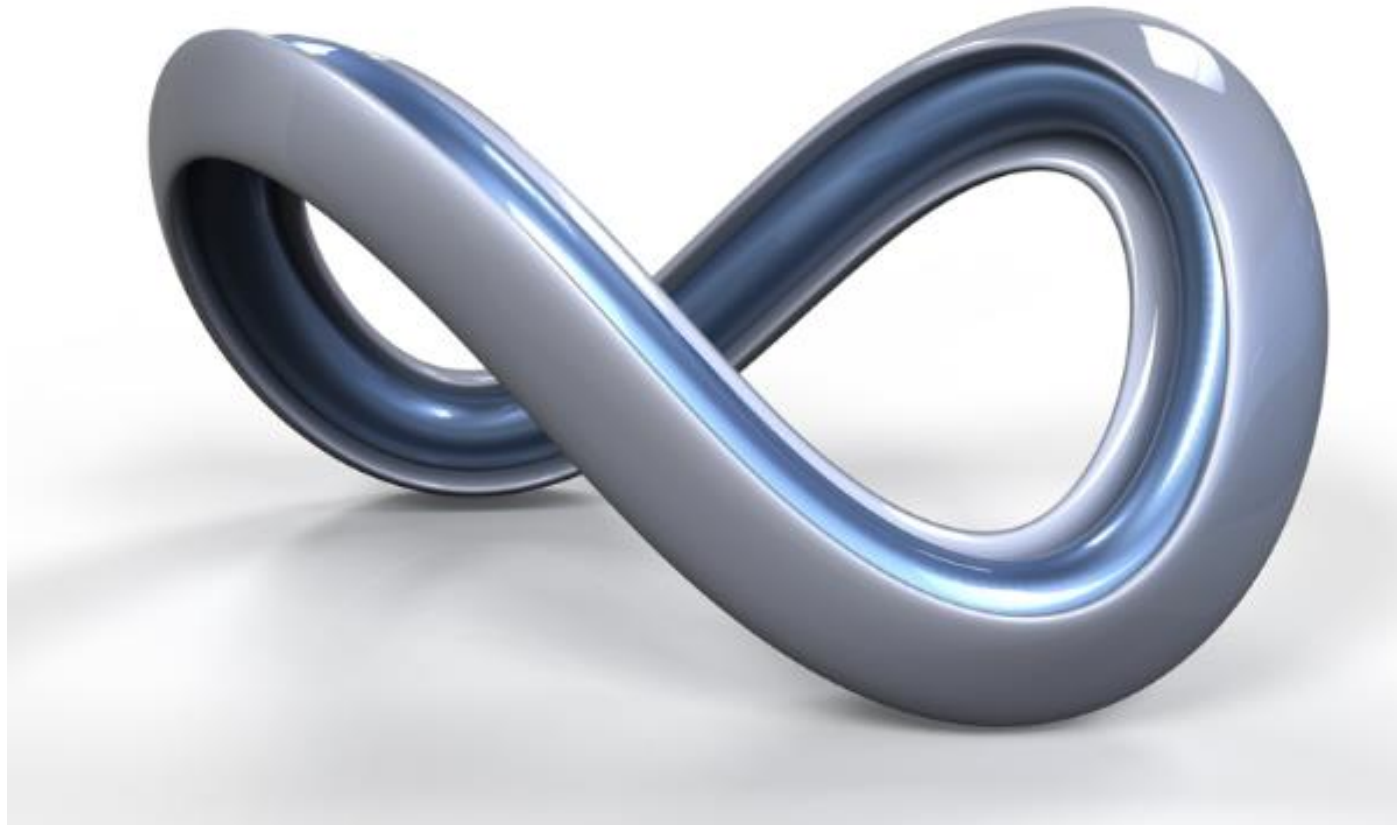


Loop Structures in VB



CMPT 110

Objectives From Study Guide 8

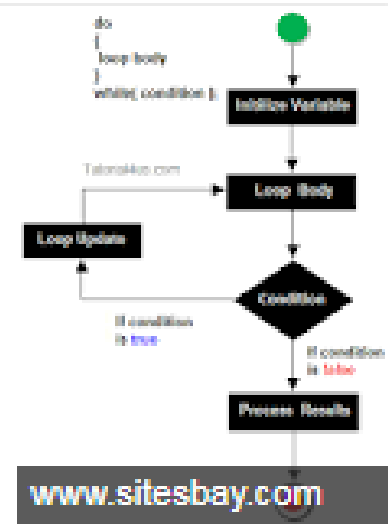
- To become familiar with what constitutes a valid loop.
- To learn the syntax for defining loops using a Do While block.
- To learn the syntax for defining loops using the For Next block.
- To understand how to define loops using recursive definitions.

Definition: Loop Structure

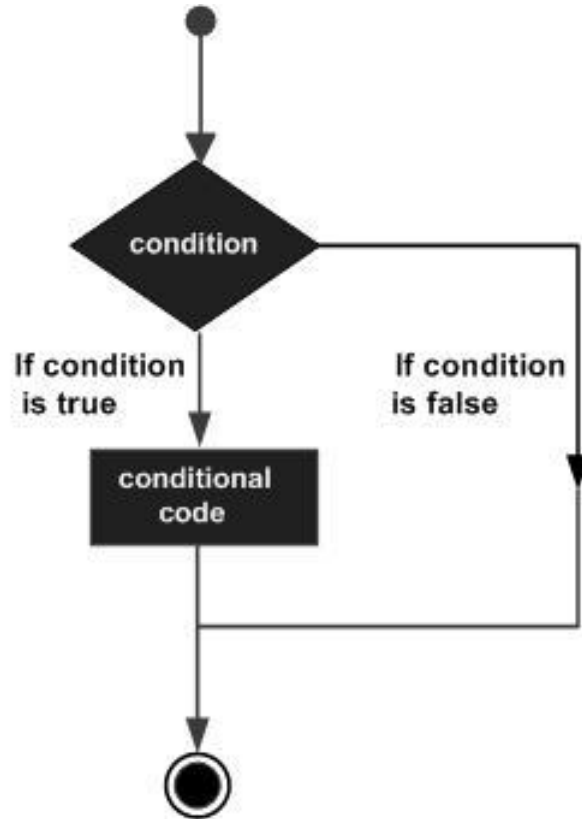
In computer programming, a **loop** is a sequence of instructions that is continually repeated until a certain condition is reached. Typically, a certain process is done, such as getting an item of data and changing it, and then some condition is checked such as whether a counter has reached a prescribed number.

What is loop? - Definition from WhatIs.com

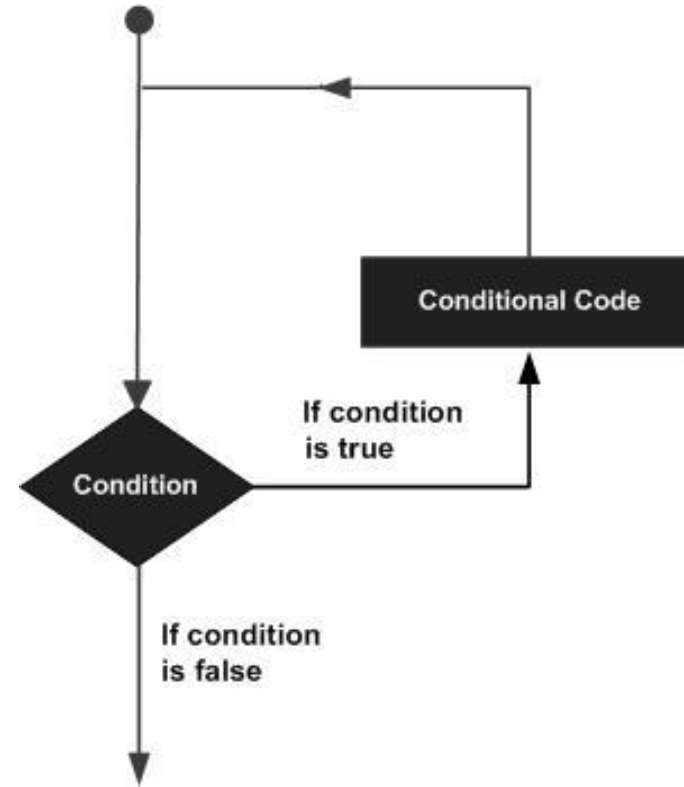
[whatis.techtarget.com/definition/loop](https://www.techtarget.com/definition/loop)



“If-Then” versus Loop Structure



“If-Then” Decision Structure



While Loop Structure (multiple Ifs)

Loop Structure Semantics

- While a condition is true, execute the statements in the block:
 - Execute them *once*; then
 - *Re-evaluate the condition; then*
 - *If the condition is still true, execute the block again!*



The key concept of iteration!!!

Loop Structure Semantics

- While a condition is true, execute the statements in the block:
 - Execute them *once*; then
 - *Re-evaluate the condition; then*
 - *If the condition is still true, execute the block again!*



The key concept of iteration!!!

The condition could be the number of iterations, OR the occurrence of an external condition.

It's Called a "Loop"

*If it's still true,
execute the
block again*

While this
condition is
true

n=1

while n < 10 :

Execute this block of code!

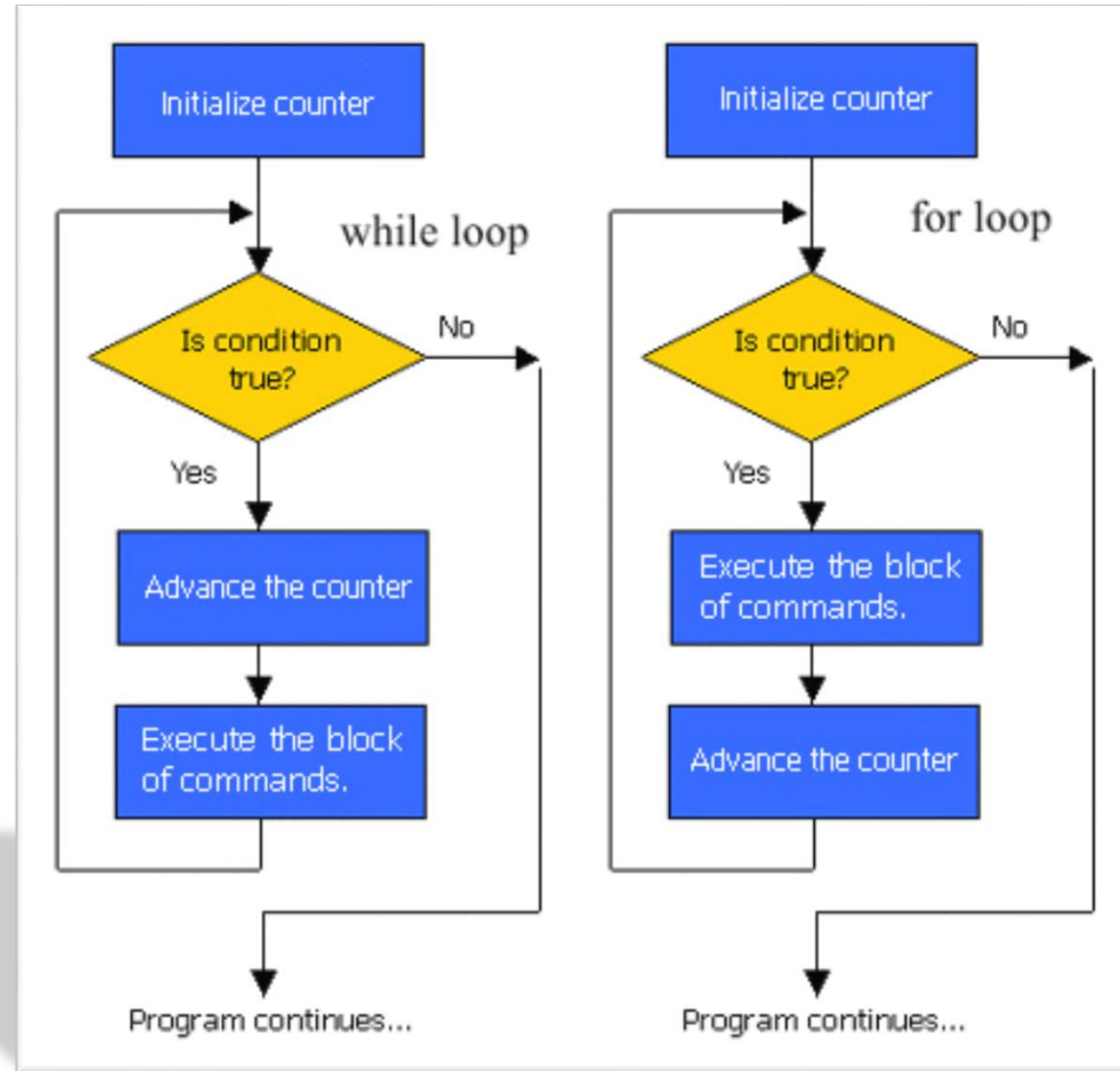
```
print "The value of n is", n  
n=n+1
```

Once the block has been executed, test the
condition again!

Different Types of Loop Structures

Loop Type	Description
while loop	Repeats a statement or group of statements while a given condition is true. It tests the condition before executing the loop body.
for loop	Executes a sequence of statements multiple times and abbreviates the code that manages the loop variable.
nested loops	You can use one or more loop inside any another while, for or do..while loop.

While-Loops versus For-Loop



The While-Loop Versus The For-Loop

- **For-loops** are used when you know exactly how many iterations are involved.

The While-Loop Versus The For-Loop

- **For-loops** are used when you know exactly how many iterations are involved.
- **While-loops** are more flexible (iterate until a condition is met, for example).
 - These are also classified as *iterative loops* (for-loop) and *conditional loops* (while-loop).

Loop Structures in VB.Net

Different Types of Loop Structures in VB

Visual Basic has three main types of loops:

- *for..next* loops,
- *do* loops and
- *while* loops.

```
For X = 1 To 100 Step 2
    Debug.Print X
Next X
```







```
Do
    Debug.Print "hello"
    x = x + 1
Loop Until x = 10
```

```
price = 2

While price < 64
    Debug.Print "Price = " & price
    price = price ^ 2
Wend

Debug.Print "Price = " & price & ": Too much for the market to bear!"
```




Loop Structures in VB.Net

Loop Type		Description
Do Loop 		It repeats the enclosed block of statements while a Boolean condition is True or until the condition becomes True. It could be terminated at any time with the Exit Do statement.
For...Next 		It repeats a group of statements a specified number of times and a loop index counts the number of loop iterations as the loop executes.
For Each...Next 		It repeats a group of statements for each element in a collection. This loop is used for accessing and manipulating all elements in an array or a VB.Net collection.
While... While 	End	It executes a series of statements as long as a given condition is True.
With... With 	End	It is not exactly a looping construct. It executes a series of statements that repeatedly refer to a single object or structure.
Nested loops 		You can use one or more loops inside any another While, For or Do loop.

Loop Control Statements in VB.Net

- Loop control statements change execution from its normal sequence. When execution leaves a scope, all automatic objects that were created in that scope are destroyed.
- Visual Basic provides the following control statements on the following slide.

Loop Control Statements in VB.Net

Control Statement	Description
Exit statement 	Terminates the loop or select case statement and transfers execution to the statement immediately following the loop or select case.
Continue statement 	Causes the loop to skip the remainder of its body and immediately retest its condition prior to reiterating.
GoTo statement 	Transfers control to the labeled statement. Though it is not advised to use GoTo statement in your program.

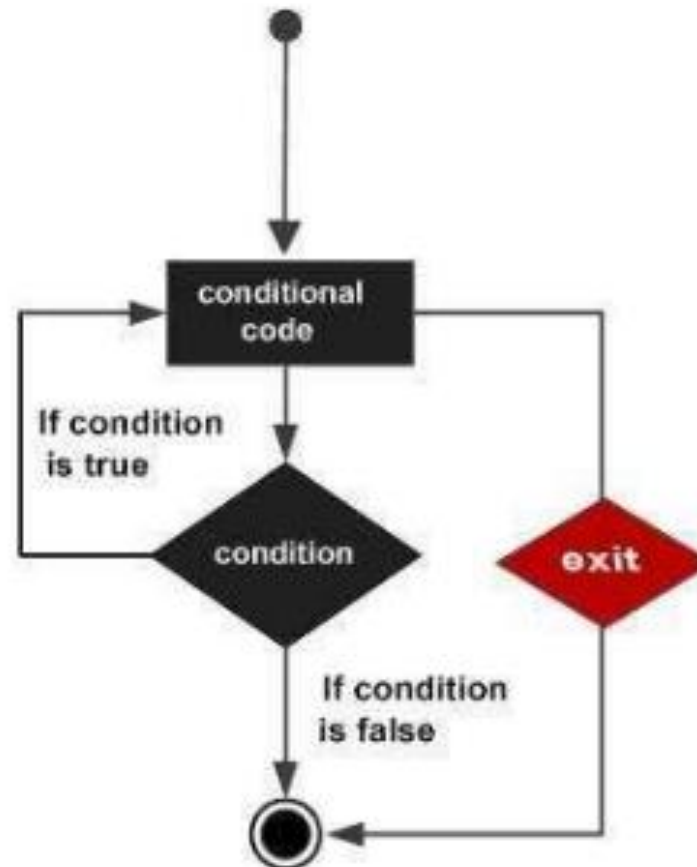
Exit Statement in VB.Net

- The Exit statement transfers the control from a procedure or block immediately to the statement following the procedure call or the block definition. It terminates the loop, procedure, try block or the select block from where it is called.
- If you are using nested loops (i.e., one loop inside another loop), the Exit statement will stop the execution of the innermost loop and start executing the next line of code after the block.
- Syntax:

```
Exit { Do | For | Function | Property | Select | Sub | Try | While }
```

Exit Statement in VB.Net

Flow Diagram:



Exit Statement in VB.Net

Example:

```
Module loops
    Sub Main()
        ' local variable definition
        Dim a As Integer = 10
        ' while loop execution '
        While (a < 20)
            Console.WriteLine("value of a: {0}", a)
            a = a + 1
            If (a > 15) Then
                'terminate the loop using exit statement
                Exit While
            End If
        End While
        Console.ReadLine()
    End Sub
End Module
```

result:

```
value of a: 10
value of a: 11
value of a: 12
value of a: 13
value of a: 14
value of a: 15
```

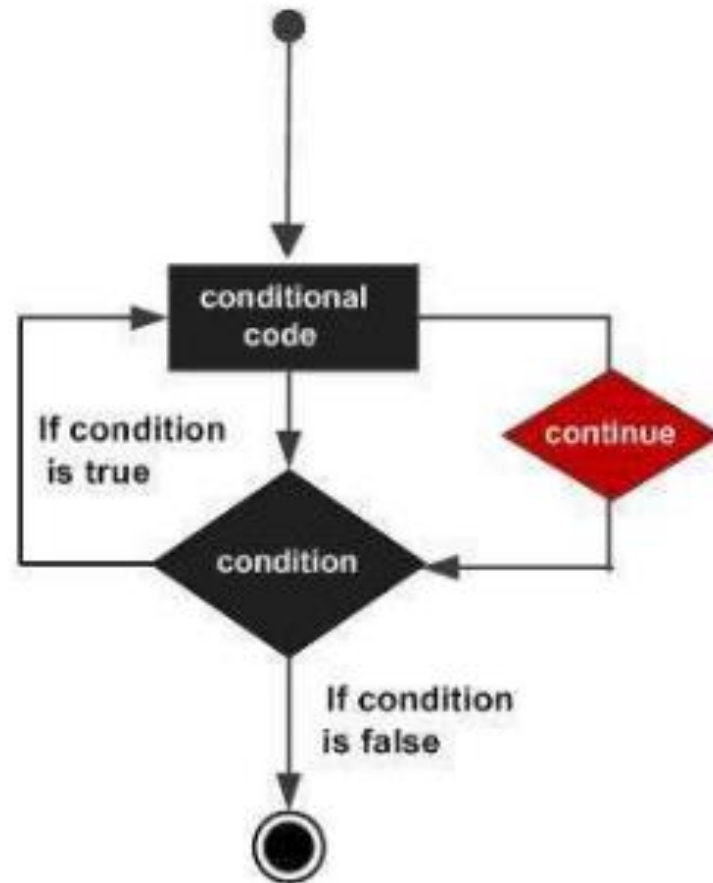
Continue Statement in VB.Net

- The Continue statement causes the loop to skip the remainder of its body and immediately retest its condition prior to reiterating. It works somewhat like the Exit statement. Instead of forcing termination, it forces the next iteration of the loop to take place, skipping any code in between.
- For the For...Next loop, Continue statement causes the conditional test and increment portions of the loop to execute. For the While and Do...While loops, continue statement causes the program control to pass to the conditional tests.
- Syntax:

```
Continue { Do | For | While }
```

Continue Statement in VB.Net

Flow Diagram:



Continue Statement in VB.Net

Example:

```
Module loops
    Sub Main()
        ' local variable definition
        Dim a As Integer = 10
        Do
            If (a = 15) Then
                ' skip the iteration '
                a = a + 1
                Continue Do
            End If
            Console.WriteLine("value of a: {0}", a)
            a = a + 1
        Loop While (a < 20)
        Console.ReadLine()
    End Sub
End Module
```

result:

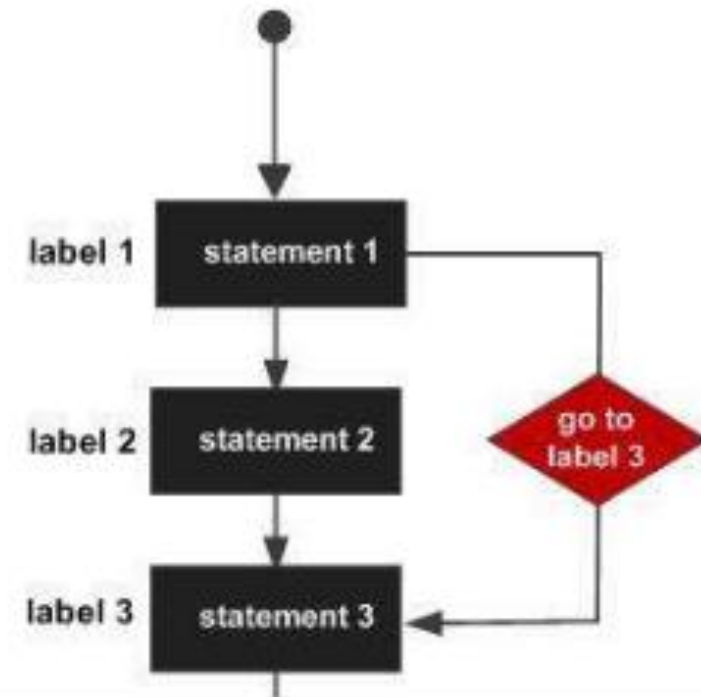
```
value of a: 10
value of a: 11
value of a: 12
value of a: 13
value of a: 14
value of a: 16
value of a: 17
value of a: 18
value of a: 19
```

GoTo Statement in VB.Net

- The GoTo statement transfers control unconditionally to a specified line in a procedure.
- Syntax:

```
GoTo label
```

Flow Diagram:



GoTo Statements in VB.Net

Example:

```
Module loops
    Sub Main()
        ' local variable definition
        Dim a As Integer = 10
    Line1:
        Do
            If (a = 15) Then
                ' skip the iteration '
                a = a + 1
                GoTo Line1
            End If
            Console.WriteLine("value of a: {0}", a)
            a = a + 1
        Loop While (a < 20)
        Console.ReadLine()
    End Sub
End Module
```

result:

```
value of a: 10
value of a: 11
value of a: 12
value of a: 13
value of a: 14
value of a: 16
value of a: 17
value of a: 18
value of a: 19
```


Recursion



Definition

- **Recursion** occurs when a thing is defined in terms of itself or of its type.
- **Recursion** in *computing science* is a method where the solution to a problem depends on solutions to smaller instances of the same problem (as opposed to *iteration*).

Recursively Defined Functions

Classic example – the Factorial Function:

$$n! = \begin{cases} 1 & \text{if } n = 0, \\ (n - 1)! \times n & \text{if } n > 0. \end{cases}$$

Recursively Defined Functions (Study Guide 8)

There is another way to define iteration that does not require any special syntax such as a "Do-While" or "For-Next" block. Consider once again computing the product of the first n positive integers, commonly called "factorial n " and [introduced as an example earlier.](#)

A common way to define how to compute factorial n (denoted by $n!$ in mathematics) is by the following "rules":

1. $\text{Factorial}(0) = 1$
2. If $n > 0$ Then $\text{Factorial}(n) = n * \text{Factorial}(n-1)$
3. If $n < 0$ Then $\text{Factorial}(n)$ is undefined.

Recursively Defined Functions (Study Guide 8)

It is possible to express recursive definitions as Visual Basic Function procedures. The factorial function, for example, can be defined recursively, using a [CASE block](#), as follows:

```
Function Factorial( n As Integer) As Integer
    Select Case n
        Case 0 ' The Basis Rule
            Return 1
        Case Is > 1 ' The Recursive Rule
            Return n * Factorial(n-1)
        Case Else
            Return 0 ' indicating an invalid value for n.
    End Select
End Function
```

Recursively Defined Sequences

Consider a physical process in which a given population, x , doubles over a particular time, t :

$$x_{t+1} = kx_t$$

The process requires an *initial condition*:












$$x_0 = a_0$$

This is known as a recursive set (a relation that is defined in terms of smaller units of itself and requiring a condition to get it started).

This is the discrete description of exponential growth.











Recursively Defined Sequences

Recall reproducing rabbits – The Fibonacci Sequence

Reproducing pairs (at least two months old)	Young pairs (less than two months old)	Month	Reproducing pairs	Young pairs	Total pairs
		1	0	1	1
		2	0	1	1
		3	1	1	2
		4	1	2	3
		5	2	3	5
	 	6	3	5	8

Recursively Defined Sequences

Recall reproducing rabbits – The Fibonacci Sequence

Reproducing pairs (at least two months old)	Young pairs (less than two months old)	Month	Reproducing pairs	Young pairs	Total pairs
		1	0	1	1
		2	0	1	1
		3	1	1	2
		4	1	2	3
		5	2	3	5
		6	3	5	8

Recursive Set

$$\begin{cases} f_1=1, f_2=1, \\ f_n = f_{n-1} + f_{n-2} \end{cases}$$

Recall from PPT2: Calculating the nth Fibonacci Number

Assembly Language

```
fib:
    mov edx, [esp+8]
    cmp edx, 0
    ja @f
    mov eax, 0
    ret

@@:
    cmp edx, 2
    ja @f
    mov eax, 1
    ret

@@:
    push ebx
    mov ebx, 1
    mov ecx, 1

@@:
    lea eax, [ebx]
    cmp edx, 3
    jbe @f
    mov ebx, ecx
    mov ecx, eax
    dec edx
    jmp @b

@@:
    pop ebx
    ret
```

Machine Language

```
8B542408 83FA0077 06B80000 0000C383
FA027706 B8010000 00C353BB 01000000
B9010000 008D0419 83FA0376 078BD98B
C84AEBF1 5BC3
```

```
unsigned int fib(unsigned int n)
{
    if (n <= 0)
        return 0;
    else if (n <= 2)
        return 1;
    else {
        int a,b,c;
        a = 1;
        b = 1;
        while (true) {
            c = a + b;
            if (n <= 3) return c;
            a = b;
            b = c;
            n--;
        }
    }
}
```

High-Level Programming Language

Calculating the nth Fibonacci Number in VB

```
Module Module1

Function Fibonacci(ByVal n)

    If n < 2 Then
        Return n
    Else
        Return Fibonacci(n - 1) + Fibonacci(n - 2)
    End If

    If n < 0 Then
        Return Val(-1)
    End If

End Function

Sub Main()

    Dim n As Double

    Console.WriteLine("Welcome!!!")
    Console.WriteLine(" Please Enter the nth term You Would like to Find ")

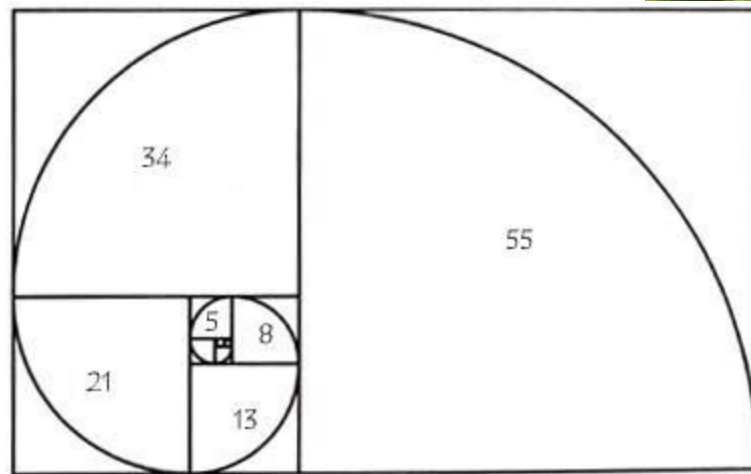
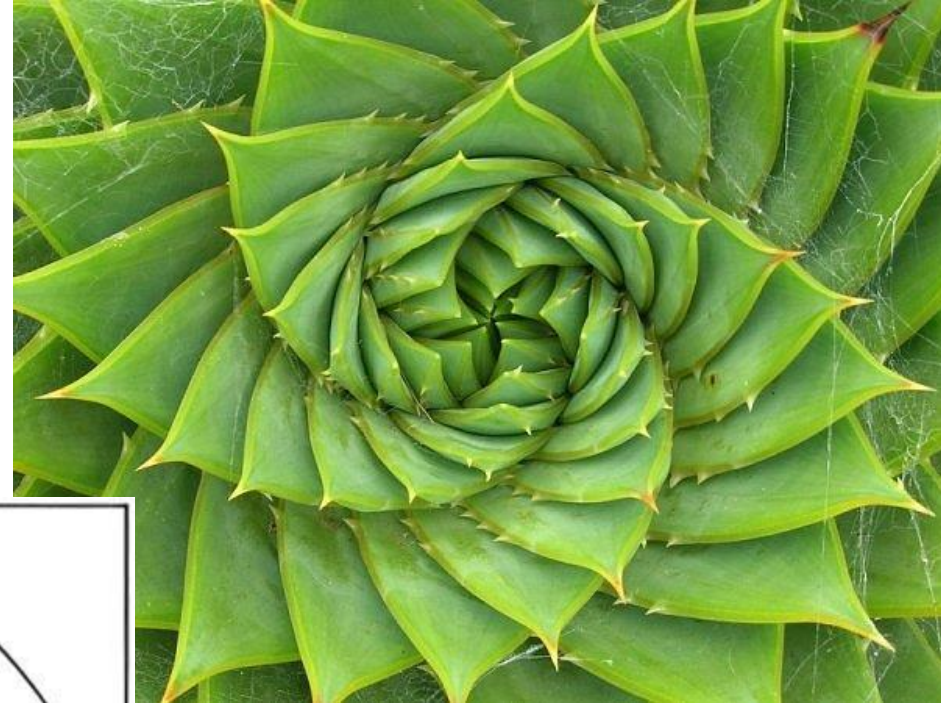
    n = Console.ReadLine()

    For n = 1 To 30
        Console.WriteLine(Fibonacci(n))
    Next
    Console.ReadKey()

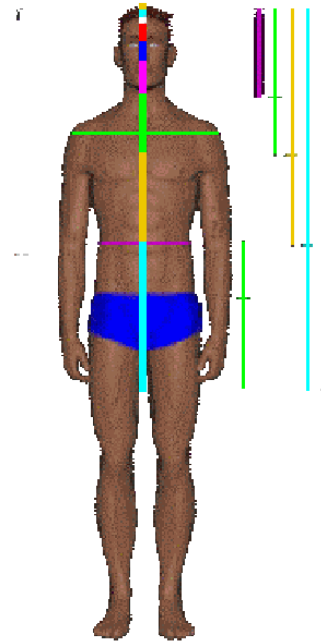
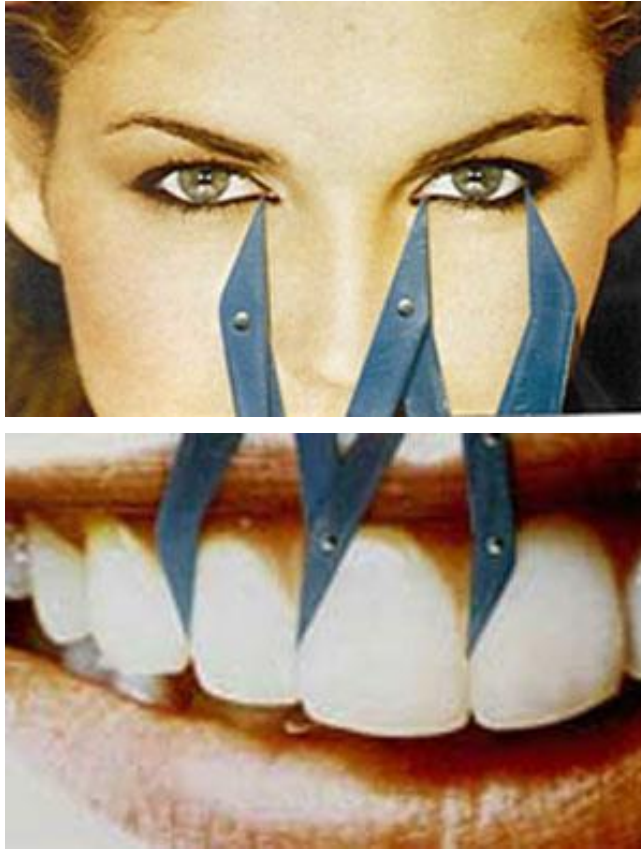
End Sub
End Module
```

Recursively Defined Sequences

Fibonacci numbers appear throughout nature



Is there an Biological Basis for Beauty?



The Golden Ratio (Fibonacci Numbers)

$$F(n) = \frac{\varphi^n - (1 - \varphi)^n}{\sqrt{5}} = \frac{\varphi^n - (-1/\varphi)^n}{\sqrt{5}}, \quad \text{where } \varphi \text{ is the golden ratio.}$$

Presentation Terminated