**6.3 Modular Programs**

In general in programming, a modular program is one in which the program is broken down into separate tasks, and each task is implemented as a function. This facilitates the top-down design approach of taking a problem, breaking it down into pieces, and implementing each as a function. The program then consists basically of calling the functions to complete each of the tasks.

A basic algorithm for many programs is:

* Get input(s)
* Calculate result(s), using the input(s)
* Print and/or display the result(s)

So, a simple outline of many programs would be to have:

* Function(s) to get the input(s)
* Function(s) to calculate result(s) using the input(s)
* Function(s) to display the result(s)

As an example, we will write a program that will prompt the user for the cost of an item in a store, and then calculate and print the total cost with a tax of 3%. The program will call 3 functions:

* A function to prompt for and return the cost
* A function to calculate the total cost including the tax
* A function to print the total cost

Here are the function definitions.

*def getcost():*

*""" Prompts for item cost. """*

*cost = float(input('Enter the item cost: '))*

*return cost*

*def calctax(cost):*

*""" Calculates and returns cost plus tax. """*

*tax = cost \* .03*

*return cost + tax*

*def printtotcost(totcost):*

*""" Prints cost plus tax. """*

*print(f'The total cost is ${totcost:.2f}')*

The functions could be called from a script, which is frequently called a ***main program***. The program consists of calls to the functions.

*# main program*

*cost = getcost()*

*totalcost = calctax(cost)*

*printtotcost(totalcost)*

The output might look like this:

Enter the item cost: 11.11

The total cost is $11.44

Of course, to be more complete, the *getcost* function would error-check to make sure that the user enters a valid cost.

Sometimes the main program is implemented as a function rather than a script.

**6.3.1 Function Stubs**

When writing a program that consists of a script calling multiple functions, the best practice is not to write the entire program and then execute it to see the results. Frequently, there are errors. When an error is encountered in a particular function, the problem may be in that function, or it might be the result of an incorrect argument passed to the function, that was obtained from another function. A more effective method for constructing the program is to use ***function stubs***, which are place-holders for the actual functions.

The approach is:

* Sketch out the algorithm
* Decide what the functions are going to do
* Write the script, consisting of the calls to the functions (including the arguments that will get passed back and forth)
* Write function stubs for the individual functions
* Change each function stub to the actual function, one at a time

This methodical method of writing a program helps cut down on mistakes, and makes it easier to find bugs when they occur.

Function stubs should mimic what the function is eventually going to do, including using the arguments and returning the correct type(s).

For example, let’s say we are going to write a program that will prompt the user for a temperature in degrees Celsius, convert that to Fahrenheit, and print the results.

We might have functions to:

* Prompt the user for degrees C, error-checking to make sure it’s valid, and return the result
* Receive the degrees C and from that calculate and return degrees F
* Print both degrees C and degrees F in a nice sentence format

The script for the main program might look like this:

*# Convert C to F*

*degC = getdegc()*

*degF = c2f(degC)*

*printCF(degC, degF)*

Example function stubs might look like this:

*def getdegc():*

*“””Prompts user for degrees Celsius.”””*

*return 33*

*def c2f(degreesc):*

*“””Converts degrees C to degrees F”””*

*return degreesc + 5*

*def printCF(degreesc, degreesf):*

*“””Prints degrees C and degrees F”””*

*print(degreesc, degreesf)*

The idea is to execute the program with the function stubs and make sure that values are being passed back and forth correctly. Eventually, after prompting the user and error-checking, the *getdegc* function will return a positive number. So, for now, we just return a positive number. Eventually, once we figure out the conversion, the *c2f* function will convert C to F. For now, we’ll just add 5 so we know that this function is correctly receiving the degrees C and using this number to calculate and return degrees F. Then, the *printCF* function will eventually print in a nice sentence format, but for now it will just print the values.

The headers for the functions should not change. Putting the actual doc strings in the function stubs is a good idea.

Then, once that is working, modify the stubs one at a time. For example, you might start by modifying the *getdegc* function to prompt the user. Then, modify that function to include the error-checking. Going about this systematically really does cut down on errors and makes it easier to find them.