**The If…Then Statement**

-This is a *conditional control structure*, which executes a set of statements when a condition is true!

If <condition> Then

<statements>

End If

-The equal sign (=) is used to determine if a condition is true or not…

Example

In the following, guess = 7 is the <condition>, and there is one statement that will execute when this condition is true:

If guess = 7 Then

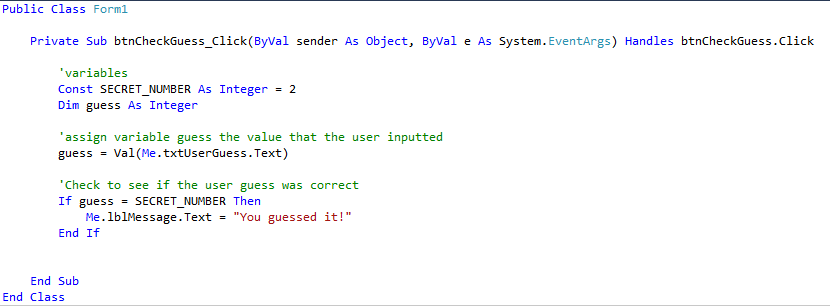
Me.lblMessage.Text = “You guessed it!”

End If

\*\*If statements are Boolean expressions, which evaluate to either true or false. We use *relational operators* to create Boolean expressions.

|  |  |
| --- | --- |
| **Operator** | **Meaning** |
| = | Equal to |
| < | Less than |
| <= | Less than or equal |
| > | Greater than |
| >= | Greater than or equal to |
| < > (no space in between) | Not equal to |

The NumberGuess application will help you understand if statements. Find it on page 94. Please create it in VB:



**Try it out:**

Review: TestGrade—Part 1 of 5 on page 94-95

**The If…Then…Else Statement**

-The If…Then statement can include an optional “else” clause that is executed when the if condition evaluates to false.

If <condition> Then

<statements>

Else

<statements>

End If

The NumberGuess application can be modified so that if the the guess is incorrect the appropriate message is displayed:

If guess = SECRET\_NUMBER Then

Me.lblMessage.Text = “You guessed it!”

Else

Me.lblMessage.Text = “Try again!”

End If

**Try it out:**

Review: TestGrade—Part 2 of 5 on page 95

**Nested If…Then…Else Statement**

What if we placed an if…then…else inside of another if statement? The result is what programmers call a *nested if statement*.

It is important to use indentation in order to make the statements easier to read!

We’ll modify our NumberGuess to include messages for “too high” and “too low”:

If guess = SECRET\_NUMBER Then

Me.lblMessage.Text = “You guessed it!”

Else

If guess < SECRET\_NUMBER Then

Me.lblMessage.Text = “Too low!”

Else

Me.lblMessage.Text = “Too high!”

End If

End If

**If…Then…ElseIf Statement**

This is used to decide among three or more actions

If <condition> Then

<statements>

ElseIf <condition> Then

<statements>

Else

<statements>

End If

\*There can be as many Elseif clauses as you like.

\*\*\*When you have elseif’s… only the FIRST TRUE Elseif statement will execute… visual basic will simply skip every elseif after the true one!!!!

\*\*The last “else” is optional. It will execute if none of the previous statements were true.

Logic is important with ElseIf statements. As soon as one clause is true, it will execute the statements and then program flow will proceed to the “End if”

**Try it out:**

Review: TestGrade—Part 3 of 5 on page 96

**The Select…Case Statement**

This decision structure uses the result of an expression to determine which statement to execute.

Sometimes it is good to use these as code may be easier to read than an ElseIf

Select Case <expression>

Case <value>

<statements>

...

Case Else

<statements>

End Select

Notes

<expression> must evaluate to a built in data type (such as an integer, double, Boolean, etc.)

\*You can have multiple Case clauses, and Case Else is optional.

\*\*<value> type should match the expression type and be a single value, or a list separated by commas, or a range separated by the keyword **To**.

Examples

The following uses Select to determine which message to display based on the user’s score in a game:

Select Case score

Case 0, 10

Me.lblMessage.Text = “Nice try”

Case 20 To 30

Me.lblMessage.Text = “Great!”

Case Else

Me.lblMessage.Text = “Invalid score”

End Select

**Using Select…Case Is**

Similar to above, Select…Case Is compare the result to a range of values. For example

Select Case score

Case Is < 10

Me.lblMessage.Text = “Nice try”

Case Is < 25

Me.lblMessage.Text = “Great!”

Case Is >= 25

Me.lblMessage.Text = “Super!”

End Select

**Try it out:**

Review: Hurricane page 97

Review: TestGrade—part 4 of 5, page 98

(Use ONLY the km/hr wind speeds for the Hurricane application!)

**Generating Random Numbers**

Many applications and programs make use of random numbers. Games, simulators, screen savers, etc.

VB has a built in function to do this, Rnd(). The function uses a special math formula to generate random numbers. However, they aren’t quite truly random, so we called them *pseudorandom (like random)*. They are random enough for what we will use them for.

Rnd() returns a number between 0 and 1. Now you’ll say, “that’s not very useful probably!”

So we have to turn to good old math and use a formula to generate random numbers in a range. To generate random numbers between say, 0 and 100, we use:

Randomize()

Int((highNumber – lowNumber + 1) \* Rnd() + lowNumber)

**Try it out:**

Review: RandomNumbers, page 99-100

**Algorithms**

We create programs to solve complex problems.

However, if a problem is complicated or long, we have to outline and design the solution to the problem.

One method of doing this is creating an algorithm

The algorithm for NumberGuess application would like this:

1. *Determine a secret number*
2. *Get a number from the player*
3. *Compare the player’s number with the secret number.*
4. *If the player’s number is the same as the scret number go to step 5, otherwise tell the player if the mumber entered was too low or too high and then go back to step 2.*
5. *Display a message telling the player the secret number was guessed.*

In computer science, we call this algorithms *pseudocode (almost code)*

The pseudocode for NumberGuess application would like this:

*Sub btnCheckGuess\_Click()*

*secretNumber = 2*

*Get guess from text box*

*If guess = secretNumber Then*

*Display “You guessed it”*

*ElseIf guess < secretNumber Then*

*Display “Too low.”*

*Else*

*Display “too high.”*

*End If*

*End Sub*

**Static Variables**

The lifetime of a local variable is the duration of the procedure in which it is delcarled.

The lifetime of a global variable is the duration of the program.

A static variable is declared using the keyword Staticinstead of Dim. It’s scope is local to the procedure in which it is declared, but its lifetime is the duration of the program.

**Try it out:**

Review: GuessingGame—part 1 of 4, page 102-104

**Compound Boolean Expressions**

Conditions with complex criteria are formed using the logical operators And and Or

As an example, consider an application that computes a discount depending on the quantity and type of purchase:

If itemNum = 200 And quantity > 50 Then

Discount = 1

End If

Another logical operator is Not.

For example, the following statements change the label because itemNum is NOT equal to 220

Dim itemNum = 180

If Not itemNum = 200 And quantity > 50 Then

Me.lblMessage.Text = “No discount given.”

End If

**Try it out:**

Review: RockPaperScissors—part 1 of 4, page 105-107

Review: RockPaperScissors—part 3 of 4, page 105-107

**Displaying a Message Box**

A message box is a dialogue box that displays a message for the user. It can alert eh user to invalid data or as a reminder.

**How to use a message box**

MessageBox.Show(message)

Where message can be a variable, constant, or a string!

An example…

If guess < MIN Or guess > MAX Then

MessageBox.Show(“Guess out of range!”)

**Try it out:**

Review: GuessingGame—part 2 of 4, page 108

Review: TestGrade—part 5 of 5, page 108

**Counters**

Many algortihms involve counting. Info such as the number of times a user enters a guess require counting each guess. Calculating the average of a set of numbers requires counting as well.

We use *counter variables* to store a number that is incremented *by the same amount* each time something happens.

We can use counters to keep track of the number of times a user clicks a button, enters a guess, or types a password.

**How to increment a counter variable**

counter = counter + 1

Since counters are used so often, VB has a special operator to save you some typing! The operator is “+=”. This can be used to increase a counter as well. It is the preferred method as it requires less typing (and being efficient is always best)

|  |  |
| --- | --- |
| counter = counter + 1 | Increases counter by 1 |
| counter += 1 | Increases counter by 1 |

If a counter must move *backwards*, we use the “-=” operator! Counters should always be Static variables, and make sure to initialize a counter when you declare it:

Static counter As Integer = 0

**Try it out:**

Review: RockPaperScissors—part 4 of 4, page 109

**The CheckBox Control**

Check boxes allow the user to select options. Unlike radio buttons, more than one can be selected at a time!

Some properties of check boxes are:

|  |  |
| --- | --- |
| **(Name)** | Identifies a control for the programmer. *Should begin with* chk |
| **Text** | Is the text displayed next to the box |
| **Checked** | Can be either true or false to display with or without a check respectively |

Related check boxes can be placed in a group box.

**How to determine if a check box is selected**

Usually an If…Then statement is used to determine if a check box is selected or cleared.

For example,

If Me.chkeLunch.Checked Then

MessageBox.Show(“Don’t forget your water!”)

Else

MessageBox.Show(“Take lunch money!”)

End If

Sometimes click procedures can be used. This procedure will execute when a box is checked and then performs actions depending on what you are trying to do.

**The Line-Continuation Character**

VB statements that are longer than 80 characters should be placed on two separate lines as good programming technique.

We use the underscore (\_), or the *line-continuation character* to do so. It must have a space before it and nothing after.

It cannot be used in quotations.

An example of a condition placed onto 2 lines instead of one long one:

If Not (Me.chkBed.Checked And Me.chkLunch.Checked \_

And Me.chkHomework.Checked And Me.chkTeeth.Checked) Then

**Try it out:**

Review: MorningToDo, page 110

**TASKS**

1. Critical Thinking Questions, Page 119

#1-9, 11-12, 16

1. Exercises, page 121-128: #2-4, 7-8, 10-14, 16-17