using System;

class Program

{

static void Main(string[] args)

{

Console.Write("Enter Name");

string name = Console.ReadLine();

Console.WriteLine("Name is " + name);

}

}

Functions :

Method Overloading

Named Parameter

Optional Parameters

Call By value

Call By reference

Params array

// Code without Method Overloading

using System;

class Program

{

static int add(int x, int y)

{

return x + y;

}

static int add1(int x, int y, int z)

{

return x + y + z;

}

static int add2(int x, int y, int z , int a)

{

return x + y + z + a;

}

static string add\_string(string a, string b)

{

return a + " " + b;

}

static float add5(int x, float f)

{

return x + f;

}

static void Main(string[] args)

{

int res = add(2, 3);

Console.WriteLine("Result of Addition " + res);

res = add1(2, 3, 5);

Console.WriteLine("Result of Addition " + res);

}

}

Method Overloading : Function names are kept same but their signature is different , number and type of parameters will differ ( Also known as Polymorphism) (One Name different forms)

Function return type is not considered in this.

Advantage : No need to remember different function names while calling them

// With Method Overloading

using System;

class Program

{

static int add(int x, int y)

{

return x + y;

}

//static float add(int x, int y)

//{

// return x + y;

//}

static int add(int x, int y, int z)

{

return x + y + z;

}

static int add(int x, int y, int z , int a)

{

return x + y + z + a;

}

static string add(string a, string b)

{

return a + " " + b;

}

static float add(int x, float f)

{

return x + f;

}

static void Main(string[] args)

{

int res = add(2, 3);

Console.WriteLine("Result of Addition " + res);

res = add(2, 3, 5);

Console.WriteLine("Result of Addition " + res);

Console.WriteLine("Full Name is " + add("Ajay", "Sood"));

}

}

// Without Named Parameters

using System;

class Program

{

static void DisplayDetails(int rn , string name , string batch

, int marks)

{

Console.WriteLine("Your RN is " + rn);

Console.WriteLine("Your Name is " + name);

Console.WriteLine("Your Batch is " + batch);

Console.WriteLine("Your Marks are " + marks);

}

static void Main(string[] args)

{

DisplayDetails(1, "Ajay", "B001", 89);

}

}

--

Named Parameters : While calling a method , when we pass parameter’ values , we pass it alongwith parameter name

Advantage : We can change sequence

It makes program easy to understand

using System;

class Program

{

static void DisplayDetails(int rn , string name , string batch

, int marks)

{

Console.WriteLine("Your RN is " + rn);

Console.WriteLine("Your Name is " + name);

Console.WriteLine("Your Batch is " + batch);

Console.WriteLine("Your Marks are " + marks);

}

static void Main(string[] args)

{

DisplayDetails(1, "Ajay", "B001", 89);

// Named Parameter

DisplayDetails(rn: 2, batch: "B002", marks: 90, name: "Deepak");

}

}

Optional Parameters : While defining method, we give parameters some default values , so If we don’t provide value to those parameters, they will tale that default value

We have to provide values from RIGHT TO LEFT

using System;

class Program

{

static void simpleInterest(int p, int r=9 , int t=10)

{

Console.WriteLine((p\*r\*t)/100);

}

//static void DisplayDetails(int rn , string name , string batch

// , int marks)

// {

// Console.WriteLine("Your RN is " + rn);

// Console.WriteLine("Your Name is " + name);

// Console.WriteLine("Your Batch is " + batch);

// Console.WriteLine("Your Marks are " + marks);

// }

static void Main(string[] args)

{

//DisplayDetails(1, "B001", "Ajay", 89);

//// Named Parameter

//DisplayDetails(rn: 2, batch: "B002", marks: 90, name: "Deepak");

simpleInterest(p:12000, t:3, r:6);

simpleInterest(p:12000, r:3);

simpleInterest(12000);

}

}

Call By Value : When the parameter is passed as value to the called method (default behaviour)

using System;

class Program

{

static void change1(int x)

{

x = 200;

Console.WriteLine("Value of x in Change1 is " + x);

}

static void Main(string[] args)

{

int x = 10;

Console.WriteLine("Value of x in Main before calling Change1 is " + x);

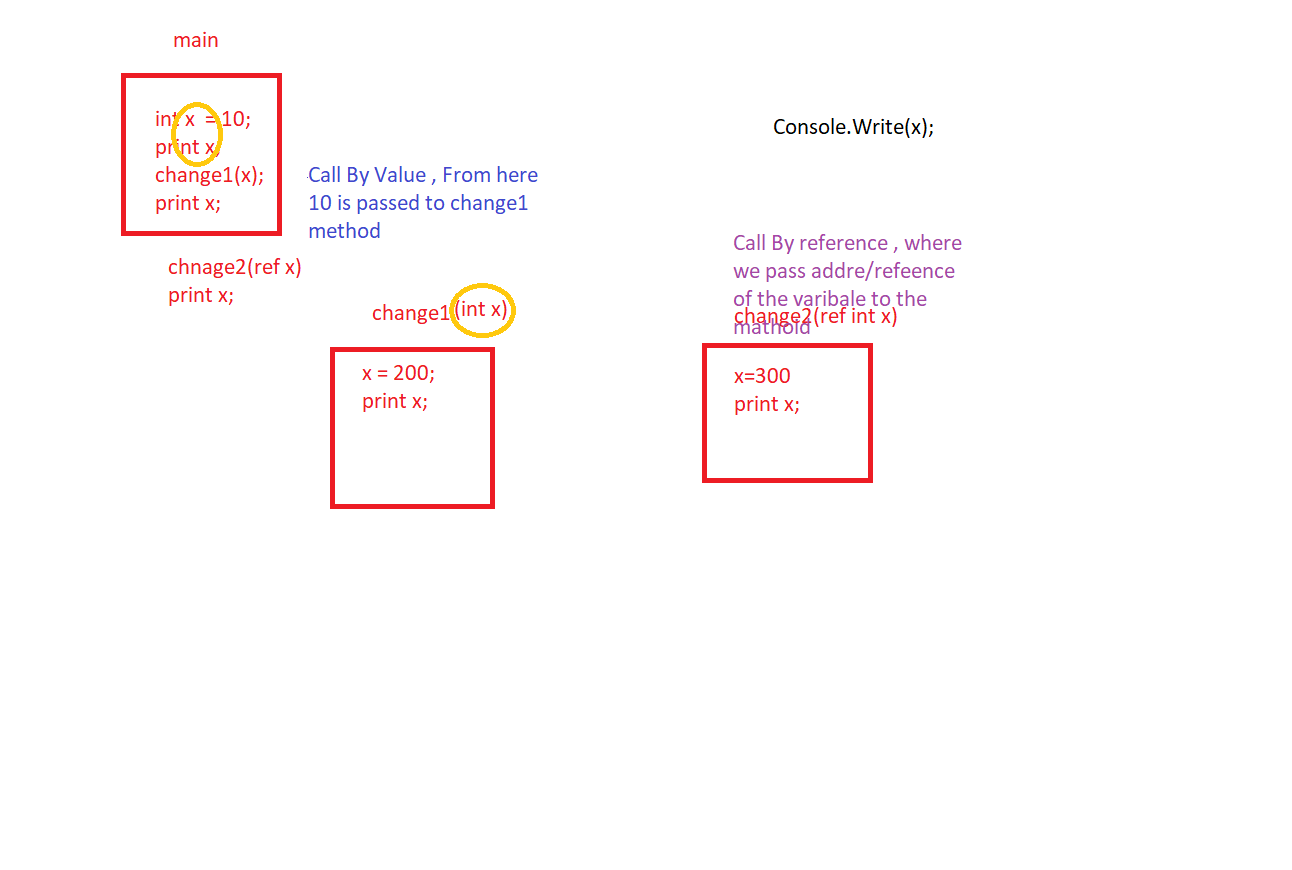
change1(x);

Console.WriteLine("Value of x in Main after calling Change1 is " + x);

}

}

Call By Reference : When we pass reference / address of the variable to the called method



using System;

class Program

{

static void change1(int x)

{

x = 200;

Console.WriteLine("Value of x in Change1 is " + x);

}

static void change2(ref int x)

{

x = 300;

Console.WriteLine("Value of x in Change2 is " + x);

}

static void Main(string[] args)

{

int x = 10;

Console.WriteLine("Value of x in Main before calling Change1 is " + x);

change1(x);

Console.WriteLine("Value of x in Main after calling Change1 is " + x);

change2(ref x);

Console.WriteLine("Value of x in Main after calling Change2 is " + x);

}

}

Call by Value : We pass value of the variable, so any changes made there are not reflected in the calling method

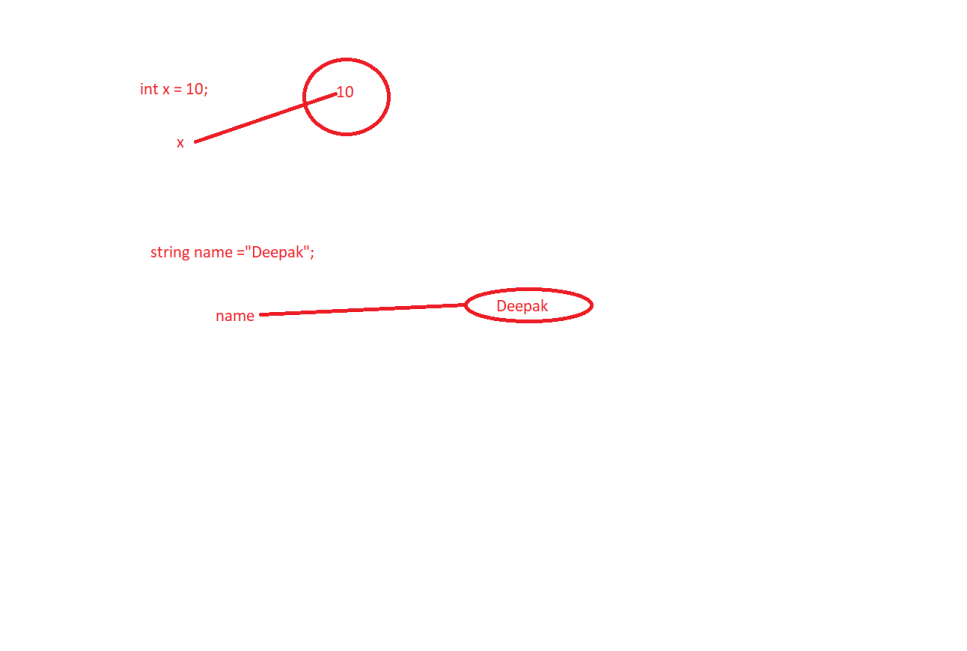
Call by Reference : We pass address of the variable, so any changes made there are reflected in the calling method

Variables of how many types

* Value Type : which stores the value . int , float , Boolean, char, struct, enum
* Reference Type : which stores the reference of the variable , string , class

When we declare varibale , they get memory and are stored somewhere in memory

Value type varibales are stored in stack

Reference type variables are stored in heap

Value type variables are stored in stack Reference type variables are stored in heap

|  |
| --- |
|  |
|  |
|  |
|  |
|  |
| + |
| 40 |
| 10  Deepak |
| 90 |

String name = “Deepak”

Garbage Collection : Removing unused memory blocks

Components of DotNet Framework