**Hello world**

using System; // System is name space

namespace CsharpConsoleApp

{

class Program

{

static void Main(string[] args)

{

Console.WriteLine("Hello World");

}

}

}

**Data Types And Variable**

class Program

{

static void Main(string[] args)

{

string message = "Hello World";

int intVar = 10;

float floatVar = 10.2f;

char charVar = 'E';

bool boolVar = false;

Console.WriteLine(message);

Console.WriteLine(intVar);

Console.WriteLine(floatVar);

Console.WriteLine(charVar);

Console.WriteLine(boolVar);

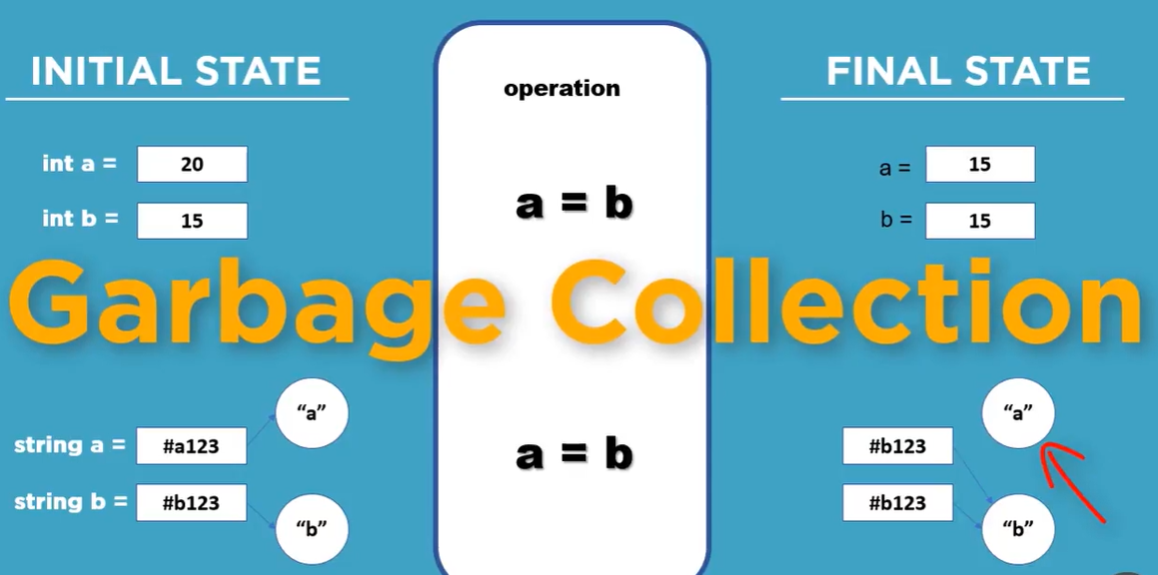
Console.ReadKey();

}

}

**Value vs Reference Type**

Example: class, objects, array, Strings, Interface etc..



**Type conversion**

1-Implicit = from smaller to larger data type

2-Explicit = Larger to small data type

// implicit

int num = 1231241234;

long newNum = num;

// explicit

double x = 3434.34;

int y = (int)x;

**Constant**

class Program

{

static void Main(string[] args)

{

// Area

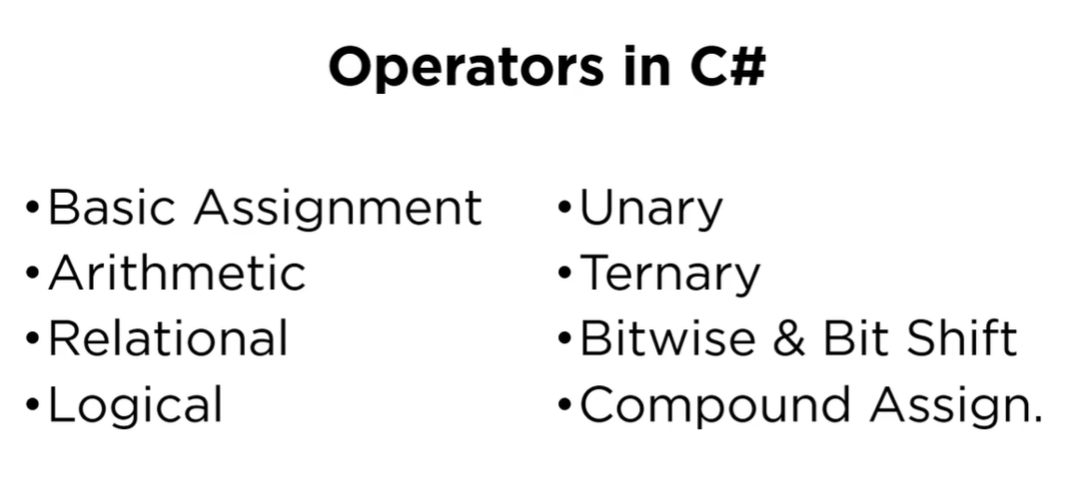
const double PI = 3.14;

PI = 100; // error cost constant can't change

}

}

Operator



**Basic assignment**

// assigning operator

int a = 10;

int b = a; //

**Arithmetic**

double firstNr = 5.4, secondNr = 4.34, result;

// add

result = firstNr + secondNr;

Console.WriteLine("{0} + {1} = {2}", firstNr, secondNr, result);

// sub

result = firstNr - secondNr;

Console.WriteLine("{0} - {1} = {2}", firstNr, secondNr, result);

// multi

result = firstNr \* secondNr;

Console.WriteLine("{0} x {1} = {2}", firstNr, secondNr, result);

// div

result = firstNr / secondNr;

Console.WriteLine("{0} / {1} = {2}", firstNr, secondNr, result);

// modul

result = firstNr % secondNr;

Console.WriteLine("{0} % {1} = {2}", firstNr, secondNr, result);

**Relation Operator ==, > ,< ,>= ,<= !=**

double firstNr = 5.4, secondNr = 4.34;

bool result;

result = firstNr == secondNr;

Console.WriteLine("{0} == {1} = {2}", firstNr, secondNr, result);

result = firstNr > secondNr;

Console.WriteLine("{0} > {1} = {2}", firstNr, secondNr, result);

result = firstNr < secondNr;

Console.WriteLine("{0} < {1} = {2}", firstNr, secondNr, result);

result = firstNr >= secondNr;

Console.WriteLine("{0} >= {1} = {2}", firstNr, secondNr, result);

result = firstNr <= secondNr;

Console.WriteLine("{0} <= {1} = {2}", firstNr, secondNr, result);

result = firstNr != secondNr;

Console.WriteLine("{0} != {1} = {2}", firstNr, secondNr, result);

**Logical Operator || And &&**

double firstNr = 10, secondNr = 4.34;

bool result;

result = (firstNr == secondNr) || (firstNr > 5);

Console.WriteLine(result);

result = (firstNr == secondNr) && (firstNr > 5);

Console.WriteLine(result);

**Unary Operator +var, -var ,++var, --var, !var**

int number = 10, result;

bool flag = true;

result = +number;

Console.WriteLine("+number = "+result);

result = -number;

Console.WriteLine("-number = "+result);

result = ++number;

Console.WriteLine("++number = " + result);

result = --number;

Console.WriteLine("--number = "+result);

Console.WriteLine("!flag = " + !flag);

**Ternary Operator variable = condition ? expression1 : expression2;**

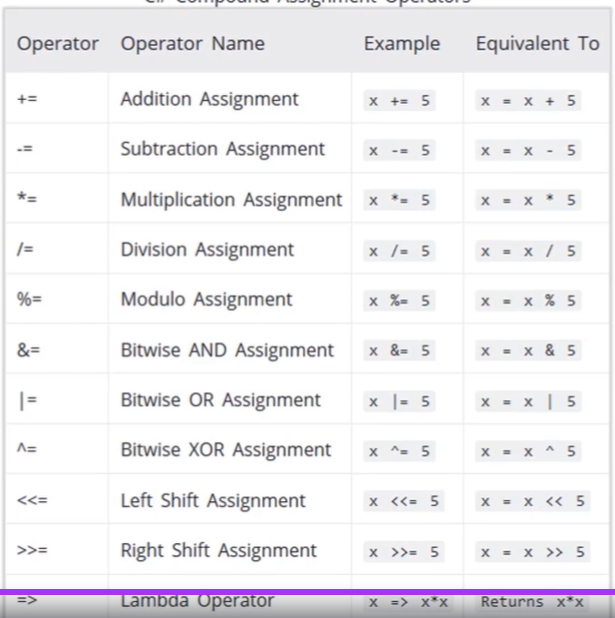
int number = 10;

string result;

result = (number > 0) ? "Greater" : "Not Greater";

Console.WriteLine("{0} is {1}", number, result);

**Compound operator**



int number = 10;

number += 5;

Console.WriteLine(number);

number &= 5;

Console.WriteLine(number); //number = 5

number ^= 5;

Console.WriteLine(number); //number = 0

**Decision Statement**

int number = 0;

if(number > 0)

{

Console.WriteLine("number is greater");

}

else

{

Console.WriteLine("number is positive");

}

switch (number)

{

case 0: Console.WriteLine("number is greater"); break;

case 1: Console.WriteLine("number is positive"); break;

default: Console.WriteLine("number is default"); break;

}

**Loop (for, while, do-while)**

for(int i = 1; i< 11; i++)

{

Console.WriteLine($"{i} - Hello!");

}

int j = 1;

while(j< 11)

{

Console.WriteLine($"{j} - Hello!");

j++;

}

j = 0;

do

{

Console.WriteLine($"{j} - Hello!");

j++;

} while (j < 11);

**Access Modifiers**

**public, private, protected, internal, protected internal**

**public:** accessed from anywhere within the application.

**private:** accessed from only within the class.

**protected:** can only be accessed from the same class and its derived classes (child).

**internal**: can be accessed only within the same assembly.

**Protected internal :** can be accessed from the same assembly and the derived class of the containing class from any other assembly