# Documentation for VB.NET: Control Flow Statements

## Select Case Statement

The `Select Case` statement is an alternative to using multiple `If-ElseIf` conditions. It allows you to compare a single expression against different values and execute corresponding code blocks.

The syntax is:

```
Select Case expression
    Case value1
        ' Statements A
    Case value2
        ' Statements B
    Case Else
        ' Statements C
End Select
```

**Behavior**:

- **Statements A** will be executed if the expression matches `value1`.
- **Statements B** will be executed if the expression matches `value2`.
- If no cases match, **Statements C** under `Case Else` will be executed.

**Example**:

```
Dim grade As String = "B"

Select Case grade
    Case "A"
        Console.WriteLine("Excellent!")
    Case "B"
        Console.WriteLine("Good Job!")
    Case "C"
        Console.WriteLine("You Passed!")
    Case Else
        Console.WriteLine("Better luck next time!")
End Select
```

In this example:

- If the grade is "A", "Excellent!" is displayed.
- If the grade is "B", "Good Job!" is displayed.
- If the grade is "C", "You Passed!" is displayed.
- For any other value, "Better luck next time!" is displayed.

## While Loop

The `While` loop repeatedly executes a block of code **while** the specified condition is `True`. The condition is checked **before** the code block runs, meaning the loop may never execute if the condition is `False` initially.

The syntax is:

```
While condition
    ' Statements A
End While
```

**Behavior**:

- **Statements A** are executed as long as the condition is `True`.
- If the condition is `False` at the start, the loop will not run at all.

**Example**:

```
Dim counter As Integer = 1

While counter <= 5
    Console.WriteLine("Counter is: " & counter)
    counter += 1
End While
```

In this example:

- The message `"Counter is: "` followed by the current value of `counter` is displayed 5 times, from 1 to 5.

## Do While Loop

The `Do While` loop repeatedly executes a block of code as long as a specified condition is `True`. It works the **same** as `While` loop.

The syntax is:

```
Do While condition
    ' Statements A
Loop
```

**Behavior**:

- **Statements A** will execute repeatedly as long as the condition remains `True`.
- If the condition is `False` at the start, the loop will **not** execute at all.

**Example**:

```
Dim counter As Integer = 10

Do While counter <= 30
    Console.WriteLine("Counter is: " & counter)
    counter += 1
Loop
```

In this example:

- The loop checks if `counter <= 30`. If this is `True`, it prints the current value of `counter`.
- Once `counter` exceeds 30, the loop terminates.
- If `counter` had been greater than 30 initially, the loop would not have run at all.

## Do Until Loop

The `Do Until` loop is the opposite of the `Do While` loop, where the loop continues until the condition becomes `True`.

The syntax is:

```
Do Until condition
    ' Statements A
Loop
```

**Behavior**:

- **Statements A** will execute repeatedly as long as the condition remains `False`.
- If the condition is `True` at the start, the loop will not run at all.

**Example**:

```
Dim value As Integer = 10

' Print a label for the sequence
Console.WriteLine("Number sequence:")

' Loop until value equals 15
Do Until value = 15
    ' Print the current value followed by a space (without a newline)
    Console.Write(value & " ")
    ' Increment the value by 1
    value += 1
Loop

' Move to the next line after the loop completes
Console.WriteLine()
Console.ReadKey()
```

In this example:

- The variable `value` is initialized to 10.
- The message **"Number sequence: "** is printed to introduce the output.
- The loop runs, printing the current value (starting from 10) followed by a space.
- The loop continues until `value` reaches 15, which means the numbers printed will be 10, 11, 12, 13, and 14.
- Finally, `Console.WriteLine()` is called to move to the next line after the sequence is printed.
- `Console.Write()` works a little bit differently than `Console.WriteLine()`.
- The key difference is that `Console.Write()` does not append a newline at the end of the output, while `Console.WriteLine()` does. This means `Console.Write()` will keep the cursor on the same line.

## Do Loop – Post-Test

The Do Loop – While/Until, also known as the "Post-test Form of Do Loop," ensures that the code block runs at least once before checking the condition. This means the loop will always execute the statements inside the loop, then evaluate the condition at the end of each iteration:

The syntax is:

```
Do
    ' Statements A
Loop While condition

' OR

Do
    ' Statements A
Loop Until condition
```

**Behavior**:

- **Statements A** are executed **at least once**, regardless of the condition.
- **After** executing the block, the condition is evaluated.
- While
    - If the condition is `True`, the loop repeats.
    - If the condition is `False`, the loop stops.
- Until
    - If the condition is `True`, the loop stops.
    - If the condition is `False`, the loop repeats.

**Example**:

```
Dim oneTimeValue As Integer = 15

Do

    Console.WriteLine("One Time Value is: " & oneTimeValue)
    oneTimeValue += 1

Loop Until oneTimeValue > 10

Console.ReadKey()
```

In this example:

- `oneTimeValue` is initialized to 15.
- The loop runs once, printing "One Time Value is: 15".
- After the first iteration, `oneTimeValue` becomes 16, and since the condition (`oneTimeValue > 10`) is True, the loop terminates.
- Although the condition was already True at the start (`15 > 10`), the loop still executes once before the check, demonstrating the post-test behavior of the Do Loop.

---

## Conclusion

This documentation provides an overview of various control flow mechanisms in VB.NET, including `Select Case`, `While`, `Do While`, and `Do Until` loops. These structures are vital for controlling the execution of your programs based on conditions. To make learning fun, we also included a humorous console art example!