

DIFFERENCE AND SIMILARITIES

C C++ C# Java Python



Created By :- Aniket Potdar

ABOUT LANGUAGE

1] C

C is a Programming language developed at AT & T's Bell Laboratories of USA in 1972. It was designed and written by a man named Dennis Ritchie. C became popular because it is reliable, simple, and easy to use. In an industry where newer language, tools and technologies emerge and vanish day in and day out, a language that has survived for more than three decades has to be really good. It is a structured programming language that is machine-independent and extensively used to write various applications, Operating Systems like Windows, and many other complex programs like Oracle database, Git, Python interpreter, and more.

2] C++

C++ is a general-purpose programming language created by Bjarne Stroustrup as an extension of the C programming language, or "C with Classes". The language has expanded significantly over time, and modern C++ now has object-oriented, generic, and functional features in addition to facilities for low-level memory manipulation. It is almost always implemented as a compiled language, and many vendors provide C++ compilers, including the Free Software Foundation, LLVM, Microsoft, Intel, Oracle, and IBM, so it is available on many platforms

3] C#

C# was designed by Anders Hejlsberg. C# a general-purpose, multi-paradigm programming language encompassing static typing, strong typing, lexically scoped, imperative, declarative, functional, generic, object-oriented (class-based), and component-oriented programming disciplines. C# was developed around 2000 by Microsoft as part of its .NET initiative and later approved as an international standard by Ecma (ECMA-334) in 2002 and ISO (ISO/IEC 23270) in 2003.

4] Java

Java is a class-based, object-oriented programming language that is designed to have as few implementation dependencies as possible. It is a general-purpose programming language intended to let application developers write once, run anywhere (WORA),[16] meaning that compiled Java code can run on all platforms that support Java without the need for recompilation.[17] Java applications are typically compiled to bytecode that can run on any Java virtual machine (JVM) regardless of the underlying computer architecture. The syntax of Java is similar to C and C++, but has fewer low-level facilities than either of them

5] Python

Python was designed by Guido van Rossum. Python is an interpreted high-level general-purpose programming language. Python's design philosophy emphasizes code readability with its notable use of significant indentation. Its language constructs as well as its object-oriented approach aim to help programmers write clear, logical code for small and large-scale projects. Python is dynamically-typed and garbage-collected. It supports multiple programming paradigms, including structured (particularly, procedural), object-oriented and functional programming.

INDEX

Sr.No	Title	Page No
1	Benefits of Languages	4
2	Difference and Similarities	7
3	Hello World Program	9
4	Data Types	12
5	Keywords	15
6	Operators	18
7	Arrays	23
8	Conditions :- 1. If else Condition 2. else if Condition 3.Nested if Condition 4. Switch Condition	26
9	Loops :- 1. While Loop 2. Do while Loop 3. For Loop	38
10	Object Oriented Programming	47

BENEFITS OF LANGUAGES

1] C

- As a middle-level language, C combines the features of both high-level and low-level languages. It can be used for low-level programming, such as scripting for drivers and kernels and it also supports functions of high-level programming languages, such as scripting for software applications etc
 - C language has a rich library which provides a number of built-in functions. It also offers dynamic memory allocation.
 - C is highly portable and is used for scripting system applications which form a major part of Windows, UNIX, and Linux operating system.
-

2] C++

- The global data and global functions are used within C++ that aren't utilized in many other high-level languages within the pc sciences and it is an advantage to the programming languages.
 - C++ program is useful for low-level programming language and really efficient for general purposes, it offers performance and memory efficiently, It offers high-level abstraction, within the language of the matter domain
 - C++ may be a system programming and features a relatively clear and mature standard
-

3] C#

- Interoperability is the process that enables the C# programs to do almost anything that a native C++ application can do. In brief, language interoperability is the ability of code to interact with code that is written using a different programming language. It can help maximize code reuse and, therefore, improve the efficiency of the development process.
 - C# language is type-safe code that can only access the memory location and has permission to execute. Therefore, it improves the security of the program
 - The major benefit of C# language is its strong memory backup. C# programming language contains high memory backup so that memory leakage problem and other such types of problem is not occurring as it happens in the case of C++ language.
-

4] Java

- Java is a secured programming language because it doesn't use Explicit pointers. Also, Java programs run inside the virtual machine sandbox. JRE also provides a classloader, which is used to load the class into JVM dynamically. It separates the class packages of the local file system from the ones that are being imported from the network.
 - Java uses a multi-threaded environment in which a bigger task can be converted into various threads and run separately. The main advantage of multi-threading is that we need not provide memory to every running thread.
 - The major benefit of C# language is its strong memory backup. C# programming language contains high memory backup so that memory leakage problem and other such types of problem is not occurring as it happens in the case of C++ language.
-

5] Python

- Python integrates the Enterprise Application Integration that makes it easy to develop Web services by invoking COM or COBRA components. It has powerful control capabilities as it calls directly through C, C++ or Java via Jython. Python also processes XML and other markup languages as it can run on all modern operating systems through same byte code.
 - With its strong process integration features, unit testing framework and enhanced control capabilities contribute towards the increased speed for most applications and productivity of applications. It is a great option for building scalable multi-protocol network applications.
 - The language has extensive support libraries and clean object-oriented designs that increase two to ten fold of programmer's productivity while using the languages like Java, VB, Perl, C, C++ and C#.
-

COMPUTER LANGUAGES

Difference and Similarities



Developed Year	1972	1979	1998	1991	1991
Developed By	Dennis Ritchie	Bjarne Stroustrup	Anders Hejlsberg	James Gosling	Guido van Rossum
Programming Paradigm	Procedural language	Object-Oriented Programming (OOP)	component oriented programming language	Pure Object-Oriented Programming (OOP)	Object-Oriented Programming (OOP)
Comments	Single Line Comment: // Multiple Line Comment: /* */	Single Line Comment: // Multiple Line Comment: /* */	Single Line Comment: // Multiple Line Comment: /* */	Single Line Comment: // Multiple Line Comment: /* */	Single Line Comment: # Multiple Line Comment: """ """
Variables	Declaration: int myNum = 15;	Declaration: int myNum = 15;	Declaration: int myNum = 15;	Declaration: int myNum = 15;	Declaration: x = 5
Type Casting	when you assign a value of one primitive data type to another type Example: int i = 17; char c = 'c'; int sum; sum = i + c; printf("Value of sum : %d\n", sum);	when you assign a value of one primitive data type to another type Example: int x = 10; char y = 'a'; x = x + y; float z = x + 1.0; cout << "x = " << x << endl << "y = " << y << endl << "z = " << z << endl;	when you assign a value of one primitive data type to another type Example: int myInt = 10; Console.WriteLine (Convert.ToString(myInt));	when you assign a value of one primitive data type to another type. Example: int myInt = 9; double myD = myInt; System.out.println(myD);	Casting in python is therefore done using constructor functions Example: x = int(1) # x will be 1 y = int(2.8) # y will be 2 z = int("3") # z will be 3



Modifiers	Access Modifiers: public , private, protected, internal Non-Access Modifiers: final, static, abstract, transient,synchronized, volatile, Strictfp	Access Modifiers: public , private, protected	Access Modifiers: public , private, protected, internal	Access Modifiers: public , private, protected, default Non-Access Modifiers: final, static, abstract, transient, synchronized, volatile	Access Modifiers: public , private, protected, default
Origin	Based on Assembly Language	Based on C Language	Based on C and Java Language	Based on C and C++ Language	Inspiration from C, Java, Perl, C++ and Lisp
Translator	Compiler Only	Compiler Only	Integrated Development Environment (IDE)	Interpreted Language (Compiler + Interpreter)	Interpreted Language
Platform Dependency	Platform Dependent	Platform Dependent	Platform Independent	Platform Independent	Platform Independent
File Generation	.exe files	.exe files	.class files	.class files	.py files
Database Connectivity	Not Supported	Not Supported	Supported	Supported	Supported
Frameworks	No Framework	Qt Framework	.Net Framework	Spring & Hibernate	Django
Databases Supported	MySQL	SQLAPI++	Microsoft SQL Server	Oracle, MySQL, Java DB, PostgreSQL, Sybase	SQLite, MySQL, Oracle, Sybase, PostgreSQL

HELLO WORLD PROGRAM

1] C

```
#include <stdio.h>
int main() {
    // printf() displays the string inside quotation
    printf("Hello, World!");
    return 0;
}
```

2] C++

```
#include <iostream>
Using namespace std;

int main() {
    // cout << displays the string inside quotation
    Cout << "Hello, World!"
    return 0;
}
```

3] C#

```
using System;

namespace HelloWorld

{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Hello World!");
        }
    }
}
```

4] Java

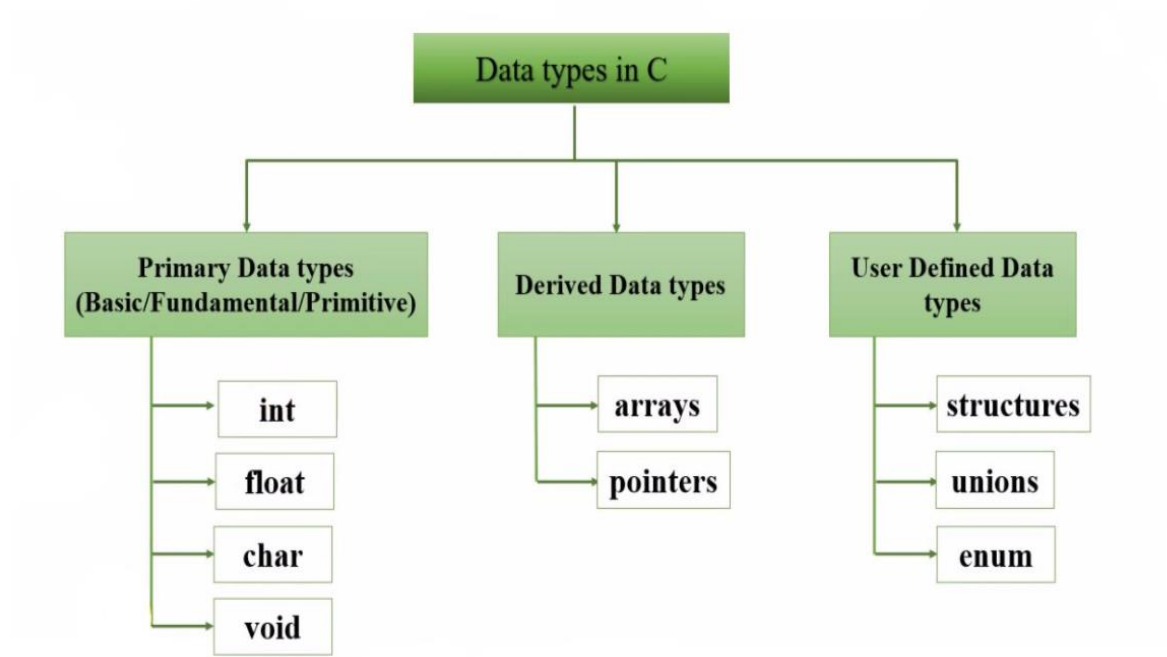
```
public class Main {
    public static void main(String[] args) {
        System.out.println("Hello World");
    }
}
```

5] Python

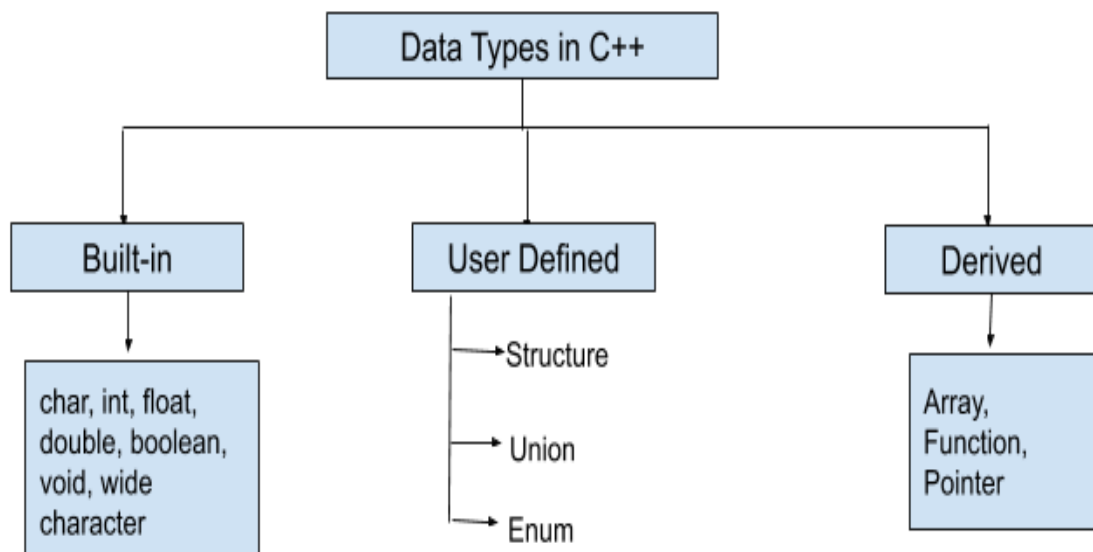
```
print("Hello, World!")
```

DATA TYPES

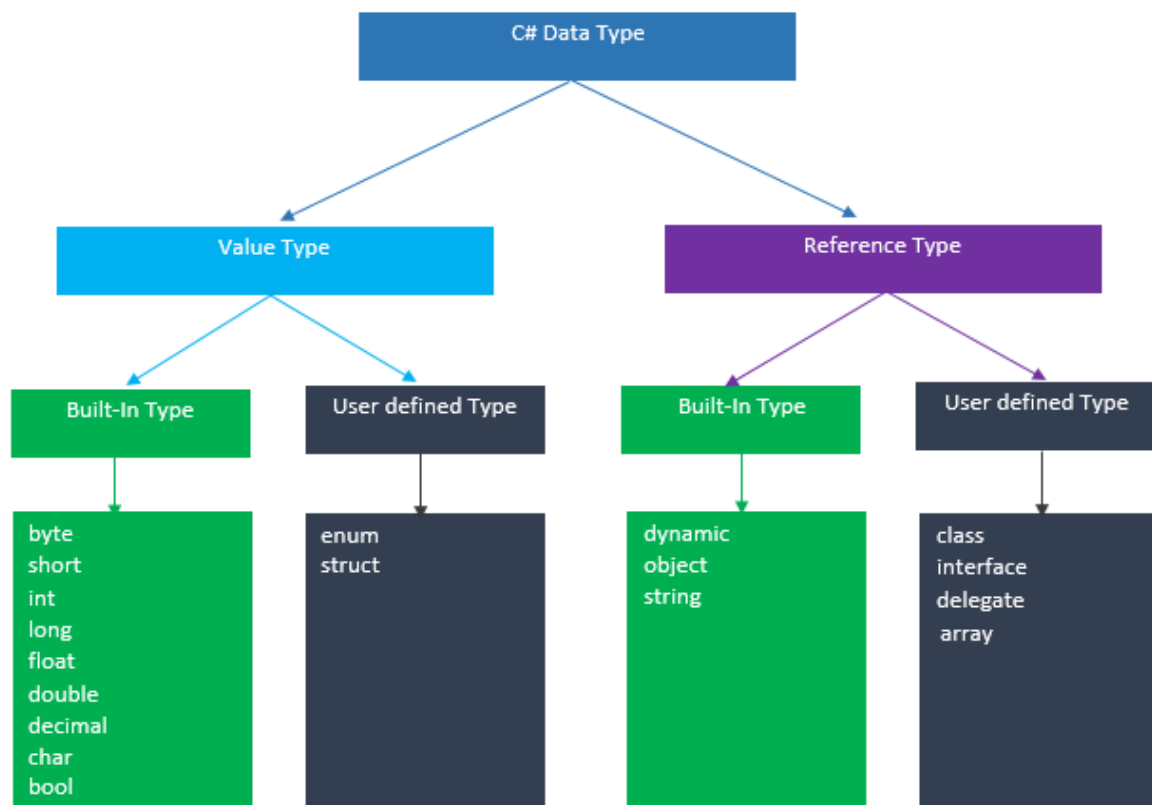
1] C



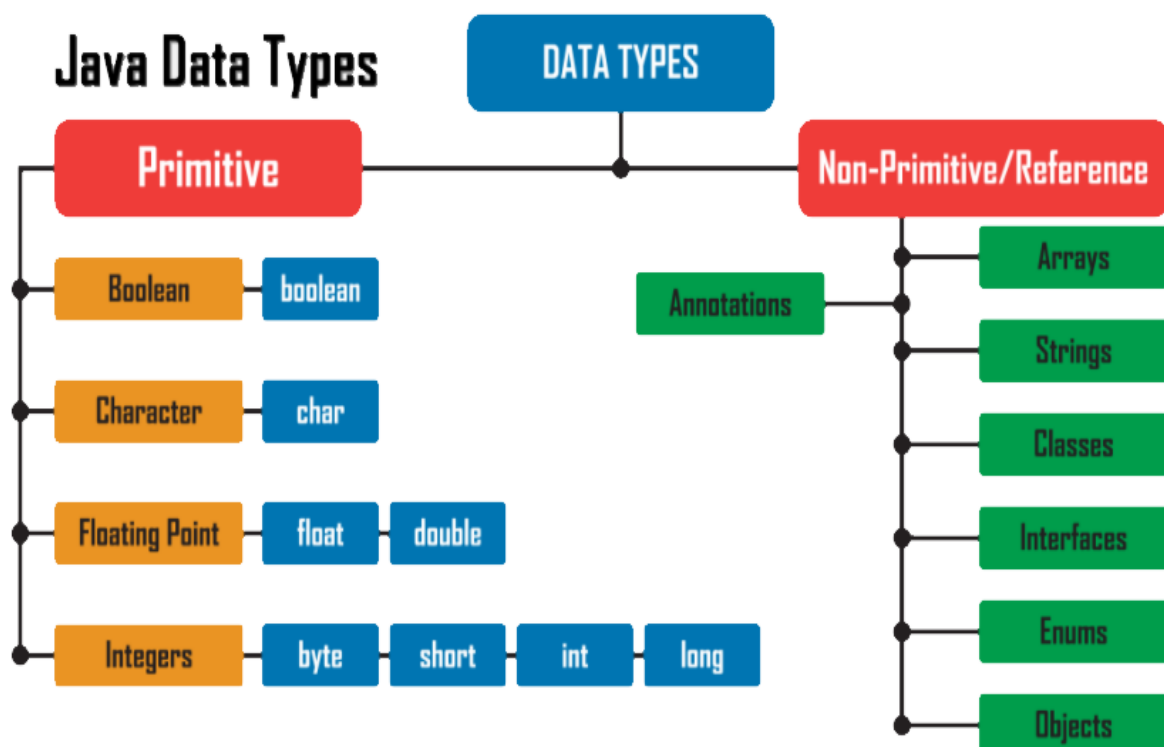
2] C++



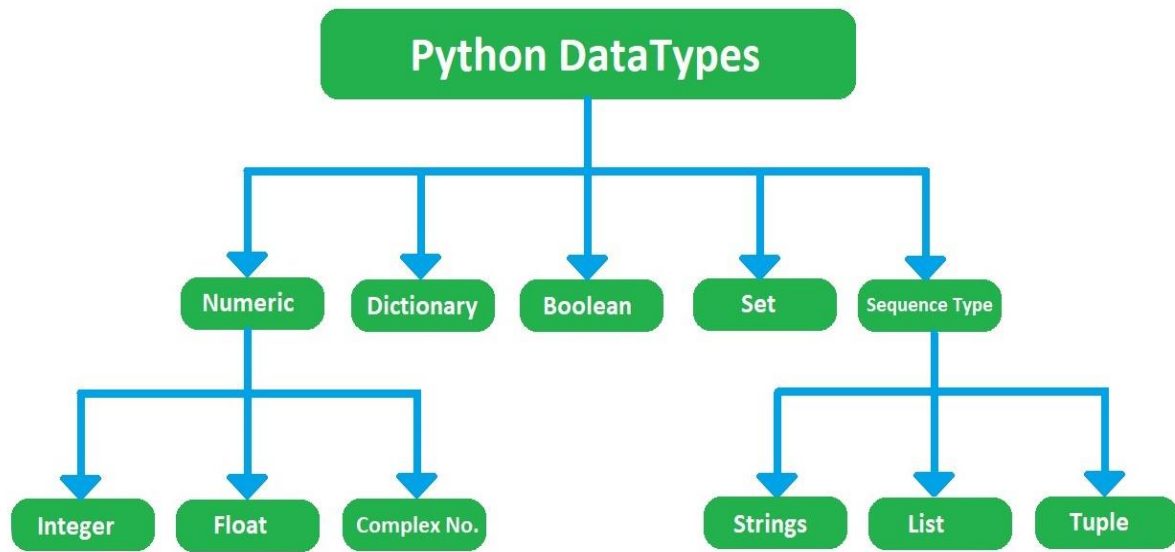
3] C#



4] Java



5] Python



KEYWORDS

1] C

Keyword in C

auto	double	int	struct	while	static	void
break	else	long	switch	continue	if	goto
case	enum	register	typedef	sizeof	volatile	
char	extern	return	union	default	signed	
const	float	short	do	unsigned	for	

2] C++

Keywords in C++

Keywords in C++ A keyword is a reserved word. You cannot use it as a variable name, constant name etc. A list of 32 Keywords in C++ Language which are also available in C language

auto	break	case	char	const	continue	default	do
double	else	enum	extern	float	for	goto	if
int	long	register	return	short	signed	sizeof	static
struct	switch	typedef	union	unsigned	void	volatile	while

Keywords in C++

Keywords in C++ A keyword is a reserved word. You cannot use it as a variable name, constant name etc. A list of 30 Keywords in C++ Language which are not available in C

asm dynamic_cast namespace reinterpret_cast bool explicit
 new static_cast false catch operator template friend
 private class this inline public throw
 const_cast delete mutable protected
 true try typeid typename using virtual wchar_t

3] C#


C# Keywords

Abstract	As	Base	Bool	Break	Ulong	Unit
Byte	Case	Catch	Char	Event	Value	While
Explicit	Extern	False	Finally	Fixed	Set	Readonly
Float	For	Foreach	Namespace	New	Short	Ref
Null	Object	Operator	Out	Override	Unchecked	Return
Params	Private	Static	String	Struct	Virtual	Sbyte
Switch	This	Throw	True	Try	Sizeof	Sealed
Typeof	Checked	Class	Const	Continue	Unsafe	Using
Decimal	Default	Delegate	Double	Do	Volatile	
Else	Enum	Get	Goto	If	Stackalloc	
Implicit	In	Int	Interface	Internal	Ushort	
Is	Lock	Long	Protected	Public	Void	

4] Java

Java Keywords				
abstract	assert	boolean	break	byte
case	catch	char	class	continue
default	do	double	else	enum
extends	final	finally	float	for
if	implements	import	instanceof	int
interface	long	native	new	package
private	protected	public	return	short
static	strictfp	super	switch	synchronized
this	throw	throws	transient	try
void	volatile	while		
<i>Keywords that are not currently used</i>				
const	goto			

5] Python

 Python Reserved Keywords				
False	class	finally	is	return
None	continue	for	lambda	try
True	def	from	nonlocal	while
and	del	global	not	with
as	elif	if	or	yield
assert	else	import	pass	
break	except	in	raise	

OPERATORS

Common Operators

Arithmetic Operators

Arithmetic operators are used to perform common mathematical operations

Operator	Name	Description	Example
+	Addition	Adds together two values	$x + y$
-	Subtraction	Subtracts one value from another	$x - y$
*	Multiplication	Multiplies two values	$x * y$
/	Division	Divides one value by another	x / y
%	Modulus	Returns the division remainder	$x \% y$
++	Increment	Increases the value of a variable by 1	$x++$
--	Decrement	Decreases the value of a variable by 1	$x--$

Assignment Operators

Assignment operators are used to assign values to variables.

Operator	Example	Same As
=	$x = 5$	$x = 5$
+=	$x += 3$	$x = x + 3$
-=	$x -= 3$	$x = x - 3$
*=	$x *= 3$	$x = x * 3$
/=	$x /= 3$	$x = x / 3$
%=	$x \% = 3$	$x = x \% 3$
&=	$x \& = 3$	$x = x \& 3$
=	$x = 3$	$x = x 3$
^=	$x \wedge = 3$	$x = x \wedge 3$
>>=	$x >> = 3$	$x = x >> 3$
<<=	$x << = 3$	$x = x << 3$

Comparison Operators

Comparison operators are used to compare two values

Operator	Name	Example
==	Equal to	x == y
!=	Not equal	x != y
>	Greater than	x > y
<	Less than	x < y
>=	Greater than or equal to	x >= y
<=	Less than or equal to	x <= y

Logical Operators

Logical operators are used to determine the logic between variables or values

Operator	Name	Description	Example
&&	Logical and	Returns true if both statements are true	x < 5 && x < 10
	Logical or	Returns true if one of the statements is true	x < 5 x < 4
!	Logical not	Reverse the result, returns false if the result is true	!(x < 5 && x < 10)

[Python Does Not have these logical operators. Python has its own Logical operators]

Different Operators

1] Arithmetic Operators

Arithmetic Operators which are in all language but not in Python

++	Increment	Increases the value of a variable by 1	++x
--	Decrement	Decreases the value of a variable by 1	--x

Arithmetic Operators which are not in all language but in Python

**	Exponentiation	x ** y
//	Floor division	x // y

2] Assignment Operators

Arithmetic Operators which are not in all language but in Python

**	Exponentiation	x ** y
//	Floor division	x // y

3] Logical Operators

Arithmetic Operators which are not in all language but in Python

Operator	Description	Example
and	Returns True if both statements are true	<code>x < 5 and x < 10</code>
or	Returns True if one of the statements is true	<code>x < 5 or x < 4</code>
not	Reverse the result, returns False if the result is true	<code>not(x < 5 and x < 10)</code>

4] Identity Operators (This Operator Only used in Python Language)

Identity operators are used to compare the objects, not if they are equal, but if they are actually the same object, with the same memory location

Operator	Description	Example
is	Returns True if both variables are the same object	<code>x is y</code>
is not	Returns True if both variables are not the same object	<code>x is not y</code>

5] Membership Operators (This Operator Only used in Python Language)

Membership operators are used to test if a sequence is presented in an object

Operator	Description	Example
in	Returns True if a sequence with the specified value is present in the object	x in y
not in	Returns True if a sequence with the specified value is not present in the object	x not in y

ARRAYS

1] C

How to declare an array?

```
dataType arrayName[arraySize];
```

For example,

```
float mark[5];
```

Here, we declared an array, `mark`, of floating-point type. And its size is 5. Meaning, it can hold 5 floating-point values.

It's important to note that the size and type of an array cannot be changed once it is declared.

2] C++

C++ Array Declaration

```
dataType arrayName[arraySize];
```

For example,

```
int x[6];
```

Here,

- `int` - type of element to be stored
- `x` - name of the array
- `6` - size of the array

3] C#

How to declare an array in C#?

In C#, here is how we can declare an array.

```
dataType[] arrayName;
```

- `dataType` - it can be [primitive data types](#) like `int`, `char`, `double`, `byte`, etc. or [C# objects](#)
- `arrayName` - it is an [identifier](#)

For example,

```
double[] data;
```

4] Java

How to declare an array in Java?

In Java, here is how we can declare an array.

```
dataType[] arrayName;
```

- `dataType` - it can be [primitive data types](#) like `int`, `char`, `double`, `byte`, etc. or [Java objects](#)
- `arrayName` - it is an [identifier](#)

For example,

```
double[] data;
```


5] Python

Creating Python Arrays

To create an array of numeric values, we need to import the `array` module. For example:

```
import array as arr
a = arr.array('d', [1.1, 3.5, 4.5])
print(a)
```

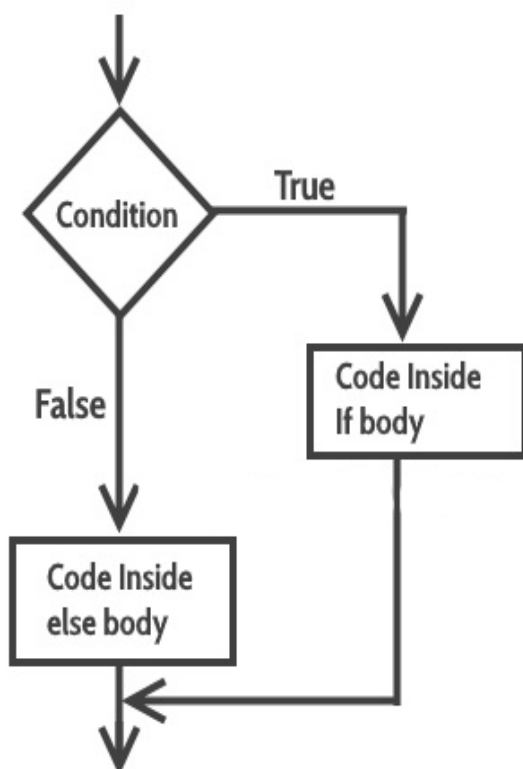
Output

```
array('d', [1.1, 3.5, 4.5])
```

CONDITIONS

If-else Condition

1] C



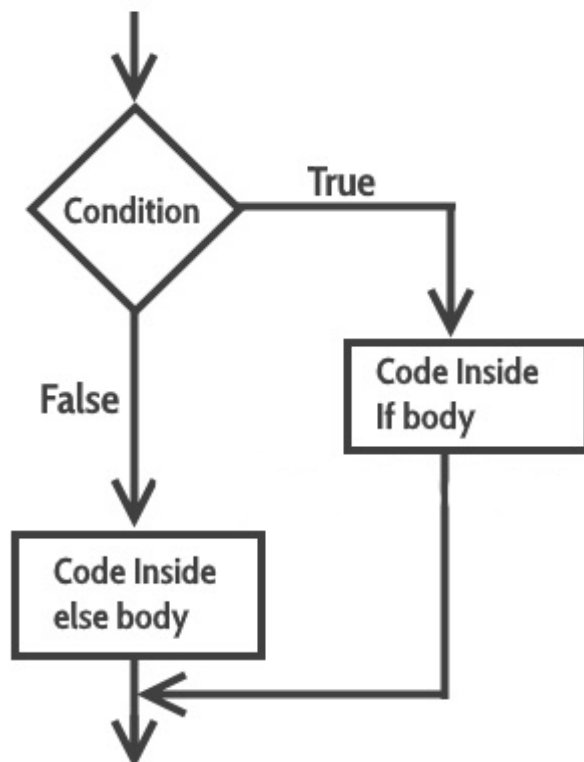
```
#include <stdio.h>
int main() {
    int number;
    printf("Enter an integer: ");
    scanf("%d", &number);

    if (number%2 == 0) {
        printf("%d is an even integer.", number);
    }
    else {
        printf("%d is an odd integer.", number);
    }

    return 0;
}
```

Output :- Enter an integer: 7
7 is an odd integer.

2] C++

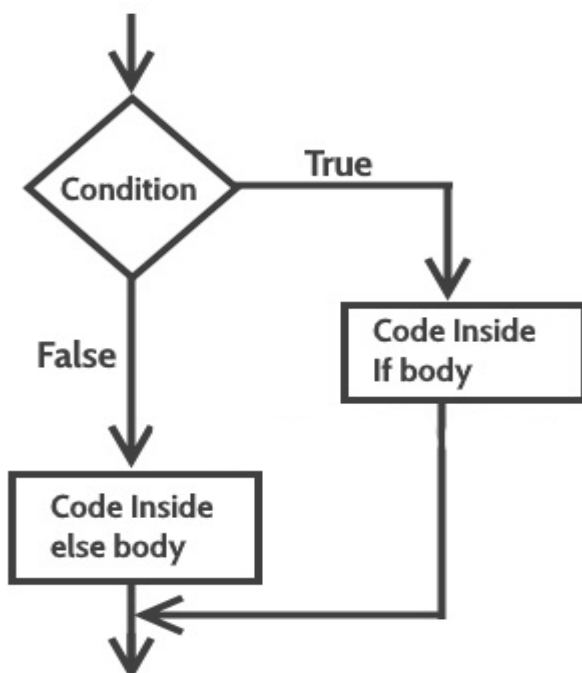


```
#include <iostream>
using namespace std;

int main() {
    int time = 20;
    if (time < 18) {
        cout << "Good day.";
    } else {
        cout << "Good evening.";
    }
    return 0;
}
```

Output :- Good evening.

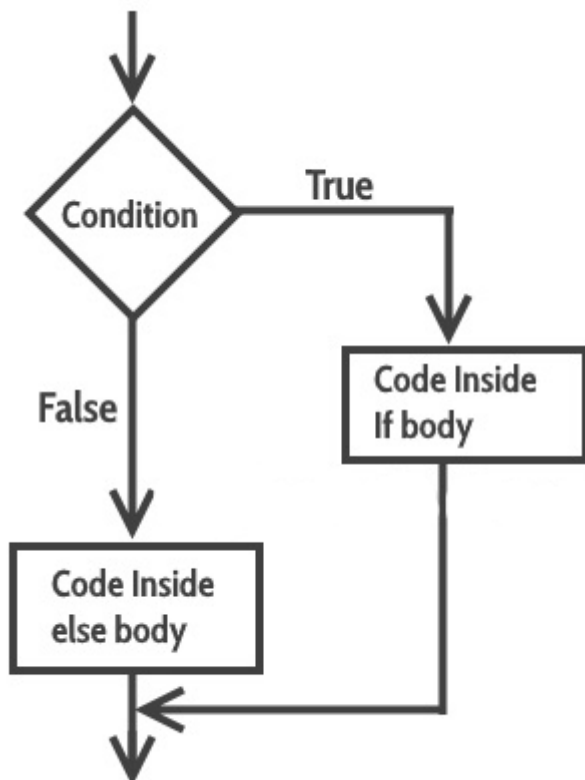
3] C#



```
using System;
namespace MyApplication
{
    class Program
    {
        static void Main(string[] args)
        {
            int time = 20;
            if (time < 18)
            {
                Console.WriteLine("Good day.");
            }
            else
            {
                Console.WriteLine("Good evening.");
            }
        }
    }
}
```

Output :- Good evening.

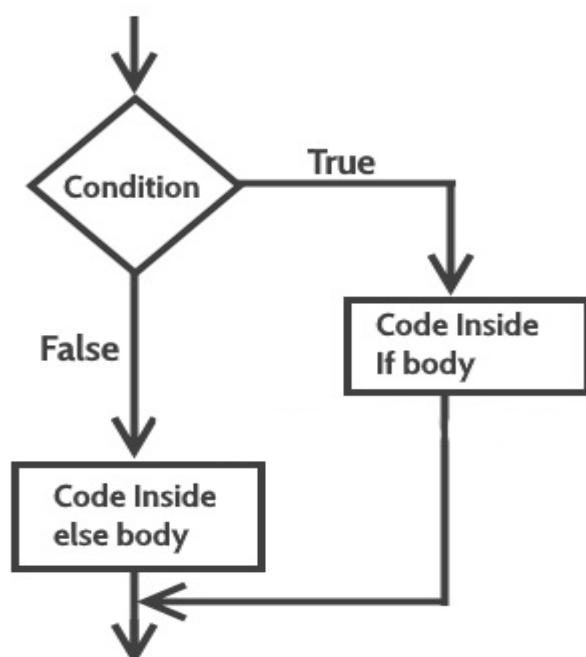
4] Java



```
public class Main {  
    public static void main(String[] args) {  
        int time = 20;  
        if (time < 18) {  
            System.out.println("Good day.");  
        } else {  
            System.out.println("Good evening.");  
        }  
    }  
}
```

Output :- Good evening.

5] Python

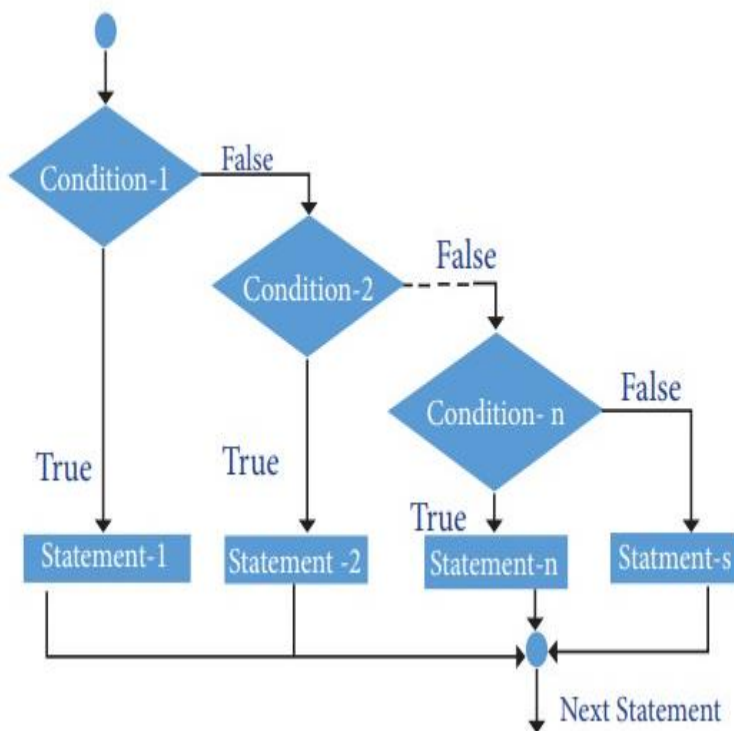


```
a = 200  
b = 33  
if b > a:  
    print("b is greater than a")  
else:  
    print("a is greater than b")
```

Output :- a is greater than b

else-if Condition

1] C



```
#include <stdio.h>

int main()
{
    int i = 20;

    if (i == 10)
        printf("i is 10");

    else if (i == 15)
        printf("i is 15");

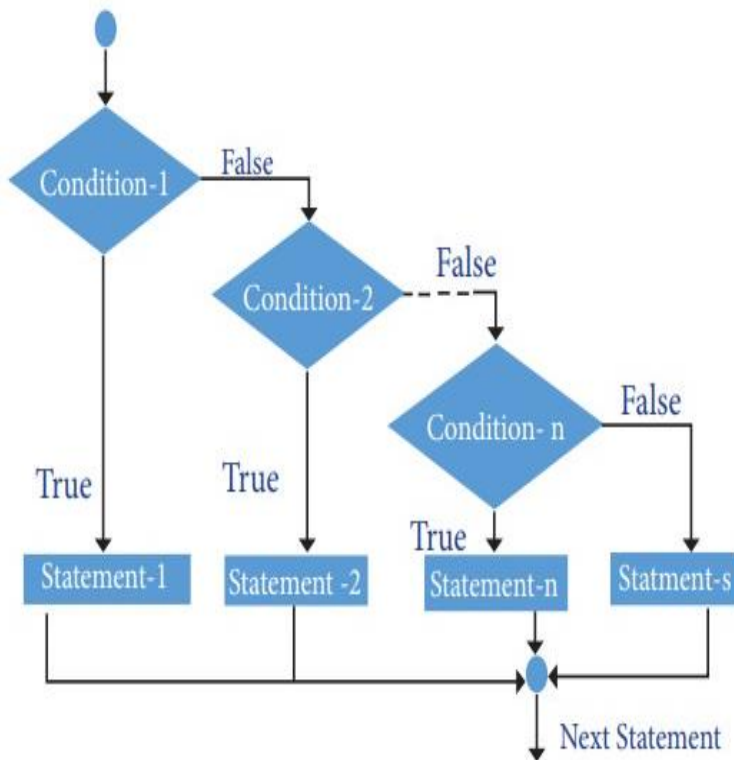
    else if (i == 20)
        printf("i is 20");

    else
        printf("i is not present");

    return 0;
}
```

Output :- i is 20

2] C++



```

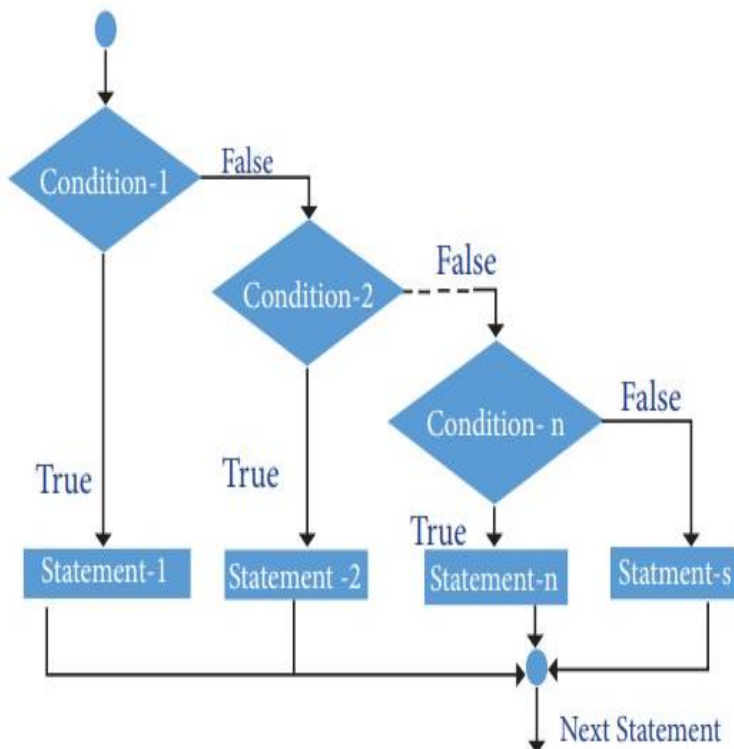
#include <iostream>
using namespace std;

int main() {
    int time = 22;
    if (time < 10) {
        cout << "Good morning.";
    } else if (time < 20) {
        cout << "Good day.";
    } else {
        cout << "Good evening.";
    }
    return 0;
}

```

Output :- Good evening

3] C#



```

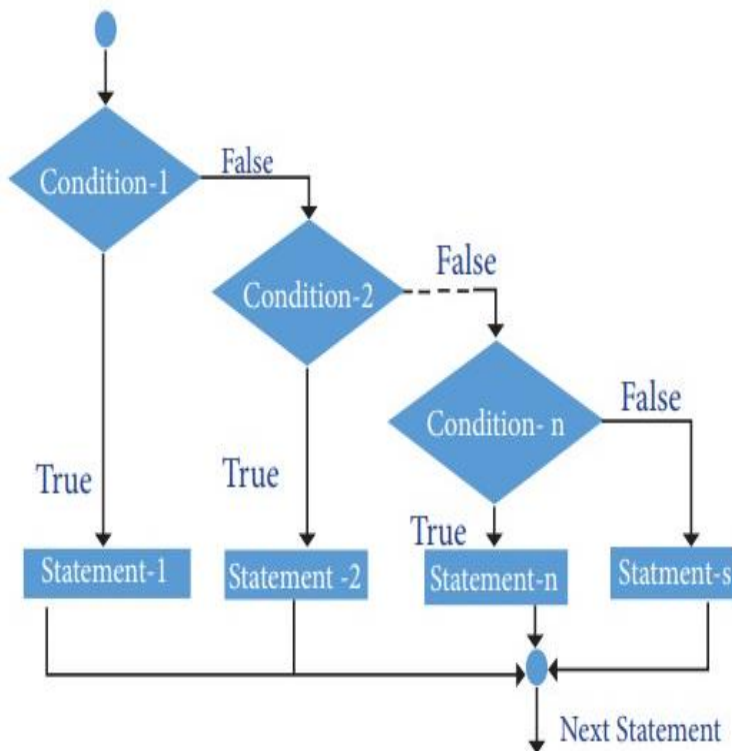
using System;

namespace MyApplication
{
    class Program
    {
        static void Main(string[] args)
        {
            int time = 22;
            if (time < 10)
            {
                Console.WriteLine("Good morning.");
            }
            else if (time < 20)
            {
                Console.WriteLine("Good day.");
            }
            else
            {
                Console.WriteLine("Good evening.");
            }
        }
    }
}

```

Output :- Good evening

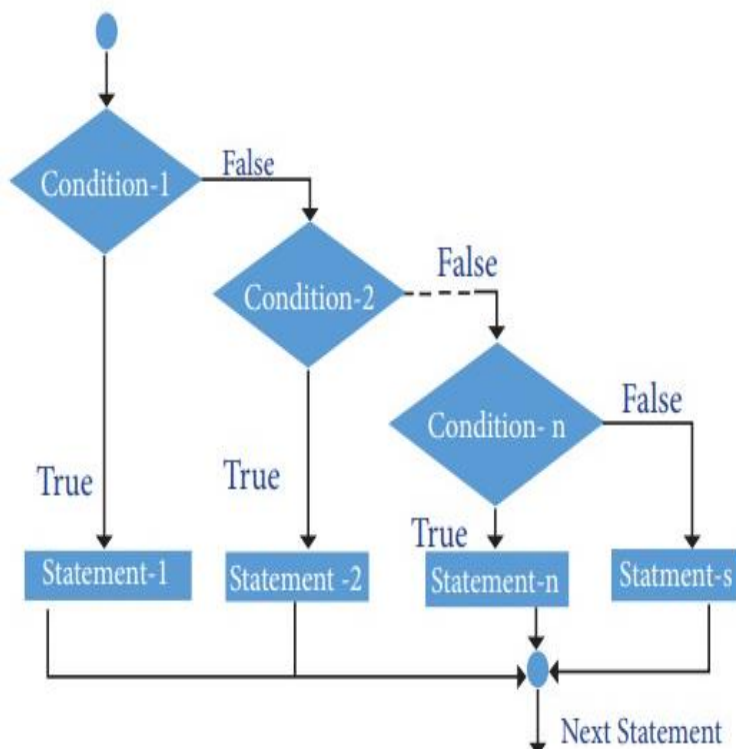
4] Java



```
public class Main {  
    public static void main(String[] args) {  
        int time = 22;  
        if (time < 10) {  
            System.out.println("Good morning.");  
        } else if (time < 20) {  
            System.out.println("Good day.");  
        } else {  
            System.out.println("Good evening.");  
        }  
    }  
}
```

Output :- Good evening.

5] Python

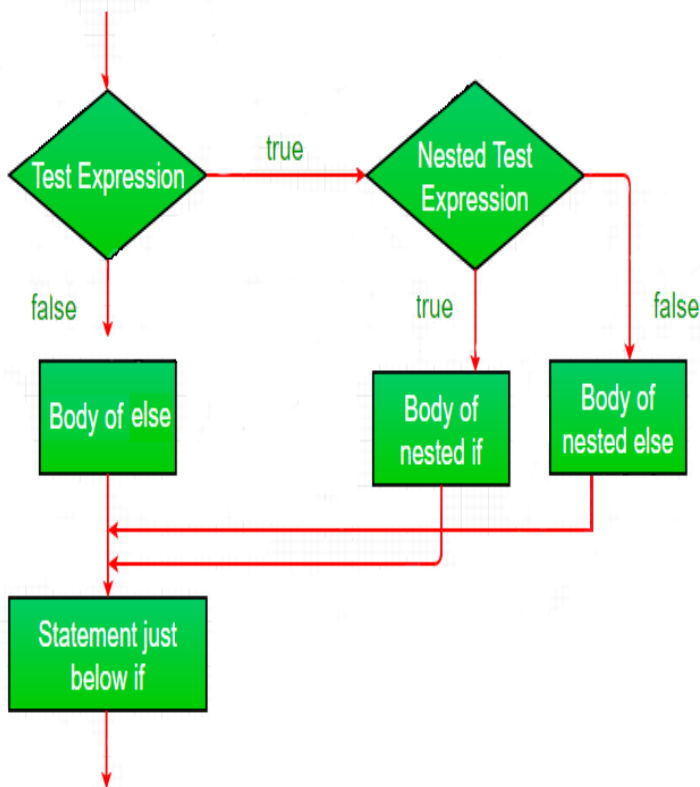


```
a = 200  
b = 33  
if b > a:  
    print("b is greater than a")  
elif a == b:  
    print("a and b are equal")  
else:  
    print("a is greater than b")
```

Output :- a is greater than b

Nested if Condition

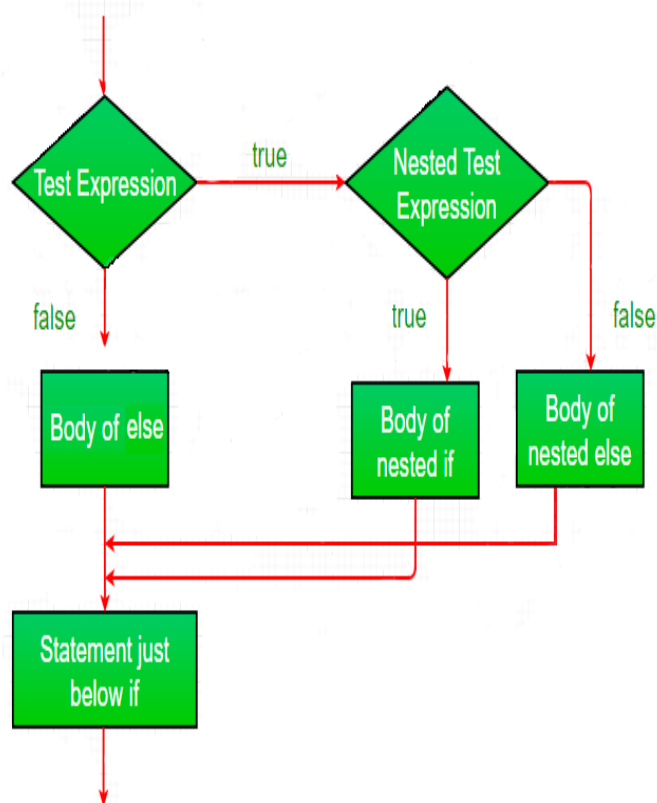
1] C



```
#include<stdio.h>
int main()
{
    int num=1;
    if(num<10)
    {
        if(num==1)
        {
            printf("The value is : %d\n",num);
        }
        else
        {
            printf("The value is greater than 1");
        }
    }
    else
    {
        printf("The value is greater than 10");
    }
    return 0;
}
```

Output :- The value is : 1

2] C++



```
#include <iostream>
using namespace std;
```

```
int main()
{
    int i = 10;

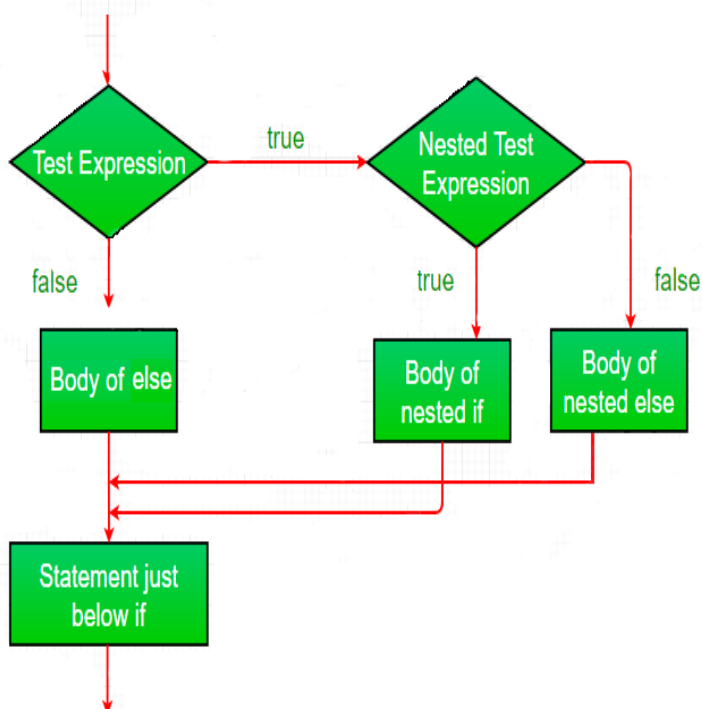
    if (i == 10)
    {
        if (i < 15)
            cout<<"i is smaller than 15\n";

        if (i < 12)
            cout<<"i is smaller than 12 too\n";
        else
            cout<<"i is greater than 15";
    }

    return 0;
}
```

Output :- i is smaller than 15
i is smaller than 12 too

3] C#



```
using System;

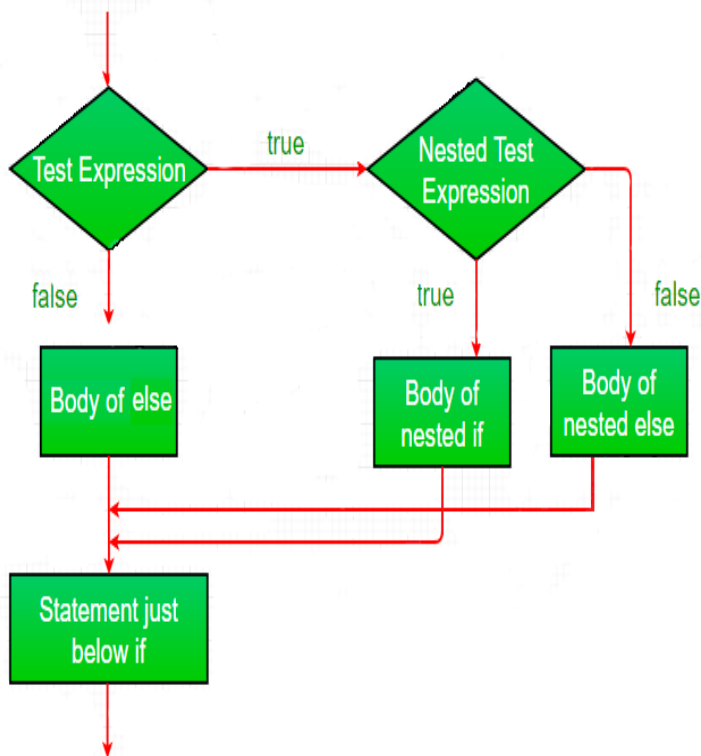
namespace DecisionMaking {

    class Program {
        static void Main(string[] args) {
            int a = 100;
            int b = 200;

            if (a == 100) {
                if (b == 200) {
                    Console.WriteLine("Value of a is 100 and b is 200");
                }
                Console.WriteLine("Exact value of a is : {0}", a);
                Console.WriteLine("Exact value of b is : {0}", b);
                Console.ReadLine();
            }
        }
    }
}
```

Output :- Value of a is 100 and b is 200
Exact value of a is : 100
Exact value of b is : 200

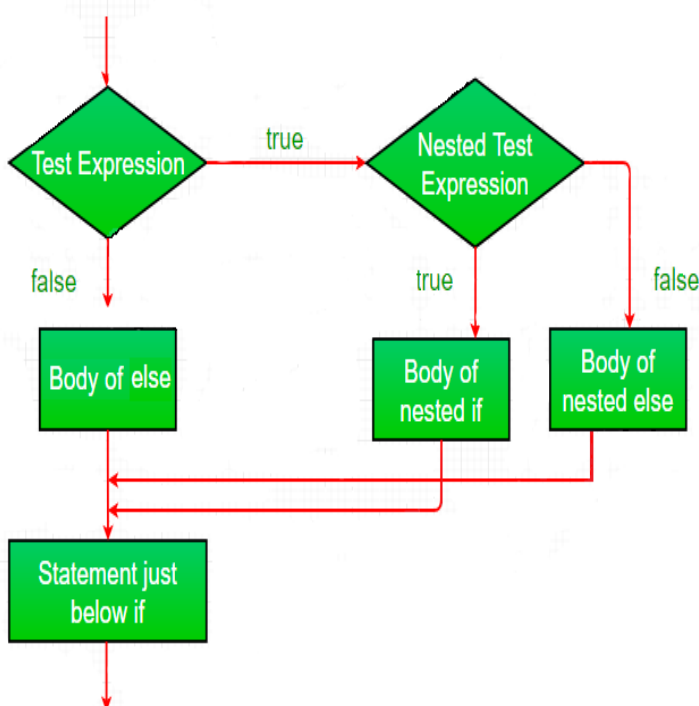
4] Java



```
public class JavaNestedIfExample {  
  
    public static void main(String[] args) {  
  
        int age=20;  
        int weight=80;  
  
        if(age>=18){  
            if(weight>50){  
                System.out.println("You are eligible to donate blood")  
            }  
        }  
    }  
}
```

Output :- You are eligible to donate blood

5] Python

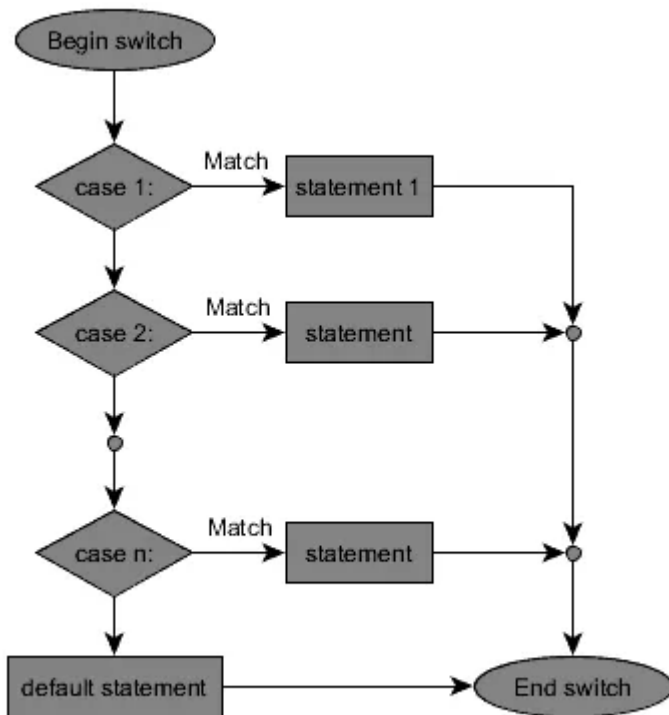


```
x = 41  
  
if x > 10:  
    print("Above ten,")  
    if x > 20:  
        print("and also above 20!")  
    else:  
        print("but not above 20.")
```

Output :- Above ten,
 and also above 20!

Switch Condition

1] C



```
#include <stdio.h>

int main () {

    char grade = 'B';

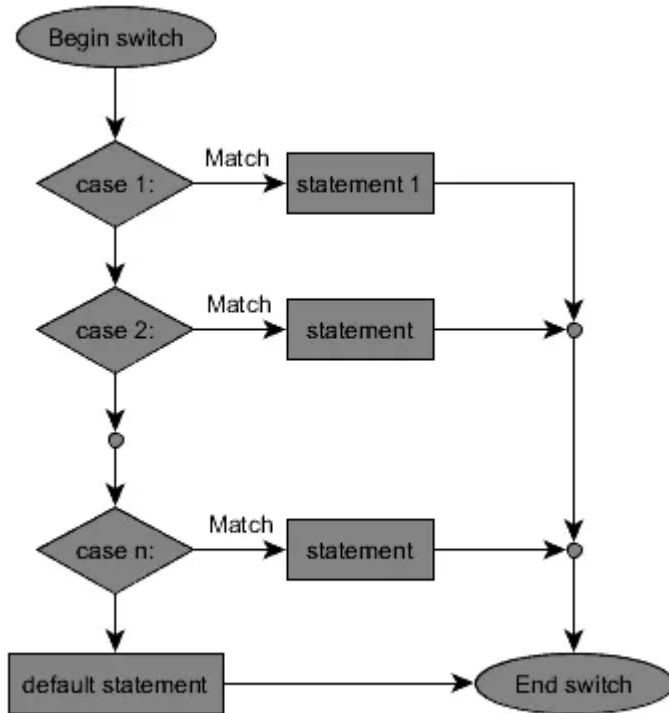
    switch(grade) {
        case 'A' :
            printf("Excellent!\n" );
            break;
        case 'B' :
            printf("Well done\n" );
            break;
        default :
            printf("Invalid grade\n" );
    }

    printf("Your grade is  %c\n", grade );

    return 0;
}
```

Output :- Well done
Your grade is B

2] C++



```

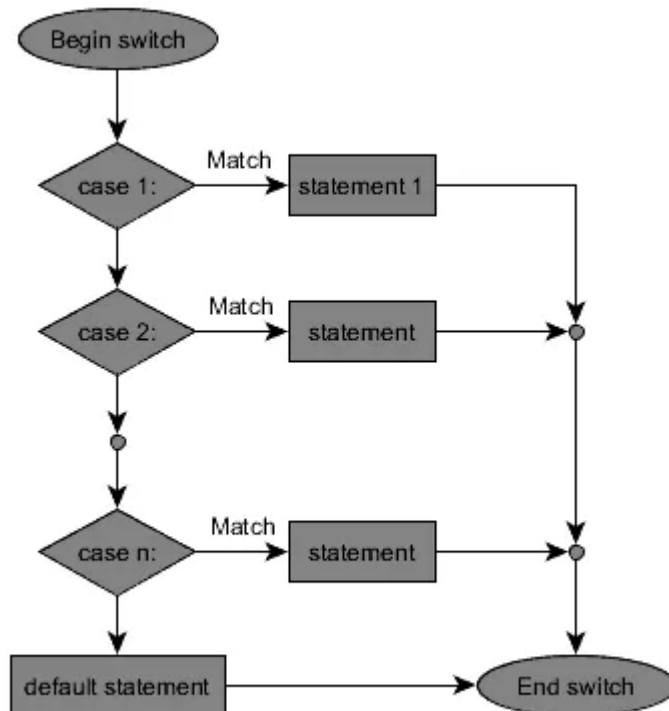
#include <iostream>
using namespace std;

int main() {
    int day = 2;
    switch (day) {
        case 1:
            cout << "Monday";
            break;
        case 2:
            cout << "Tuesday";
            break;
        case 3:
            cout << "Wednesday";
            break;
    }
    return 0;
}

```

Output :- Tuesday

3] C#



```

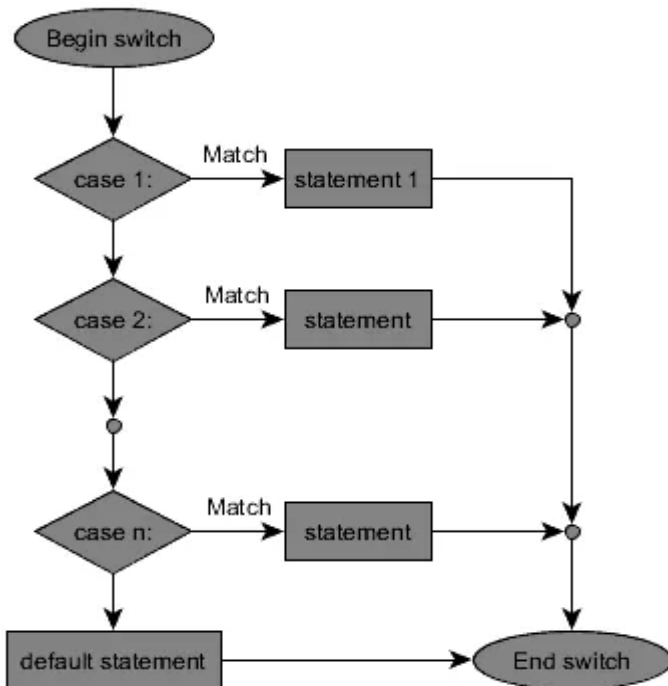
using System;

namespace MyApplication
{
    class Program
    {
        static void Main(string[] args)
        {
            int day = 2;
            switch (day)
            {
                case 1:
                    Console.WriteLine("Monday");
                    break;
                case 2:
                    Console.WriteLine("Tuesday");
                    break;
            }
        }
    }
}

```

Output :- Tuesday

4] Java



```
public class Main {  
    public static void main(String[] args) {  
        int day = 3;  
        switch (day) {  
            case 1:  
                System.out.println("Monday");  
                break;  
            case 2:  
                System.out.println("Tuesday");  
                break;  
            case 3:  
                System.out.println("Wednesday");  
                break;  
        }  
    }  
}
```

Output :- Wednesday

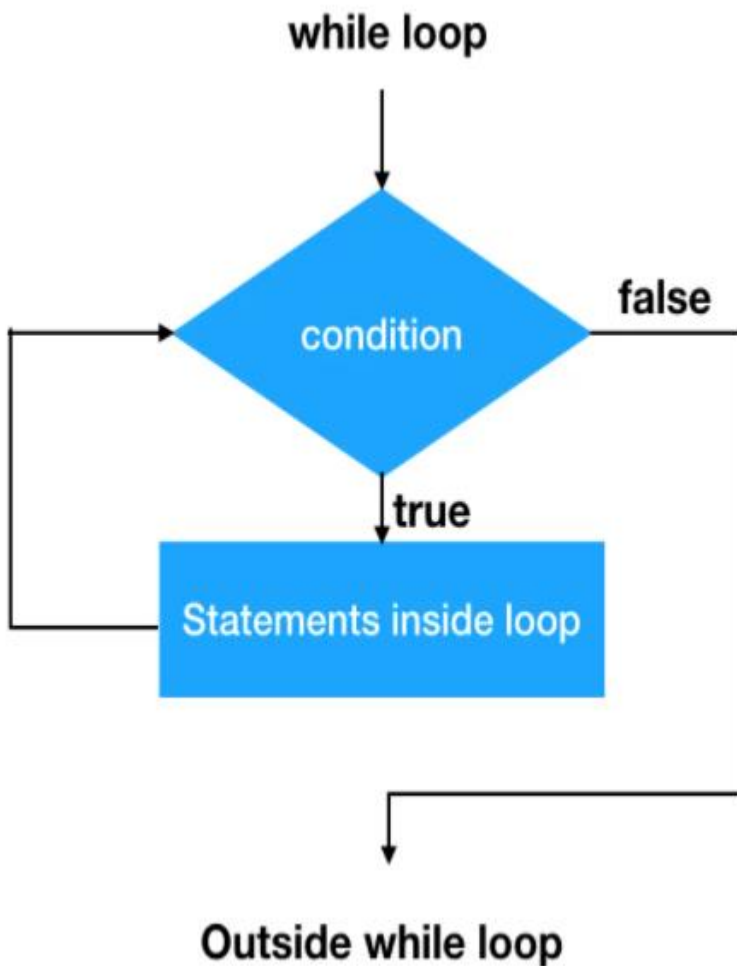
5] Python

[Python does not have the switch condition]

LOOPS

While Loop

1] C



```
#include <stdio.h>

int main () {

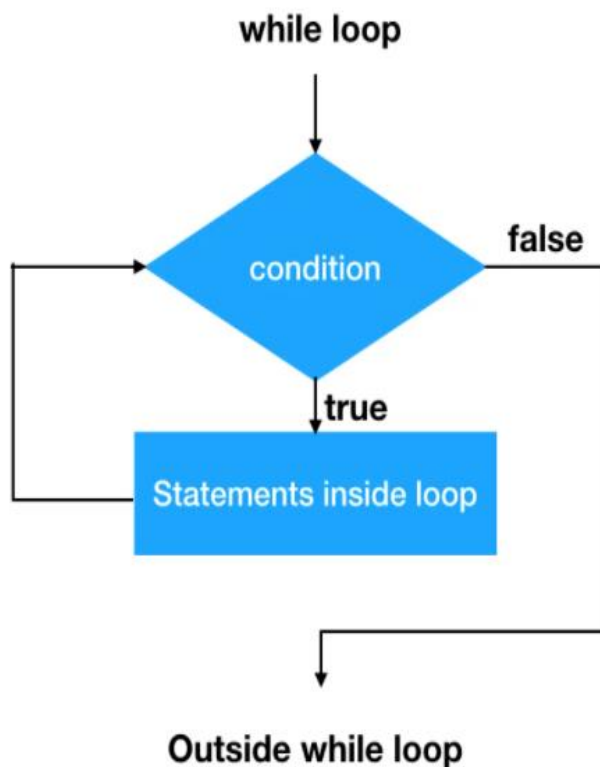
    int a = 10;

    while( a < 15 ) {
        printf("value of a: %d\n", a);
        a++;
    }

    return 0;
}
```

Output :- value of a: 10
value of a: 11
value of a: 12
value of a: 13
value of a: 14

2] C++

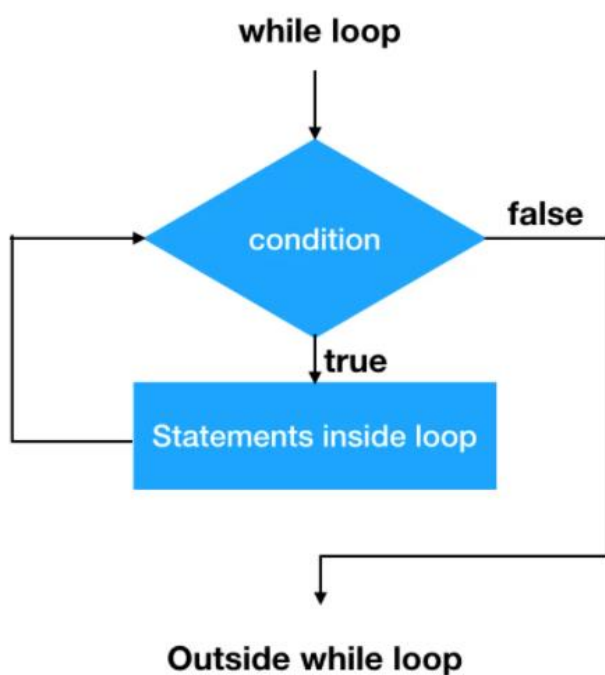


```
#include <iostream>
using namespace std;

int main() {
    int i = 0;
    while (i < 5) {
        cout << i << "\n";
        i++;
    }
    return 0;
}
```

Output :- 0
1
2
3
4

3] C#

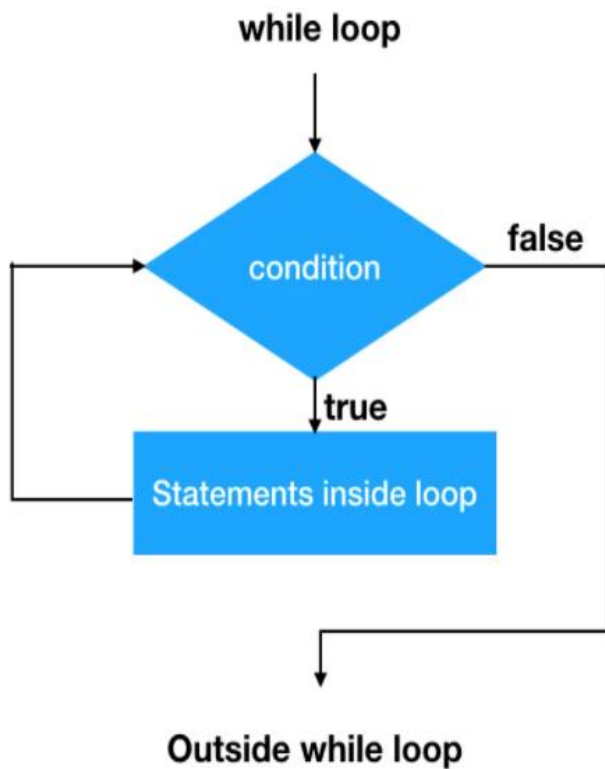


```
using System;

namespace MyApplication
{
    class Program
    {
        static void Main(string[] args)
        {
            int i = 0;
            while (i < 5)
            {
                Console.WriteLine(i);
                i++;
            }
        }
    }
}
```

Output :- 0
1
2
3
4

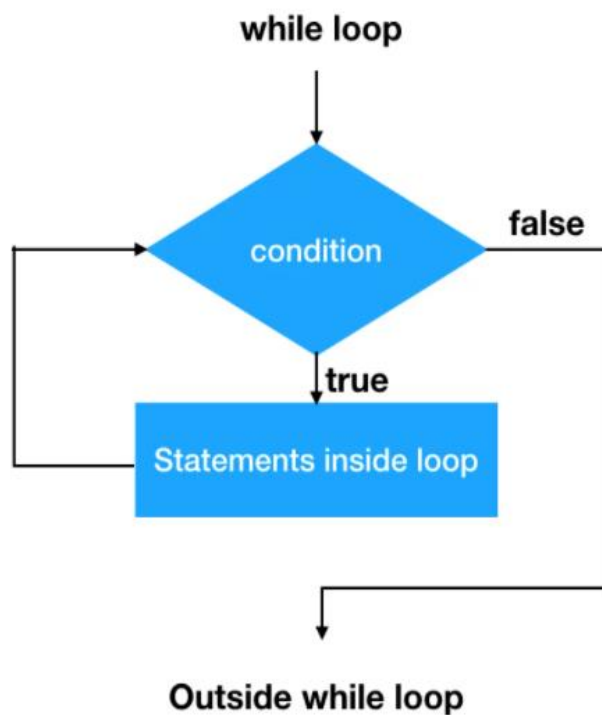
4] Java



```
public class Main {  
    public static void main(String[] args) {  
        int i = 0;  
        while (i < 5) {  
            System.out.println(i);  
            i++;  
        }  
    }  
}
```

Output :- 0
1
2
3
4

5] Python

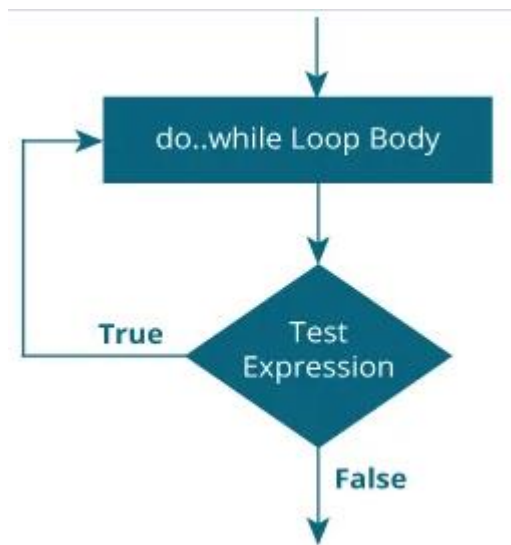


```
i = 0  
while i < 5:  
    print(i)  
    i += 1
```

Output :- 0
1
2
3
4

Do-While Loop

1] C



```
#include <stdio.h>
int main()
{
    double number, sum = 0;

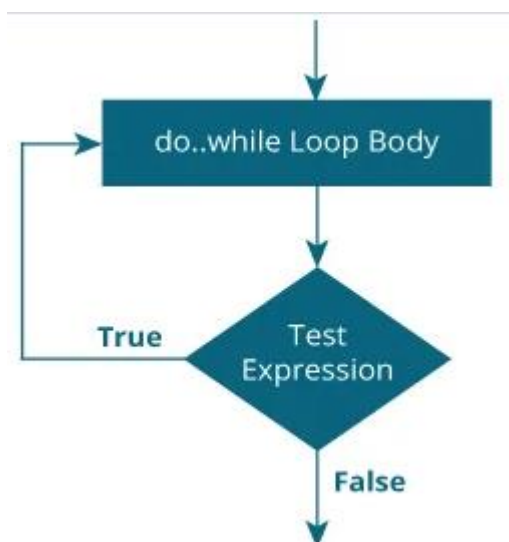
    do
    {
        printf("Enter a number: ");
        scanf("%lf", &number);
        sum += number;
    }
    while(number != 0.0);

    printf("Sum = %.2lf", sum);

    return 0;
}
```

Output :- Enter a number: 1.5
Enter a number: 2.4
Enter a number: -3.4
Enter a number: 4.2
Enter a number: 0
Sum = 4.70

2] C++

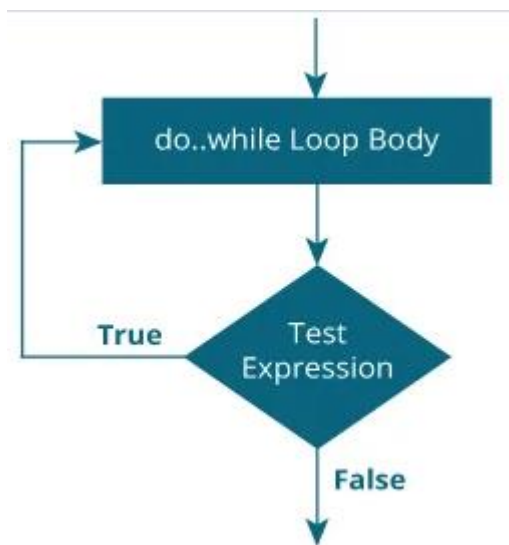


```
#include <iostream>
using namespace std;

int main() {
    int i = 0;
    do {
        cout << i << "\n";
        i++;
    }
    while (i < 5);
    return 0;
}
```

Output :- 0
1
2
3
4

3] C#

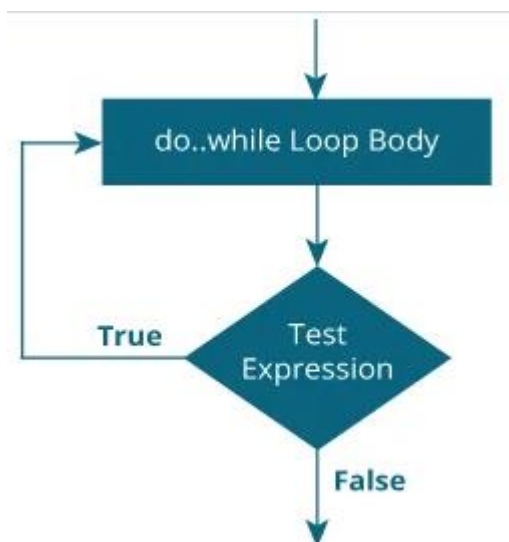


```
using System;

namespace MyApplication
{
    class Program
    {
        static void Main(string[] args)
        {
            int i = 0;
            do
            {
                Console.WriteLine(i);
                i++;
            } while (i < 5);
        }
    }
}
```

Output :- 0
1
2
3
4

4] Java



```
public class Main {
    public static void main(String[] args) {
        int i = 0;
        do {
            System.out.println(i);
            i++;
        } while (i < 5);
    }
}
```

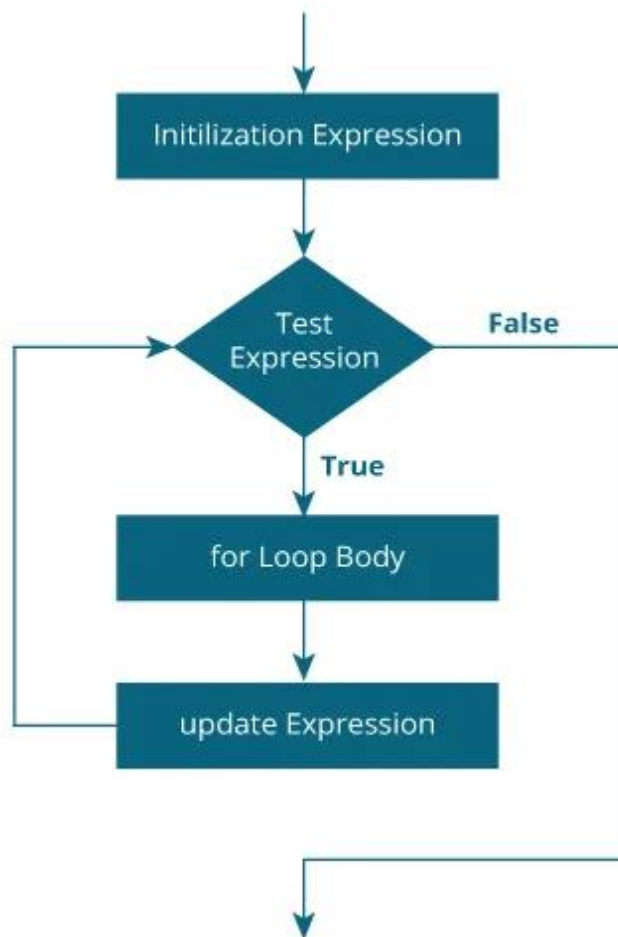
Output :- 0
1
2
3
4

5] Python

[Python does not have Do While Loop]

For Loop

1] C



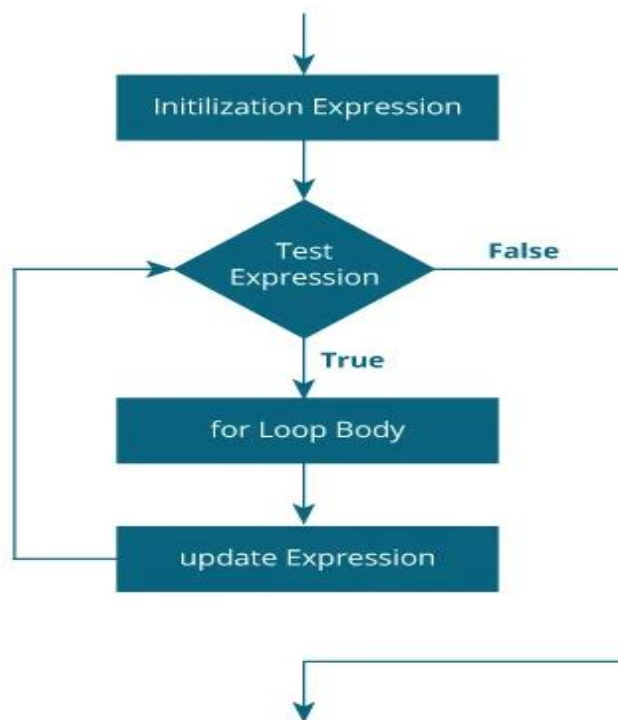
```
#include <stdio.h>

int main() {
    int i;

    for (i = 1; i < 11; ++i)
    {
        printf("%d ", i);
    }
    return 0;
}
```

Output :- 1 2 3 4 5 6 7 8 9 10

2] C++

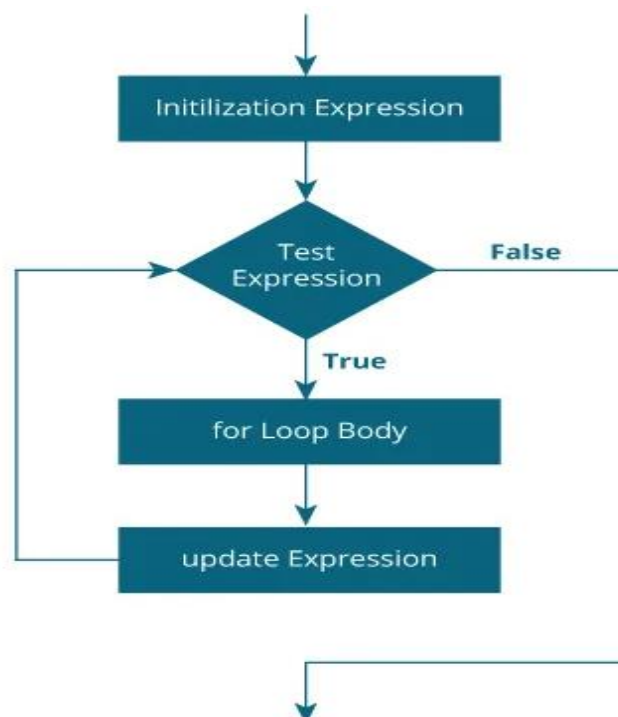


```
#include <iostream>
using namespace std;

int main() {
    for (int i = 0; i < 5; i++) {
        cout << i << "\n";
    }
    return 0;
}
```

Output :- 0
1
2
3
4

3] C#

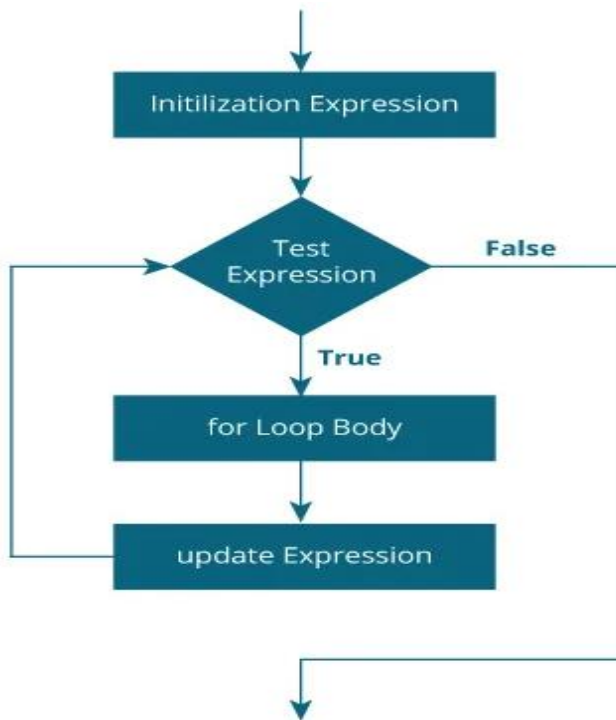


```
using System;

namespace MyApplication
{
    class Program
    {
        static void Main(string[] args)
        {
            for (int i = 0; i < 5; i++)
            {
                Console.WriteLine(i);
            }
        }
    }
}
```

Output :- 0
1
2
3
4

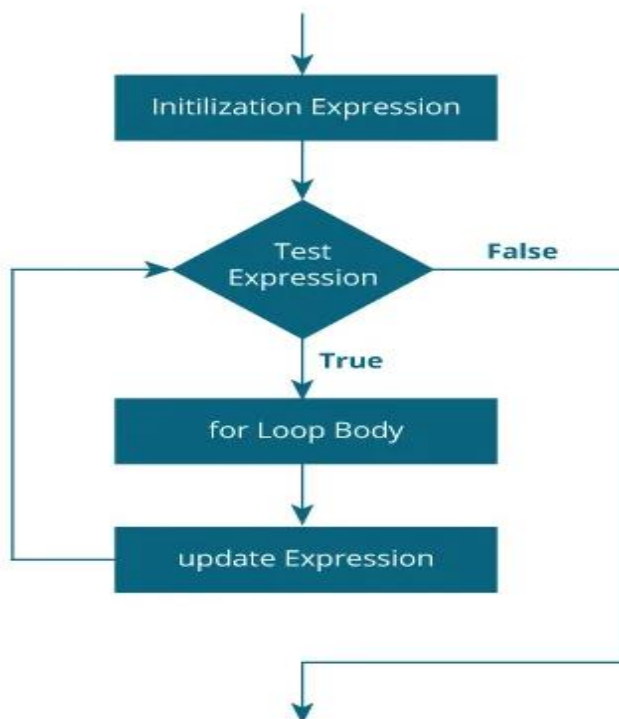
4] Java



```
public class Main {  
    public static void main(String[] args) {  
        for (int i = 0; i < 5; i++) {  
            System.out.println(i);  
        }  
    }  
}
```

Output :- 0
1
2
3
4

5] Python



```
fruits = ["apple", "banana", "cherry"]  
for x in fruits:  
    print(x)
```

Output :- apple
banana
cherry

OBJECT ORIENTED PROGRAMMING

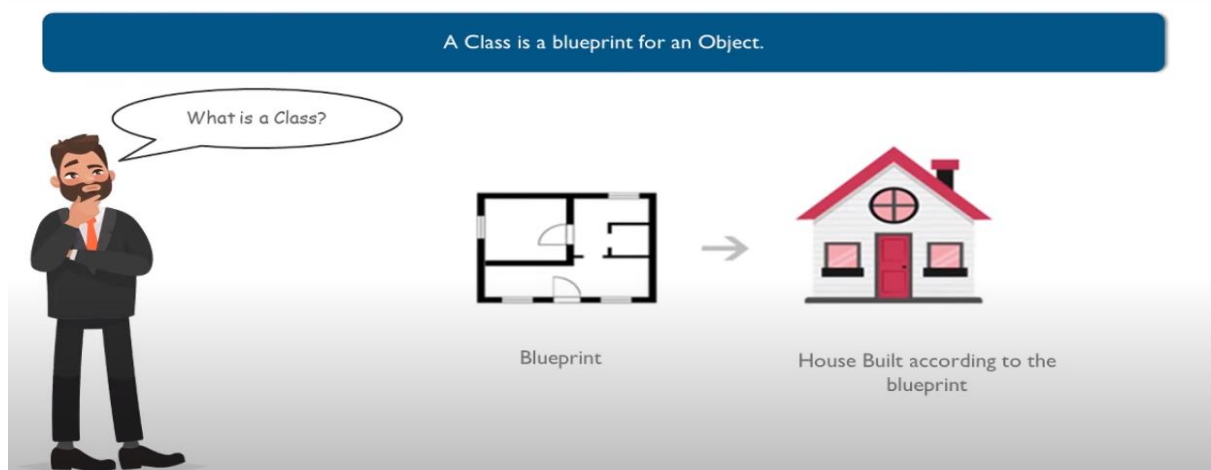
Object-oriented programming (OOP) is a programming paradigm based on the concept of "objects", which can contain data and code: data in the form of fields (often known as attributes or properties), and code, in the form of procedures.

A feature of objects is that an object's own procedures can access and often modify the data fields of itself (objects have a notion of this or self). In OOP, computer programs are designed by making them out of objects that interact with one another.[1][2] OOP languages are diverse, but the most popular ones are class-based, meaning that objects are instances of classes, which also determine their types.

[C Language is a Functional Language. It is Not an Object-oriented programming Language (OOP)]

[It does not Object-oriented programming Language (OOP) Concept]

Classes and Objects



C++ supports the object-oriented (OO) style of programming that allows you to divide complex problems into smaller sets by creating objects. An object is an instance of a class, which can be used to access member variables and functions of a class.



```
ClassName objName; //Syntax for Object Creation
```

How to Create Classes and Object

1] C++

```
#include <iostream>
using namespace std;

class Room { // create a class

public:
    double length;
    double breadth;
    double height;

    double calculateArea() {
        return length * breadth;
    }

    double calculateVolume() {
        return length * breadth * height;
    }
};
```



```

int main() {

    Room room1; // create object of Room class

    room1.length = 42.5;
    room1.breadth = 30.8;
    room1.height = 19.2;

    cout << "Area of Room = " << room1.calculateArea() << endl;
    cout << "Volume of Room = " << room1.calculateVolume() << endl;

    return 0;
}

```

2] C#

```

class Room {

    using System;

    namespace MyApplication
    {
        class Car { // create a class
            string color = "red";

            static void main(String[] args) {

                Car myObj1 = new Car(); // create object of Room class

                Console.WriteLine(myObj1.color);

            }
        }
    }
}

```

3] Java

```
class Lamp { // create a class

    boolean isOn;

    void turnOn() {

        isOn = true;
        System.out.println("Light on? " + isOn);

    }

    void turnOff() {

        isOn = false;
        System.out.println("Light on? " + isOn);

    }
}

class Main {
    public static void main(String[] args) {

        // create objects led and halogen
        Lamp led = new Lamp();
        Lamp halogen = new Lamp();

        led.turnOn();

        halogen.turnOff();

    }
}
```

4] Python

```
class Person: # create a new object of Person class
    "This is a person class"
    age = 10

    def greet(self):
        print('Hello')

# create a new object of Person class
harry = Person()

print(Person.greet)

print(harry.greet)

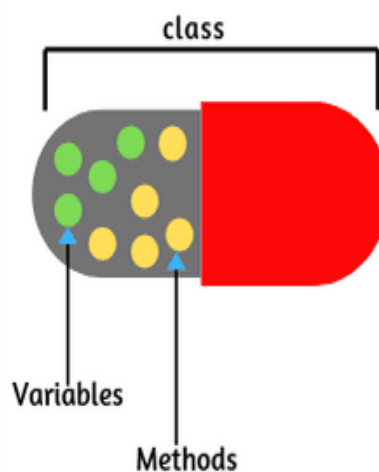
harry.greet()
```

The Four Principles of Object-Oriented-Programming



1] Encapsulation :- In object-oriented programming (OOP), encapsulation refers to the bundling of data with the methods that operate on that data, or the restricting of direct access to some of an object's components. Encapsulation is used to hide the values or state of a structured data object inside a class, preventing direct access to them by clients in a way that could expose hidden implementation details or violate state invariance maintained by the methods.

```
class
{
    data members
    +
    methods (behavior)
}
```



Example of Encapsulation

1] C++

```
#include <iostream>
using namespace std;

class Employee {
private:
    int salary;

public:
    void setSalary(int s) {
        salary = s;
    }
    int getSalary() {
        return salary;
    }
};

int main() {
    Employee myObj;
    myObj.setSalary(50000);
    cout << myObj.getSalary();
    return 0;
}
```

Output :- 50000

2] C#

```
using System;

namespace RectangleApplication {
    class Rectangle {
        public double length;
        public double width;

        public double GetArea() {
            return length * width;
        }
        public void Display() {
            Console.WriteLine("Length: {0}", length);
            Console.WriteLine("Width: {0}", width);
            Console.WriteLine("Area: {0}", GetArea());
        }
    }

    class ExecuteRectangle {
        static void Main(string[] args) {
            Rectangle r = new Rectangle();
            r.length = 4.5;
            r.width = 3.5;
            r.Display();
            Console.ReadLine();
        }
    }
}
```

Output :- Length: 4.5
 Width: 3.5
 Area: 15.75

3] Java

```
class Encapsulate {
    private String geekName;
    private int geekRoll;
    private int geekAge;

    public int getAge() { return geekAge; }
    public String getName() { return geekName; }
    public int getRoll() { return geekRoll; }
    public void setAge(int newAge) { geekAge = newAge; }
    public void setName(String newName) {
        geekName = newName; }
    public void setRoll(int newRoll) { geekRoll = newRoll; }
}

public class TestEncapsulation {
    public static void main(String[] args)
    {
        Encapsulate obj = new Encapsulate();

        obj.setName("Harsh");
        obj.setAge(19);
        obj.setRoll(51);

        System.out.println("Geek's name: " + obj.getName());
        System.out.println("Geek's age: " + obj.getAge());
        System.out.println("Geek's roll: " + obj.getRoll());
    }
}
```

Output :- Geek's name: Harsh
 Geek's age: 19
 Geek's roll: 51

4] Python

```
class Base:
    def __init__(self):

        # Protected member
        self._a = 2

class Derived(Base):
    def __init__(self):

        Base.__init__(self)
        print("Calling protected member of base class: ")
        print(self._a)

obj1 = Derived()

obj2 = Base()

print(obj2.a)
```

Output :- Calling protected member of base class: 2

2] Abstraction :- Data Abstraction is the property by virtue of which only the essential details are displayed to the user. The trivial or the non-essentials units are not displayed to the user. Ex: A car is viewed as a car rather than its individual components.

Data Abstraction may also be defined as the process of identifying only the required characteristics of an object ignoring the irrelevant details. The properties and behaviours of an object differentiate it from other objects of similar type and also help in classifying/grouping the objects.

Abstraction

Abstraction



- ❑ Hides the implementation details and only provides the functionality to the user
- ❑ You can achieve abstraction using [Abstract classes](#) and [Interfaces](#)



Without
Abstraction



With
Abstraction

Example of Abstraction

1] C++

```
using System;

namespace MyApplication
{
    // Abstract class
    abstract class Animal
    {
        public abstract void animalSound();
        public void sleep()
        {
            Console.WriteLine("Zzz");
        }
    }

    class Pig : Animal
    {
        public override void animalSound()
        {
            Console.WriteLine("The pig says: wee wee");
        }
    }

    class Program
    {
        static void Main(string[] args)
        {
            Pig myPig = new Pig(); // Create a Pig object
            myPig.animalSound();
            myPig.sleep();
        }
    }
}
```

Output :- The pig says: wee wee
Zzz

2] C#

```
using System;

namespace MyApplication
{
    // Abstract class
    abstract class Animal
    {
        public abstract void animalSound();
        public void sleep()
        {
            Console.WriteLine("Zzz");
        }
    }

    class Pig : Animal
    {
        public override void animalSound()
        {
            Console.WriteLine("The pig says: wee wee");
        }
    }

    class Program
    {
        static void Main(string[] args)
        {
            Pig myPig = new Pig(); // Create a Pig object
            myPig.animalSound();
            myPig.sleep();
        }
    }
}
```

Output :- The pig says: wee wee
Zzz

3] Java

```
abstract class Bike{
    Bike(){System.out.println("bike is created");}
    abstract void run();
    void changeGear(){System.out.println("gear changed");}
}

class Honda extends Bike{
    void run(){System.out.println("running safely..");}
}

class TestAbstraction2{
    public static void main(String args[]){
        Bike obj = new Honda();
        obj.run();
        obj.changeGear();
    }
}
```

Output :- bike is created
 running safely..
 gear changed

4] Python

```
from abc import ABC, abstractmethod
class Car(ABC):
    def mileage(self):
        pass

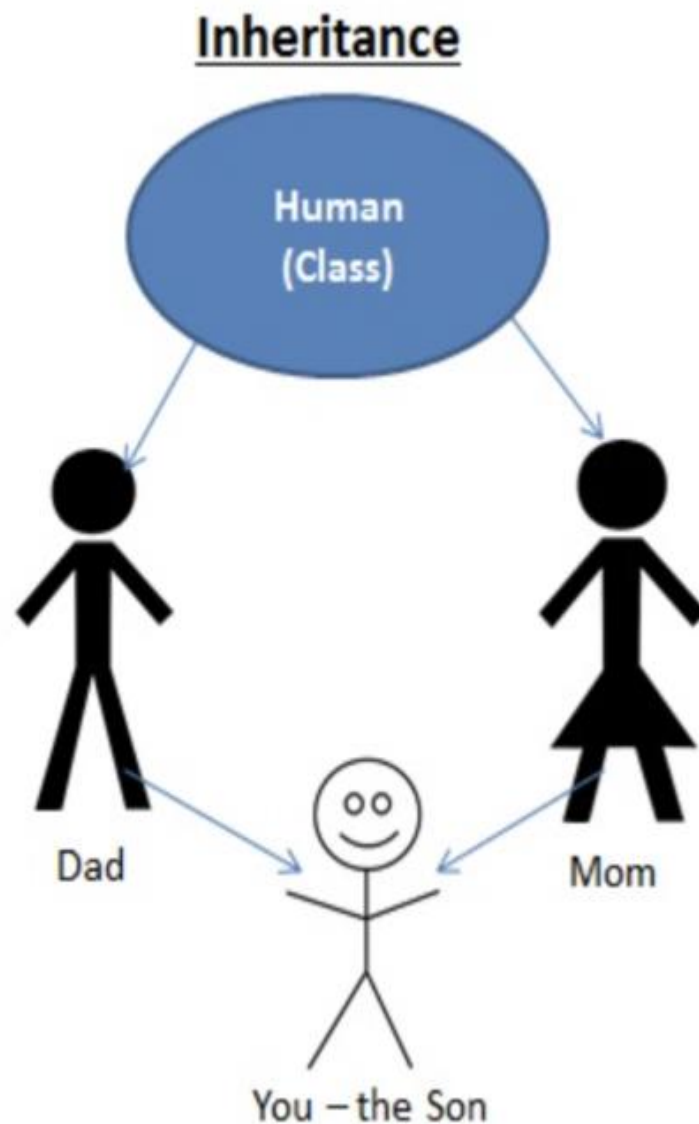
class Tesla(Car):
    def mileage(self):
        print("The mileage is 30kmph")
class Suzuki(Car):
    def mileage(self):
        print("The mileage is 25kmph ")
class Duster(Car):
    def mileage(self):
        print("The mileage is 24kmph ")

class Renault(Car):
    def mileage(self):
        print("The mileage is 27kmph ")

t= Tesla ()
t.mileage()
r = Renault()
r.mileage()
s = Suzuki()
s.mileage()
d = Duster()
d.mileage()
```

Output :- The mileage is 30kmph
The mileage is 27kmph
The mileage is 25kmph
The mileage is 24kmph

3] Inheritance :- Inheritance in Java is a mechanism in which one object acquires all the properties and behaviours of a parent object. It is an important part of OOPs (Object Oriented programming system).



Example of Inheritance

1] C++

```
#include <iostream>
#include <string>
using namespace std;

// Base class
class Vehicle {
public:
    string brand = "Ford";
    void honk() {
        cout << "Tuut, tuut! \n" ;
    }
};

// Derived class
class Car: public Vehicle {
public:
    string model = "Mustang";
};

int main() {
    Car myCar;
    myCar.honk();
    cout << myCar.brand + " " + myCar.model;
    return 0;
}
```

Output :- Tuut, tuut!
Ford Mustang

2] C#

```
using System;

namespace InheritanceApplication {
    class Shape {
        public void setWidth(int w) {
            width = w;
        }
        public void setHeight(int h) {
            height = h;
        }
        protected int width;
        protected int height;
    }

    // Derived class
    class Rectangle: Shape {
        public int getArea() {
            return (width * height);
        }
    }
    class RectangleTester {
        static void Main(string[] args) {
            Rectangle Rect = new Rectangle();

            Rect.setWidth(5);
            Rect.setHeight(7);

            // Print the area of the object.
            Console.WriteLine("Total area: {0}", Rect.getArea());
            Console.ReadKey();
        }
    }
}
```

Output :- Total area: 35

3] Java

```
class Vehicle {  
    protected String brand = "Ford";  
    public void honk() {  
        System.out.println("Tuut, tuut!");  
    }  
}  
  
class Car extends Vehicle {  
    private String modelName = "Mustang";  
    public static void main(String[] args) {  
        Car myFastCar = new Car();  
        myFastCar.honk();  
        System.out.println(myFastCar.brand + " " + myFastCar.modelName);  
    }  
}
```

Output :- Tuut, tuut!
 Ford Mustang

4] Python

```
class Person:
    def __init__(self, fname, lname):
        self.firstname = fname
        self.lastname = lname

    def printname(self):
        print(self.firstname, self.lastname)
```

#Use the Person class to create an object, and then execute the printname method:

```
x = Person("John", "Doe")
x.printname()
```

Object :- John Doe

4] Polymorphism :- Polymorphism is the ability of any data to be processed in more than one form. The word itself indicates the meaning as **poly** means many and **morphism** means types. Polymorphism is one of the most important concept of object oriented programming language. The most common use of polymorphism in object-oriented programming occurs when a parent class reference is used to refer to a child class object. Here we will see how to represent any function in many types and many forms.

Save a new Contact



1] C++

```
#include <iostream>
#include <string>
using namespace std;
// Base class
class Animal {
public:
    void animalSound() {
        cout << "The animal makes a sound \n" ;
    }
};

class Pig : public Animal {
public:
    void animalSound() {
        cout << "The pig says: wee wee \n" ;
    }
};

class Dog : public Animal {
public:
    void animalSound() {
        cout << "The dog says: bow wow \n" ;
    }
};

int main() {
    Animal myAnimal;
    Pig myPig;
    Dog myDog;
    myAnimal.animalSound();
    myPig.animalSound();
    myDog.animalSound();
    return 0;
}
```

Output :- The animal makes a sound
The pig says: wee wee
The dog says: bow wow

2] C#

```
using System;
namespace MyApplication
{
    class Animal // Base class (parent)
    {
        public void animalSound()
        {
            Console.WriteLine("The animal makes a sound");
        }
    }

    class Pig : Animal // Derived class (child)
    {
        public void animalSound()
        {
            Console.WriteLine("The pig says: wee wee");
        }
    }

    class Dog : Animal // Derived class (child)
    {
        public void animalSound()
        {
            Console.WriteLine("The dog says: bow wow");
        }
    }

    class Program
    {
        static void Main(string[] args)
        {
            Animal myAnimal = new Animal(); // Create a Animal object
            Animal myPig = new Pig(); // Create a Pig object
            Animal myDog = new Dog(); // Create a Dog object

            myAnimal.animalSound();
            myPig.animalSound();
            myDog.animalSound();
        }
    }
}
```

Output :- The animal makes a sound
 The animal makes a sound
 The animal makes a sound

3] Java

```
class Animal {  
    public void animalSound() {  
        System.out.println("The animal makes a sound");  
    }  
}  
  
class Pig extends Animal {  
    public void animalSound() {  
        System.out.println("The pig says: wee wee");  
    }  
}  
  
class Dog extends Animal {  
    public void animalSound() {  
        System.out.println("The dog says: bow wow");  
    }  
}  
  
class Main {  
    public static void main(String[] args) {  
        Animal myAnimal = new Animal();  
        Animal myPig = new Pig();  
        Animal myDog = new Dog();  
  
        myAnimal.animalSound();  
        myPig.animalSound();  
        myDog.animalSound();  
    }  
}
```

Output :- The animal makes a sound
 The pig says: wee wee
 The dog says: bow wow

4] Python

```
class Cat:
    def __init__(self, name, age):
        self.name = name
        self.age = age

    def info(self):
        print(f"I am a cat. My name is {self.name}. I am {self.age} years old.")

    def make_sound(self):
        print("Meow")

class Dog:
    def __init__(self, name, age):
        self.name = name
        self.age = age

    def info(self):
        print(f"I am a dog. My name is {self.name}. I am {self.age} years old.")

    def make_sound(self):
        print("Bark")

cat1 = Cat("Kitty", 2.5)
dog1 = Dog("Fluffy", 4)

for animal in (cat1, dog1):
    animal.make_sound()
    animal.info()
    animal.make_sound()
```

Output :- Meow
I am a cat. My name is Kitty. I am 2.5 years old.
Meow
Bark
I am a dog. My name is Fluffy. I am 4 years old.
Bark
