they would like the computer to work with.

To do this, we'll use Python's built-in `input()` function that accepts user-generated input from the keyboard. Inside of the parentheses of the function we can pass a string to prompt the user. We'll assign the user's input to

For this program, we would like the user to input two numbers, so let's have the program prompt for two numbers. When asking for input, we should include a space at the end of our string so that there is a space between the user's input and the prompting string.

After writing our two lines, we should save the program before we run it. We can call this program `calculator.py` and in a terminal window, we can run the program in our programming environment by using the command `python calculator.py`. You should be able to type into the terminal window in response to each prompt.

```
Output
Enter your first number: 5
Enter your second number: 7
```

If you run this program a few times and vary your input, you'll notice that you can enter whatever you want when prompted, including words, symbols, whitespace, or just the enter key. This is because `input()` takes data in as strings and doesn't know that we are looking for a number.

We would like to use a number in this program for 2 reasons: 1) to enable the program to perform mathematical calculations, and 2) to validate that the user's input is a numerical string.

Depending on our needs of the calculator, we may want to convert the string that comes in from the `input()` function to either an integer or a float. For us, whole numbers suit our purpose, so we'll wrap the `input()` function in the `int()` function to convert the input to the integer data type.

calculator.py

```
number_1 = int(input('Enter your first number: '))
number_2 = int(input('Enter your second number: '))
```

Now, if we input two integers we won't run into an error:

```
Output
Enter your first number: 23
Enter your second number: 674
```

But, if we enter letters, symbols, or any other non-integers, we'll encounter the following error:

```
Output
Enter your first number: sammy
Traceback (most recent call last):
  File "testing.py", line 1, in <module>
```


So far, we have set up two variables to store user input in the form of integer data types. You can also experiment with converting the input to floats.

Step 2 — Adding operators

Before our program is complete, we'll add a total of 4 mathematical operators: `+` for addition, `-` for subtraction, `*` for multiplication, and `/` for division.

As we build out our program, we want to make sure that each part is functioning correctly, so here we'll start with setting up addition. We'll add the two numbers within a print function so that the person using the calculator will be able to see the output.

calculator.py

```
number_1 = int(input('Enter your first number: '))
number_2 = int(input('Enter your second number: '))

print(number_1 + number_2)
```

Let's run the program and type in two numbers when prompted to ensure that it is working as we expect:

```
Output
Enter your first number: 8
Enter your second number: 3
11
```

The output shows us that the program is working correctly, so let's add some more context for the user to be fully informed throughout the runtime of the program. To do this, we'll be using string formatters to help us properly format our text and provide feedback. We want the user to receive confirmation about the numbers they are entering and the operator that is being used alongside the produced result.

calculator.py

```
number_1 = int(input('Enter your first number: '))
number_2 = int(input('Enter your second number: '))
```


Now, when we run the program, we'll have extra output that will let the user confirm the mathematical expression that is being performed by the program.

```
Output
Enter your first number: 90
Enter your second number: 717
90 + 717 =
807
```

Using the string formatters provides the users with more feedback.

At this point, you can add the rest of the operators to the program with the same format we have used for addition:

calculator.py

```
number_1 = int(input('Enter your first number: '))
number_2 = int(input('Enter your second number: '))

# Addition
print('{} + {} = '.format(number_1, number_2))
print(number_1 + number_2)

# Subtraction
print('{} - {} = '.format(number_1, number_2))
print(number_1 - number_2)

# Multiplication
print('{} * {} = '.format(number_1, number_2))
print(number_1 * number_2)

# Division
print('{} / {} = '.format(number_1, number_2))
print(number_1 / number_2)
```

We added the remaining operators, `-`, `*`, and `/` into the program above. If we run the program at this point, the program will execute all of the operations above. However, we want to limit the

program to only perform one operation at a time. To do this, we'll be using conditional statements.

Step 2 — Adding conditional statements

different operators. So, let's start by adding some information at the top of the program, along with a choice to make, so that the person knows what to do.

We'll write a string on a few different lines by using triple quotes:

```
'''
Please type in the math operation you would like to complete:
+ for addition
- for subtraction
* for multiplication
/ for division
'''
```

We are using each of the operator symbols for users to make their choice, so if the user wants division to be performed, they will type `/`. We could choose whatever symbols we want, though, like `1 for addition`, or `b for subtraction`.

Because we are asking users for input, we want to use the `input()` function. We'll put the string inside of the `input()` function, and pass the value of that input to a variable, which we'll name `operation`.

calculator.py

```
operation = input('''
Please type in the math operation you would like to complete:
+ for addition
- for subtraction
* for multiplication
/ for division
''')

number_1 = int(input('Enter your first number: '))
number_2 = int(input('Enter your second number: '))

print('{} + {} = '.format(number_1, number_2))
print(number_1 + number_2)

print('{} - {} = '.format(number_1, number_2))
print(number_1 - number_2)
```

```
print('{} * {} = '.format(number_1, number_2))
print(number_1 * number_2)
```


At this point, if we run our program it doesn't matter what we input at the first prompt, so let's add our conditional statements into the program. Because of how we have structured our program, the `if` statement will be where the addition is performed, there will be 3 else-if or `elif` statements for each of the other operators, and the `else` statement will be put in place to handle an error if the person did not input an operator symbol.

calculator.py

```
operation = input('''
Please type in the math operation you would like to complete:
+ for addition
- for subtraction
* for multiplication
/ for division
''')

number_1 = int(input('Enter your first number: '))
number_2 = int(input('Enter your second number: '))

if operation == '+':
    print('{} + {} = '.format(number_1, number_2))
    print(number_1 + number_2)

elif operation == '-':
    print('{} - {} = '.format(number_1, number_2))
    print(number_1 - number_2)

elif operation == '*':
    print('{} * {} = '.format(number_1, number_2))
    print(number_1 * number_2)

elif operation == '/':
    print('{} / {} = '.format(number_1, number_2))
    print(number_1 / number_2)

else:
    print('You have not typed a valid operator, please run the program again.')
```

To walk through this program, first it prompts the user to put in an operation symbol. We'll say the user inputs `*` to multiply. Next, the program asks for 2 numbers, and the user inputs `58` and `40`. At this point, the program shows the equation performed and the product.


```
Please type in the math operation you would like to complete:
+ for addition
- for subtraction
* for multiplication
/ for division
*
Please enter the first number: 58
Please enter the second number: 40
58 * 40 =
2320
```

Because of how we structure the program, if the user enters `%` when asked for an operation at the first prompt, they won't receive feedback to try again until after entering numbers. You may want to consider other possible options for handling various situations.

At this point, we have a fully functional program, but we can't perform a second or third operation without running the program again, so let's add some more functionality to the program.

Step 4 — Defining functions

To handle the ability to perform the program as many times as the user wants, we'll define some functions. Let's first put our existing code block into a function. We'll name the function `calculate()` and add an additional layer of indentation within the function itself. To ensure the program runs, we'll also call the function at the bottom of our file.

calculator.py

```
# Define our function
def calculate():
    operation = input('''
Please type in the math operation you would like to complete:
+ for addition
- for subtraction
* for multiplication
/ for division
''')

    number_1 = int(input('Please enter the first number: '))
    number_2 = int(input('Please enter the second number: '))
```

```
    if operation == '+':
        print('{} + {} = '.format(number_1, number_2))
        print(number_1 + number_2)
```



```
        print(number_1 - number_2)

    elif operation == '*':
        print('{} * {} = '.format(number_1, number_2))
        print(number_1 * number_2)

    elif operation == '/':
        print('{} / {} = '.format(number_1, number_2))
        print(number_1 / number_2)

    else:
        print('You have not typed a valid operator, please run the program again.')

# Call calculate() outside of the function
calculate()
```

Next, let's create a second function made up of more conditional statements. In this block of code, we want to give the user the choice as to whether they want to calculate again or not. We can base this off of our calculator conditional statements, but in this case we'll only have one `if`, one `elif`, and one `else` to handle errors.

We'll name this function `again()`, and add it below our `def calculate():` code block.

calculator.py

```
...
# Define again() function to ask user if they want to use the calculator again
def again():

    # Take input from user
    calc_again = input('''
Do you want to calculate again?
Please type Y for YES or N for NO.
''')

    # If user types Y, run the calculate() function
    if calc_again == 'Y':
        calculate()
```

```
    # If user types N, say good-bye to the user and end the program
    elif calc_again == 'N':
        print('See you later.')
```



```
# Call calculate() outside of the function
calculate()
```

Although there is some error-handling with the else statement above, we could probably do a little better to accept, say, a lower-case `y` and `n` in addition to the upper-case `Y` and `N`. To do that, let's add the string function `str.upper()`:

calculator.py

```
...
def again():
    calc_again = input('''
Do you want to calculate again?
Please type Y for YES or N for NO.
''')

    # Accept 'y' or 'Y' by adding str.upper()
    if calc_again.upper() == 'Y':
        calculate()

    # Accept 'n' or 'N' by adding str.upper()
    elif calc_again.upper() == 'N':
        print('See you later.')

    else:
        again()
...
```

At this point, we should add the `again()` function to the end of the `calculate()` function so that we can trigger the code that asks the user whether or not they would like to continue.

calculator.py

```
def calculate():
    operation = input('''
Please type in the math operation you would like to complete:
+ for addition
- for subtraction
* for multiplication
```

```
    / for division
    ''')

    number_1 = int(input('Please enter the first number: '))
```



```
    if operation == '+':
        print('{} + {} = '.format(number_1, number_2))
        print(number_1 + number_2)

    elif operation == '-':
        print('{} - {} = '.format(number_1, number_2))
        print(number_1 - number_2)

    elif operation == '*':
        print('{} * {} = '.format(number_1, number_2))
        print(number_1 * number_2)

    elif operation == '/':
        print('{} / {} = '.format(number_1, number_2))
        print(number_1 / number_2)

    else:
        print('You have not typed a valid operator, please run the program again.')

    # Add again() function to calculate() function
    again()

def again():
    calc_again = input('''
Do you want to calculate again?
Please type Y for YES or N for NO.
''')

    if calc_again.upper() == 'Y':
        calculate()
    elif calc_again.upper() == 'N':
        print('See you later.')
    else:
        again()

calculate()
```

You can now run your program with `python calculator.py` in your terminal window and you'll be able to calculate as many times as you would like.

Step 5 — Improving the code

We now have a nice, fully functional program. However, there is a lot more that you can do to improve this code. You can add a welcome function, for example, that welcomes people to the


```
def welcome():
    print('''
Welcome to Calculator
''')
...
# Don't forget to call the function
welcome()
calculate()
```

There are opportunities to introduce more error-handling throughout the program. For starters, you can ensure that the program continues to run even if the user types `plankton` when asked for a number. As the program is right now, if `number_1` and `number_2` are not integers, the user will get an error and the program will stop running. Also, for cases when the user selects the division operator (`/`) and types in `0` for their second number (`number_2`), the user will receive a `ZeroDivisionError: division by zero` error. For this, you may want to use exception handling with the `try ... except` statement.

We limited ourselves to 4 operators, but you can add additional operators, as in:

```
...
    operation = input('''
Please type in the math operation you would like to complete:
+ for addition
- for subtraction
* for multiplication
/ for division
** for power

% for modulo
''')
...
# Don't forget to add more conditional statements to solve for power and modulo
```

Additionally, you may want to rewrite part of the program with a loop statement.

There are many ways to handle errors and modify and improve each and every coding project. It is important to keep in mind that there is no single correct way to solve a problem that we are

presented with.

Conclusion

nmand line. on other projects that require user input on the command line.

We are interested in seeing your solutions to this simple command-line calculator project! Please feel free to post your calculator projects in the comments below.

Next, you may want to create a text-based game like tic-tac-toe or rock-paper-scissors.

By: Lisa Tagliaferri

Related Tutorials

How To Set Up Jupyter Notebook with Python 3 on Ubuntu 18.04

How To Install and Configure pgAdmin 4 in Server Mode

How To Build a Neural Network to Recognize Handwritten Digits with TensorFlow

How to Install, Run, and Connect to Jupyter Notebook on a Remote Server

How To Set Up a Jupyter Notebook with Python 3 on Debian 9

19 Comments

Leave a comment...

Log In to Comment

Forst *November 23, 2016*

1 The easiest way might be the best :)

```
$ python3
>>> 2 + 4
6
```

Hispanion *November 25, 2016*

2 Excelent tutorial! i learn a lot just coding this, thank you.

ltagliaferri MOD *November 25, 2016*

2 Awesome! Thanks for taking the time to comment :)

rsnrdy18 *December 2, 2016*

0 hello,
I found this tutorial very helpful, thank you
but, I have doubt, In the code just above step 5 the again function is called before the function is defined and yet the programme is not showing error, am I missing something...?

ltagliaferri MOD *December 2, 2016*

1 Hi there, thanks for your comment!

fined, yes, but neither `calculate()` nor `again()` are called until the last line of the program file. Calling something that's not defined is a **runtime error**, so since the call to `again()` is in a definition prior to being defined it does not produce an error. Both `calculate()` and `again()` are not executed until the bottom of the program, after both have been defined.

You may prefer to have the `again()` definition above the `calculate()` definition which is a valid way to construct the program. I put `calculate()` above because it is the core purpose of the program, but you could switch it :)

rsnrdy18 *December 2, 2016*

1 yeah makes sense now....thanks for the reply

nelsonmarcos *December 6, 2016*

1 Although this tutorial seems to be trivial, it cover important concepts like recursion and strings interpolation.

It was pretty fun to do it.

Thanks and waiting for the next!

ltagliaferri MOD *December 6, 2016*

1 Awesome, would love to see if you came up with a more complicated calculator based on what was started here :)

Matematicas *March 25, 2017*

0 Hi Lisa: great program . Would you have a program for adding fractions of different denominators ??? in Python ??? I am trying to make something for children 5 to 7 grades aprox.... so they can actually see the fraction displayed , operations, and get few questions before seeying the result, thanks ! greetings from Santiago of Chile , Richard teaching private maths to small monsters...

ltagliaferri MOD *March 27, 2017*

0 Hi Richard,

Here is something based on the program above with functions for adding fractions:

```
from fractions import Fraction
```



```
    print('Sum of fractions: {0}'.format(x+y))


def calculate():
    x = Fraction(input('Enter your first fraction: '))
    y = Fraction(input('Enter your second fraction: '))
    add_fractions(x, y)
    again()


def again():
    calc_again = input('''
Do you want to calculate again?
Please type Y for YES or N for NO.
''')

    if calc_again.upper() == 'Y':
        calculate()
    elif calc_again.upper() == 'N':
        print('See you later.')
    else:
        again()

calculate()
```

petersonmotors *January 18, 2018*

0 can someone show me how to add more conditional statements to solve for power and module

thanks so much

RoyHobbs *July 3, 2018*

0 Just follow the syntax pattern of the other conditional operations, Peterson. Let me know if this hint helps you. I'd be happy to provide you with more hints if and when needed.

- Roy

Great tutorial!!

My solution uses far less code and is very simplistic.

I also included an option for users to use exponentiation, along with the basic mathematical operations.

def main( ):
print("{:^72}".format('Interactive Python Calculator'))
print("{:^72}".format('---------------------------'))
print("This interactive program will allow you to type a mathematical\n"
"expression, then display the value of that expression. You may\n"
"perform the following operations: addition (+), subtraction(-)\n"
"multiplication (*), division (/), or exponentiation (**). You may\n"
"exit the program at any time by typing the following: exit(), followed by\n"
"pressing the Enter key, then clicking the OK button in the popup window Kill.")

```
for i in range(100):
    express = eval(input('\nEnter your mathematical expression: '))
    print('The value of your expression is',express)
    print( )


main( )
```

ItsMe88 *February 26, 2018*

0 To get operation and calc_again to correctly run

You need to add:
raw_input('''

instead of
input('''

RoyHobbs *July 3, 2018*

1 Excellent article/tutorial, Lisa! Looks like I'm going to be spending some worthwhile time on this website.

ltagliaferri MOD *October 11, 2018*

0 Thank you, glad you are finding the Community tutorials useful!

kaluwaavril *October 10, 2018*

1 THANK YOU. I am new to programming and after reviewing countless "simple" calculator tutorials, I found yours to be the clearest. I liked how you built upon each example to introduce the concepts - I did not feel overwhelmed because of this approach. Keep up the great work

ltagliaferri MOD *October 11, 2018*

0 Awesome, I am glad you're finding the building approach to be useful and that it is helping you get into programming! I hope you are enjoying learning about and developing software :)

adrianlum7447 *about 15 hours ago*

0 hi, if I will like to have number*1 and number*2 to be bigger than 0. How should i amend the program.