How to Make a Python Calculator

Creating a basic calculator program in Python is a great way to get familiar with the language and its logic.

Written by **Max Reynolds**
Published on Mar 01 2023

Image: Shutterstock / Built In

Making a calculator is a classic beginner project for those who are just starting to learn to program with Python. This project can be completed in a relatively short amount of time, making it a great stepping stone for building more complex programs and exploring other areas of software development.

WHY BUILD A PYTHON CALCULATOR?

Building a Python calculator allow for hands-on experience with the language and provides an opportunity to understand the basics of coding logic, arithmetic operations and user-input functions.

We will build a basic calculator that takes in two input numbers and an operator from the user, but the project can be built upon in the future if you are interested in exploring more complex logic or graphical user interfaces.

MORE FROM HAN REYNOLDS:
Differences Between SQL and SOQL Explained

Make a Python Calculator

Here's what you need to get started with this exercise.

1. A text editor or integrated development environment. I like VS Code. Others include Pycharm, Spyder, vim, etc.
2. Python installed on your computer. (See python.org)
3. A basic understanding of Python syntax, variables, and data types (integer, float, boolean, etc.)

USER INPUT IN PYTHON

Let's start by getting familiar with input and output in Python, or I/O. For collecting user input in Python, we can use the `input()` function. For output, we use `print()`.

Let's take a user's input and print it out.

```
inp = input('Welcome, please enter a number: ')
print('You entered', inp)
```

First our program prompts the user with "Welcome, please enter a number" and then prints the user's input (stored in the variable `inp`).

Note: to run a Python program, put the code into a file with the .py extension. In the command line, run `python <filename>.py`.

Our user input stored in `inp` is a string. This is fine if we are just printing out the user's input. But we need to convert it if we want to do any sort of math with the input. Let's convert our input to a `float`.

```
inp = float(input('Welcome, please enter a number: '))
print('You entered', inp)
```

DEFINING OPERATORS

Now let's talk about the math we want to build into the calculator. We'll allow our calculator to use five operations: addition, subtraction, multiplication, division and exponents.

Let's prompt the user to enter their two numbers and an operator. We can store the three inputs and then output the whole equation by passing multiple arguments to `print()`.

```
number1 = float(input('Enter first number: '))
op = input('Enter operator (+,-,*,/,**): ')
number2 = float(input('Enter second number: '))
print(number1,op,number2)
```

Note: we don't convert `op` to a float.

CONDITIONALLY SELECT AN OPERATION

We now have two numbers and an operator selected by the user. Now we need to convert that input into Python code to compute the calculation.

To do this, we need to create a function. Our function will take in three arguments: the two numbers and the operator string. We'll call the function `calculate`.

```
def calculate(n1,n2,op):
    if op == '+':
        result = n1+n2
    elif op == '-':
        result = n1-n2
    elif op == '*':
        result = n1*n2
    elif op == '/':
        result = n1/n2
    elif op=='**':
        result = n1**n2

    return result
```

The calculate function uses conditional statements, which allow you to execute a certain block of code only if a certain condition is met. In our case, for example, we only need to add the two numbers if the `+` operator is passed in. Depending on the operator, we store the calculation result using the `result` variable and return it.

As you can see, the basic math operations in Python using typical operators. The only irregular one is exponential, which uses `**` rather than `^`, as `n^3` is the equivalent of n^3.

Let's call our function and put it all together!

```
def calculate(n1,n2,op):
    if op == '+':
        result = n1+n2
    elif op == '-':
        result = n1-n2
    elif op == '*':
        result = n1*n2
    elif op == '/':
        result = n1/n2
    elif op=='**':
        result = n1**n2

    return result


number1 = float(input('Enter first number: '))
op = input('Enter operator (+,-,*,/,**): ')
number2 = float(input('Enter second number: '))
print(number1,op,number2)
result=calculate(number1,number2,op)
print('=',result)
```

The input and output will look something like this:

```
Enter first number: 5
Enter operator (+,-,*,/,**): +
Enter second number: 3
5.0 + 3.0 = 8.0
```

MORE SOFTWARE ENGINEERING PERSPECTIVES:
A Handy Guide to Python

Python Calculator Bells and Whistles

We now have a basic functional Python calculator. But we can add a few simple things to make it a bit more user-friendly.

Say we want to perform multiple calculations without having to re-run our script. One way to do this is to create a variable called `continue_calculating`. As long as `continue_calculating` is `True`, we keep on performing calculations.

```
continue_calculating = True
while continue_calculating == True:
    number1 = float(input('Enter first number: '))
    op = input('Enter operator (+,-,*,/,**): ')
    number2 = float(input('Enter second number: '))
    result=calculate(number1,number2,op)
    print('=',result)
    yes_or_no = input('Continue? (y/n): ')
    if yes_or_no == 'n':
        continue_calculating = False
```

We've done a few things here. First, we create `continue_calculating` and initialize it as `True`. Next we start a `while` loop which continues as long as `continue_calculating` is `True`. After performing the calculation, we ask the user if they want to continue using `input` again. If they enter 'n', then the process ends. If they enter 'y', we start the loop over again and do another calculation.

If our calculation result is equivalent to an integer (e.g. 3.0), we may want to just print the output without the decimal. We can use the built-in `is_integer` function for this, and our output becomes a bit cleaner.

```
if result.is_integer():
    result = int(result)
```

Finally, we can raise an error if the user enters an invalid operator:

```
if op == '+':
    result = n1+n2
elif op == '-':
    result = n1-n2
elif op == '*':
    result = n1*n2
elif op == '/':
    result = n1/n2
elif op=='**':
    result = n1**n2
else:
    raise ValueError('Invalid operator')
```

Our finished calculator script:

```
def calculate(n1,n2,op):
    if op == '+':
        result = n1+n2
    elif op == '-':
        result = n1-n2
    elif op == '*':
        result = n1*n2
    elif op == '/':
        result = n1/n2
    elif op=='**':
        result = n1**n2
    else:
        raise ValueError('Invalid operator')

    if result.is_integer():
        result = int(result)

    return result

continue_calculating = True
while continue_calculating == True:
    number1 = float(input('Enter first number: '))
    op = input('Enter operator (+,-,*,/,**): ')
    number2 = float(input('Enter second number: '))
    print(number1,op,number2)
    result=calculate(number1,number2,op)
    print('=',result)
    yes_or_no = input('Continue? (y/n): ')
    if yes_or_no == 'n':
        continue_calculating = False
```

Creating a basic calculator program in Python is a great starting point for beginners who are looking to familiarize themselves with the language and its logic. This project covered some basic concepts of variables, data types, user input, functions and conditional statements.

The calculator project can be completed in a relatively short amount of time and can be expanded upon by adding more complex logic or graphical user interfaces. For example, you might add the ability for a user to push buttons instead of entering text. Or, maybe you want to parse a single equation string instead of three separate inputs. Either way, I hope this project was a fun and helpful experience.

RECENT EXPERT CONTRIBUTORS ARTICLES

How to Build a Resilient Global Team Culture

Do You Really Need That New Tech?

How Web Intelligence Can Empower Environmental Activism

Expert Contributors | Software Engineering Perspectives