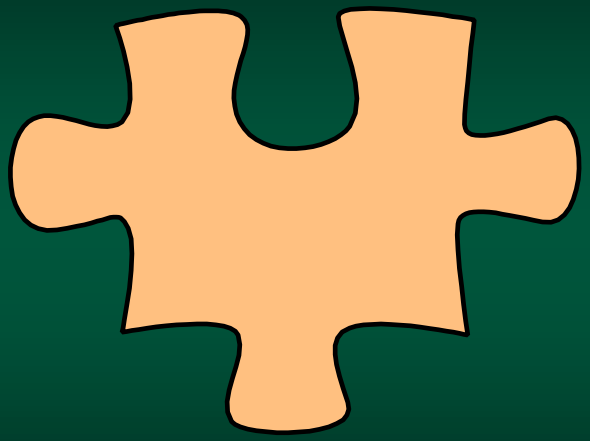
Chapter 3

Modules

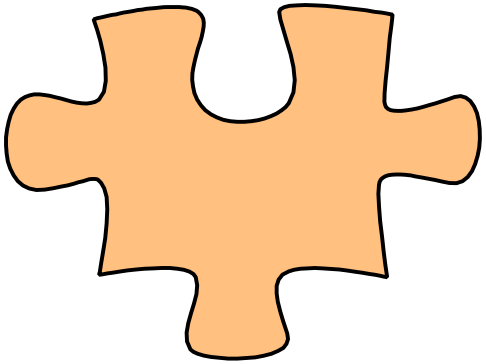


Introduction to Modules

Chapter 3.1

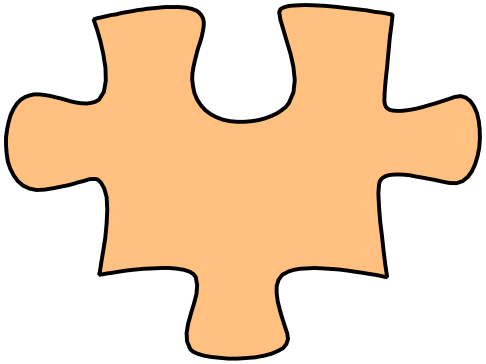
Introduction to Modules

- 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



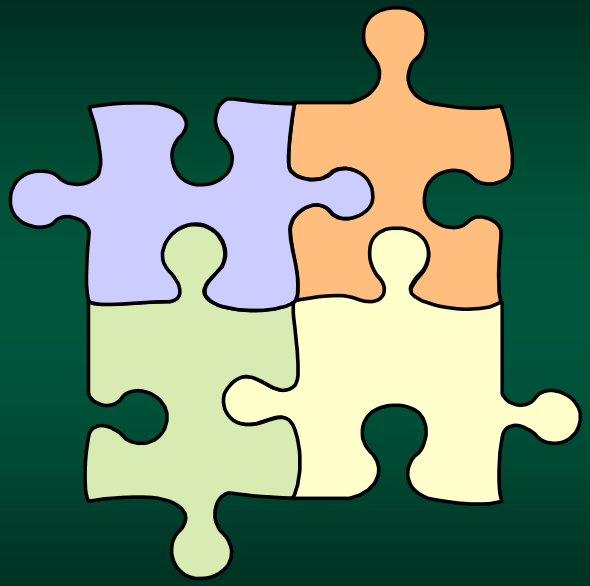
Introduction to Modules

- This approach is used by professional developers is part of modern program design



5 Benefits of Modules

1. Simpler code
2. Code reuse
3. Better testing
4. Faster development
5. Easier facilitation of teamwork

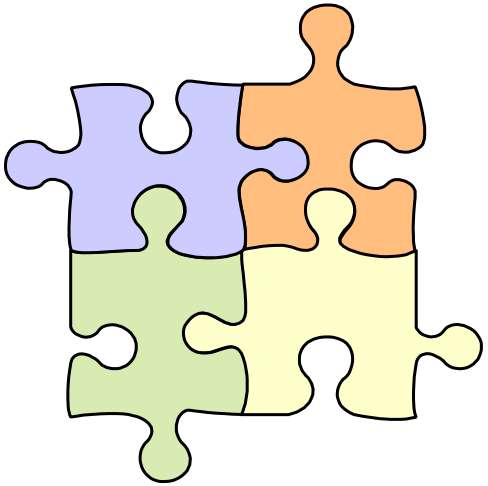


Defining and Calling a Module

Chapter 3.2

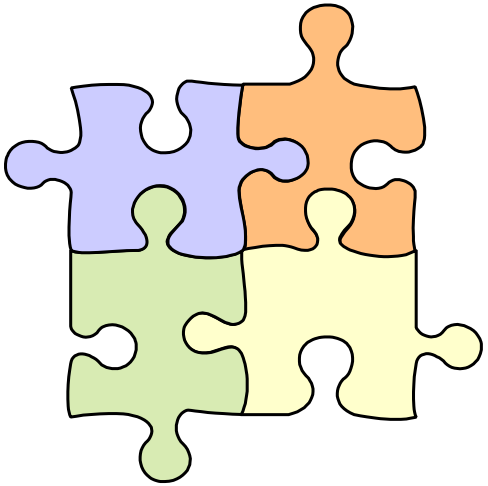
Defining and Calling a Module

- After a module is written, it can be used from anywhere in the program
- Modules can be used multiple times – whenever needed



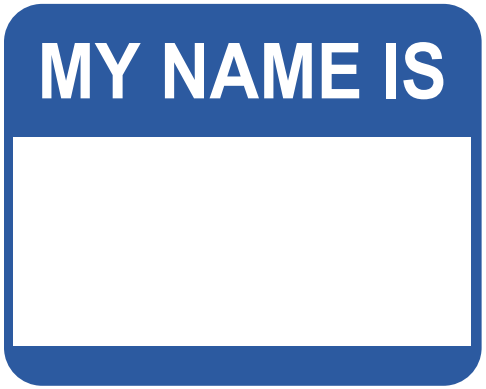
Defining and Calling a Module

- When a module is *called*, execution is transferred to the module – which then runs
- Once the module completes, execution returns to the line after the call



Naming a Module

- A module's name should be descriptive so that a reader can guess what it does
- Names must follow the same rules as variable identifiers
  - no spaces in a module name.
  - no punctuation.
  - cannot begin with a number.



Defining a Module

- A *module definition* contains the code
- Everything between "Module" and "End Module" is the code related to the module

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

Header and Body

- Definition contains two parts
- Header
  - the starting point of the module
  - contains the name (and more later)
- Body
  - the statements within a module
  - these are executed whenever you call the module

Header and Body

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

Header

Body

Calling a Module

- To execute the module, you write a statement that calls it
- The book uses the keyword "Call" followed by the name of the module

```
Call showMessage()
```

Example Program

```
Module Main
    Call hello
    Display "Bye"
End Main

Module hello
    Display "Hello"
End Module
```

We start in Main

Example Program

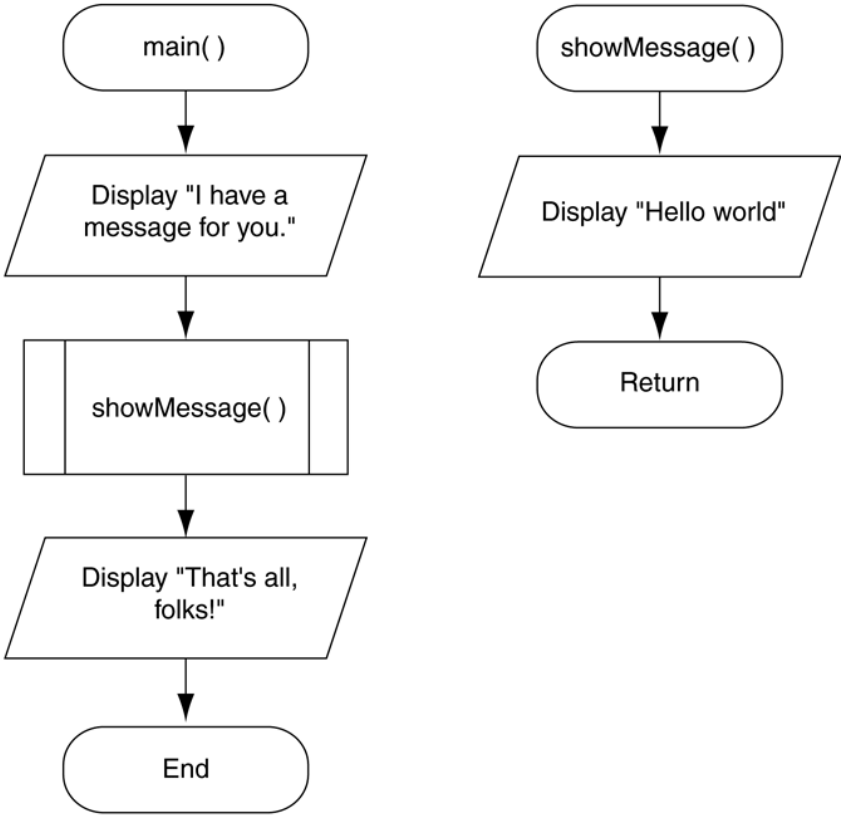
```
Hello
Bye
```

Displayed by hello()

Displayed after hello() finished

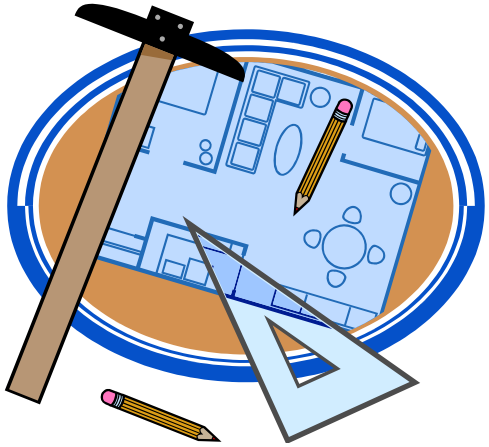
Modules in Flowcharts

- When flowcharting a program with modules, each module is drawn separately
- Calls are denoted by a rectangle with two vertical bars



Top-Down Design

- *Top-down design* is used to break down an algorithm into modules
- The idea is that the same code shouldn't be written multiple times

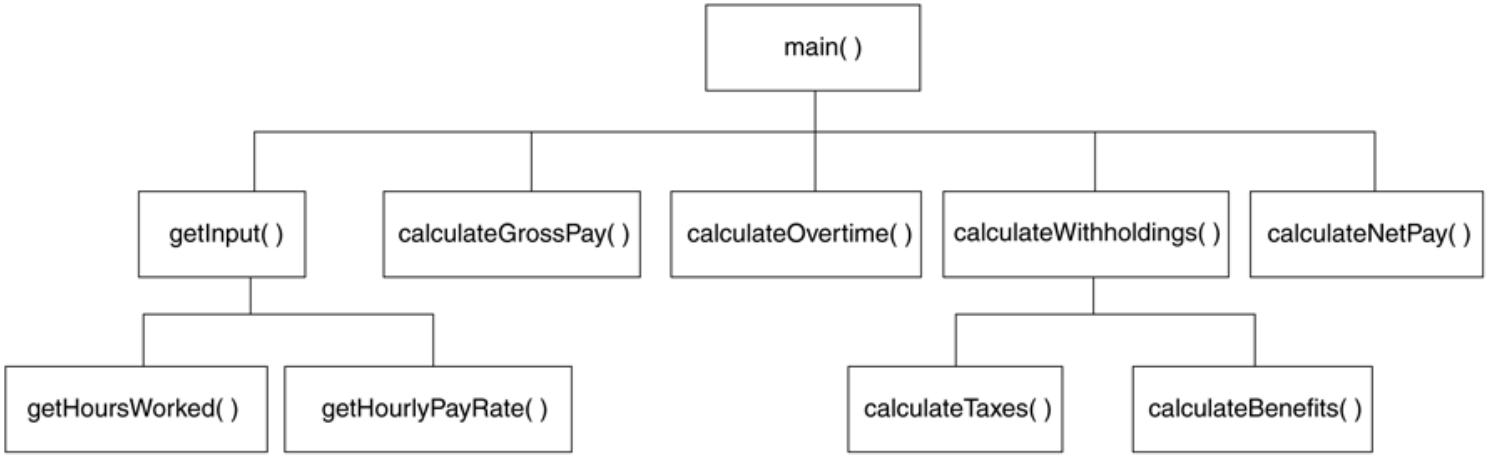


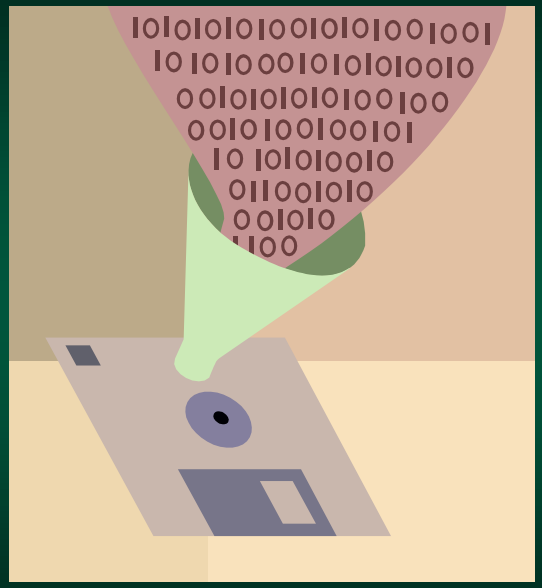
Steps of Top-Down Design

1. The overall task is broken down into a series of subtasks
2. Each of the subtasks is repeatedly examined to determine if it can be further broken down
3. Each subtask is coded

Hierarchy Chart

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



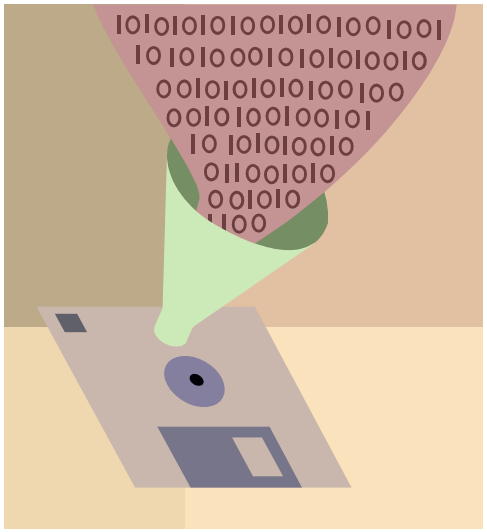


Local Variables

Chapter 3.3

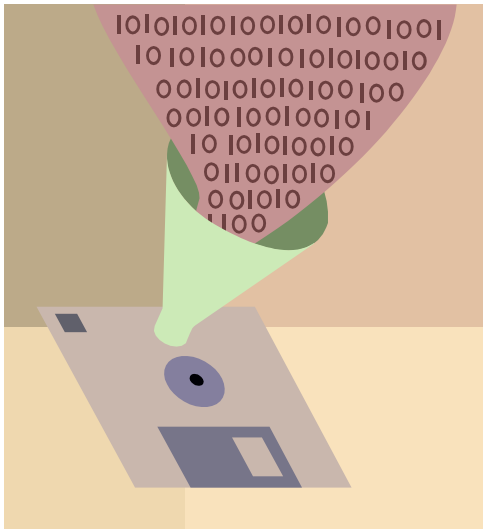
Local Variables

- *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.



Local Variables

- Variables with the same scope must have different names.
- Otherwise, how does the program know which one to use?



Example Program

```
Module sayHello()
    Declare String name

    Display "Enter your name: "
    Input name
    Display "Hello ", name
End Module
```

Local variable

Example Program

```
Module Main()
    Call sayHello()
    Call sayHello()
End Module
```

We can it call twice!

Example Program

```
Enter your name
"Herky"
Hello Herky
Enter your name
"Moe Howard"
Hello Moe Howard
```

Passing Arguments to Modules

Chapter 3.4

Passing Arguments to Modules

- So far, the modules have been incredibly simple
- However, sometimes, one or more pieces of data need to be sent to a module

Passing Arguments to Modules

- 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.

Passing Rules

- You can have multiple arguments
- Multiple arguments can be passed sequentially into a parameter list

Some Terminology

- The argument and the receiving parameter variable *must have same data type*
- Basically
  - arguments are *passed* to the parameters
  - they match, in order, on a one-to-one basis
  - arguments ➔ parameters

Example

```
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
```

Parameter Example

```
Module main()
    Declare String name

    Input name
    Call printHello(name)
End Main

Module printHello(String n)
    Display "Hello", n
End Module
```

n is a local variable
It is set to the value
of name.

Parameter Example

```
Moe
```

User typed this

```
Hello Moe
```

Types of Passing

Chapter 3.4

Types of Passing

- There are a number of different was of pass values into a module
- In most programming languages, there are two ways: *pass by value* and *pass by reference*

Pass by Value

- *Pass by Value* means that only a copy of the argument's value is passed into the module
- Think of it as "one directional communication" between the caller and module

Pass by Value

- Since it is just a copy (not the original) you can set the value, but the source will not be changed
- The argument is really a local variable that is automatically Set for you

Pass By Value

```
Module main()
    Declare Integer x

    Set x = 42
    Call changeIt(x)
    Display "Result is ", x
End Main

Module changeIt(Integer value)
    Set value = 100
End Module
```

'Value' is a copy. It is Set to the argument

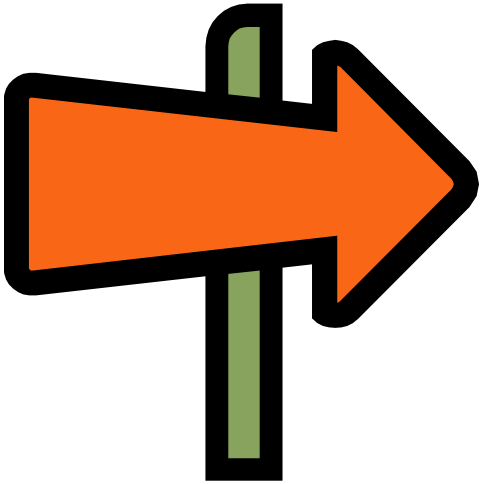
Pass By Value

```
Result is 42
```

The changeIt module only changed the local variable (by value). The x variable wasn't changed.

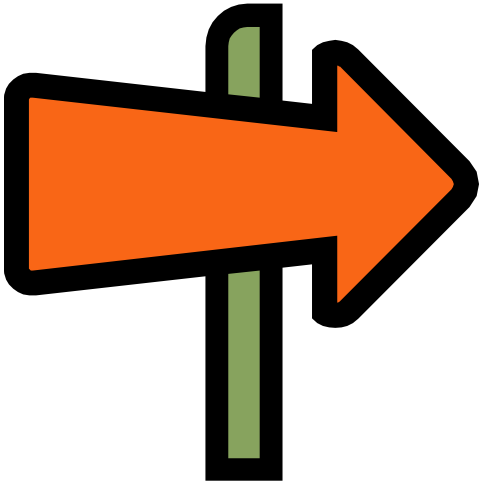
Pass by Reference

- *Pass by Reference* means that the argument itself is passed rather than just its value
- Think of it as "bi-directional communication" between the caller and module



Pass by Reference

- Module's parameter variable is an *alias* of the caller variable
- So, they are the same thing – but with different names
- The called module can modify the value of the argument



Pass By Value

```
Module main()
    Declare Integer x

    Set x = 42
    Call changeIt(x)
    Display "Result is ", x
End Main

Module changeIt(Integer ref value)
    Set value = 100
End Module
```

By reference means is the same variable with a different name.

Pass By Value

```
Result is 100
```

The module updated the variable x. It was called 'Value' in the module.

Global Variables & Constants

Chapter 3.5

Global Variables & Constants

- Some variables and constants can be defined so every module can use them
- As "global", they are basically shared

Global Variables

- 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

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.

Global Constant Example

```
Constant Real PI = 3.14

Module main()
    ...
End Module
```

Global Variable Example

```
Declare String value

Module main()
    Call readValue()
    Display "Hello ", value
End Module

Module readValue()
    Input value
End Module
```

Outputted

Read here

Parameter Example

```
Moe
```

User typed this

```
Hello Moe
```