IT LEGEND

Create legends Programming

C# Statements and Control Flow

| www.itlegend.net | 01013855500 | Ceo@itlegend.net

Table of Content

What is C# Statements?

- A statement is a basic unit of execution of a program, and the program consists of multiple statements.
- There is many types of statements like
  - Declaration Statement
  - Expression Statement
  - (there are other types we will cover them later)

types of statements

Declaration

Expression

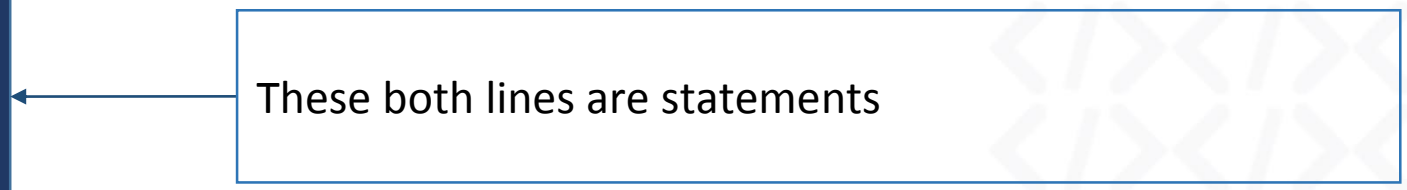
Control Flow

Method Calling

C# Statements Examples

Ex.

```
int age = 21;
int marks = 90;
```



C# Statements Examples

Declaration Statement

Declaration statements are used to declare and initialize variables.

```
char ch;
int maxValue = 55;
```

Both char ch;
and int maxValue = 55;
are declaration statements.

Expression Statement

An expression followed by a semicolon is called an expression statement.

```
area = 3.14 * radius * radius;
/* Method call is an expression*/
System.Console.WriteLine("Hello");
```

Here, 3.14 * radius * radius is an expression,
area = 3.14 * radius * radius;
is an expression statement.

System.Console.WriteLine("Hello");
is both an expression and a statement.

What is C# Blocks?

- A block is a combination of zero or more statements that is enclosed inside curly brackets { }

C# Block Examples

```
using System;

namespace Blocks
{
        class BlockExample
        {
                public static void Main(string[] args)
                {
                        double temperature = 42.05;
                        if (temperature > 32)
                        {       // Start of block
                                Console.WriteLine("Current temperature = {0}", temperature);
                                Console.WriteLine("It's hot");
                        }       // End of block
                }
        }
}
```

Types of Statements

Simple Statements

Are any expression that terminates with a semicolon

EX.

```
var1 = var2 + var3;
```

Such as, **Declaration Statements**

Compound Statements

Related statements can be grouped together in braces to form a compound statement or block

EX.

```
{
        int i = 4;
        Console.WriteLine (i);
}
```

Such as, **Blocks**

Control Statements

Are the statements that can change the flow of execution

Loops Statements

Jump Statements

Conditions Statements

Exception Handling Statements

What is C# Expressions?

- An expression in C# is a combination of operands(variables, literals, **Method calls**) and operators that can be evaluated to a single value. To be precise, an expression must have at least one operand but may not have any operator.

C# Expression Examples

Ex.1

```
double temperature;
temperature = 42.05;

int a, b, c, sum;
sum = a + b + c;
```

Here, 42.05 is an expression.
Also, temperature = 42.05 is an expression too.

There, a + b + c is an expression

Ex.2

```
if (age>=18 && age<58)
            Console.WriteLine("Eligible to work");
```

Here, (age>=18 && age<58) is an expression that returns a Boolean value.

"Eligible to work" is also an expression.

Operators

Arithmetic

Relational

Logical

Assignment

Arithmetic Operators

Assume variable A holds 10 and variable B holds 20

Operator	Description	Example
+	Adds two operands	A + B = 30
-	Subtracts second operand from the first	A - B = -10
*	Multiplies both operands	A * B = 200
/	Divides numerator by de-numerator	B / A = 2
%	Modulus Operator and remainder of after an integer division	B % A = 0
++	Increment operator increases integer value by one	A++ = 11
--	Decrement operator decreases integer value by one	A-- = 9

Arithmetic Operators Example

```
using System;
public class ArithmaticOperators
{
   static void Main(string[] args) {
      int a = 21;
      int b = 10;
      int c;

      c = a + b;
      Console.WriteLine("Line 1 - Value of c is {0}", c);

      c = a - b;
      Console.WriteLine("Line 2 - Value of c is {0}", c);

      c = a * b;
      Console.WriteLine("Line 3 - Value of c is {0}", c);
```

Note:

Rest of code in the next slide

IT LEGEND
Create legends Programming

Arithmetic Operators Example

```
        c = a / b;
        Console.WriteLine("Line 4 - Value of c is {0}", c);

        c = a % b;
        Console.WriteLine("Line 5 - Value of c is {0}", c);

        c = a++;
        Console.WriteLine("Line 6 - Value of c is {0}", c);

        c = a--;
        Console.WriteLine("Line 7 - Value of c is {0}", c);
        Console.ReadLine();
    }
}
```

Output:

```
Line 1 - Value of c is 31
Line 2 - Value of c is 11
Line 3 - Value of c is 210
Line 4 - Value of c is 2
Line 5 - Value of c is 1
Line 6 - Value of c is 21
Line 7 - Value of c is 22
```

Relational Operators

Assume variable A holds 10 and variable B holds 20

Operator	Description	Example
==	Checks if the values of two operands are equal or not, if yes then condition becomes true.	(A == B) is not true.
!=	Checks if the values of two operands are equal or not, if values are not equal then condition becomes true.	(A != B) is true.
>	Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true.	(A > B) is not true.
<	Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true.	(A < B) is true.
>=	Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true.	(A >= B) is not true.
<=	Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true.	(A <= B) is true.

Relational Operators Example

```
using System;
class RelationalOperators {
  static void Main(string[] args) {
    int a = 21;
    int b = 10;
    if (a == b) {
      Console.WriteLine("Line 1 - a is equal to b");
    } else {
      Console.WriteLine("Line 1 - a is not equal to b");}
    if (a < b) {
      Console.WriteLine("Line 2 - a is less than b");
    } else {
      Console.WriteLine("Line 2 - a is not less than b");}
    if (a > b) {
      Console.WriteLine("Line 3 - a is greater than b");
    } else {
      Console.WriteLine("Line 3 - a is not greater than b"); }
```

Output:

```
Line 1 - a is not equal to b
Line 2 - a is not less than b
Line 3 - a is greater than b
```

Logical Operators

Assume variable A holds Boolean value true and variable B holds Boolean value false

Operator	Description	Example
&&	Called Logical AND operator. If both the operands are non zero then condition becomes true.	(A && B) is false.
||	Called Logical OR Operator. If any of the two operands is non zero then condition becomes true.	(A || B) is true.
!	Called Logical NOT Operator. Use to reverses the logical state of its operand. If a condition is true then Logical NOT operator will make false.	!(A && B) is true.

Logical Operators Example

```
using System;
namespace logicalOperators {
  class LogicalOperatorsCls {
    static void Main(string[] args) {
      bool a = true;
      bool b = true;
      if (a && b) {
        Console.WriteLine("Line 1 - Condition is true");
      }
      if (a || b) {
        Console.WriteLine("Line 2 - Condition is true");
      }
```

Note:

Rest of code in the next slide

Logical Operators Example

```
/* lets change the value of  a and b */
        a = false;
        b = true;
        if (a && b) {
           Console.WriteLine("Line 3 - Condition is true");
        } else {
           Console.WriteLine("Line 3 - Condition is not true");
        }
        if (!(a && b)) {
           Console.WriteLine("Line 4 - Condition is true");
        }
        Console.ReadLine();      }
 }
}
```

Output:

```
Line 1 - Condition is true
Line 2 - Condition is true
Line 3 - Condition is not true
Line 4 - Condition is true
```

Assignment Operators

Operator	Description	Example
=	Simple assignment operator, Assigns values from right side operands to left side operand	C = A + B assigns value of A + B into C
+=	Add AND assignment operator, It adds right operand to the left operand and assign the result to left operand	C += A is equivalent to C = C + A
-=	Subtract AND assignment operator, It subtracts right operand from the left operand and assign the result to left operand	C -= A is equivalent to C = C - A
*=	Multiply AND assignment operator, It multiplies right operand with the left operand and assign the result to left operand	C *= A is equivalent to C = C * A
/=	Divide AND assignment operator, It divides left operand with the right operand and assign the result to left operand	C /= A is equivalent to C = C / A
%=	Modulus AND assignment operator, It takes modulus using two operands and assign the result to left operand	C %= A is equivalent to C = C % A

Assignment Operators

Operator	Description	Example
<<=	Left shift AND assignment operator	C <<= 2 is same as C = C << 2
>>=	Right shift AND assignment operator	C >>= 2 is same as C = C >> 2
&=	Bitwise AND assignment operator	C &= 2 is same as C = C & 2
^=	bitwise exclusive OR and assignment operator	C ^= 2 is same as C = C ^ 2
|=	bitwise inclusive OR and assignment operator	C |= 2 is same as C = C | 2

Assignment Operators Example

```
using System;
namespace assignmentOperators {
  class assignmentOperatorCls  {
    static void Main(string[] args) {
      int a = 21;
      int c;
      c = a;
      Console.WriteLine("Line 1 - =  Value of c = {0}", c);

      c += a;
      Console.WriteLine("Line 2 - += Value of c = {0}", c);

      c -= a;
      Console.WriteLine("Line 3 - -=  Value of c = {0}", c);
      c *= a;
      Console.WriteLine("Line 4 - *=  Value of c = {0}", c);
```

Note:

Rest of code in the next slide

Assignment Operators Example

```
c /= a;
Console.WriteLine("Line 5 - /=  Value of c = {0}", c)
c = 200;
c %= a;
Console.WriteLine("Line 6 - %=  Value of c = {0}", c);
c <<= 2;
Console.WriteLine("Line 7 - <<=  Value of c = {0}", c);
c >>= 2;
Console.WriteLine("Line 8 - >>=  Value of c = {0}", c);
c &= 2;
Console.WriteLine("Line 9 - &=  Value of c = {0}", c);
c ^= 2;
Console.WriteLine("Line 10 - ^=  Value of c = {0}", c);
c |= 2;
Console.WriteLine("Line 11 - |=  Value of c = {0}", c);
Console.ReadLine();}}}
```

Output:

```
Line 1 - =  Value of c = 21
Line 2 - += Value of c = 42
Line 3 - -=  Value of c = 21
Line 4 - *=  Value of c = 441
Line 5 - /=  Value of c = 21
Line 6 - %=  Value of c = 11
Line 7 - <<=  Value of c = 44
Line 8 - >>=  Value of c = 11
Line 9 - &=  Value of c = 2
Line 10 - ^=  Value of c = 0
Line 11 - |=  Value of c = 2
```

Miscellaneous Operators

Operator	Description	Example
sizeof()	Returns the size of a data type.	sizeof(int), returns 4.
typeof()	Returns the type of a class.	typeof(StreamReader);
&	Returns the address of an variable.	&a; returns actual address of the variable.
*****	Pointer to a variable.	*a; creates pointer named 'a' to a variable.
? :	Conditional Expression	If Condition is true ? Then value X : Otherwise value Y
is	Determines whether an object is of a certain type.	If(Ford is Car) // checks if Ford is an object of the Car class.
as	Cast without raising an exception if the cast fails.	Object obj = new StringReader("Hello");StringReader r = obj as StringReader;

Operator Precedence

Category	Operator	Associativity
Postfix	() [] -> . ++ - -	Left to right
Unary	+ - ! ~ ++ - - (type)* & sizeof	Right to left
Multiplicative	* / %	Left to right
Additive	+ -	Left to right
Shift	<< >>	Left to right
Relational	< <= > >=	Left to right
Equality	== !=	Left to right

Operator Precedence

Category	Operator	Associativity
Bitwise AND	&	Left to right
Bitwise XOR	^	Left to right
Bitwise OR	|	Left to right
Logical AND	&&	Left to right
Logical OR	||	Left to right
Conditional	?:	Right to left
Assignment	= += -= *= /= %=>>= <<= &= ^= |=	Right to left
Comma	,	Left to right

Loops Statements

while

do-while

for

Infinite Loop

IT LEGEND
Create legends Programming

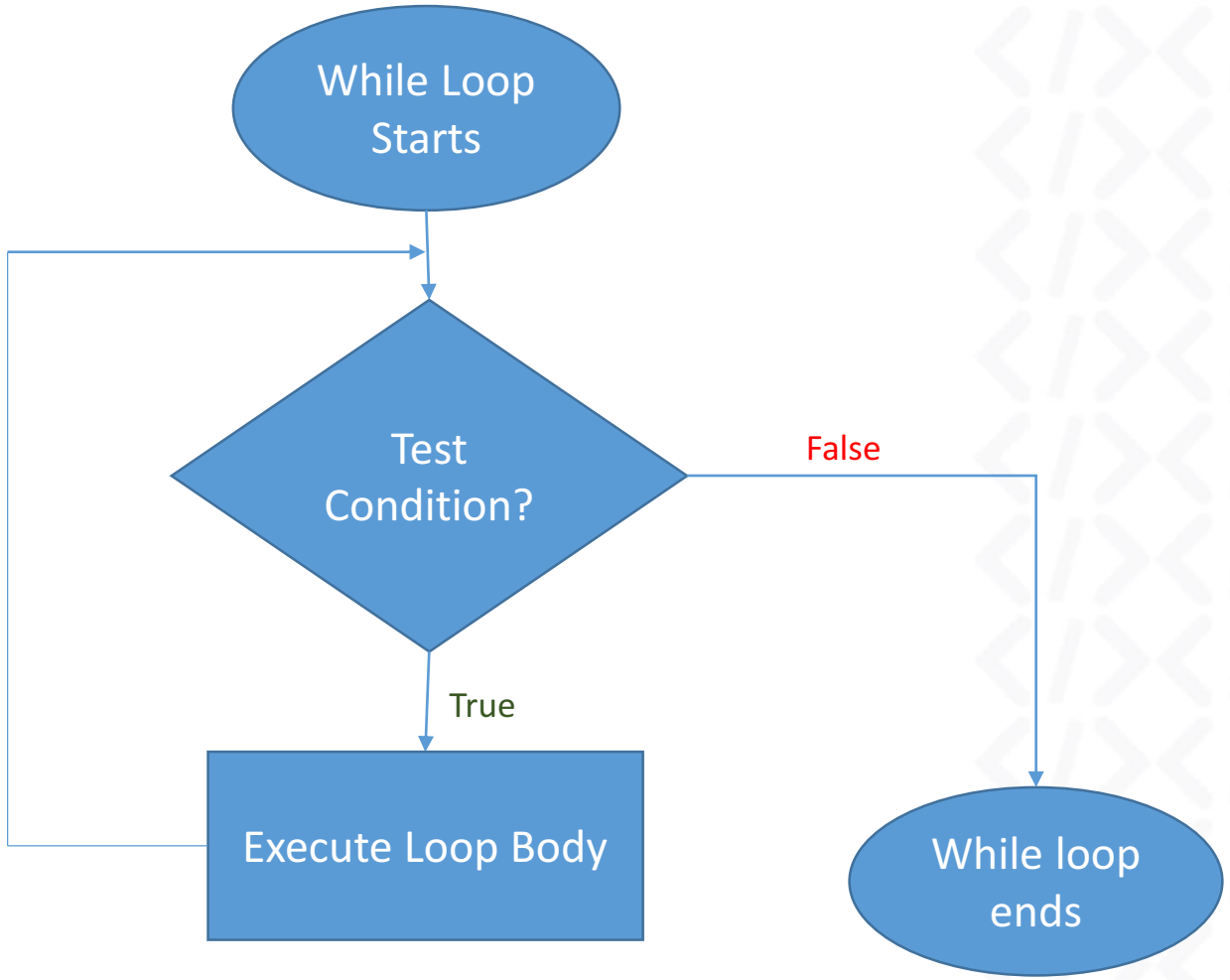
While Loop

- While Loop test condition is given in the beginning of the loop and all statements are executed till the given Boolean condition satisfies when the condition becomes false, the control will be out from the while loop.

Syntax:

```
while (boolean condition)
{
   loop statements...
}
```

While Loop Flowchart



While Loop Example

```
// C# program to illustrate while loop
using System;
public class whileLoopDemo
{
        public static void Main()
        {
                int x = 1;

                // Exit when x becomes greater than 4
                while (x <= 4)
                {
                        Console.WriteLine(“Welcome at IT LEGEND");
                        // Increment the value of x for
                        // next iteration
                        x++;
                }
        }
}
```

Output:

```
Welcome at IT LEGEND
Welcome at IT LEGEND
Welcome at IT LEGEND
Welcome at IT LEGEND
```

IT LEGEND
Create legends Programming

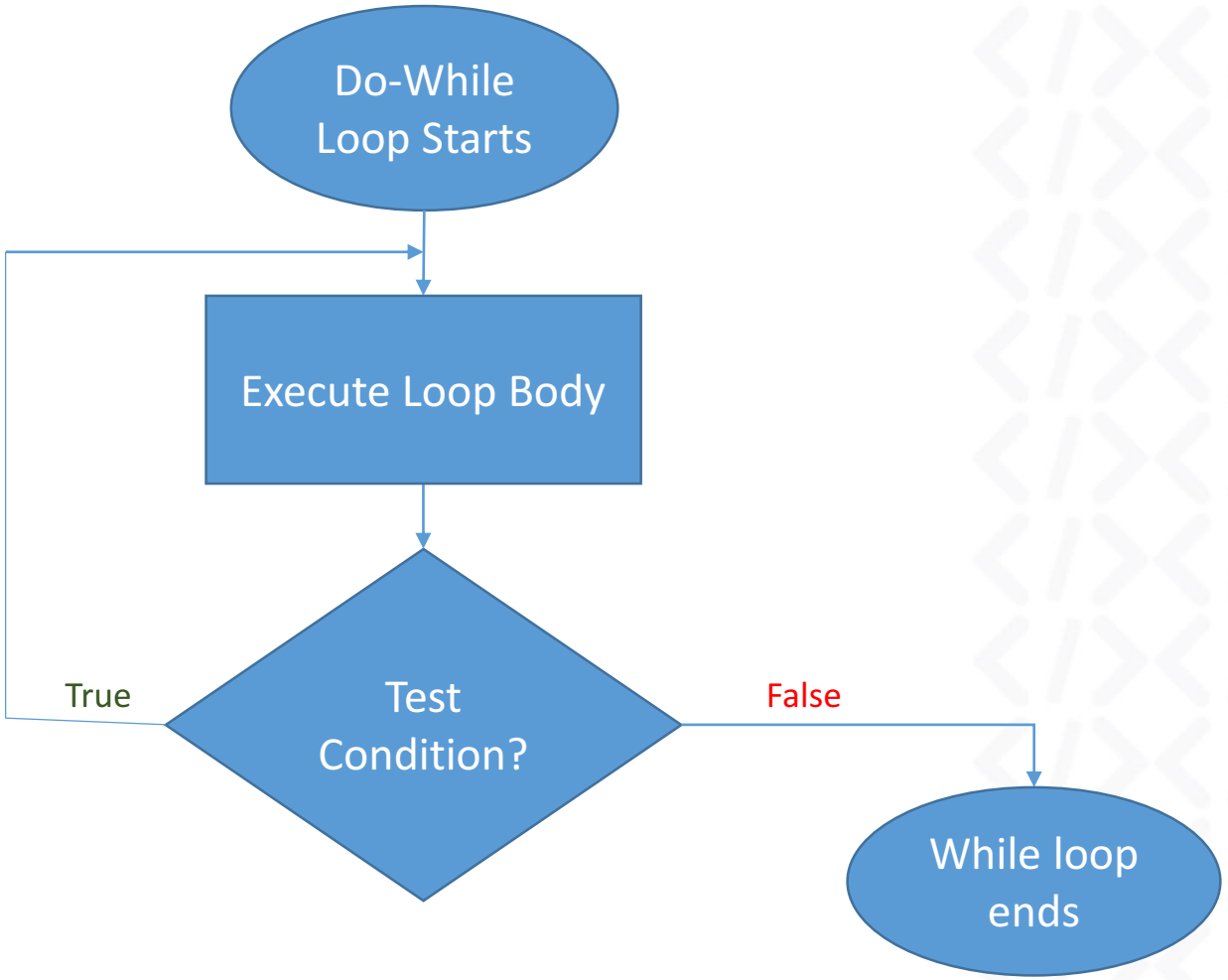
Do-While

- Do-while loop is similar to while loop with the only difference that it checks the condition after executing the statements, i.e it will execute the loop body one time for sure because it checks the condition after executing the statements.

Syntax:

```
do
{
   statements..
}while (condition);
```

Do-While Loop Flowchart



IT LEGEND
Create legends Programming

Do While Loop Example

```
// C# program to illustrate do-while loop
using System;

class dowhileloopDemo
{
    public static void Main()
    {
        int x = 21;
        do
        {
            // The line will be printed even
            // if the condition is false
            Console.WriteLine(“Welcome at IT LEGEND");
            x++;
        }
        while (x < 20);
    }
}
```

Output:

```
Welcome at IT LEGEND
```

IT LEGEND
Create legends Programming

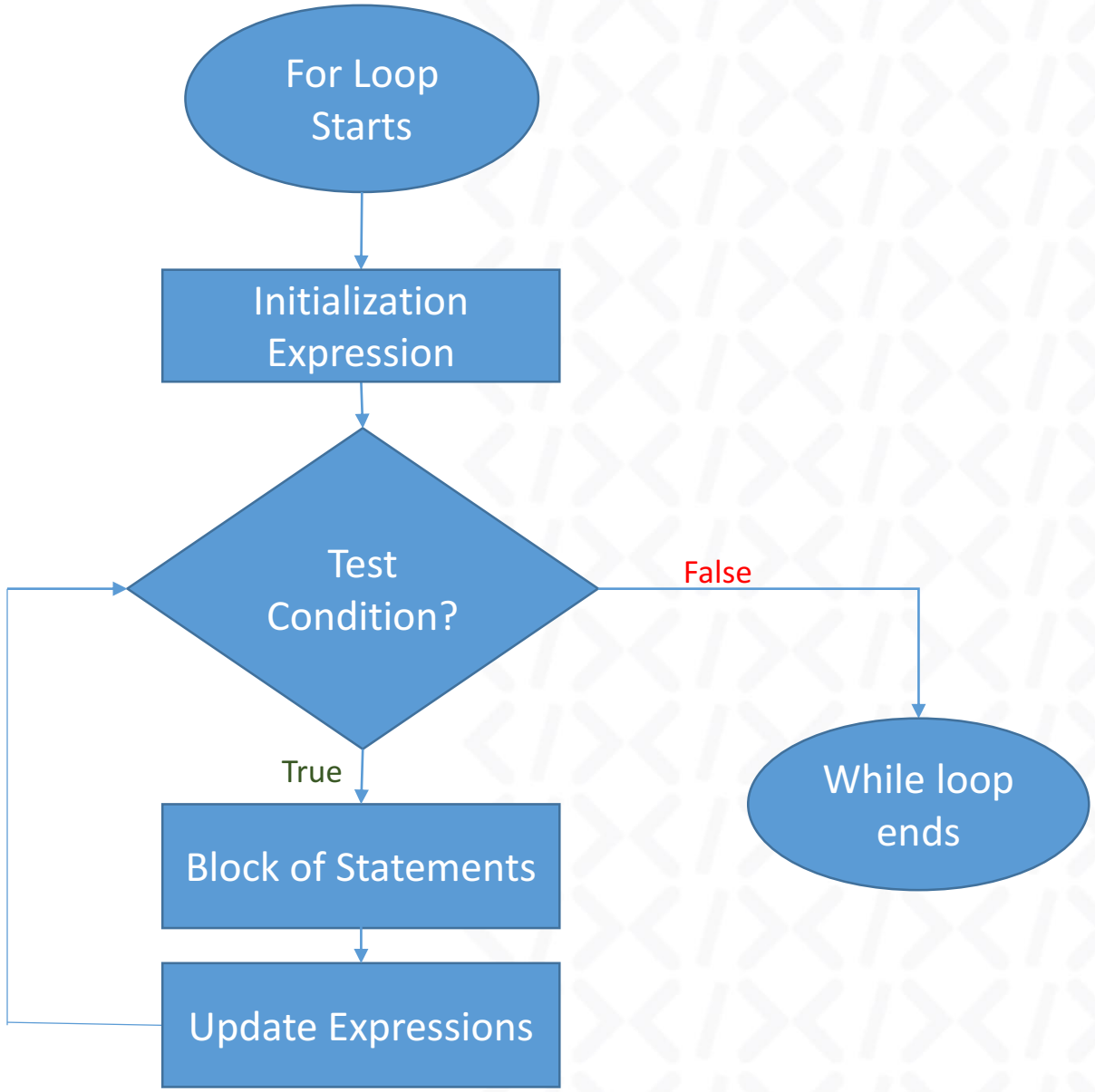
For Loop

- for loop has similar functionality as while loop but with different syntax. for loops are preferred when the number of times loop statements are to be executed is known beforehand. The loop variable initialization, condition to be tested, and increment/decrement of the loop variable is done in one line in for loop thereby providing a shorter, easy to debug structure of looping.
- Syntax:

```
for (loop variable initialization ; testing condition ; increment / decrement)
{
   // statements to be executed
}
```

For Loop Flowchart

1. Initialization of loop variable
2. Testing Condition
3. Increment / Decrement



IT LEGEND
Create legends Programming

For Loop Example

```
using System;
class forLoopDemo
{
    public static void Main()
    {
        // for loop begins when x=1
        // and runs till x <=4
        for (int x = 1; x <= 4; x++)
            Console.WriteLine("Welcome at IT LEGEND");
    }
}
```

Output:

```
Welcome at IT LEGEND
Welcome at IT LEGEND
Welcome at IT LEGEND
Welcome at IT LEGEND
```

Infinite Loop

- The loops in which the test condition does not evaluate false ever tend to execute statements forever until an external force is used to end it and thus they are known as infinite loops.

```
// C# program to demonstrate infinite loop
using System;

class infiniteLoop
{
        public static void Main()
        {
                // The statement will be printed
                // infinite times
                for(;;)
                Console.WriteLine(“Welcome at IT LEGEND");
        }
}
```

Output:

```
Welcome at IT LEGEND
Welcome at IT LEGEND
Welcome at IT LEGEND
Welcome at IT LEGEND
Welcome at IT LEGEND
Welcome at IT LEGEND
Welcome at IT LEGEND
Welcome at IT LEGEND
.....
```

Nested Loop

- When loops are present inside the other loops, it is known as nested loops.

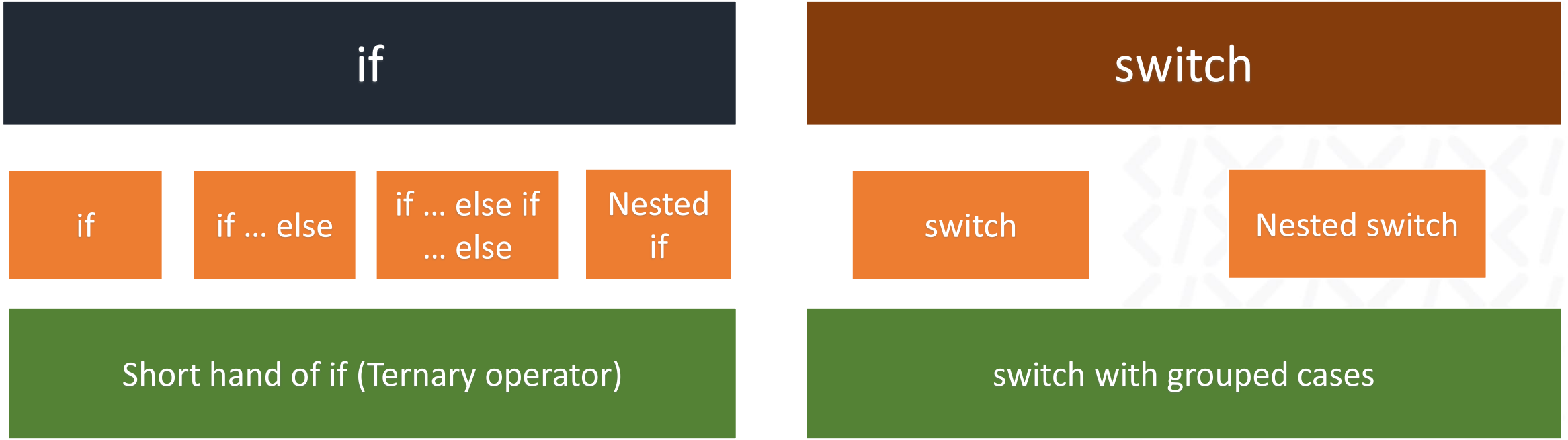
```
// C# program to demonstrate nested loops
using System;

class nestedLoops
{
        public static void Main()
        {
                // loop within loop printing Welcome at IT LEGEND
                for(int i = 2; i < 3; i++)
                        for(int j = 1; j < i; j++)
                                Console.WriteLine("Welcome at IT LEGEND");
        }
}
```

Output:

```
Welcome at IT LEGEND
```

Conditional(Selection) Statements



IT LEGEND
Create legends Programming

IF

- The if statement checks the given condition. If the condition evaluates to be true then the block of code/statements will execute otherwise not.

Syntax:

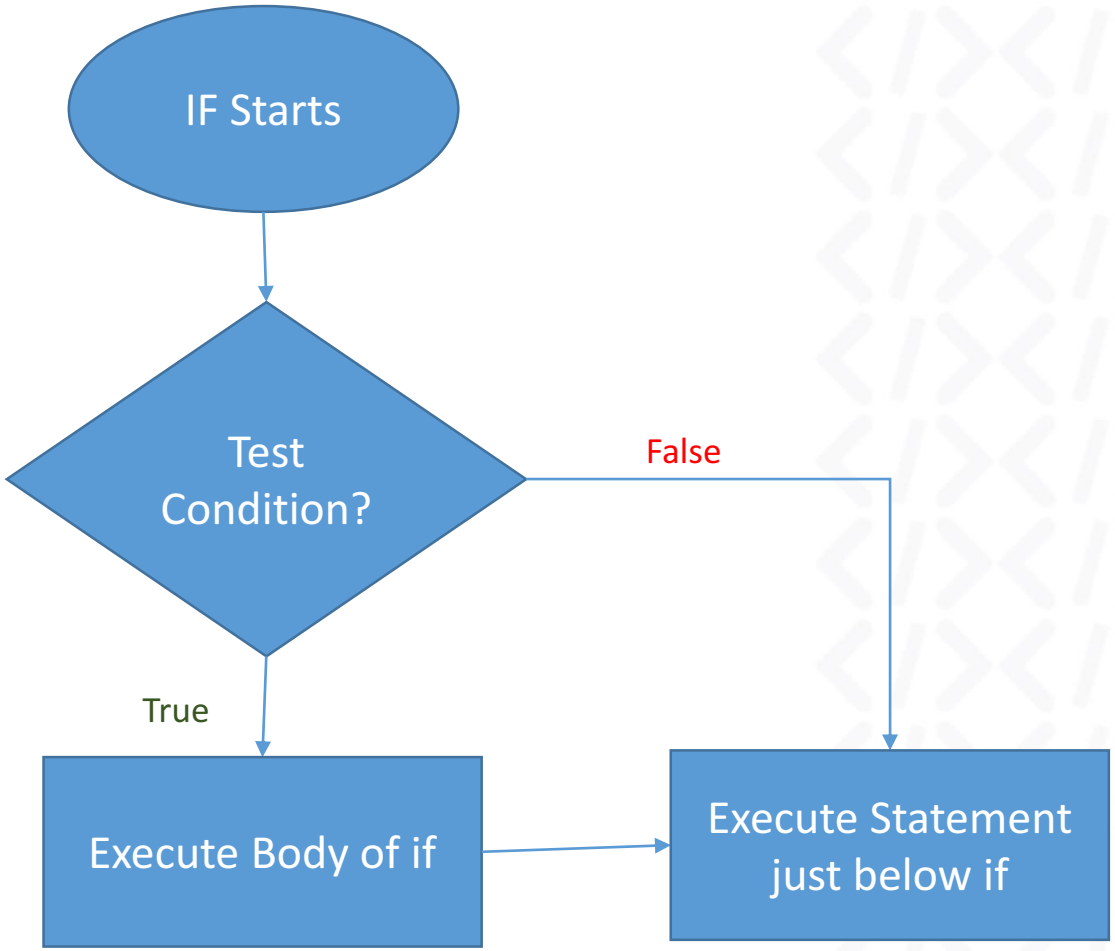
```
if(condition)
    {
       //code to be executed
    }
```

NOTE: If the curly brackets { } are not used with if statements then the statement just next to it is only considered associated with the if statement.

```
if (condition)
   statement 1;
statement 2;
```

IT LEGEND
Create legends Programming

IF Flowchart



IF Example

```
// C# program to illustrate if statement
using System;

        public static void Main(string[] args)
        {
                string name = "IT LEGEND";
                if (name == "IT LEGEND") {
                        Console.WriteLine("Welcome at IT LEGEND");
                }
        }
```

Output:

```
Welcome at IT LEGEND
```

IT LEGEND
Create legends Programming

IF-Else

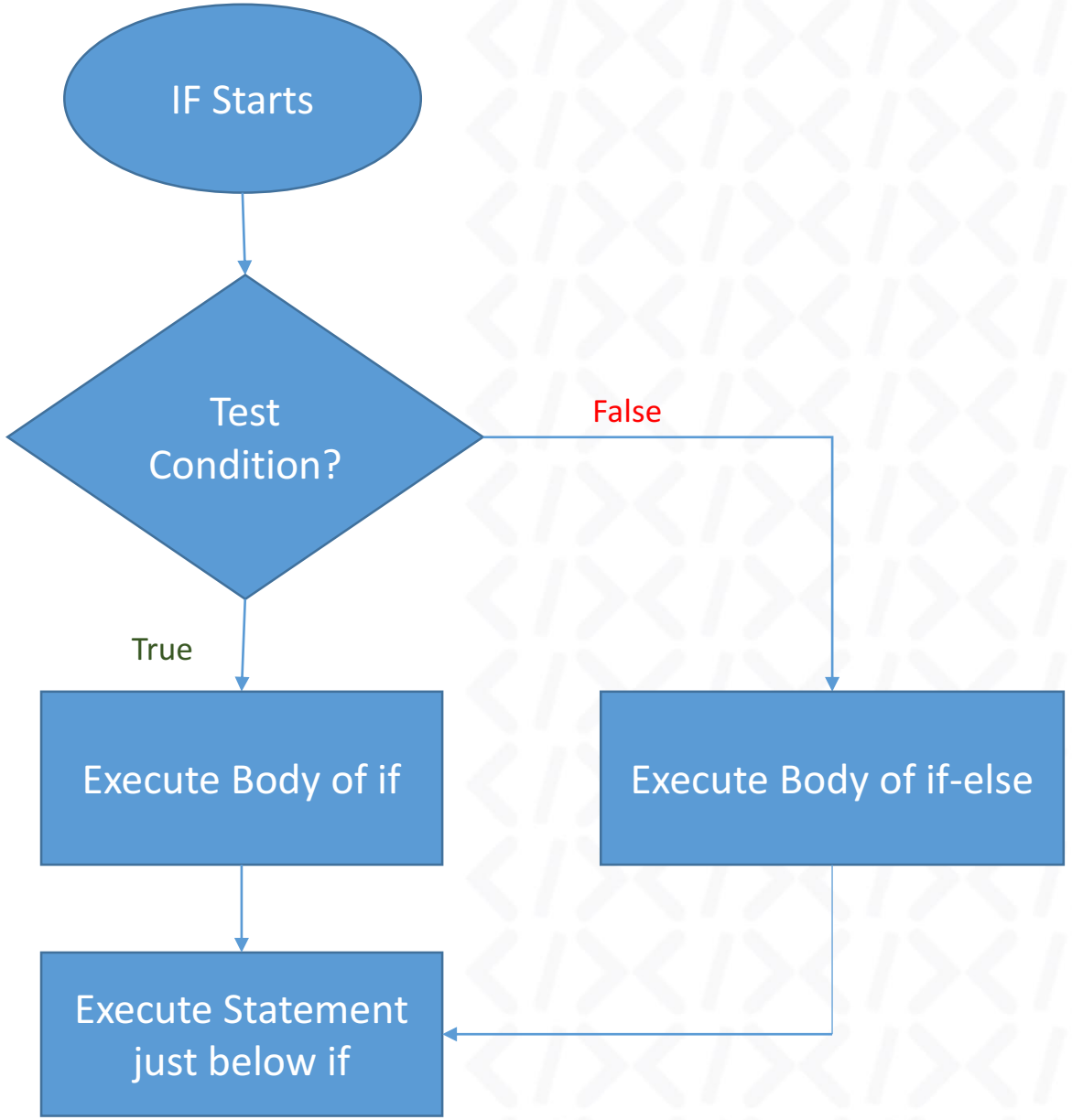
- The if statement evaluates the code if the condition is true but what if the condition is not true, here comes the else statement. It tells the code what to do when the if condition is false.

Syntax:

```
if(condition)
   {
     // code if condition is true
   }
   else
   {
     // code if condition is false
   }
```

IT LEGEND
Create legends Programming

IF-Else Flowchart



IT LEGEND
Create legends Programming

IF-Else Example

```
using System;
public static void Main(string[] args)
   {
      string name = “ITLegend";
      if (name == " ITLegend ") {
         Console.WriteLine("Welcome at IT LEGEND");
      }
      else {
         Console.WriteLine(“Not ITLegend");
      }
   }
```

Output:

Welcome at IT LEGEND

IT LEGEND
Create legends Programming

IF-Else If...-Else (Ladder Statement)

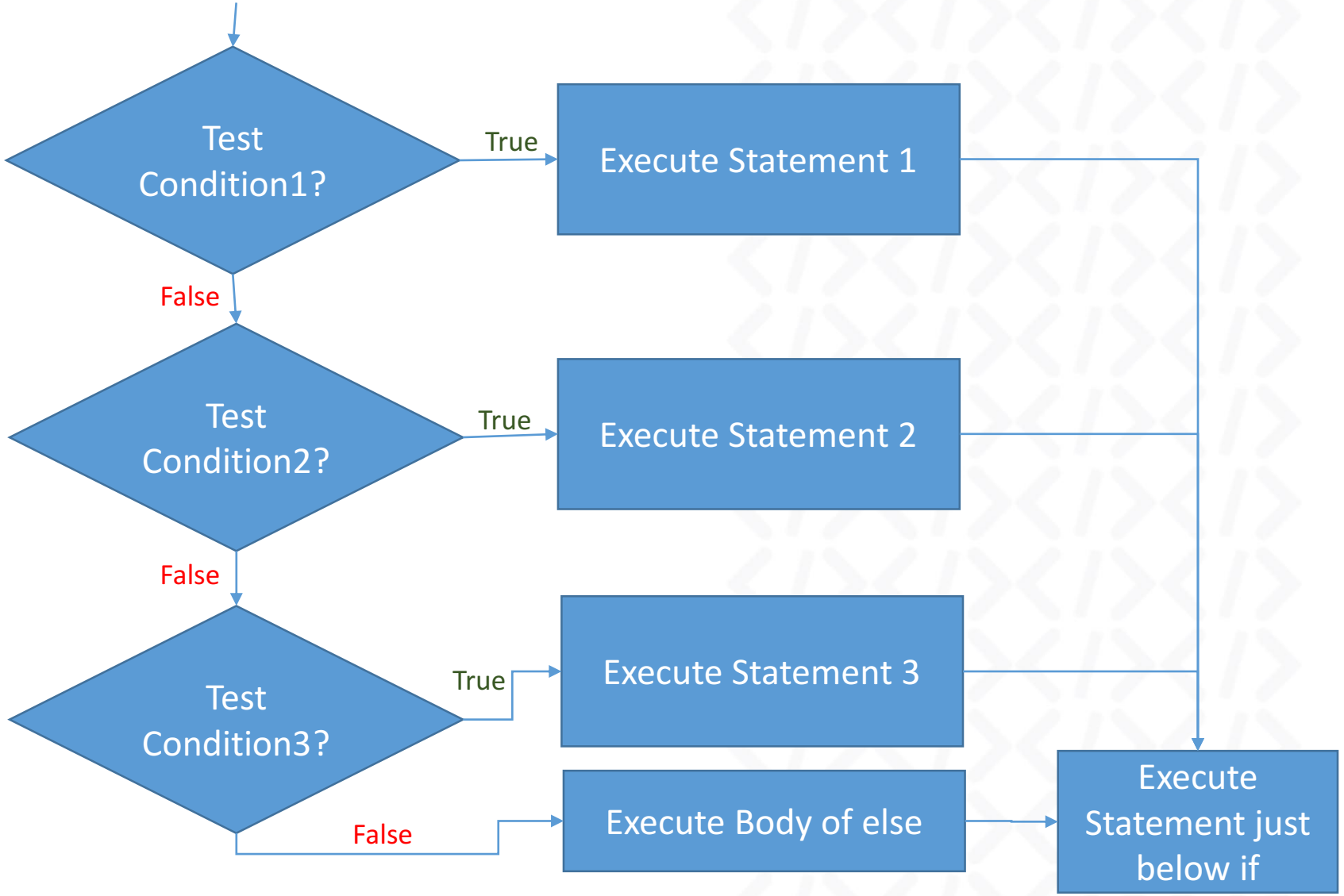
- The if-else-if ladder statement executes one condition from multiple statements. The execution starts from top and checked for each if condition. The statement of if block will be executed which evaluates to be true. If none of the if condition evaluates to be true then the last else block is evaluated

Syntax:

```
if(condition1)
{
      // Statement 1 executed if condition1 is true
}
else if(condition2)
{
       // Statement 2 executed if condition2 is true
}
else if(condition3)
{
       // Statement 3 executed if condition3 is true
}
...
else
{
        // code to be executed if all the conditions are false
}
```

IT LEGEND
Create legends Programming

IF-Else If-Else Flowchart



IF-Else If-Else Example

```
using System;

public static void Main(String[] args)
{
      int i = 20;

      if (i == 10)
         Console.WriteLine("i equals 10");
      else if (i == 15)
         Console.WriteLine("i equals 15");
      else if (i == 20)
         Console.WriteLine("i equals 20");
      else
         Console.WriteLine("i is not present");
}
```

Output:

```
i equals 20
```

IT LEGEND
Create legends Programming

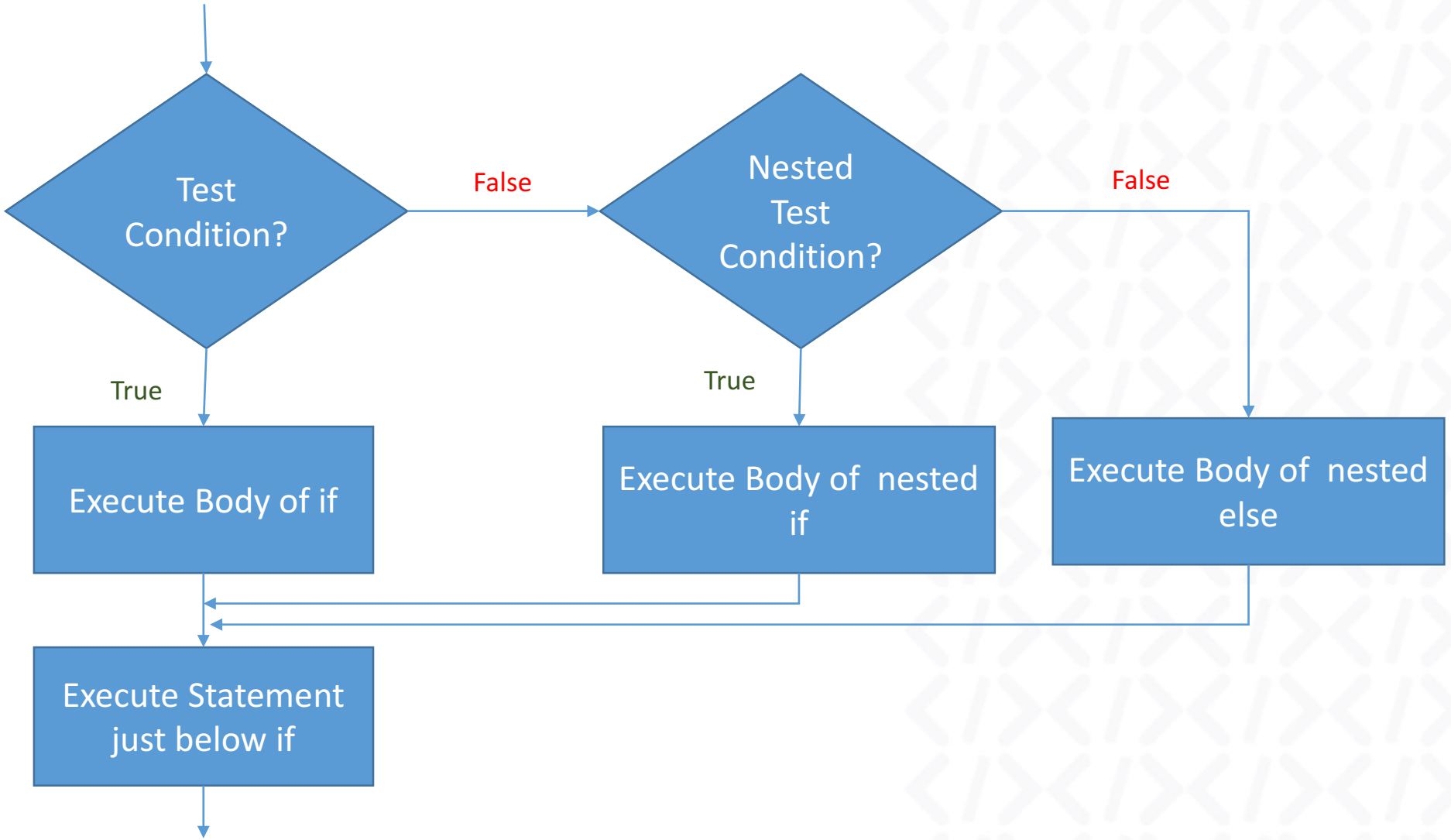
Nested – If

- if statement inside an if statement is known as nested if. if statement in this case is the target of another if or else statement. When more than one condition needs to be true and one of the condition is the sub-condition of parent condition, nested if can be used

Syntax:

```
if (condition1)
      {
          // code to be executed
          // if condition2 is true
          if (condition2)
          {
             // code to be executed
             // if condition2 is true
          }
      }
```

Nested-If Flowchart



IT LEGEND
Create legends Programming

Nested-if Example

```
using System;
 class GFG {
    public static void Main(String[] args)
    {
        int i = 10;
        if (i == 10) {

            // Nested - if statement
            // Will only be executed if statement
            // above it is true
            if (i < 12)
                Console.WriteLine("i is smaller than 12 too");
            else
                Console.WriteLine("i is greater than 15");
        }
    }
}
```

Output:

```
i is smaller than 12 too
```

IT LEGEND
Create legends Programming

Short-Hand of if-else (Ternary Operator(?:))

- basically used to replace multiples lines of codes with a single line. And it will return one of two values depending on the value of a Boolean expression

Syntax:

```
variable_name = (condition) ? TrueExpression :  FalseExpression;
```

Ternary Operator(?:) Example

```
using System;
class GFG{
static public void Main()
{

    // Declaring and initializing variables
    int x = 5;
    int y = 10;

    // Short-hand if-else statement
    string result = (x == y) ? "Both integers are equal" : "Not equal";

    // Display result
    Console.WriteLine(result);
}
}
```

Output:

```
Not equal
```

IT LEGEND
Create legends Programming

Nested Ternary Operator(?:) Example

```
using System;
class GFG{
static void Main(string[] args)
{

  int a = 23, b = 90;

  string result = a > b ? "a is greater than b" :
                  a < b ? "a is less than b" :
                  a == b ? "a is equal to b" : "Invalid";

  Console.WriteLine(result);
}
}
```

Output:

```
a is less than b
```

IT LEGEND
Create legends Programming

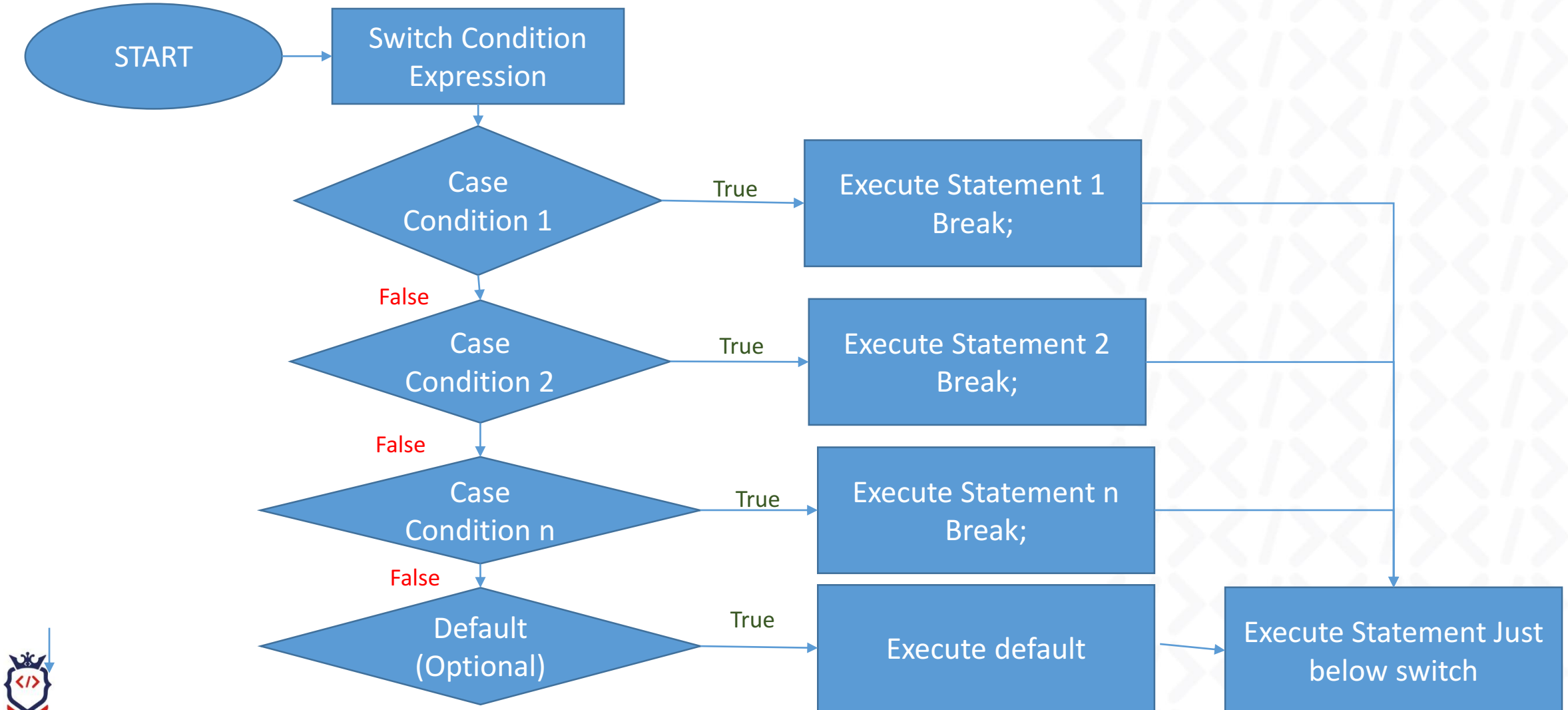
Switch

- Switch statement is an alternative to long if-else-if ladders. The expression is checked for different cases and the one match is executed. break statement is used to move out of the switch. If the break is not used, the control will flow to all cases below it until break is found or switch comes to an end. There is default case (optional) at the end of switch, if none of the case matches then default case is executed.

Syntax:

```
switch (expression)
 {
case value1: // statement sequence
            break;
case value2: // statement sequence
            break;
.
.
.
case value3: // statement sequence
            break;
default: // default statement sequence
}
```

Switch Flowchart



Switch Example

```
using System;
public class GFG
{ public static void Main(String[] args)
   {
      int number = 30;
      switch(number)
      {
      case 10: Console.WriteLine("case 10");
               break;
      case 20: Console.WriteLine("case 20");
               break;
      case 30: Console.WriteLine("case 30");
               break;
      default: Console.WriteLine("None matches");
               break;
      }
   }
}
```

Output:

```
case 30
```

Nested Switch Example

```
using System;
public class NestedSwitch
{public static void Main(String[] args)
   {   int j = 5;
       switch (j)
       {case 5: Console.WriteLine(5);
                switch (j - 1)
               {case 4: Console.WriteLine(4);
                        switch (j -1)
                       {case 3: Console.WriteLine(3);
                                break;}
                 break;}
              break;
          case 10: Console.WriteLine(10);
                   break;
          case 15: Console.WriteLine(15);
                   break;
          default: Console.WriteLine(100);
                   break;}}}
```

Output:

```
5
4
3
```

Grouped Cases Switch Example

```
using System;
{  public class SwitchCase
   {public static void Main(string[] args)
      {char ch;
         Console.WriteLine("Enter an alphabet");
         ch = Convert.ToChar(Console.ReadLine());
         switch(Char.ToLower(ch))
         {  case 'a':
            case 'e':
            case 'i':
            case 'o':
            case 'u':
               Console.WriteLine("Vowel");
               break;
            default:
               Console.WriteLine("Not a vowel");
               break;}}}}
```

Input: a

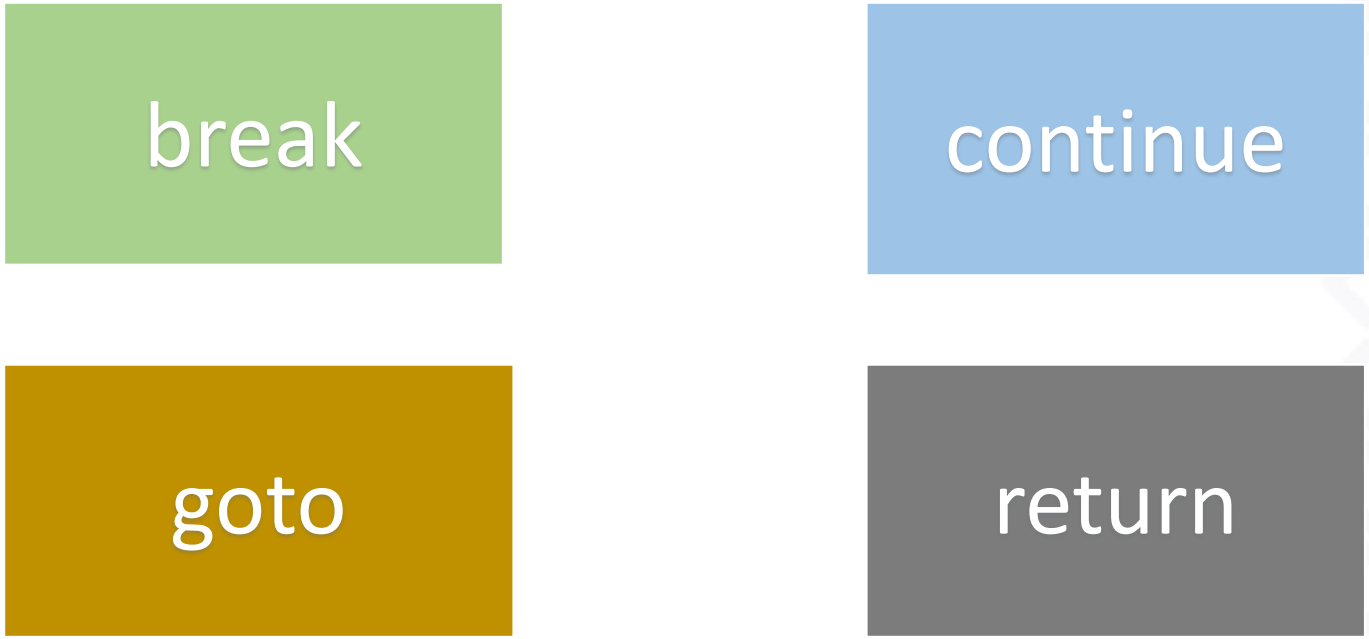
Output:

Vowel

IT LEGEND
Create legends Programming

Jump Statements

- Jump statements are used to transfer control from one point to another point in the program due to some specified code while executing the program.



Break Statement

- The break statement is used to terminate the loop or statement in which it present. After that, the control will pass to the statements that present after the break statement, if available. If the break statement present in the nested loop, then it terminates only those loops which contains break statement.

Break Example

```
using System;
class breakExample{
    // Main Method
    static public void Main()
    {
        // Welcome ItLegend is printed only 2 times
        // because of break statement
        for (int i = 1; i < 4; i++)
        {
            if (i == 3)
                break;
            Console.WriteLine("Welcome ItLegend");
        }
    }
}
```

Output:

```
Welcome ItLegend
Welcome ItLegend
```

IT LEGEND
Create legends Programming

Continue Statement

- This statement is used to skip over the execution part of the loop on a certain condition. After that, it transfers the control to the beginning of the loop. Basically, it skips its following statements and continues with the next iteration of the loop.

Continue Example

```
using System;
class ContinueExample{
    // Main Method
    public static void Main()
    {
        // This will skip 4 to print
        for (int i = 1; i <= 10; i++) {
            // if the value of i becomes 4 then
            // it will skip 4 and send the
            // transfer to the for loop and
            // continue with 5
            if (i == 4)
                continue;
            Console.WriteLine(i);
        }
    }
}
```

Output:

```
1
2
3
5
6
7
8
9
10
```

IT LEGEND
Create legends Programming

goto Statement

- This statement is used to transfer control to the labeled statement in the program. The label is the valid identifier and placed just before the statement from where the control is transferred.

goto Example

```
using System;
class gotoExample{
    static public void Main()
    {int number = 20;
        switch (number) {
        case 5:
            Console.WriteLine("case 5");
            break;
        case 10:
            Console.WriteLine("case 10");
            break;
        case 20:
            Console.WriteLine("case 20");
            // goto statement transfer
            // the control to case 5
            goto case 5;
        default:
            Console.WriteLine("No match found");}}}
```

Output:

```
case 20
Case 5
```

IT LEGEND
Create legends Programming

return Statement

- This statement terminates the execution of the method and returns the control to the calling method. It returns an optional value. If the type of method is void, then the return statement can be excluded.

return Example

```
using System;
class returnExample{
   // creating simple addition function
   static int Addition(int a)
   {
      // add two value and
      // return the result of addition
      int add = a + a;
      // using return statement
      return add;
   }
   // Main Method
   static public void Main()
   {
      int number = 2;
      // calling addition function
      int result = Addition(number);
      Console.WriteLine("The addition is {0}", result);}}
```

Output:

```
The addition is 4
```

IT LEGEND
Create legends Programming

Exception Handling Statements

- An exception is defined as an event that occurs during the execution of a program that is unexpected by the program code. The actions to be performed in case of occurrence of an exception is not known to the program. In such a case, we create an exception object and call the exception handler code. The execution of an exception handler so that the program code does not crash is called exception handling. Exception handling is important because it gracefully handles an unwanted event, an exception so that the program code still makes sense to the user.

Exception Handling Statements

Keyword	Definition
try	Used to define a try block. This block holds the code that may throw an exception.
catch	Used to define a catch block. This block catches the exception thrown by the try block.
finally	Used to define the finally block. This block holds the default code.
throw	Used to throw an exception manually.