Chapter 4: -Modular Programming

4.1 Chapter objectives

By the end of this topic, learners should be able to:

- Demonstrate understanding what modular programming is
- Explain the benefits associated with modular programming
- Demonstrate understanding on how to define and invoke functions in python
- Understand the difference between an argument and parameters.
- Use of lambda and recursive functions in python

4.2 Introduction

Modular programming is an art of software engineering whereby programmers/software engineers break large and complex programs into manageable portions known as modules. Each of the module will perform a dedicated/specific task and it's upon the programmer to decide when to break the program into these small manageable modules.

Different programming languages refer to these modules using different terms. In C, python, C++, we call these modules functions, In Java, C# we call them methods while in Visual Basic we call them sub procedures/functions. Ideally, each of the module is supposed to perform a specific function hence avoiding code duplication, simplifying program maintenance and debugging as well as promoting program portability

In the previous sections, we have encountered a programming approach where all the code (statements) has been written inside a single. In the next sections we will learn how to program in a modular way by writing additional user defined functions.

4.3 Benefits associated with modular programming

- Less code duplication since each module will perform a specific task.
- It's also easy to maintain the program since the programmer is working with small program segments/module.

- Simplifies debugging since each program logic can be verified separately
- Modules allow you to reusable the same code over and again in different programs with less/little code duplication

4.4 Defining a method in python

In Python a function is defined using the def keyword as shown in the following syntax

```
def my_function(<parameter-list>):
        Declaration of local variables;
        function-body
        return return_value;
```

A method definition in consists of two components; *header and method body.*

- ***function_name:*** This is a unique name of the function which should follow the rules of naming identifiers.
- ***Parameter-list:*** A parameter is like a placeholder. When a function is invoked, you pass a value to the parameter. This value is referred to as actual parameter/argument. Parameters are optional i.e.; a function may contain no parameters. Parameters can be used like local variables within the function body

  The programmer can also declare additional local variables and constants within the method body. The parameters and variables of a function are only accessible within that method.
- ***Function -body:*** these are python line of statements that shall be executed by the function upon being called/invoked
- ***Return return_value:*** this line is optional and its included only when a function is intended to return a value. It's always placed at the end of method. ***Return_value*** denotes the value the function shall return after execution.

Example 1 of method definition

Let's assume you wish to define a method which accepts marks of a student then computes the grade of the grade of the student. The method shall return the grade of the student to the calling block

```
def grading(marks):
    grade = ""
    if marks >= 70 and marks <= 100:
        grade = "A"
    if marks >= 60 and marks < 70:
        grade = "A"
    if marks >= 50 and marks < 60:
        grade = "c"
    if marks >= 40 and marks < 50:
        grade = "D"
    if marks >= 0 and marks < 40:
        grade = "F"
    return grade
```

Example 2: define a method that accepts travel time, travel date and destination then determine if there is a bus that matches the three criteria.

If a bus exists, we should return the status i.e. true or false with true indicating the bus is available and false indicating that there is no bus that matches the three criteria

```
def schedule(time, date,  destination):
    available = False
    if time == "8:00" and date == "01/10/2022" and
destination == "kigali":
        available = True
    elif time == "2:00" and date == "01/10/2022" and
destination == "US":
        available = True
    else:
        available = False
    return available
```

A function is supposed to be defined in a separate independent block; i.e. that means we can't define a method inside another event

4.5 function invocation/call

function definition is completely nothing since by itself it does nothing unless instructed to execute. For a function to execute, it needs to be called/invoked otherwise the interpreter will not be aware of its existence. The general syntax for calling a function in pythonis as follows

<variable-name>=function-name (<argument list>);

- *Variable-name:* This is optional is only included if a function returns a value after execution. The data type of the variable name should be the same as that of the function return data type
- *function-name:-* This refers to the function we are invoking/calling in our program. It should be already defined within the program otherwise the interpreter will generate syntax error

- *Argument-list:* this are the actual values we pass to the method during the point of calling the method. The number of arguments are optional and this depends if the corresponding method has parameters or not

Once the function call is encountered in the program, execution branches to the body of the function definition . Copies of the values of function arguments are stored in the memory locations allocated to the parameters. The statements of the method body are then executed up to the last return statement then control returns to the calling block. Any value returned by the function is stored by to the variable on the left of the assignment operator where the method was invoked. Execution then resumes immediately after the function call.

For example, to call our earlier method for grading, we can use the following statement

```
marks = int (input ("Input student marks\n"))
print ("You scored is:\t" + grading(marks))
```

In the above example, the marks are read by the user from the keyboard using an **input ( )** then converted into integer before being passed to the parameter (*marks*) in a function grading.

The body of the function is executed and a value assigned to the variable grade. The return statement passes this value back to the calling block. The memory allocated to the parameters and local variables is then destroyed to free memory.

The net effect of executing the function in our example is that the variable *grade* has been assigned the value of the corresponding grade depending on the marks we pass to the method.

To call our second method, we can use the following statement

```
time = input("Enter time of travel\n")
date = input("Enter date of travel\n")
destination = input("Enter your destination\n")
status = schedule(time, date, destination)
if status == True:
```

```
    print("A matching bus was found")
else:
    print("A matching bus was not found")
```

4.6 function arguments

Arguments /actual parameters are values you pass to a function parameter when you call the method. The calling code supplies the arguments in parentheses immediately following the function name.

Arguments are specified after the function name, inside the parentheses. You can add as many arguments as you want, just separate them with a comma.

The following example has a function has one argument (yourname). When the function is called, we pass along a name, which is used inside the function to print the full name:

```
def my_name(myName):
  print("Name is:\t" + myName)

my_name("joel")
my_name("jerry")
my_name("cynthia")
```

In most python documentations, arguments are often shortened to args

4.6.1 Number of Arguments

A function must be called with the correct number of arguments. That means if your function expects 1 argument, you have to call the function with1 argument, not more, and not less otherwise your program shall generate errors

```
def myName(fname, onames):
  print(fname + " " + onames)
  myName("Chris", "Edward")
```

This function expects 2 arguments, and gets 2 arguments. If you try to call the function with zero, 1 ,3 e.t.c arguments, you will get an error:

4.6.2 Arbitrary Arguments, *args

If at all you do not know how many arguments that will be passed into your function in advance, add a * before the parameter name in the function definition.

```
def grading(*grades):
  print("Your grade is:\t " + grades[1])
grading("A", "B", "C","D","F")
```

In python documentations, Arbitrary Arguments are often shortened to *args

This way the function will receive a tuple of arguments, and can access the items accordingly

4.6.3 Keyword Arguments

In python, its possible to pass arguments with the key = value syntax. This ensures the order of the arguments does not matter.

```
def myName(fname, onames):
  print("Your first name is" + fname)
myName(onames="Chris", fname="Edward")
```

In python documentations, the Keyword kwargs has been adopted.

4.6.4 Default Parameter Value

We might wish to define a default parameter value when defining our python functions. This can be achieved as follows

```
def myCountry(name,country="Kenya"):
    print("My name is\t" + name + "\tand I am
from\t"+country)

myCountry("Jane","South Sudan")
myCountry("Joel","Tanzania")
myCountry("Jerry")
myCountry("Joe","Burundi")
myCountry("Mary","Dr congo")
```

If we call, if we don't supply the country in the list of arguments, it uses the default value of kenya

4.6.5 Passing a List as an Argument

We can send any data types of argument to a function (string, number, list, dictionary etc.), and it will be treated as the same data type inside the function body. If we send a dictionary, it shall be when it reaches the function.

```
def myCars(cars):
  for j in cars:
    print(j)

cars= ["volvo", "Benz", "Prado","Audi"]

myCars(cars)
```

4.7 Parameters

A parameter/formal parameter represents a value that a function expects you to pass when you call it and it's defined during method declaration. The name of each parameter serves as a local variable within the method and can be used the same way you use any other variable.

1. Each argument must correspond to the parameter in the same position in the list.
2. The number of arguments must be the same as the number of parameters.
3. Parameters and arguments must be of the same data type
4. Parameters and may bear different names.
5. Method arguments can be constants and mathematical expressions.

4.8 Functions that return values

If a function returns a value, we use the return statement at the end of the function body. Examples of such functions are found in section 4.5 of this chapter. You can make reference to the two examples.

4.9 The pass statements

Typically, function definitions cannot be empty, but if for some reason you have a function definition with no content, put in the pass statement to avoid getting an error. An example is shown below

```
def emptyFunction():
  pass
```

4.10 Lambda function

A lambda function is a small anonymous function. We refer to it as anonymous since we do declare it like other usual functions, using the def keyword.

Lambda expressions can accept an unlimited number of arguments; however, they only return one value.

They can't have numerous expressions or instructions in them.

Lambda Syntax

```
lambda [argument1 [,argument2... .argumentn]] :
expression
```

Below is a lambda function used for getting the sum of two numbers

```
# Defining a function
x = lambda a, b: a + b;

# Calling the function and passing values
print("Value is : ", x(10, 40))
print("Value is : ", x(19, 4))
```

The following example displays the sum of three numbers using a lambda function

```
# Defining a function
x = lambda a, b, c: a* c * b;
# Calling the function and passing values
print("The product of three numbers is : ", x(3, 5,7))
```

4.11 Recursion Functions

Python accepts function recursion, which means a defined function can call itself. Recursive functions are very common in mathematical and programming concept.

The developer however needs to be very careful since he can end up writing a function which never terminates, or one that uses excess amounts of memory or processor power.

However, if such programs are written with the correct precise, they remain very efficient and mathematically elegant approach to programming.

Example, using recursive function, write a python program that finds the factorial of a number ***n***

4.12 Chapter summary

4.13 Chapter exercise

4.14 Further reading suggestions