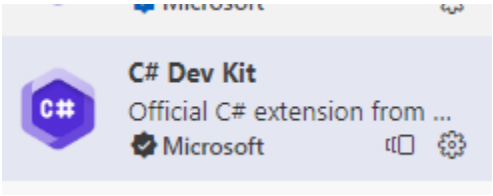
1. Install VS code
2. Open view->Extension->C# development kit



3. Install .Net sdk: https://dotnet.microsoft.com/en-us/download/dotnet/thank-you/sdk-8.0.100-windows-x64-installer
4. To create project type following command in terminal
   dotnet new console -o "firstapp"

5. You can check app and default Program.cs file edit file for first program
6. On terminal type

dotnet run

Program 1:

Create a program that takes user input for two numbers and performs basic arithmetic operations (addition, subtraction, multiplication, and division). This program demonstrates the use of primitive data types, expressions, and control structures.

```
using System;
class HelloWorld
{
  static void Main()
   {

        int caseSwitch ;
        char ch='y';
        while(ch=='y')
        {
           Console.WriteLine("\nSelect case 1 to 4 for");

           caseSwitch   =Convert.ToInt32(Console.ReadLine());
           int num1,num2,num3;
           Console.WriteLine("Enter two numbers");
           num1=Convert.ToInt32(Console.ReadLine());
           num2=Convert.ToInt32(Console.ReadLine());
           num3=0;
           switch (caseSwitch)
           {
           case 1:
              num3=num1+num2;
           break;
```

```
            case 2:
               num3=num1-num2;
            break;
            case 3:
               num3=num1*num2;
            break;
            case 4 :
               num3=num1/num2;
            break;
            default:
               Console.WriteLine("Value didn't match earlier.");
            break;
            }
            String res=String.Format("Result of {0} and {1} are {2}",num1,num2,num3);
            Console.WriteLine(res);
            Console.WriteLine("Do u want continue");
            ch= Console.ReadKey().KeyChar;
     }

    }
}
```

//explanation

- using System means that we can use classes from the **System namespace.**
- **namespace** is used to organize your code, and it is a container for classes and other namespaces.
- **Console** is a class of the **System namespace**, which has a **WriteLine**() method that is used to output/print text.
    - Note: when we create console application new version automatically refer builtin System so no need to add but older version need this namspace

Program 2:

a. Write a c# program to implement Inventory system in Game using List Data structure.

```
using System;
using System.Collections.Generic;


class Inventory
{
```

```
static void Main()
{
   string name;
   List<string>item=new List<string>();
   string values;
   Console.WriteLine("Enter Your Name");
   name=Console.ReadLine();
   Console.WriteLine("Enter number of items you want to add in list");
   int n=Convert.ToInt32(Console.ReadLine());

   for(int i=0;i<n;i++)
   {
      Console.WriteLine("Enter item name");
      values=Console.ReadLine();
      item.Add(values);
   }
   Console.WriteLine("Item in Inventory are:");
   string str;
   for(int i=0;i<item.Count;i++)
   {
      str=String.Format("{0}-{1} | ",i,item[i]);
      Console.Write(str);
   }
   char ch='y';
   int pos;
   Console.WriteLine("\nEnter more item in position\n");
   while(ch=='y')
   {
      Console.WriteLine("Enter position");
      pos=Convert.ToInt32(Console.ReadLine());
      Console.WriteLine("Enter Item");
      values=Console.ReadLine();
      item.Insert(pos,values);
      Console.WriteLine("Do u want continue Adding y/n");
      ch= Console.ReadKey().KeyChar;

   }
   Console.WriteLine("Item in Inventory are:");
   for(int i=0;i<item.Count;i++)
   {
      str=String.Format("{0}-{1} | ",i,item[i]);
      Console.Write(str);
   }
   Console.WriteLine("Do u want Delete item y/n");
   ch= Console.ReadKey().KeyChar;

   if(ch=='y')
   {
       Console.WriteLine("Enter item name to remove");
```

```
            values=Console.ReadLine();
            item.Remove(values);
        }
       Console.WriteLine("{0} Your  Final Inventory list",name);
        foreach(string val in item)
        {
            Console.Write(String.Format("{0}|",val));
        }
    }
}
```

b. Create a program that demonstrates the use of a dictionary to store and retrieve key-value pairs

A Dictionary<TKey, TValue> is a generic collection that consists of elements as key/value pairs that are not sorted in an order

Syntax:

```
Dictionary<int, string> country = new Dictionary<int, string>();
```

Here, country is a dictionary that contains int type keys and string type val

```
using System;
using System.Collections;
class Program
{
    public static void Main()
    {
        // create a dictionary
        Dictionary<int, string> country = new Dictionary<int,
string>();
        Console.WriteLine("Enter number of items you want to add in
list");
        int n=Convert.ToInt32(Console.ReadLine());
        int countryCode;
        string countryName;
        for(int i=0;i<n;i++)
        {
```

```
            Console.WriteLine("Enter Country Code");
            countryCode=Convert.ToInt32(Console.ReadLine());
            Console.WriteLine("Enter Country Name");
            countryName=Console.ReadLine().ToString();
            country.Add(countryCode,countryName);
        }


        Console.WriteLine("Dictionary Data is");
            foreach (KeyValuePair<int, string> entry in country)
        {
            Console.WriteLine(entry.Key + " : " + entry.Value);
        }

        Console.WriteLine("Enter key to fin value");
        int key=Convert.ToInt32(Console.ReadLine());
        Console.WriteLine("Value having key {0}  is {1}: "
,key,country[key]);
    }
}
```

//other operations can be performed on dictionary are:

- Change Elements
- Remove Elements

Program 3: Demonstrating creation of Delegates and Events in c#.

```
using System;

// Step 1: Define a delegate for the operation
public delegate void MyDelegate(int a, int b);

// Step 2: Define a class that performs the operation and raises an event
class A
{
```

```
    // Define an event of type MathOperationHandler
    public event MyDelegate MyEvent;

    // Method to perform the addition operation
    public void Addition(int a, int b)
    {
        int result = a + b;
        Console.WriteLine($"The result of {a} + {b} is {result}");
    if(MyEvent==null)
    {

    Console.WriteLine("Empty Events");

    }
        // Check if there are any subscribers to the event
    if (MyEvent != null)
        {
            // Invoke the event, passing the result
            MyEvent(a, b);
        }
    }
}

// Step 3: Define a class that subscribes to the event
class B
{
    // Method that matches the signature of the MathOperationHandler delegate
    public void Multiplication(int a, int b)
    {
        int result = a * b;
        Console.WriteLine($"The result of {a} * {b} : {result}");
    }
}

class Program
{
    static void Main()
    {
        // Create an instance of the math operation class and the result logger
        A a = new A();
        B b = new B();

        // Subscribe the LogResult method to the OperationPerformed event
```

```
        a.MyEvent += b.Multiplication;
        a.Addition(5, 3);

        // Perform the addition operation
         a.MyEvent -= b.Multiplication;

        // Perform the addition operation
        a.Addition(5, 5);

        Console.ReadLine();
    }
}
```

//below program with events and delegates to understand why we need them not for lab record

```
using System;

class A
{

    // Method to perform the addition operation
    public void Addition(int a, int b)
    {
        int result = a + b;
        Console.WriteLine($"The result of {a} + {b} is {result}");
    }

}
class B
{
    // Method that matches the signature of the MathOperationHandler delegate
    public void Multiplication(int a, int b)
    {
        int result = a * b;
        Console.WriteLine($"The result of the {a} *{b} is {result}");
    }
}

class Program
```

```
{
   static void Main()
   {
      // Create an instance of the math operation class and the result logger
      A a = new A();
      B b = new B();

      // Subscribe the LogResult method to the OperationPerformed event
      a.Addition(5, 3);

      b.Multiplication(5,3);
      a.Addition(5, 5);
      b.Multiplication(5,5);

      Console.ReadLine();
   }
}
```

Program 4:

A. Create a 2D Game environment with an orthographic camera .

Sol:

Step 1: Add assets from unity asset store or photoshop file
Step 2: design environment using tilemap
Step 3: Add required colliders and assets in environment
Step 4: create player

B. Add cinemachine and give Player movement Left and Right

//Add cinemachine

Step 1: Add cinemachine as package
Step2: Select gameobject->cinemachine and add 2D camera
Step 3: Add player at lookAt and Follow
//Add movement
Step 1: create c# scripts
Step 2: Add script to the player
Step 3: Write code to move player

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class PlayerMovement : MonoBehaviour
{
    // Start is called before the first frame update
    public float moveSpeed=5f;
     Rigidbody2D rb;
        void Start()
    {
        rb=GetComponent<Rigidbody2D>();
           }

    // Update is called once per frame
    void Update()
    {
         float horizontalInput = Input.GetAxis("Horizontal");
         MoveLeftRight(horizontalInput);
             }
    void MoveLeftRight(float x)
    {
         rb.velocity = new Vector2(x * moveSpeed, rb.velocity.y);
    }
    }
```

Program 5: Program: Create animation idle,run, jump and add in game using scripting.

Steps:

1. Create Animation by Adding Framers
2. Open Animation Controller and create transition for each state
3. Add parameter and condition in Inspector window
4. Add required parameter in c# script and code

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
```

```
public class PlayerMovement : MonoBehaviour
{
    // Start is called before the first frame update
    public float moveSpeed=5f;
     Rigidbody2D rb;
     //Animation
     Animator anim;
     //Jump
     public float jumpHeight=8f;
    void Start()
    {
        rb=GetComponent<Rigidbody2D>();
        anim=GetComponent<Animator>();
    }

    // Update is called once per frame
    void Update()
    {
         float horizontalInput = Input.GetAxis("Horizontal");
         MoveLeftRight(horizontalInput);
         SetAnimation(horizontalInput);//calling

    }
    void MoveLeftRight(float x)
    {
         rb.velocity = new Vector2(x * moveSpeed, rb.velocity.y);
    }
    void SetAnimation(float x)//defination
    {
        if(x>0||x<0)
        {
            anim.SetBool("run",true);
        }
        else
        {
            anim.SetBool("run",false);
        }
```

```
    }
}
```

Program 6 : Create Physics system like gravity,collider on player and perform player jump on keyboard input.

Steps 1: Add required physics for player by adding Rigidbody2D
Step2: Add required Colliders for player and Ground
Step 3: Generate event OnCollisionEnter2D
Step 4: create required variable and code in script
//Flip Player

Step 1: Check Sprite renderer component
Step 2: Create object of Sprite renderer
Step 3: Code for flip player

```
using System;
using System.Collections;
using System.Collections.Generic;
using Unity.VisualScripting;
using UnityEngine;

public class PlayerMovement : MonoBehaviour
{
    // Start is called before the first frame update
    public float moveSpeed=5f;
     Rigidbody2D rb;
     //Animation
     Animator anim;

      //ground check
    bool isGrounded;
    //for Flip
    [SerializeField]
    SpriteRenderer sp;
     public float jumpHeight=8f;
```

```
    void Start()
    {
        rb=GetComponent<Rigidbody2D>();
        anim=GetComponent<Animator>();
        sp=GetComponent<SpriteRenderer>();
    }

    // Update is called once per frame
    void Update()
    {
         float horizontalInput = Input.GetAxis("Horizontal");
         MoveLeftRight(horizontalInput);
         SetAnimation(horizontalInput);//calling
         if(Input.GetButtonDown("Jump")&&isGrounded)
         {
            PlayJump();
         }


    }
    void MoveLeftRight(float x)
    {
         rb.velocity = new Vector2(x * moveSpeed, rb.velocity.y);
         if(x>0)
         {
            sp.flipX=false;
         }
         else{
            sp.flipX=true;
         }
    }
    void SetAnimation(float x)//defination
    {
        if(x>0||x<0)
        {
            anim.SetBool("run",true);
        }
        else
```

```
        {
            anim.SetBool("run",false);
        }
        if(isGrounded)
        {
        anim.SetBool("isJumping",false);
        }
        else
        {
         anim.SetBool("isJumping",true);
        }

    }
    void PlayJump()
    {
        rb.velocity = new Vector2(rb.velocity.x,jumpHeight);
    }
       public void OnCollisionEnter2D()
    {
        isGrounded=true;
        Debug.Log("colleding");
    }
    public void OnCollisionExit2D()
    {
        isGrounded=false;
    Debug.Log(" Leaving colleding");
    }
}
```

Program 7 : Create score for player when user collects coins here two separate classes need to be created.

Steps:
Step 1: Create new script Gamemanager.cs
Step2: create new script collector.cs
Step 3: Add coin and animate it
Step 4: Drag and drop coin in game environment

Step 5: Code Gamemanager.cs

```
using UnityEngine;
public class Gamemanager : MonoBehaviour
{
    int score=0;

public  void AddCoin(int val)
{
    score=score+val;
    Debug.Log(score);
}

}
```

Step 5: code for coin script when player touches it it will Destroy

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class coin : MonoBehaviour
{
    Gamemanager gm;
    int score=10;
    void Start()
    {
        gm=FindObjectOfType<Gamemanager>();
    }
    void OnTriggerEnter2D(Collider2D other)
    {
        if(other.tag=="Player")
        {
            gm.AddCoin(score);
            Destroy(gameObject);
```

```
        }
    }
}
```

Program 8:
A. Add UI and show score in UI.

Step 1: Right click in Hierarchy window->UI->TextMesh Pro
Step 2: Click import
Step 3: Set textbox
Step 4: Edit Gamemanager script
Step5: Attach UI to Gamemanager

```
using UnityEngine;
using TMPro;
public class Gamemanager : MonoBehaviour
{
    public TextMeshProUGUI txt;
    int score=0;
    // Start is called before the first frame update
    void Start()
    {
        txt.text=score.ToString();
    }
public  void AddCoin(int val)
{
    score=score+val;
    Debug.Log(score);
     txt.text=score.ToString();
}

}
```

B. Create Button UI and call different methods on the button.

Step 1: Add Panel by Rightclick->UI
Step 2: Add two ButtonMesh Pro and set it in Game
Step 3: Name button as Quit and Restart
Step 4 : Edit Gamemanager script to display UI and handle button

```
using UnityEngine;
using TMPro;
using UnityEngine.SceneManagement;
public class Gamemanager : MonoBehaviour
{//code for score
    public TextMeshProUGUI txt;
    int score=0;
    // Start is called before the first frame update
    void Start()
    {
        txt.text=score.ToString();
    }
    public  void AddCoin(int val)
    {
        score=score+val;
        Debug.Log(score);
        txt.text=score.ToString();
    }
    //Code for button
    public void Restart()
    {
        SceneManager.LoadScene(0);
    }
   public void Quit()
   {
    Application.Quit();
   }
}
```

Program 9:
Program: Add player movement on Joystick and touch input for mobile.

//For Joystick:

Step1: Import asset for Joystick from Unity asset store
https://assetstore.unity.com/packages/tools/input-management/joystick-pack-107631

Step 2: Create new scene with Player and ground
Step 3 :Drag the joystick on scene
Step 4: Create new Script JMovement

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Jmovement : MonoBehaviour
{
     float speed = 5.0f;
 public Joystick jstick;
  Rigidbody2D playerRB;
   float movement, verticalmove;

    // Start is called before the first frame update
    void Start()
    {
        playerRB = GetComponent<Rigidbody2D>();

    }

    // Update is called once per frame
    void Update()
    {
         movement = jstick.Horizontal * speed;
```

```
        GetAxis(movement);
    }
    public void GetAxis(float movement)
    {
        //To move player vector2 use 2 parameter one for x and other
for y
        if (movement > 0.2f)
        {
            playerRB.velocity = new Vector2(movement,
playerRB.velocity.y);//only we need x axis;

        }
        else if(movement < 0.2f)
        {
            playerRB.velocity = new Vector2(movement,
playerRB.velocity.y);//only we need x axis;
        }
    }

}
```

Create Drag and Drop which u can test better way in simulator

Step 1 : Create gameobject circle
Step 2 : Add Circle Collider2D on it
Step 3: Create Script DragDrop

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class DragDrop : MonoBehaviour
{
     bool isDragging;
    Collider2D objectCollider;

    // Start is called before the first frame update
```

```
    void Start()
    {
        objectCollider = GetComponent<Collider2D>();

        isDragging = false;
    }

    // Update is called once per frame
    void Update()
    {
        DragAndDrop();
    }

    void DragAndDrop()
    {
        Vector2 mousePosition =
Camera.main.ScreenToWorldPoint(Input.mousePosition);

        if (Input.GetMouseButtonDown(0))
        {
            if (objectCollider ==
Physics2D.OverlapPoint(mousePosition))
            {
                isDragging = true;
            }
            else
            {
                isDragging = false;
            }

        }
        if (isDragging)
        {
            this.transform.position = mousePosition;
        }

        if (Input.GetMouseButtonUp(0))
```

```
        {

            isDragging = false;
        }
    }
}
```