|  |
| --- |
| Day 20 Assignment  By  J Siva Naga Prasanna |

|  |
| --- |
| 1.Research and understand the scope of variables in c#? |
| * The Scope of the variable determines the accessibility of the variable to a particular part of the application. * Variables can be declared within the class, method, and code block of a loop, condition   Types:   * Class Level Scope   A variable declared within a class is known as a field. It has a class-level scope that can be accessed anywhere in the class   * Method Level Scope   A variable declared within a method has a method level Scope. It is also known as a local variable of the method where it is declared. It cannot be accessed outside the method.   * Code-Block Level Scope   A variable declared within a loop or any block within brackets has the code-block level scope |
|  |

|  |
| --- |
| 2.What are the delegates in C# .Write the points discussed about the delegates in the class |
| Delegate is a function pointer.   * Using delegates we can call (or) point to one or more methods. * When declaring a delegate: * Return and parameters must match with the methods you want to point using the delegate.   Benefits:   * Using single call from delegate, all your methods pointing to delegate will be called.   Types :   * Single-cast delegate. * Multi-cast delegate. |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  namespace day20pro2  {  public delegate void MyCaller(int a, int b);  internal class Program  {  public static void Add(int a, int b)  {  Console.WriteLine(a + b);  }  public static void Mul(int a, int b)  {  Console.WriteLine(a \* b);  }  public static void Div(int a, int b)  {  Console.WriteLine(a / b);  }  static void Main(string[] args)  {  MyCaller mc = new MyCaller(Add);  mc += Mul;  mc += Div;  // 17, 19  mc(17, 19);  Console.WriteLine("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");  //23,26  mc(23, 26);  Console.WriteLine("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");  //34,38  mc(34, 38);  Console.ReadLine();  }  }  } |
|  |

|  |
| --- |
| **3. What are Nullable types in C#**  **WACP to illustrate Nullable types**  **Write some properties of Nullable types (like HasValue)** |
| Nullable type allows you to assign a null value to a variable.    Properties of nullable types:   * Nullable <T> type allows assignment of null to value types. * ? operator is a shorthand syntax for nullable types. * Use value property to get the value of nullable type. * Use Hasvalue property to check whether value is assigned to nullable type or not. * Static Nullable class is a helper class to compare nullable types |
| Code:  using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  namespace day20pro3  {  internal class Program  {  static void Main(string[] args)  {  byte? input = 5;  if (input.HasValue)  Console.WriteLine(input \* input);  else  Console.WriteLine("No vaiue");  Console.ReadLine();  }  }  } |
| Output: |

|  |
| --- |
| **4. out, ref - parameters**  **research on these two types of parameters**  **write a C# program to illustrate the same** |
| Ref:   * Ref is used to state that the parameter passed may be modified by the method. * By default, a reference type passed into a method will have any changes made to its values reflected outside the method as well.   If you assign the reference type to a new reference type inside the method, those changes will only be local to the method |
| Code:  using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  namespace day20pro4  {  internal class Program  {  public static string GetNextName(ref int id)  {  string returnText = "Next-" + id.ToString();  id += 3;  return returnText;  }  static void Main(string[] args)  {  int i = 231;  Console.WriteLine("Previous value of integer i:" + i.ToString());  string test = GetNextName(ref i);  Console.WriteLine("Current value of intger i:" + i.ToString());  Console.ReadLine();  }  }  } |
| Output: |

|  |
| --- |
| Out keyword:   * Out keyword is used to state that the parameter passed must be modified by the method. * Ref requires the parameter to have been initialized before being passed to a method. * The modifier does not require this and is typically not initialized prior to being used in a method. |
| Code:  using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  namespace day20pro5  {  internal class Program  {  public static string GetNextNameByOut(out int id)  {  id = 2;  string returnText = "Next-" + id.ToString();  return returnText;  }  static void Main(string[] args)  {  int i = 4;  Console.WriteLine("Previous value of integer i:" + i.ToString());  string test = GetNextNameByOut(out i);  Console.WriteLine("Current value of intger i:" + i.ToString());  Console.ReadLine();  }  }  } |
| Output: |