Programming with Microsoft Visual Basic.NET

Lesson 2

What have we learnt in last lesson?

- Overview of Microsoft .NET Framework
- A quick look for the Visual Studio
- Create a simple project using Visual Basic
- Declare and assign Variables and Constants
- Using Stepping for Debug

Arithmetic Expression

Calculation

Arithmetic Expression

- Calculations can be performed with variables, constants, properties of certain objects, and numeric literals
- Do not use strings in calculations
- Values from Text property of Text Boxes
  - Are strings, even if they contain numeric data
  - Must be converted to a numeric data type before performing a calculation

Operator	Operation	Precedence number
^	exponentiation (raises a number to a power)	1
–	negation	2
*, /	multiplication and division	3
\	integer division	4
Mod	modulus	5
+, –	addition and subtraction	6

Using Arithmetic Expression

INCLUDE ARITHMETIC EXPRESSIONS IN ASSIGNMENT STATEMENTS

Example 1

```
age = age + 1
```

adds the integer 1 to the contents of the Integer **age** variable, then assigns the result to the **age** variable

Example 2

```
quarters = change \ 25
```

uses the integer division operator to divide the contents of the Integer **change** variable by the integer 25, then assigns the result to the Integer **quarters** variable

Example 3

```
bonus = sales *.05
```

multiplies the contents of the Double **sales** variable by the Double number .05, then assigns the result to the Double **bonus** variable

Example 4

```
price = price * Convert.ToDecimal(1.04)
```

converts the Double number 1.04 to Decimal, then multiplies the result by the contents of the Decimal **price** variable, then assigns the result to the **price** variable

Example 5

```
bonusTextBox.Text =
    Convert.ToString(sales * .05)
```

multiplies the contents of the Double **sales** variable by the Double number .05, then converts the result (a Double number) to the String data type before assigning it to the bonusTextBox

Using Calculations in Code

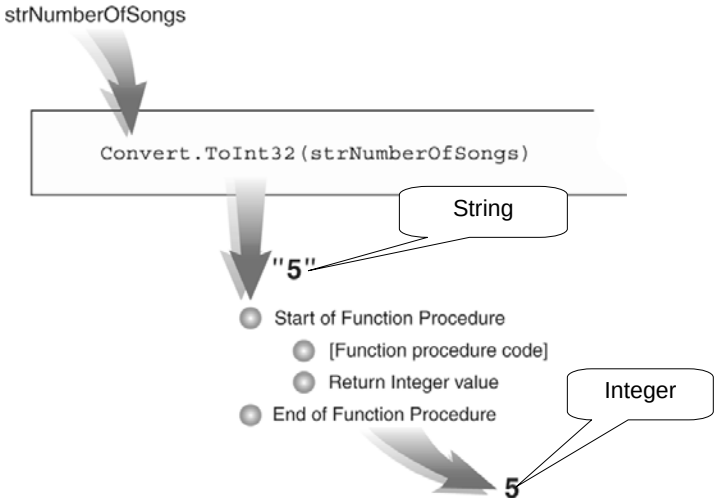
- Perform calculations in assignment statements
- What appears on right side of assignment operator is assigned to item on left side
- Assignment operators
  - = assign
  - += add and assign
  - -= sub and assign
  - *= multiply and assign
  - /= divide and assign
  - \= integer division and assign
  - ^= power and assign
  - &= append and assign

E.g.
a = a + 2 ➔ a += 2
a = a - 2 ➔ a -= 2
a = a * 2 ➔ a *= 2
a = a / 2 ➔ a /= 2
a = a \ 2 ➔ a \= 2
a = a ^ 2 ➔ a ^= 2
a = a & “2” ➔ a &= “2”

Type Conversion

TryParse and Convert

Data Type Conversion



Using the Convert Class

- Convert class:
  - Contains methods for converting numeric values to specific data types
  - Use the dot member access operator to separate the class name from the method name
- Commonly used methods of the Convert class include:
  - ToDouble
  - ToDecimal
  - ToInt32
  - ToString

How to use Convert Class?

USE THE CONVERT CLASS METHODS

Syntax

Convert.*method*(*value*)

Example 1

```
Dim rate As Decimal = Convert.ToDecimal(.05)
```

converts the Double number .05 to Decimal before storing it in a Decimal variable named rate

Example 2

```
Dim testScore As Integer = 98
totalLabel.Text = Convert.ToString(testScore)
```

converts the contents of an Integer variable named testScore to String, and then assigns the result to the totalLabel's Text property

Using the TryParse Method

- **TryParse** method
  - Exist in every numeric data type's class
  - Used to convert a string to that numeric data type
- Basic syntax of **TryParse** method has two arguments:
  - String: string value to be converted
  - Variable: location to store the result
- If **TryParse** conversion is successful, the method stores the value in the variable. Otherwise, a 0 is stored in the numeric variable

How to Use TryParse Method?

USE THE BASIC SYNTAX OF THE TRYPARSE METHOD

Syntax

dataType.**TryParse**(*string, variable*)

Example 1

```
Dim sales As Double
Double.TryParse(salesTextBox.Text, sales)
```

If the string entered in the salesTextBox can be converted to a Double number, the TryParse method converts the string and stores the result in the sales variable; otherwise, it stores the number 0 in the sales variable. Examples of strings that the method can convert to Double include "34", "12.55", "-4.23", "1,457.99", and " 33 " (notice the space before and after the 33). The strings will be converted to the numbers 34, 12.55, -4.23, 1457.99, and 33, respectively. Examples of strings that the method cannot convert to Double include "$5.67", "(4.23)", "7.88-", "7%", "122o", "1 345", and "" (the empty string).

Example 2

```
Dim num As Integer
Integer.TryParse(numTextBox.Text, num)
```

If the string entered in the numTextBox can be converted to an Integer number, the TryParse method converts the string and stores the result in the num variable; otherwise, it stores the number 0 in the num variable. Examples of strings that the method can convert to Integer include "6", " 7 " (notice the space before and after the 7), and "-896". The strings will be converted to 6, 7, and –896, respectively. Examples of strings that the method cannot convert to Integer include "5,889", "$78", "(11)", "4-", "7.5", and "" (the empty string).

Option for Compiler Checking

- The Visual Basic compiler provides several options for checking your code at compile time.
  - **Option Explicit** determines whether variables must be explicitly declared.
  - **Option Strict** determines whether explicit narrowing conversions and late binding are allowed.
  - **Option Infer** enables type inference for member-level (local) variables.
  - **Option Compare** specifies the method that is used for string comparisons: binary (case-sensitive) or text (case-insensitive).

Option Explicit

- When Option Explicit is turned off, you can use any variable name without first declaring it.
- In VB .NET the Option Explicit is turned on by default for all new projects. All variables are declared before being used.
- When Option Explicit is turned on, you must declare both var1 and var2 before writing
  - *var1 = var2 + 1*

Option Strict

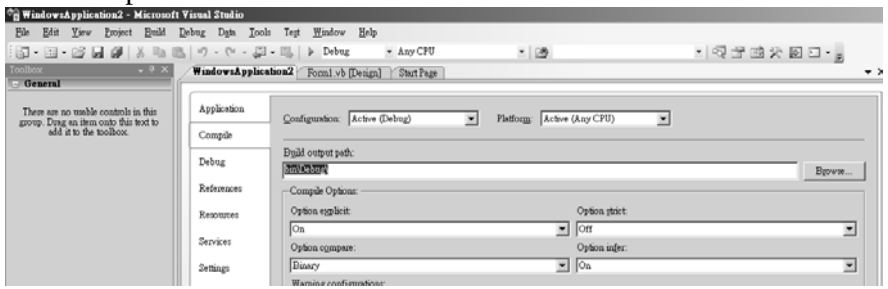
- You must use the statement “Option Strict On” in projects to ensure that:
  - All variables are declared prior to use
  - Data conversions are not done from a wider type to a narrower type
    - E.g. a decimal number 5.43 is not automatically converted to an integer 5
  - Option Strict does not allow any implicit conversions from a wider data type to a narrower one or between String and numeric data types.

Option Infer

- Option Infer Off statement ensures that every variable is declared with a data type
- Option Infer Off warns if a variable declaration does not include a data type

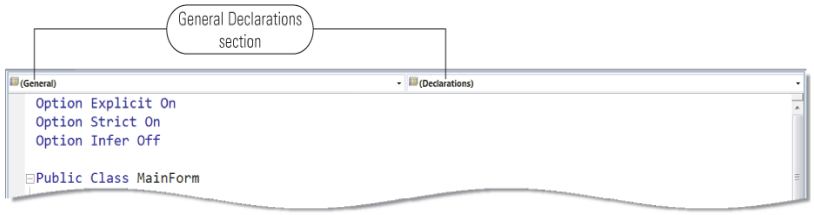
Setting for Option Explicit, Option Infer & Option Strict by Configuration

- Open the project properties dialog box and select “Compile” tab page.
- By default:
  - Option Explicit and Option Infer is turned on
  - Option Strict is turned off.



Setting for Option Explicit, Option Infer & Option Strict by Programming

- Option statements entered in the General Declarations section



Rules and Examples of Type Conversion

1. Strings will not be implicitly converted to numbers. The Code Editor will display a warning message when a statement in your code attempts to use a string where a number is expected. In the following statements, hours is a Double variable.

Incorrect — hours = hoursTextBox.Text

Correct — Double.TryParse(hoursTextBox.Text, hours)

2. Numbers will not be implicitly converted to strings. The Code Editor will display a warning message when a statement in your code attempts to use a number where a string is expected. In the following statements, grossPay is a Decimal variable.

Incorrect — grossLabel.Text = grossPay

Correct — grossLabel.Text = Convert.ToString(grossPay)

3. Wider data types will not be implicitly demoted to narrower data types. The Code Editor will display a warning message when a statement in your code attempts to use a wider data type where a narrower data type is expected.

Incorrect — Dim rate As Decimal = .05

Correct — Dim rate As Decimal = Convert.ToDecimal(.05)

4. Narrower data types will be implicitly promoted to wider data types.

Correct — Dim bonusRate As Double = 6

Relational and Logical Operators

Compound Decision

Relational Operators

- Relational operators are binary – they require an operand on both sides of the operator
- Result of a relational expression will always be Boolean
- They are evaluated from left to right with no order of operations

	Relational Operator	Meaning	Example	Resulting Condition
1	=	Equal to	8 = 8	True
2	<>	Not equal to	6 <> 6	False
3	>	Greater than	7 > 9	False
4	<	Less than	4 < 6	True
5	>=	Greater than or equal to	3 >= 3	True
6	<=	Less than or equal to	7 <= 5	False

Logical Operators

- Used for joining Boolean expressions
  - **Not –** makes a False condition True and vice versa
  - **And –** will yield a True if and only if both expressions are True
  - **Or –** will yield a True if one or the other or both expressions are True

Logical Operator	Order
Not	Highest Precedence
And, AndAlso	Next Precedence
Or, OrElse, Xor	Last Precedence

Example

- To test if n falls between 2 and 5:
  - (2 < n) And (n < 5)
- A complete relational expression must be on either side of the logical operators And and Or.
- The following is NOT a valid way to test if n falls between 2 and 5:
  - (2 < n < 5)

Order of Operations

- The order of operations for evaluating Boolean expressions is:
  - Arithmetic Operators
    - Parenthesis
    - Exponentiation
    - Division and multiplication
    - Addition and subtraction
  - Relational Operators
  - Logical Operators
    - Not
    - And
    - Or

Control Structures

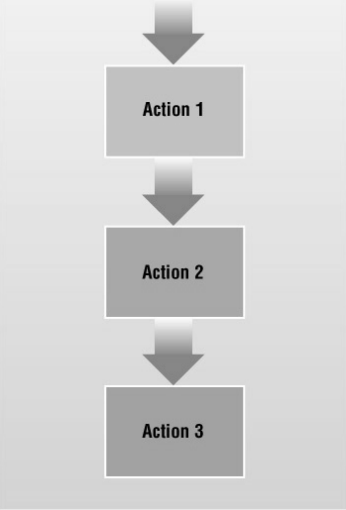
Sequence, Selection and Iteration Structure

Control Structure

- Three types of control structure, derived from structured programming:
  - Sequences of instructions
  - Selection of alternative instructions (or groups of instructions)
  - Iteration (repetition) of instructions (or groups of instructions)

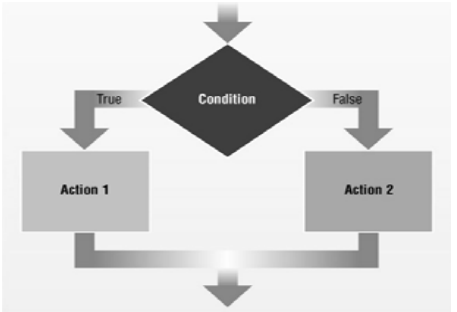
The Sequence Structure

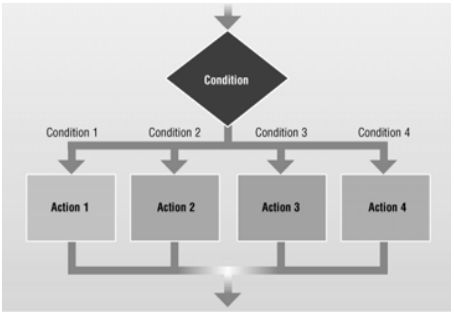
- Directs computer to process program instructions in a particular order
- Set of step-by-step instructions that accomplish a task



The Selection Structure

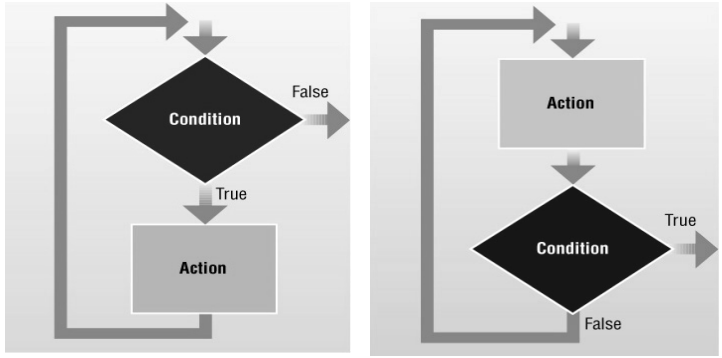
- Also called the **Decision Structure**, Makes a decision and then takes appropriate action based on that decision
- Used every time you drive your car and approach an intersection





The Iteration Structure

- Directs computer to repeat one or more instructions until some condition is met
- Also referred to as a **Loop**, **Repeating** or **Iteration**



Decision

If…Then…Else / Select Case

If … Then … Else Block

- Programs need to do different things in response to different conditions.
- The If...Then statement allows you to evaluate a condition and to then run different sections of code based on the results of that condition.

Will be executed if condition is True

Will be executed if condition is False

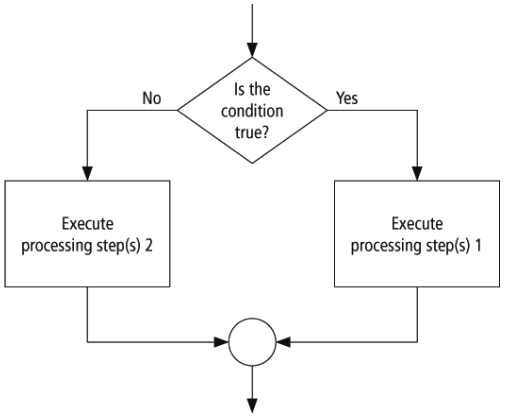
If *condition* **Then**
action1
Else
action2
End If

If *condition* **Then**
action1
End If
Statement 2
Statement 3

Regardless of whether the condition in the If statement is True or False, these statements will be executed

Workflow for If … Then … Else Block

```
If condition is true Then
  Processing step(s) 1
Else
  Processing step(s) 2
End If
```



ElseIf Block

- An extension of the If block allows for more than two possible alternatives with the inclusion of ElseIf clauses.
  - There is no space between the word "Else" and "If“ and only one "End If" is required.

```
If condition1 Then
  action1
ElseIf condition2 Then
  action2
ElseIf condition3 Then
  action3
Else
  action4
End If
```


Simplified Nested If Statement

- When one If block is contained inside another If block, the structure is referred to as nested If blocks.
- Care should be taken to make If blocks easy to understand.

```
If cond1 Then
  If cond2 Then
    action
  End If
End If
```

Nested If

```
If cond1 And cond2 Then
  action
End If
```

Less Confusing

Select Case Terminology

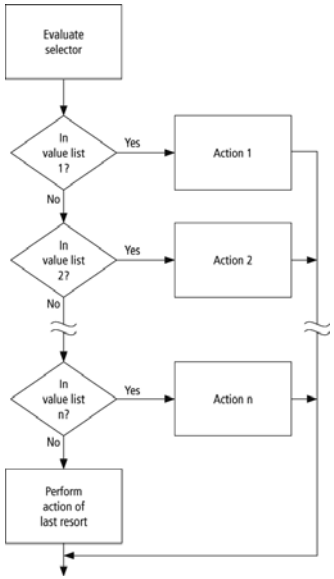
- A decision-making structure that simplifies choosing among several actions.
- Avoids complex nested If constructs.
- If blocks make decisions based on the truth value of a condition; Select Case choices are determined by the value of an expression called a **Selector**.

```
Select Case Selector
  Case Value1
      Action1
  Case Value2 to Value3
      Action 2
  Case Else
      Action of last resort
End Select
```

Selector

Value Lists

Workflow for Select Case Block



Rules for Select Case

- Case Else (and its action) is optional
- Each value list contains one or more of the following types of items separated by commas:
  - A literal;
  - A variable;
  - An expression;
  - An inequality sign preceded by Is and followed by a literal, variable, or expression;
  - A range expressed in the form a To b, where a and b are literals, variables, or expressions.

Iteration

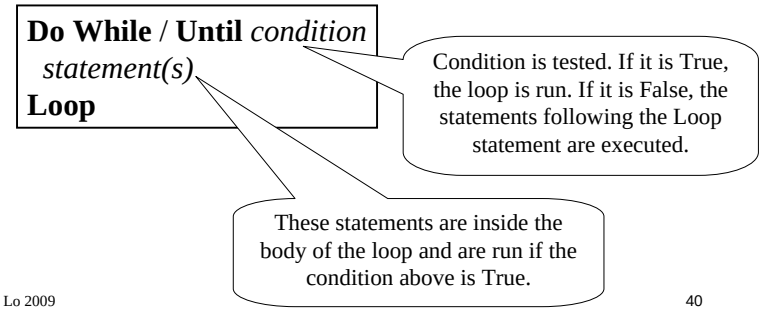
Pre Test Loop / Pro Test Loop / For Loop

Do Loop

Type of Do Loop	Explanation	Example
Do While … Loop	The Do While ... Loop evaluates the condition, and if the condition is true, then it evaluates the statements following the condition. When it has finished doing this, it evaluates the condition again and if the condition is true, it evaluates the statements again. It continues repeating this process until the condition is false.	**Do While** *condition* *statements* **Loop**
Do Until ... Loop	The Do Until ... Loop is similar to the Do While ... Loop except it keeps evaluating the statements until the condition is true rather than while it is true.	**Do Until** *condition* *statements* **Loop**
Do ... Loop While	The Do ... Loop While evaluates the statements only once. It then evaluates the condition, and if the condition is true, evaluates the statements again. This process continues until the condition is false.	**Do** *statements* **Loop While** *condition*
Do ... Loop Until	Similar to Do ... Loop While except that it evaluates the statements until the condition is true.	**Do** *statements* **Loop Until** *condition*

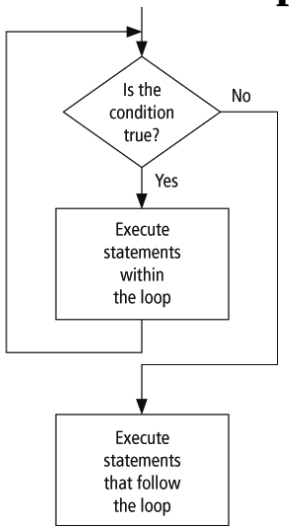
Pre Test Loops

- A loop is one of the most important structures in programming.
- Used to repeat a sequence of statements a number of times.
- The Do loop repeats a sequence of statements either as long as or until a certain condition is true.



Workflow for Pre Test Loop

Do While condition is true
  Processing step(s)
Loop



Example for Pre Test Loop

```
Dim intScore As Integer = 0
Do While intScore < 5
    intScore += 1
Loop
```

Loop Iteration	Value of intScore	Result of Condition Tested
1	intScore = 0	True
2	intScore = 1	True
3	intScore = 2	True
4	intScore = 3	True
5	intScore = 4	True
6	intScore = 5	False

Post Test Loop

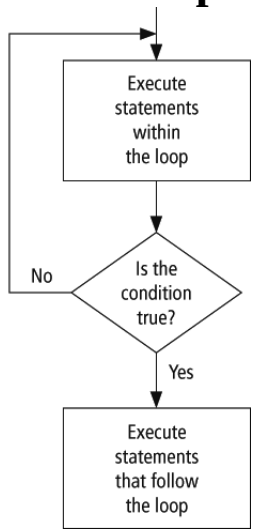
- A Do statement precedes the sequence of statements, and a Loop statement follows the sequence of statements.
- The condition, preceded by either the word “While” or the word “Until”, follows the word “Do” or the word “Loop”.
- Be careful to avoid infinite loops – loops that never end.
- VB.NET allows for the use of either the **While** keyword or the **Until** keyword at the top or the bottom of a loop.

Do
  statement(s)
Loop Until / While *condition*

Loop is executed once and then the condition is tested. If it is False, the loop is run again. If it is True, the statements following the Loop statement are executed.

Workflow for Post Test Loop

Do
  statement(s)
Loop Until condition is true



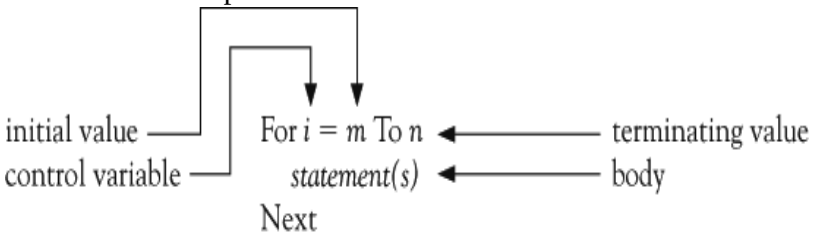
Example for Post Test Loop

```
22          Dim intScore As Integer = 0
23          Do
24              intScore = intScore + 1
25          Loop While intScore < 5
```

Loop Iteration	Value of intScore at Start of the Loop	Value of intScore When Checked	Result of Condition Tested
1	intScore = 0	intScore = 1	True
2	intScore = 1	intScore = 2	True
3	intScore = 2	intScore = 3	True
4	intScore = 3	intScore = 4	True
5	intScore = 4	intScore = 5	False

For…Next Loops

- You can use a For...Next loop when a section of code is to be executed an exact number of times
- For…Next loops enable you to evaluate a sequence of statements multiple times.
- For and Next statements must be paired and a counter is used to control the loop



Start, Stop, and Step values

- Consider a loop beginning with
  - **For** *i* = *m* **To** *n* **Step** s.
- The loop will be executed exactly once if *m* equals *n* no matter what value *s* has.
- The loop will not be executed at all if *m* is greater than *n* and *s* is positive, or if *m* is less than *n* and *s* is negative.

Altering the Control Variable

- The value of the control variable should not be altered within the body of the loop;
  - doing so might cause the loop to repeat indefinitely
  - or have an unpredictable number of repetitions.
- Non-integer step values can lead to round-off errors with the result that the loop is not executed the intended number of times.

Example for For…Next Loop

```
For intNumber = 1 To 4
    'Body of loop
Next
```

Loop Iteration	Value of intNumber	Process
1	`intNumber = 1`	Executes the code inside the loop
2	`intNumber = 2`	Executes the code inside the loop
3	`intNumber = 3`	Executes the code inside the loop
4	`intNumber = 4`	Executes the code inside the loop
5 (exits the loop)	`intNumber = 5`	The control variable value exceeds the ending value, so the application exits the For...Next loop. This means the statement(s) following the Next command are executed.

Creating Random Number

The Random Function

Generating Random Integers

- **Pseudo-random number generator:** a device that produces a sequence of numbers that meets certain statistical requirements for randomness
- **Random object:** represents a pseudo-random number generator
- **Random.Next** method:
  - Generates a random integer
  - Can specify a minimum and maximum value

How to Generate a Random Number?

GENERATE RANDOM NUMBERS

Syntax

Dim *randomObjectName* **As New Random**

*randomObjectName***.Next(***minValue***,** *maxValue***)**

Example 1

```
Dim number As Integer
Dim randomGenerator As New Random
number = randomGenerator.Next(1, 51)
```

Creates a Random object named randomGenerator, then assigns (to the number variable) a random integer that is greater than or equal to 1, but less than 51.

Example 2

```
Dim number As Integer
Dim randomGenerator As New Random
number = randomGenerator.Next(-10, 0)
```

Creates a Random object named randomGenerator, then assigns (to the number variable) a random integer that is greater than or equal to -10, but less than 0.