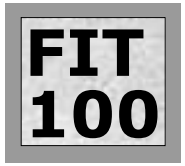
Programming Basics



To specify algorithms, we must be precise. To be precise, we need a language that is more exact than English. A programming language offers this advantage. All programming languages have a basic set of features.

FIT 100

Recapping *Alphabetize CDs*

- *Alphabetize CDs* illustrates an intuitively understandable process not involving a computer
- The *Alphabetize CDs* program demonstrated several features of algorithms and programs …
  - The program illustrated the 5 properties of algorithms -- input and output specs, definiteness, effectiveness, finiteness
  - In order to reference the different slots, we used two "pointers" called *Alpha* and *Bet*
  - *Alpha* referenced all slots but the last, and for each slot *Alpha* referenced, *Bet* referenced each slot to its right
  - Can you "visualize" *Alphabetize CDs'* processing strategy?

Alphabetize CDs illustrates nearly all of the programming concepts to be covered in FIT100, but it did so in English

FIT 100

An Approach To Programming

- Though Alphabetize CDs was precise enough for a person to execute successfully, computers demand greater precision from programs
- The plan …
  - Adopt a better notation than English to express algorithms
    - General ideas are given in lecture
    - VB6 will be used in lecture and lab
  - Discuss standard ways of using a programming language
  - Practice the ideas by writing programs
  - Add a few more language features and describe their use
  - Practice with a few more programs

FIT 100

Variables

- In normal language, names are (usually) tightly fixed to their values --
  - "penny" means 1 cent … it doesn't change its meaning, and sometimes refer to $8.41 or a time zone or an action
- In computing names must change values
  - Example: *Alpha* and *Bet* in *Alphabetize CDs* changed
  - Names must change values in a program because programs specify a *transformation* of input into output … as the transformation proceeds the things named change values
- *variable* is term for program names that change value

Variables are analogous to titles in normal language since titles are expected to change values: president, mayor, James Bond

FIT 100 On Variable Names

- The term “variable” reminds us the value changes
- The names used for variables are arbitrary provided
  - Variable names must begin with a letter
  - Variable names can contain any letter, numeral or _
  - Variable names should be meaningful and accurate
    - total, averageOverClass, etc, but not x, o0OO0o, etc
  - Most languages are *case sensitive*: a ≠ A

Convention: In all programming for FIT100, variables should start with lowercase letters so as to avoid confusion with other names in VB6 … *ignore this convention at your peril*

FIT 100 On Variable Values

- A variable can be thought of as a “named container”

- Variables name computer memory locations, so the value of a variable is the quantity stored in its memory
- Variables can take on different *types* of values
  - Whole numbers or *integers*: **`2, -9, 1048576`**
  - Character sequences or *strings*: **`“2”, “&^%$#@”, “ ”`**
  - Decimal numbers or *doubles*: **`2.0, 3.14159, -999.99`**
- A variable’s values have a specific type
- Variables are *declared* and their type is specified
  - **`Dim averageOverClass As Double`**

FIT 100 Assignment

- Computers must be told what value to assign to variables, using an *assignment statement* such as

```
averageOverClass = 21.14
mayor = "Paul Schell"
```

- The general form of an assignment statement is

  <*variable name*> <*assignment symbol*> <*expression*>

  - Languages use different assignment symbols: **`= := ←`**
  - Read assignment as "is assigned", or "becomes" or "gets"
  - All three components must always be present

- Fundamental property of assignment

  The "flow" of information is always right-to-left

  - **`destination = source`**
  - **`changedVariable = value`**

Meta-brackets < > enclose language defining terms

FIT 100

Expressions

- Expressions are formulae made from variables and operators, e.g. calculator operations: +, -, *, /, ^
  - **`weeks = days / 7`** *divide value of days by 7*
  - **`grossPay = hours * rate`** *multiply the two values*
  - **`area = pi * radius ^ 2`** *π times radius squared*
- Fundamental rule of assignment

  The expression is evaluated before the assignment is made
  - **`score = score + 3`**
  - **`shotClock = shotClock - 1`**

Computing is NOT *algebra*: Though = is used in assignment statements, it means "becomes" whereas in algebra it means equality. So, **`score = score + 3`** is essential to computing, but meaningless in algebra

FIT 100 Operators

- Most programming languages have more operators than a pocket calculator
  - Operators like **+** taking 2 operands are called *binary*: **`a + b`**
  - Operators like **-** taking 1 operand are called *unary*: **`- a`**
- A very useful operator is *concatenate*, & in VB6, which connects two strings together:
  - **`plural = “dog” & “s”`**
- The relational operators are:
  - a < b less than a > b greater than
  - a <= b less than or equal to a >= b greater than or equal
  - a = b equal to a <> b not equal

FIT 100

Conditionals

- Programs must frequently test if some condition holds, e.g. are two CDs in alphabetical order
- Conditional statements have been invented to make tests
  - **`If temp < 32 Then waterState = “frozen”`**
- General form of basic conditional:

  `If <`*`T/F expression`*`> Then <`*`assignment statement`*`>`
- The meaning is that the <*T/F expression*> is *evaluated*
  - If the outcome is true, then the assignment statement is performed
  - If the outcome is false, then the assignment statement is skipped

FIT 100

More Complex Conditionals

- The basic conditional is too limited, so generalize it
- General form of an If-statement

```
If <T/F expression> Then
   <statement list>
End If
```

List terminator, one word

- Example:

```
If temp >= 212 Then
   state = “gaseous”
   form = “steam”
End If
```

FIT 100 General Conditional Statement

- When operations must be performed for the true outcome and different operations are need for a false outcome, use the If-Then-Else statement
- General form

```
If <T/F expression> Then
   <statement list>
Else
   <statement list>
End If
```

```
If sideUp = sideCalled Then
   coinTossWinner = hostTeam
   firstHalfOffense = hostTeam
   secondHalfOffense = visitorTeam
Else
   coinTossWinner = visitorTeam
   firstHalfOffense = visitorTeam
   secondHalfOffene = hostTeam
End If
```

FIT 100

Example of If-Then-Else

- An advantage of the general conditional is that the statement lists can contain other conditionals

```
If flip1 = guess1 Then
   If flip2 = guess2 Then
      score = “win win”
   Else
      score = “win lose”
   End If
Else
   If flip2 = guess2 Then
      score = “lose win”
   Else
      score = “lose lose”
   End If
End If
```