Chapter 13
Implementing a Calculator Using Functions

13.1 Introduction

In this chapter we will step through the development of another Python program; this time the program will be to provide a simple calculator which can be used to add, subtract, multiple and divide numbers. The implementation of the calculator is base on the Function Decomposition performed earlier in the book in the Introduction to Structured Analysis chapter.

The calculator will be implemented using Python functions to help modularise the code.

13.2 What the Calculator Will Do

This will be a purely command driven application that will allow the user to specify

- the operation to perform and
- the two numbers to use with that operation.

When the program starts up it can use a loop to keep processing operations until the user indicates that they wish to terminate the application.

We can also use an `if` statement to select the operation to perform etc.

As such it will also build on several other features in Python that we have already been working with.

J. Hunt, *A Beginners Guide to Python 3 Programming*,
Undergraduate Topics in Computer Science,
https://doi.org/10.1007/978-3-030-20290-3_13

13.3 Getting Started

The first step will be to create a new Python file. If you are using the PyCharm IDE you can do it using the New>PythonFile menu option (look back at the number guess game chapter if you can't remember how to do this). The file can be called anything you like, but *calculator* seems like a reasonable name.

In the newly created (and empty) `calculator.py` file type in a welcome print message such as:

```
print('Simple Calculator App')
```

Now run the `calculator.py` program (again if you don't remember how to do that look back at the Number Guess Game chapter).

You should see the message printed out in the Python console. This verifies that the file has been created properly and that you can run the Python code you will define in it.

13.4 The Calculator Operations

We are going to start by defining a set of functions that will implement the add, subtract, multiply and divide operations.

All of these functions take two numbers and return another number. We have also given each function a docstring to illustrate their use, although in practice the functions are so simple and self-describing that the docstring is probably redundant.

The functions are listed below; you can now add them to the `calculator.py` file:

```
def add(x, y):
    """ Adds two numbers """
    return x + y

def subtract(x, y):
    """ Subtracts two numbers """
    return x - y

def multiply(x, y):
    """ Multiples two numbers """
    return x * y

def divide(x, y):
    """Divides two numbers"""
    return x / y
```

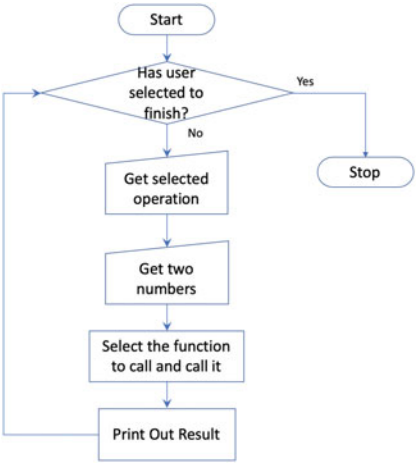
We now have the basic functions needed by the calculator.

13.5 Behaviour of the Calculator

We can no explore what the operation of the calculator program should be.

Essentially, we want to allow the user to be able to select the operation they want to perform, provide the two numbers to use with the operation and then for the program to call the appropriate function. The result of the operation should then be presented to the user.

We then want to ask the user whether they want to continue to use the calculator or to exit the program. This is illustrated below in flow chart form:



Based on this flowchart we can put in place the skeleton of the logic for the calculator's processing cycle.

We will need a `while` loop to determine whether the user has finished or not and a variable to hold the result and print it out.

The following code provides this skeleton.

```
finished = False
while not finished:
    result = 0
    # Get the operation from the user
    # Get the numbers from the user
    # Select the operation
    print('Result:', result)
    print('==================')
    # Determine if the user has finished

print('Bye')
```

If you try to run this right now then you will find that this code will loop forever as the user is not yet prompted to say if they wish to continue or not. However, it does provide the basic framework; we have

- a variable, `finished`, with a Boolean flag in it to indicate if the user has finished or not. This is referred to as a flag because it is a Boolean value and because it is being used to determine whether to terminate the main processing loop or not.
- a variable to hold the result of the operation and the two numbers.
- the `while` loop representing the main processing loop of the calculator.

13.6 Identifying Whether the User Has Finished

We could address several of the remaining areas next; however, we will select the last step—that of determining if the user has finished or not. This will allow us to start to run the application so that we can test out the behaviour.

To do this we need to prompt the user to ask them if they want to continue using the calculator.

At one level this is very straight forward; we could ask the user to input 'y' or 'n' to indicates *yes* I have finished or *no* I want to keep going.

We could therefore use the input function as follows:

```
user_input = input('Do you want to finish (y/n): ')
```

We could then check to see if they have entered a 'y' character and terminate the loop.

However, anytime we take any input from something outside of our program (such as the user) we should verify the input. For example, what should the program do if the user enters 'x' or the number '1'? One option is to treat anything that is not a 'y' as being—*I want to keep going*. However, this is opening up our simple program to bad practices and (in a much larger system) to potential security issues and certainly to potential hacker attacks.

It is a much better idea to verify that the input is what is expected and to reject any input until it is either a 'y' or an 'n'.

This means that the code is more complex than a single input statement; there is for example an implied loop here and as well as some idea of input validation.

This means that this is an ideal candidate for a function that will *encapsulate* this behaviour into a separate operation. We can then test this function which is always a good idea. It also means that where we use the function, we have a level of *abstraction*. That is we can name the function appropriately which will make it easier to see what we intended, instead of having a mass of code in one place.

We will call the function `check_if_user_has_finished`; this name makes it very clear what the purpose of the function is. It also means that when we use it in our main processing loop its role in that loop will be obvious.

The function is given below:

```
def check_if_user_has_finished():
    """
    Checks that the user wants to finish or not.
    Performs some verification of the input."""
    ok_to_finish = True
    user_input_accepted = False
    while not user_input_accepted:
        user_input = input('Do you want to finish (y/n): ')
        if user_input == 'y':
            user_input_accepted = True
        elif user_input == 'n':
            ok_to_finish = False
            user_input_accepted = True
        else:
            print('Response must be (y/n), please try again')
    return ok_to_finish
```

Notice the use of two variables that are local to the function:

- the first variable (`ok_to_finish`) holds the result of the function; whether it is OK to finish or not. It is given a default value of `True`; this follows the fail closed approach—which suggests that it is always better to fail by closing down an application or connection. In this case it means that if something goes wrong with the code (if it contains a software bug or logic error) the user will not keep looping forever.
- the second variable (`user_input_accepted`) is used to indicate whether the user has provided an acceptable input or not (i.e. have they entered 'y' or 'n') until they do the loop inside the function will repeat.

The loop itself is interesting as we are looping while the user input has not been accepted; note that we can (almost) read the while loop as plain English text. This is both a feature of Python (it is intended to be easily readable) and also of the use of a meaningful name for the variable itself.

Within the loop we obtain the input from the user; check to see if it is 'y' or 'n'. If it is either of these options, we set the `user_input_accepted` flag to `True`. Otherwise the code will print out a message indicating that the only acceptable input is a 'y' or 'n'.

Notice that we only set the `ok_to_finish` variable to `False` if the user inputs a 'n'; this is because the `ok_to_finish` variable by default has a value of `True` and thus there is no need to reassign `True` to it if the user select 'n'.

We can now add this function into our main processing loop in place of the last comment:

```
finished = False
while not finished:
    result = 0
    # Get the operation from the user
    # Get the numbers from the user
    # Select the operation
    print('Result:', result)
    print('==================')
    finished = check_if_user_has_finished(()


print('Bye')
```

We can now run the application.

You may wonder why we would do this at this point as it does not yet do any calculations for us; the answer is that we can verify that the overall behaviour of the main loop works and that the `check_if_user_has_finished()` function operates correctly.

13.7 Selecting the Operation

Next let us implement the function used to obtain the operation to perform.

Again, we want to name this function in such a way as to help with the comprehensibility of our program. In this case we are asking the user to select which operation they want to perform, so let's call the function `get_operation_choice`.

This time we need to present a list of options to the user and then ask them to make a selection. Again, we want to write our function defensively, so that it makes sure the user only inputs a valid option; if they do not then the function prompts them for another input. This means our function will have a loop and some validation code.

There are four options available to the user: Add, Subtract, Multiply and Divide. We will therefore number then 1 to 4 and ask the user to select an option between 1 and 4.

There are several ways in which we can verify that they have entered a number in this range, including

- converting the string entered into a number and using numerical comparison (but then we need to check that they entered an integer),
- having multiple `if` and `elif` statements (but that seems a bit long winded) or
- by checking that the entered character is one of a set of values (which is the approach we will use).

To check that a value is *in* a set of other value (that it is one of the values in the set) you can use the 'in' operator, for example:

```
user_selection in ('1', '2', '3', '4')
```

This will return `True` if (and only if) `user_selection` contains one of the strings '1', '2', '3' or '4'.

We can therefore use it in our function to verify that the user entered a valid input.

The `get_operation_choice` function is shown below:

```
def get_operation_choice():
    input_ok = False
    while not input_ok:
        print('Menu Options are:')
        print('\t1. Add')
        print('\t2. Subtract')
        print('\t3. Multiply')
        print('\t4. Divide')
        print('-----------------')
        user_selection = input('Please make a selection: ')
        if user_selection in ('1', '2', '3', '4'):
            input_ok = True
        else:
            print('Invalid Input (must be 1 - 4)')
    print('-----------------')
    return user_selection
```

Work through this function and make sure you are comfortable with all its elements. The '\t' character is a special character denoting a Tab.

We can now update our main calculator loop with this function:

```
finished = False

while not finished:
    result = 0
    menu_choice = get_operation_choice()

    # Get the numbers from the user
    # Select the operation
    print('Result:', result)
    print('==================')
    finished = check_if_user_has_finished(()

print('Bye')
```

13.8 Obtaining the Input Numbers

Next we need to obtain two numbers from the user to use with the selected operation.

In our introduction to Functions in Python chapter we looked at a function (the `get_integer_input()` function) that could be used to take input from the user and convert it (safely) into an integer; if the user entered a non-number then this function would prompt them to enter an actual number. We can reuse the function here.

However, we need to ask the user for two numbers; we will therefore create a function which uses the `get_integer_input()` function to prompt the user for two numbers and then return both numbers. Both functions are shown here:

```
def get_numbers_from_user():
    num1 = get_integer_input('Input the first number: ')
    num2 = get_integer_input('Input the second number: ')
    return num1, num2

def get_integer_input(message):
    value_as_string = input(message)
    while not value_as_string.isnumeric():
        print('The input must be an integer')
        value_as_string = input(message)
    return int(value_as_string)
```

Having one function call another function is very common and indeed we have already been doing this; the `input()` function has been used several times, the only difference here is that we have written the `get_integer_input()` function ourselves.

When we can the `get_numbers_from_user()` function we can store the results returned into two variables; one for each result; for example:

```
n1, n2 = get_numbers_from_user()
```

We can now add this statement to the main calculator loop:

```
finished = False
while not finished:
    result = 0
    menu_choice = get_operation_choice()
    n1, n2 = get_numbers_from_user()
    # Select the operation
    print('Result:', result)
    print('==================')
    finished = check_if_user_has_finished(()

print('Bye')
```

13.9 Determining the Operation to Execute

We are now almost there and can update our main calculation loop with some logic to determine the actual operation to invoke. To do this we will use an `if` statement with the optional `elif` parts. The `if` statement will be conditional on the operation selected and will then call the appropriate function (such as add, subtract etc.) as shown here:

```
if menu_choice == '1':
    result = add(n1, n2)
elif menu_choice == '2':
    result = subtract(n1, n2)
elif menu_choice == '3':
    result - multiply(n1, n2)
elif menu_choice == '4':
    result = divide(n1, n2)
```

Each part of the `if` statement calls a different function; but they all store the value returned into the `result` variable.

We can now add this to the calculation loop to create our fully functional calculator loop:

```
finished = False
while not finished:
    result = 0
    menu_choice = get_operation_choice()
    n1, n2 = get_numbers_from_user()
    if menu_choice == '1':
        result = add(n1, n2)
    elif menu_choice == '2':
        result = subtract(n1, n2)
    elif menu_choice == '3':
        result - multiply(n1, n2)
    elif menu_choice == '4':
        result = divide(n1, n2)
    print('Result:', result)
    print('==================')
    finished = check_if_user_has_finished(()

print('Bye')
```

13.10 Running the Calculator

If you now run the calculator you will be prompted as appropriate for input. You can try and break the calculator by entering characters when numbers are requested, or values out of range for the operations etc. and it should be resilient enough to handle these erroneous inputs, for example:

```
Simple Calculator App
Menu Options are:
  1. Add
  2. Subtract
  3. Multiply
  4. Divide
-----------------
Please make a selection: 5
Invalid Input (must be 1 - 4)
Menu Options are:
  1. Add
  2. Subtract
  3. Multiply
  4. Divide
-----------------
Please make a selection: 1
-----------------
Input the first number: 5
Input the second number: 4
Result: 9
=================
Do you want to finish (y/n): y
Bye
```

13.11 Exercises

For this chapter the exercises relate to extensions to the calculator:

1. Add an option to apply the modulus (`%`) operator to the two numbers input by the user. This will involve defining an appropriate function and adding this as an option to the menu. You will also need to extend the main calculator control loop to handle this option.
2. Add a power of (**) option to the calculator.
3. Modify the program to take floating point numbers instead of simple integers.
4. Allow the choice of division operator or integer division operator (this have both '`/`' and '`//`' available.