Data Types

- In computers, data such as integers, real numbers, character, strings, and Boolean are represented and stored in memory differently.
- Computer languages provide appropriate different data types according to the size of storage.
- There are four fundamental data types:
  - Integral types
  - Floating-point numbers
  - Characters
  - String
  - Boolean values

Integral Types

- An integer type has a value as a whole number either negative or positive, e.g., 123, +123, −123, 22333
- Visual Basic .NET has different integral types depending on the size of the storage. For examples,
  - Byte -- 1 byte
  - Short -- 2 bytes
  - Integer -- 4 bytes
  - Long -- 8 bytes
Floating-Point Numbers

- A floating number or a real number is any signed or unsigned number having an integer part and a fractional part, with a decimal point in between e.g., 18.0, +9.2, 35.25, 0.57, −138.2, 4., .8
- Visual Basic supports two different categories of floating point numbers:
  - **Single** -- Numbers which are stored using the single data type are called single precision numbers.
  - **Double** -- Numbers that are stored using the double data type are called double precision numbers.
- A double precision number uses twice the amount of storage as that used by a single precision number.

Exponential Notation

- Exponential notation is commonly used to express both very large and very small floating point numbers in a compact form, e.g., 1.74536E−12, 3.652442E4, 7E20
- It is similar to scientific notation.

<table>
<thead>
<tr>
<th>Decimal Notation</th>
<th>Exponential Notation</th>
<th>Scientific Notation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1234</td>
<td>1.234E3</td>
<td>1.234 × 10^3</td>
</tr>
<tr>
<td>123456</td>
<td>1.23456E5</td>
<td>1.23456 × 10^5</td>
</tr>
<tr>
<td>.01234</td>
<td>1.234E −2</td>
<td>1.234 × 10^−2</td>
</tr>
<tr>
<td>.0001234</td>
<td>1.234E −4</td>
<td>1.234 × 10^−4</td>
</tr>
</tbody>
</table>

Strings

- A *string* consists of one or more characters that are enclosed within double quotes.
- The number of characters within a string determines the length of the string. For examples, “apple”, “S”, “45”, “Joe.Nobody” and “HELLO”.
- String *concatenation* means the joining of two or more strings into a single string.
- Visual Basic provides two symbols, the ampersand ( &) and the plus sign (+), for performing string concatenation. “Joe” & “C” & “Nobody” ⇒ “Joe.C.Nobody”
- Numeric values and/or strings can be displayed using either a call to the *MessageBox.Show()* method or a text box.

ASCII Code

```
|   | 0000000 | 0000001 | 0000010 | 0000011 | 0000100 | 0000101 | 0000110 | 0000111 | 0001000 | 0001001 | 0001010 | 0001011 | 0001100 | 0001101 | 0001110 | 0001111 | 0010000 | 0010001 | 0010010 | 0010011 | 0010100 | 0010101 | 0010110 | 0010111 | 0011000 | 0011001 | 0011010 | 0011011 | 0011100 | 0011101 | 0011110 | 0011111 | 0100000 | 0100001 | 0100010 | 0100011 | 0100100 | 0100101 | 0100110 | 0100111 | 0101000 | 0101001 | 0101010 | 0101011 | 0101100 | 0101101 | 0101110 | 0101111 |
|---|---------|---------|---------|---------|---------|---------|---------|---------|---------|---------|---------|---------|---------|---------|---------|---------|---------|---------|---------|---------|---------|---------|---------|---------|---------|---------|---------|---------|---------|---------|---------|---------|---------|---------|---------|---------|---------|---------|---------|---------|---------|---------|---------|---------|
| A | 0000000 | N       | 0001110 |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |
| B | 0000010 | O       | 0011111 |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |
| C | 0000011 | P       | 0100000 |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |
| D | 0000100 | Q       | 0100001 |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |
| E | 0000101 | R       | 0100010 |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |
| F | 0000110 | S       | 0100011 |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |
| G | 0000111 | T       | 0101000 |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |
| H | 0001000 | U       | 0101001 |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |
| I | 0001001 | V       | 0101010 |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |
| J | 0001010 | W       | 0101011 |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |
| K | 0001011 | X       | 0101100 |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |
| L | 0001100 | Y       | 0101101 |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |
| M | 0001101 | Z       | 0101101 |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |         |
```
Boolean

- There are only two Boolean data values in Visual Basic.
- These values are the constants: **True** and **False**.
- The Boolean constants are used in decision-making statements.

### Fundamental VB .NET Data Types

<table>
<thead>
<tr>
<th>Type</th>
<th>Size in Bytes</th>
<th>Description</th>
<th>Range of Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Byte</td>
<td>1</td>
<td>8-bit unsigned integer</td>
<td>0 - 255</td>
</tr>
<tr>
<td>Integer</td>
<td>4</td>
<td>32-bit signed integer</td>
<td>-2147483648 - 2147483647</td>
</tr>
<tr>
<td>Long</td>
<td>8</td>
<td>64-bit signed integer</td>
<td>-9223372036854775808 - 9223372036854775807</td>
</tr>
<tr>
<td>Short</td>
<td>2</td>
<td>16-bit signed integer</td>
<td>-32,768 - 32,767</td>
</tr>
<tr>
<td>Single</td>
<td>4</td>
<td>32-bit floating point variable</td>
<td>-3.4E38 - 3.4E+38</td>
</tr>
<tr>
<td>Double</td>
<td>8</td>
<td>64-bit floating point variable</td>
<td>-1.7E308 - 1.7E+308</td>
</tr>
<tr>
<td>Decimal</td>
<td>16</td>
<td>128-bit floating point variables</td>
<td>-</td>
</tr>
<tr>
<td>Char</td>
<td>2</td>
<td>16-bit Unicode characters</td>
<td>0 - 65,535</td>
</tr>
<tr>
<td>String</td>
<td>Varies</td>
<td>Non-Numeric Type</td>
<td></td>
</tr>
<tr>
<td>Boolean</td>
<td>1</td>
<td>Non-Numeric Type</td>
<td>False or True</td>
</tr>
<tr>
<td>Data</td>
<td>8</td>
<td>Non-Numeric Type</td>
<td>00:00:00 on January 1, 0001 – 11:59:59 on December 31, 9999</td>
</tr>
<tr>
<td>Object</td>
<td>4</td>
<td>Non-Numeric Type</td>
<td>References to an object</td>
</tr>
</tbody>
</table>

### Arithmetic Operations

- Visual Basic provides a number of numeric operators, which can be used to perform operations such as additions, subtractions, multiplications, etc.
- A simple numeric expression is an expression that has a numeric operator connecting two arithmetic operands. This type of expression has the syntax:
  - `operand operator operand`
- A binary operator is an operator that requires two operands.
- A unary operator is an operator that requires only one operand.
### Precedence of Arithmetic Operations

<table>
<thead>
<tr>
<th>Operator</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>^</td>
<td>Exponentiation</td>
</tr>
<tr>
<td>−</td>
<td>Negation</td>
</tr>
<tr>
<td>* /</td>
<td>Multiplication and Division</td>
</tr>
<tr>
<td>\</td>
<td>Integer Division</td>
</tr>
<tr>
<td>Mod</td>
<td>Modulus -- return remainder</td>
</tr>
<tr>
<td>+</td>
<td>Addition and Subtraction</td>
</tr>
</tbody>
</table>

### Rules of Forming Expressions

The followings are rules when writing expressions in Visual Basic:

- Two binary operators must never be placed adjacent to one another.
  
  \[20 \times 20 \times -40\] is invalid.

- Parentheses may be used to form groupings, and all expressions enclosed within parentheses are evaluated first.
  
  \[(10 + 20) (30)\] is invalid.

- Parentheses cannot be used to indicate multiplication.
  
  \[(10 + 20) (30)\] is invalid.

### Expression Types

- **Integer expression** -- is an arithmetic expression containing only integers.

- **Floating point expression** -- is an arithmetic expression containing only floating point numbers.

- **Mixed-mode expression** -- is an arithmetic expression containing both integers and floating point numbers.

### Types of Arithmetic Expressions

- combining floating point numbers with either integers or floating point numbers
  
  \[\Rightarrow\] floating point

- combining integers or floating point numbers with \n  
  \[\Rightarrow\] integer

- combining integers with either +, −, *, \, Mod
  
  \[\Rightarrow\] integer
# Arithmetic Expression

<table>
<thead>
<tr>
<th>Expression</th>
<th>Result Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>11 + 4 - 27</td>
<td>Integer</td>
</tr>
<tr>
<td>11 / 4.6 * 0.879</td>
<td>Floating Point</td>
</tr>
<tr>
<td>11 / (4.6 * 0.879)</td>
<td>Floating Point</td>
</tr>
<tr>
<td>11.36 + 4.6 * 0.879</td>
<td>Floating Point</td>
</tr>
</tbody>
</table>

# Integer Division Operator

VB .NET uses back slash (\) as integer division operator.

<table>
<thead>
<tr>
<th>Expression</th>
<th>Actual Value</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>11\4</td>
<td>11\4 = 2.75</td>
<td>2</td>
</tr>
<tr>
<td>11\4.6</td>
<td>11\5 = 2.20</td>
<td>2</td>
</tr>
<tr>
<td>11.36\4</td>
<td>11\4 = 2.75</td>
<td>2</td>
</tr>
<tr>
<td>11.36\4.6</td>
<td>11\5 = 2.20</td>
<td>2</td>
</tr>
</tbody>
</table>

# Mod Operator

The modulus operator Mod yields the remainder of the result of dividing its first operand by its second.

<table>
<thead>
<tr>
<th>Expression</th>
<th>Actual Value</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>11 Mod 4</td>
<td>11 + 4 = 2 r 3</td>
<td>3</td>
</tr>
<tr>
<td>20 Mod 4</td>
<td>20 + 4 = 4 r 0</td>
<td>0</td>
</tr>
<tr>
<td>11 Mod 4.6</td>
<td>11 + 4.6 = 2 r 1.8</td>
<td>1.8</td>
</tr>
<tr>
<td>11.36 Mod 4.6</td>
<td>11.36 + 4.6 = 2 r 2.16</td>
<td>2.16</td>
</tr>
</tbody>
</table>

# Variables

- A variable is a name given by the programmer to a memory storage location.
- The value stored in a variable can be changed and referenced.
- The rules that the programmer must follow when selecting a variable name are:
  - The name must begin with a letter or an underscore.
  - The name can only consist of letters, numeric digits, or underscores.
  - The name cannot exceed 16,383 characters.
  - The name cannot be a Visual Basic keyword, such as Integer.
- A variable name should be chosen to indicate what it represents -- Width, Length, Area, Total_Sum, EmployeeIncome, Interest_Rate.
Declaration Statements

- A declaration statement is a step used for naming a variable and specifying the data type that can be stored in it.
- Declaration statements placed within a procedure are non-executable statements that have the general syntax:
  ```
  Dim variable-name As data-type
  ```
  where
  - `data-type` designates a Visual Basic data types and
  - `variable-name` is a variable name selected by the programmer.

Examples

```vbnet
Dim Number1 As Integer
Dim Wages As Single
Dim LastName As String
Dim MiddleInitial As Char
Dim Empty As Boolean
```

Single-line Declarations

Visual Basic allows combining multiple declarations into one statement, using the syntax:

```vbnet
Dim var1 As data-type, var2 As data-type, ..., varn As data-type
```

Examples:

```vbnet
Dim FirstName As String, LastName As data-type,
    MiddleInitial As Char, Test1 As Integer, Test2 As Integer
Dim Test1, Test2, Test3, FinalExam As Integer,
    Average As Single
```

Initialization

- An initialization is the process of storing a value in a variable for the first time.
- In Visual Basic, by default, all numeric variables are initialized to zero and all strings are initialized to zero-length strings (strings containing no characters).
- Declaration statements perform both a software and a hardware function.
  - From a software perspective, declaration statements provide a convenient, up-front list of all variables and their data types.
  - The hardware function that declaration statements perform is that they inform Visual Basic of the physical memory storage that must be reserved for each variable.
- Programmers use variable names to reference their contents.
Named Constants or Literals

- **Constants** are numbers that have a special meaning in the context of a particular application.
- **Const** statement is used to assign symbolic names to constants.
- The syntax for this statement within a form’s procedure is:
  
  ```vbscript
  Const named-constant As data-type = expression
  ```
- A named constant is a symbolic name assigned to a constant.
- A named constant cannot be altered after it is defined.
- The rules for selecting a constant’s name are the same as the rules for selecting a variable’s name.
- All Const statements must be placed in the Declarations section of the form or within a procedure, but for clarity they are usually placed immediately after the procedure’s declaration.

Examples

```vbscript
Const HourlyRate As Byte = 15
Const Weekly As Byte = 40
Const PI As Single = 3.141
Const SalesTax As Double = .0825
```

Assignment Statements

- **Assignment statements** are statements that tell the computer to store a value into a variable.
- An assignment statement can be used to both assign a value to a variable and to perform calculations.
- The assignment statement has the following syntax:
  
  ```vbscript
  variable = expression
  ```
- An expression is any combination of constants, variables, and operators that can be evaluated to yield a value.
- Assignment statements always have an equal (=) sign and exactly one variable name immediately to the left of the equal sign.
- The value on the right of the equal sign is assigned to the variable on the left of the equal sign.

Examples

```vbscript
Count = 0
Count = Count + 1
Sum = Sum + Data
Area = Height * Width
AverageScore = TotalSum / NemberOfScores
Solution = Math.Sqrt(B^2 - 2*A*C)
```
Assignment Statements

Consider the following actions that a computer will perform:
- put 65 in location 12345
- put 49 in location 67890
- add the content of location 12345 to the content of location 67890
- store the sum in location 76543

The programming version is as follows:
- Data1 = 65
- Data2 = 49
- Sum = Data1 + Data2

Assignment Variations

An assignment variation statement is an assignment statement in which the variable on the left of the equal sign is also used on the right of the equal sign. For example:

```
Total = Total + 34.7
Sum = Sum + Number
```

Accumulating

To accumulate subtotals when data is entered one number at a time, assignment statements such as `Total = Total + 34.7` are used. These statements have the syntax:

```
variable = variable + new value
```

Example:
```
SumScore = SumScore + Score
```

Counting

- The counting statement is a variation of the accumulating assignment statement. It has the following form:
  
  Variable = Variable + fixed number

- After a counting statement is executed, the value of the variable is increased by a fixed amount.

- Example:
  
  Count = Count + 1
  Count += 1
Type Conversions

- When an assignment statement is used, the expression on the right side of the assignment operator is converted to the data type of the variable to the left of the assignment operator.
- In addition, Visual Basic also provides for explicit user-specified type conversions.

Examples

<table>
<thead>
<tr>
<th>Expression</th>
<th>Assigned Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dim Average As Integer</td>
<td>Average = 10</td>
</tr>
<tr>
<td>Dim NumberString As Integer</td>
<td>NumberString = 100</td>
</tr>
<tr>
<td>Dim Number As Single</td>
<td>Number = 100.0</td>
</tr>
</tbody>
</table>

Intrinsic (Built-in) Functions

- Intrinsic functions are preprogrammed routines provided by Visual Basic .NET that can be used in a procedure.
- These include mathematical functions, functions to carry out conversion between data types, formatting functions, and string manipulation functions.
- A number of mathematical functions are found in the System.Math class of Visual Basic .NET.
- Functions operate in a manner similar to sub procedures, except for the fact that a function always directly returns a single value.
Examples

<table>
<thead>
<tr>
<th>Function Name</th>
<th>Description</th>
<th>Returned Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Math.Sqrt(9)</td>
<td>Square root</td>
<td>3</td>
</tr>
<tr>
<td>Math.Abs(-42)</td>
<td>Absolute value</td>
<td>42</td>
</tr>
<tr>
<td>Math.Round(4.779, 2)</td>
<td>Round</td>
<td>4.78</td>
</tr>
<tr>
<td>Math.Log(24.212)</td>
<td>Log</td>
<td>3.186...</td>
</tr>
</tbody>
</table>

Common Programming Errors and Problems

- The most common errors associated with the material presented in this chapter that students should be aware of are:
- Misspelling the name of a method.
- Forgetting to close string messages to be displayed by the `MessageBox.Show` method within double quote symbols.
- Incorrectly typing the letter O for the number zero (0), and vice versa.
- Incorrectly typing the letter l, for the number 1, and vice versa.
- Forgetting to declare all the variables used in a program.
- Storing an inappropriate data type value in a declared variable.
- Using a variable in an expression before an explicit value has been assigned to the variable.
- Using an intrinsic function without providing the correct number of arguments of the proper data type.
- Being unwilling to test an event procedure in depth.