Chapter 3:
Modules

Starting Out with Programming Logic & Design

Second Edition

by Tony Gaddis

Addison-Wesley
is an imprint of

PEARSON

Chapter Topics

3.1 Introduction

3.2 Defining and Calling a Module

3.3 Local Variables

3.4 Passing Arguments to Modules

3.5 Global Variables and Global Constants

3.1 Introduction

- A module is a group of statements that exists within a program for the purpose of performing a specific task.
- Most programs are large enough to be broken down into several subtasks.
- Divide and conquer: It’s easier to tackle smaller tasks individually.

3.1 Introduction

5 benefits of using modules

- Simpler code
  - Small modules easier to read than one large one
- Code reuse
  - Can call modules many times
- Better testing
  - Test separate and isolate then fix errors
- Faster development
  - Reuse common tasks
- Easier facilitation of teamwork
  - Share the workload

3.2 Defining and Calling a Module

- The code for a module is known as a module definition.

```
Module showMessage()
   Display "Hello world."
End Module
```

- To execute the module, you write a statement that calls it.

```
Call showMessage()
```

3.2 Defining and Calling a Module

- A module's name should be descriptive enough so that anyone reading the code can guess what the module does.
- No spaces in a module name.
- No punctuation.
- Cannot begin with a number.

3.2 Defining and Calling a Module

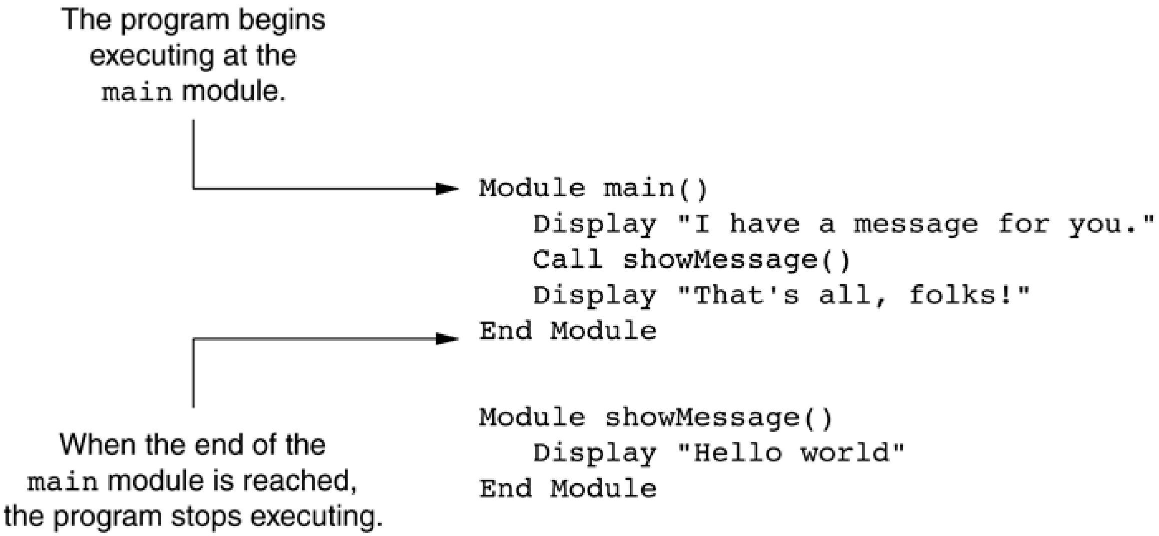
- Definition contains two parts
  - A header
    - The starting point of the module
  - A body
    - The statements within a module

```
Module name( )
     Statement
     Statement
     Etc.
End Module
```

3.2 Defining and Calling a Module

- A call must be made to the module in order for the statements in the body to execute.

Figure 3-2 The `main` module



The program begins executing at the main module.

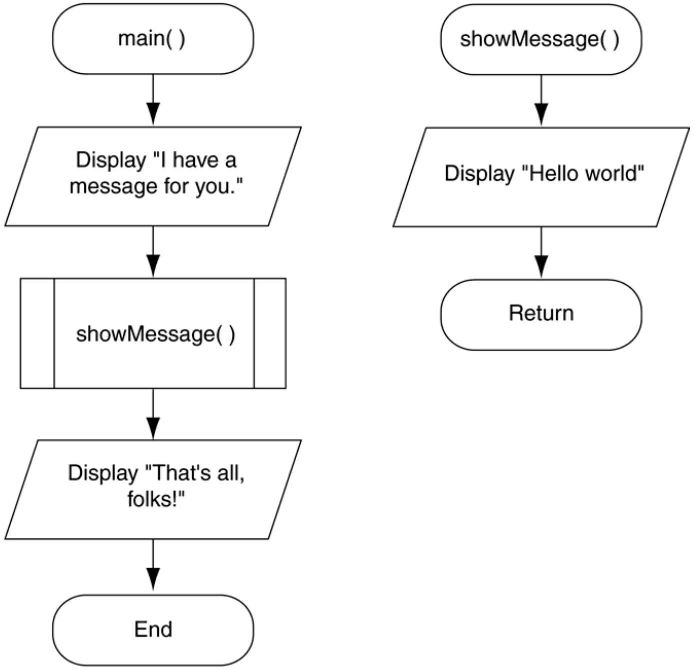
```
Module main()
   Display "I have a message for you."
   Call showMessage()
   Display "That's all, folks!"
End Module

Module showMessage()
   Display "Hello world"
End Module
```

When the end of the main module is reached, the program stops executing.

3.2 Defining and Calling a Module

- When flowcharting a program with modules, each module is drawn separately.

Figure 3-6 Flowchart for Program 3-1



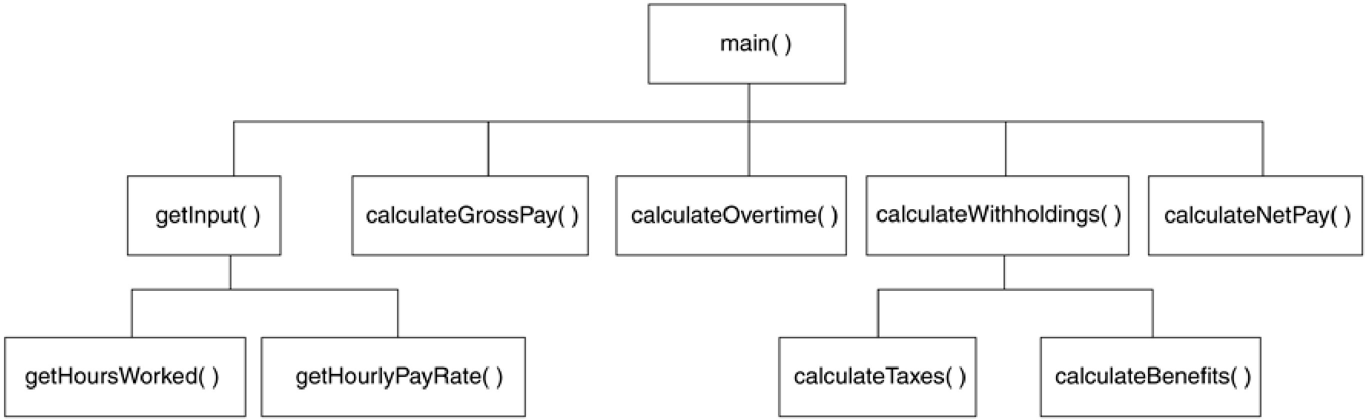
3.2 Defining and Calling a Module

- A top-down design is used to break down an algorithm into modules by the following steps:
  - The overall task is broken down into a series of subtasks.
  - Each of the subtasks is repeatedly examined to determine if it can be further broken down.
  - Each subtask is coded.

3.2 Defining and Calling a Module

- A hierarchy chart gives a visual representation of the relationship between modules.
- The details of the program are excluded.

Figure 3-7 A hierarchy chart



3.3 Local Variables

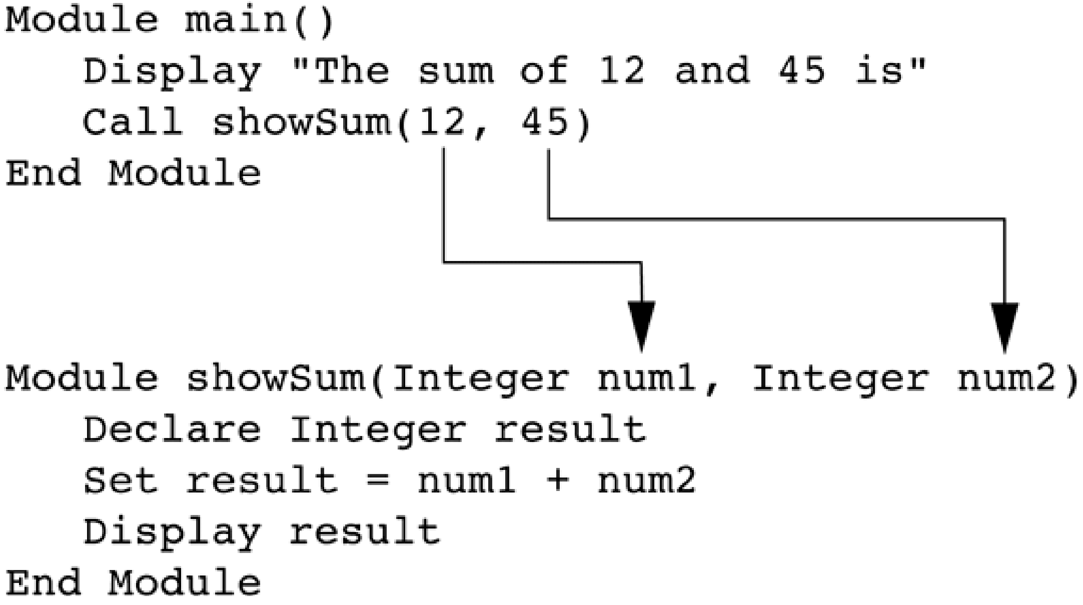
- A **local variable** is declared inside a module and cannot be accessed by statements that are outside the module.
- **Scope** describes the part of the program in which a variable can be accessed.
- Variables with the same scope must have different names.

3.4 Passing Arguments to Modules

- Sometimes, one or more pieces of data need to be sent to a module.
- An **argument** is any piece of data that is passed into a module when the module is called.
- A **parameter** is a variable that receives an argument that is passed into a module.
- The argument and the receiving parameter variable must be of the same data type.
- Multiple arguments can be passed sequentially into a **parameter list**.

3.4 Passing Arguments to Modules

Figure 3-14 Two arguments passed into two parameters



```
Module main()
   Display "The sum of 12 and 45 is"
   Call showSum(12, 45)
End Module


Module showSum(Integer num1, Integer num2)
   Declare Integer result
   Set result = num1 + num2
   Display result
End Module
```

3.4 Passing Arguments to Modules

Pass by Value vs. Pass by Reference

- Pass by **Value** means that only a copy of the argument's value is passed into the module.
  - One-directional communication: Calling module can only communicate with the called module.
- Pass by **Reference** means that the argument is passed into a reference variable.
  - Two-way communication: Calling module can communicate with called module; and called module can modify the value of the argument.

3.5 Global Variables & Global Constants

- A **global variable** is accessible to all modules.
- Should be avoided because:
  - They make debugging difficult
  - Making the module dependent on global variables makes it hard to reuse module in other programs
  - They make a program hard to understand

3.5 Global Variables & Global Constants

- A **global constant** is a named constant that is available to every module in the program.
- Since a program cannot modify the value of a constant, these are safer than global variables.