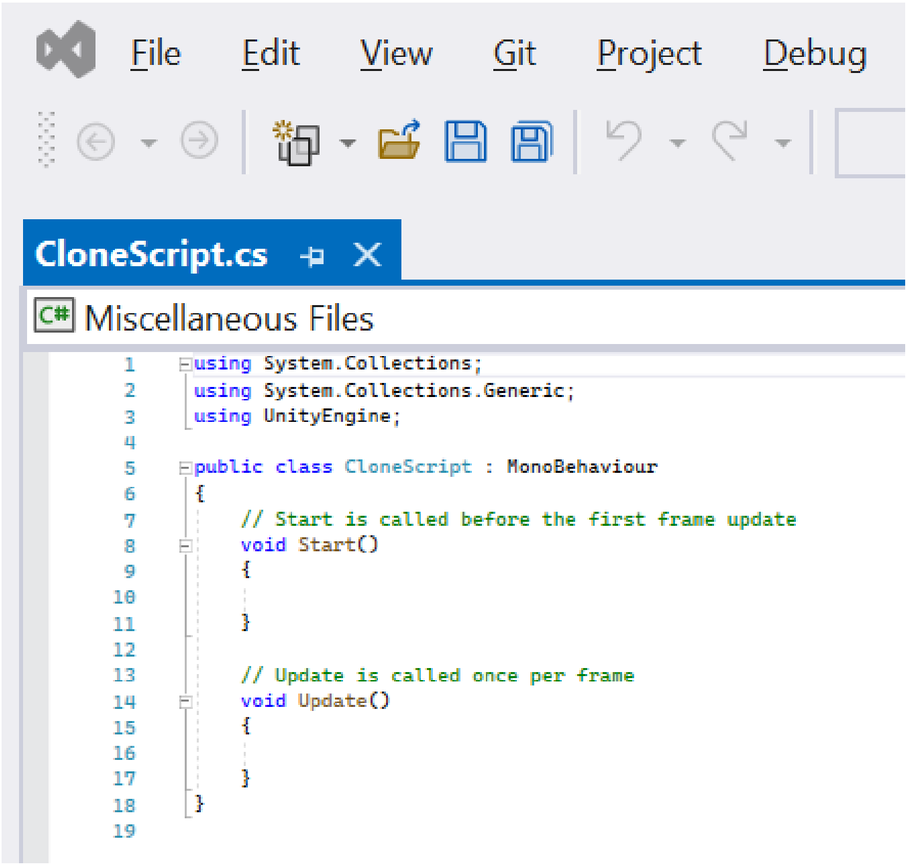
C# SCRIPTS

Within Unity

Script Source Code

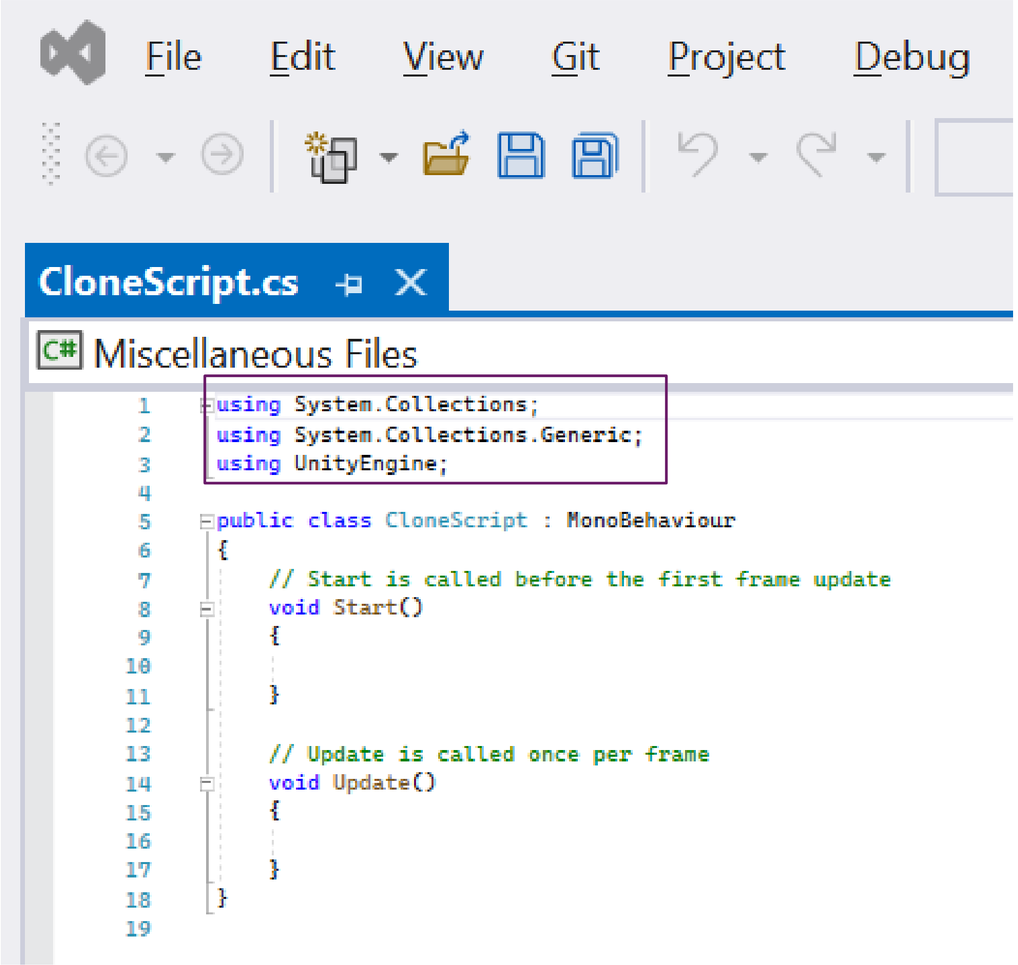
- This is the “source code” of a script.
- A script is a small program, or part of a program, that is executed by a host program.
- Scripts are often interpreted, but not always (e.g., Unity compiles them).
- Coding, programming, and scripting are all the same.



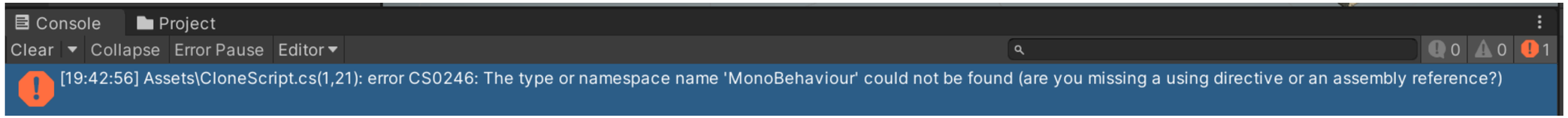
Using Declarations

The "using" declaration says what library parts are used in the current file.

- Extra functionality is provided by libraries.
- Libraries are organized using "namespaces".
- For example: two things called "List" might exist in different libraries.
- Namespaces disambiguate: "MyLibrary.List" and "System.Collections.List".
- Delete them and see what happens.



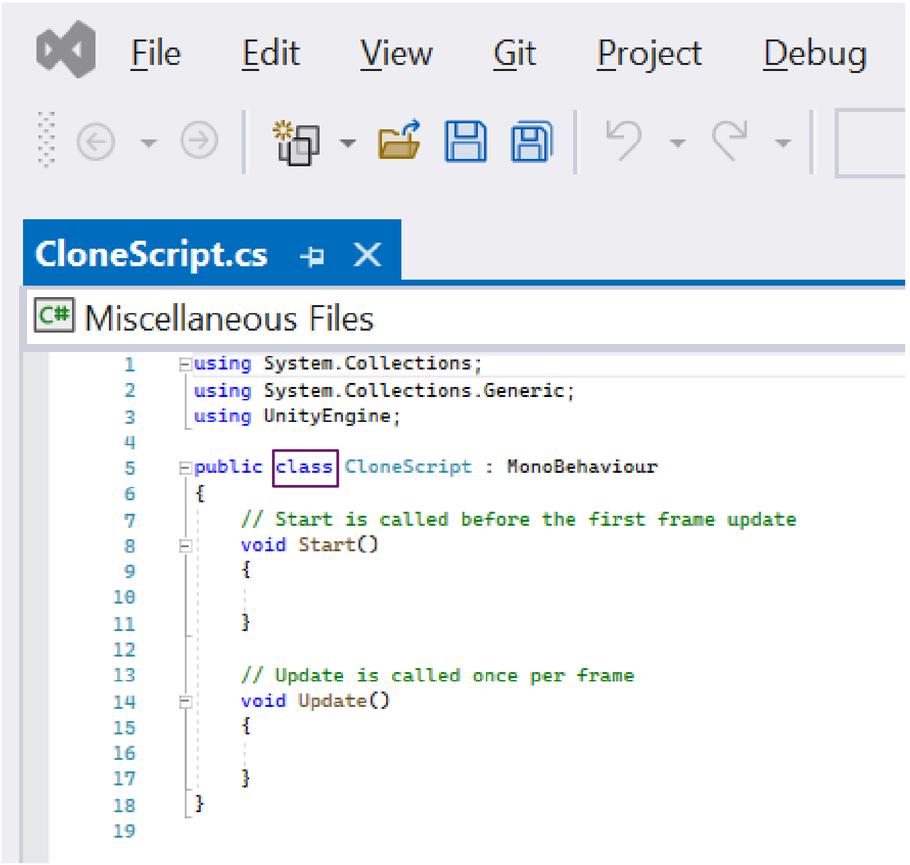
Don’t Panic!



- This is a very common type of error, often because of misspelling, or a missing “using directive”.
- Try double-clicking it.
- Now fix the mistake.

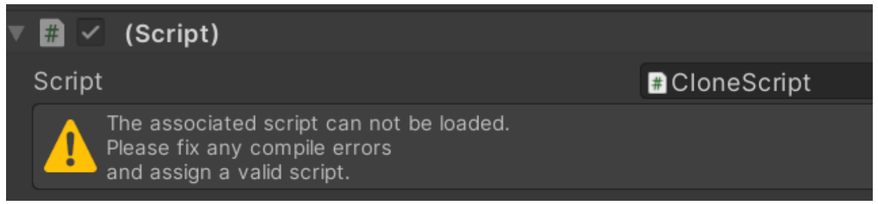
Class

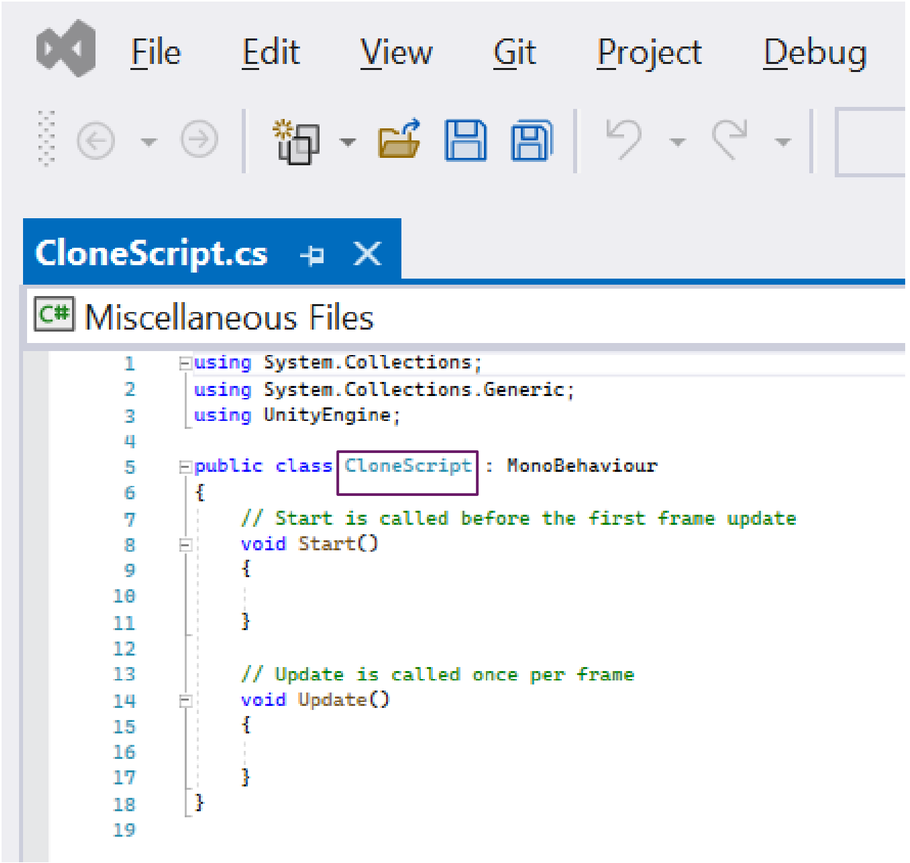
- A class describes a particular type of object.
- Objects contain data and provide operations for accessing or transforming that data.
- This is part of object-oriented programming (OOP) and will be covered in depth on later classes.
- In Unity, all scripts are classes.



Class Name

- The name of the script class must match the file name.
- Try changing the name.
- You should see this in the inspector:



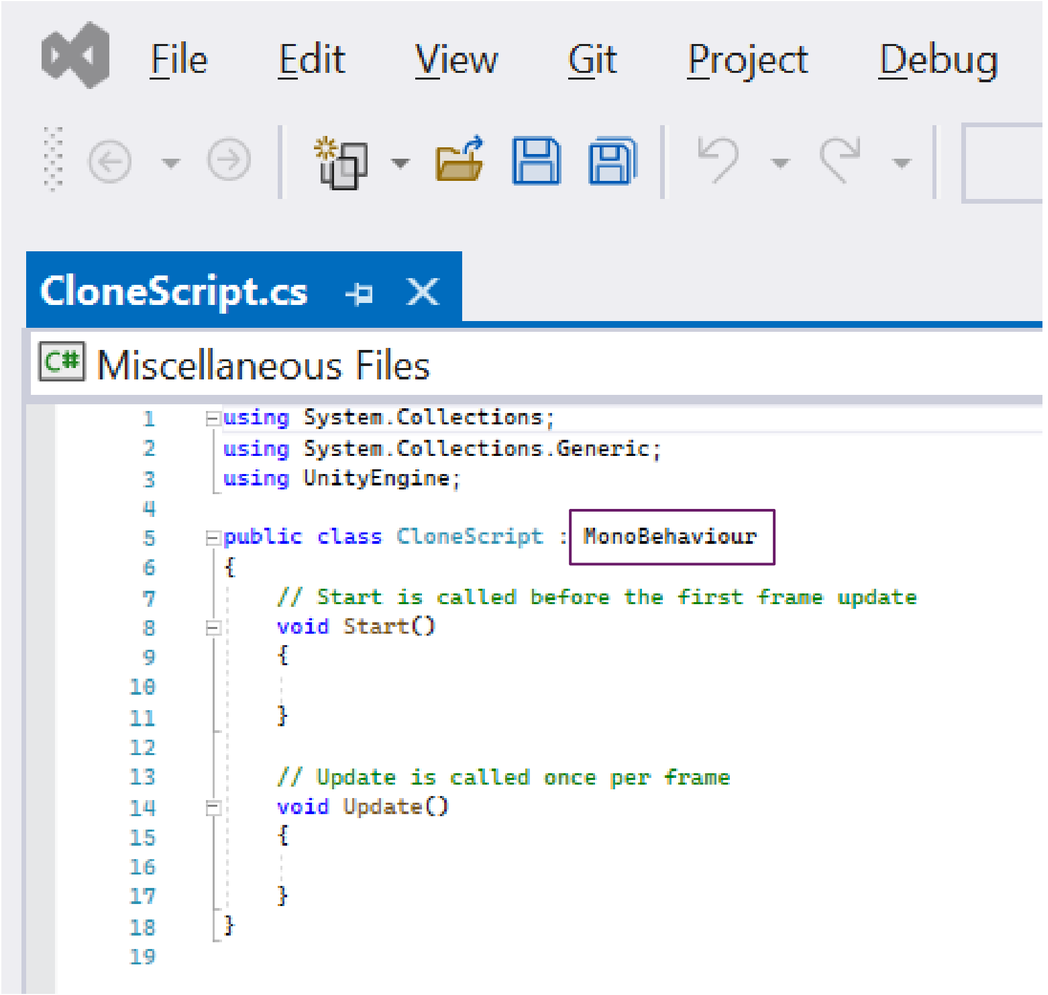


Inheritance

In Unity all scripts are classes that inherit from a Unity class called “MonoBehavior”.

This means that the things that a MonoBehavior can do, and the values that it contains, are accessible to the new class.

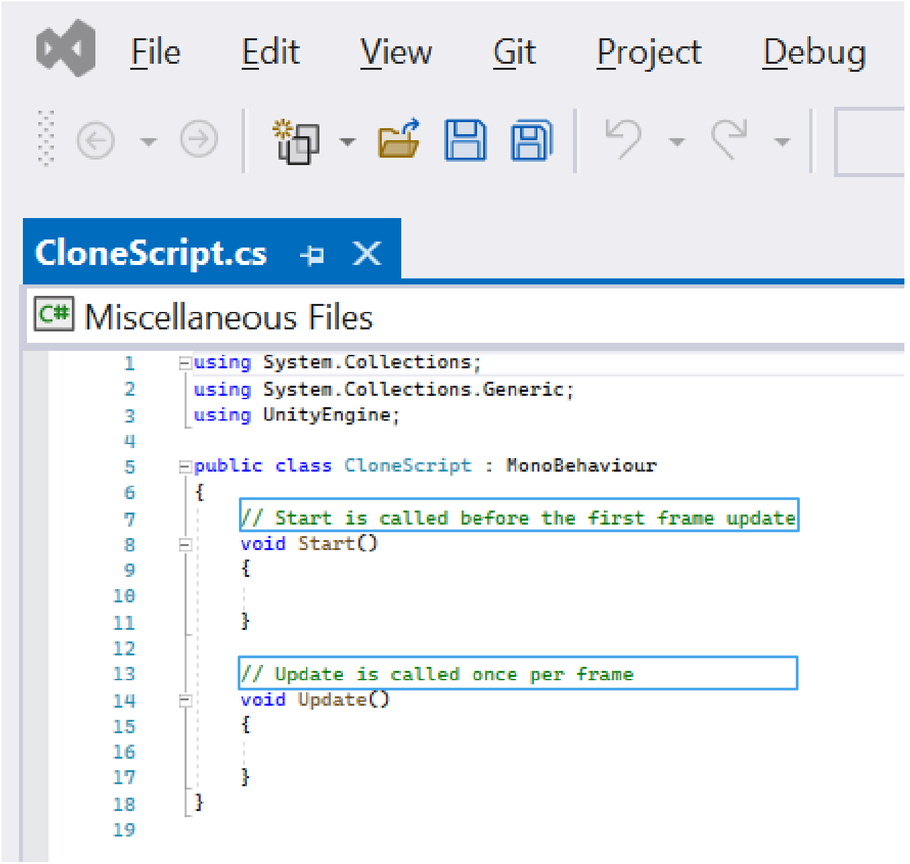
This is called inheritance and is a feature of object-oriented programming (OOP).




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

public class CloneScript : MonoBehaviour
{
    // Start is called before the first frame update
    void Start()
    {

    }

    // Update is called once per frame
    void Update()
    {

    }
}
```

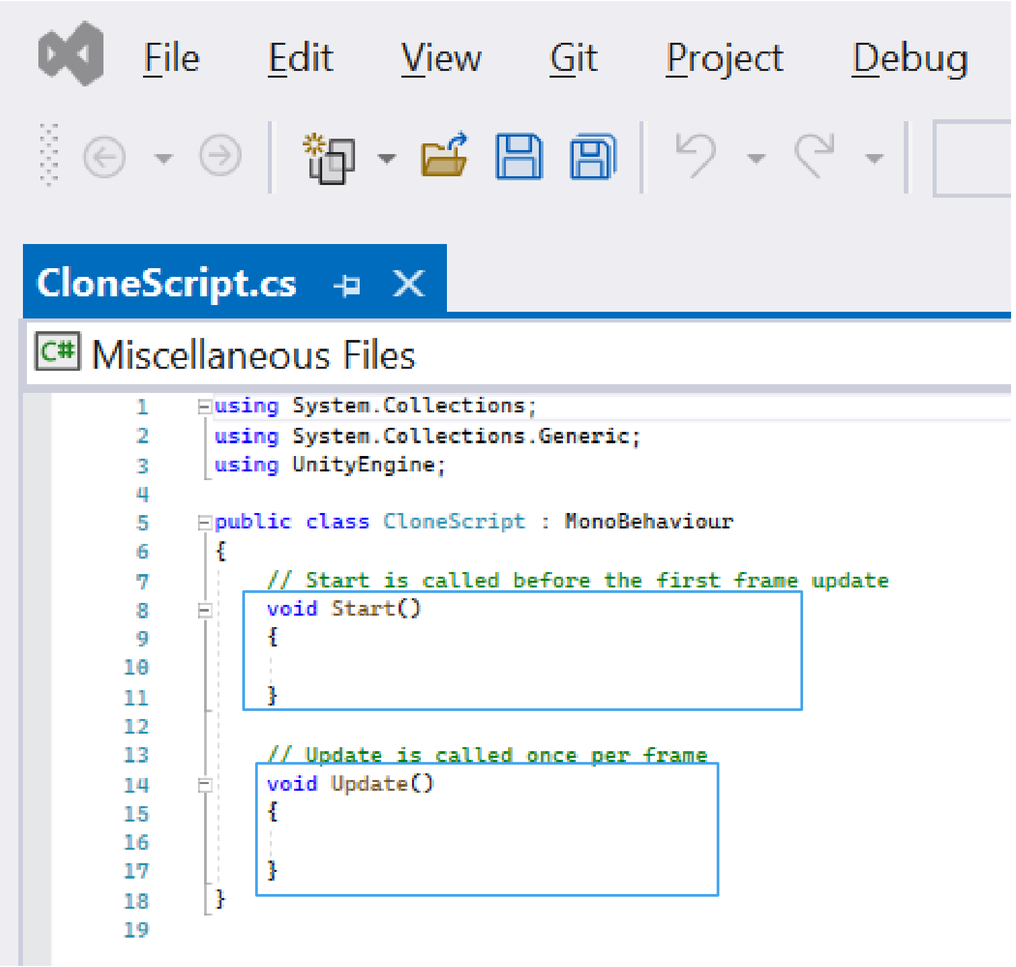
Comments

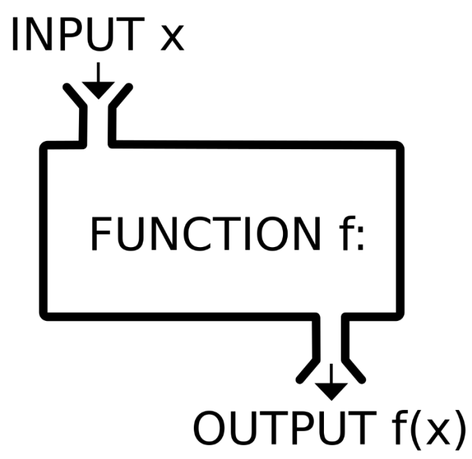
- Comments are for humans and have no impact on programs.
- The compiler ignores everything from the "//" until the end of the line.
- C# supports multi-line block comments using "/*" and "*/"
- The C# compiler can generated help documents if you use XMLDoc format.

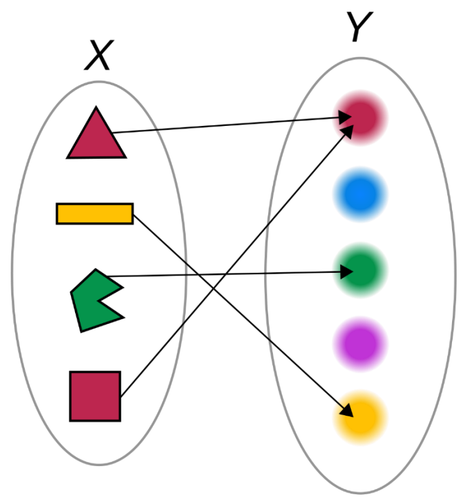


Functions

- The start function is called by Unity once when the game starts.
- The update function is called by Unity every frame of the game.
- They are not called when the game object is deactivated.







Functions in Code versus Mathematics

In Mathematics, a function from a set X to a set Y assigns to each element of X exactly one element of Y.

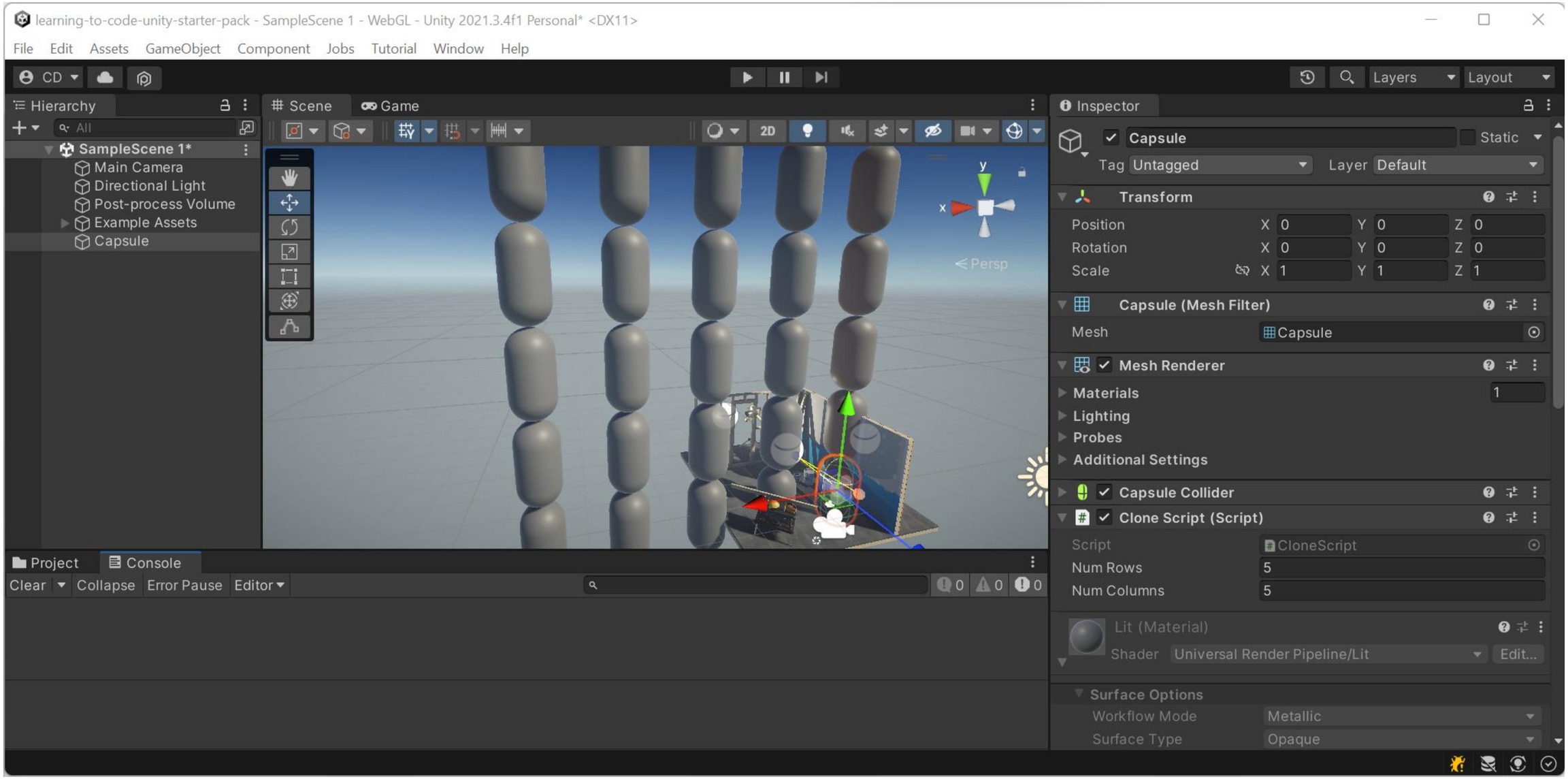
In computer languages what are called "functions" are actually "subroutines".

Unlike mathematical functions:

- They might not return a value ("void").
- They might have zero, one, or more inputs.
- They might have side-effects
- Calling them with the same input might yield different results.

Type in your First Script

```
using UnityEngine;

[ExecuteInEditMode]
public class CloneScript : MonoBehaviour
{
    public int NumRows = 5;
    public int NumColumns = 5;

    public void Update()
    {
        for (var column = 0; column < NumColumns; ++column)
        {
            for (var row = 0; row < NumRows; ++row)
            {
                Graphics.DrawMesh(
                    GetComponent<MeshFilter>().sharedMesh,
                    new Vector3(column * 2, row * 2, 0),
                    Quaternion.identity,
                    GetComponent<MeshRenderer>().sharedMaterial,
                    0);
            }
        }
    }
}
```


learning-to-code-unity-starter-pack - SampleScene 1 - WebGL - Unity 2021.3.4f1 Personal* <DX11>
File Edit Assets GameObject Component Jobs Tutorial Window Help
CD
Layers
Layout
Hierarchy
All
SampleScene 1*
Main Camera
Directional Light
Post-process Volume
Example Assets
Capsule
Scene
Game
2D
Persp
Project
Console
Clear
Collapse
Error Pause
Editor
Inspector
Capsule
Static
Tag Untagged
Layer Default
Transform
Position X 0 Y 0 Z 0
Rotation X 0 Y 0 Z 0
Scale X 1 Y 1 Z 1
Capsule (Mesh Filter)
Mesh Capsule
Mesh Renderer
Materials 1
Lighting
Probes
Additional Settings
Capsule Collider
Clone Script (Script)
Script CloneScript
Num Rows 5
Num Columns 5
Lit (Material)
Shader Universal Render Pipeline/Lit
Edit...
Surface Options
Workflow Mode Metallic
Surface Type Opaque

Attribute

Meta-information for the compiler or host environment.

Available through Reflection.

Tells the script to run in editor even when the game is not playing.

```
1   using UnityEngine;
2
3   [ExecuteInEditMode]
4   public class CloneScript : MonoBehaviour
5   {
6       public int NumRows = 5;
7       public int NumColumns = 5;
8
9       public void Update()
10      {
11          for (var column = 0; column < NumColumns; ++column)
12          {
13              for (var row = 0; row < NumRows; ++row)
14              {
15                  Graphics.DrawMesh(
16                      GetComponent<MeshFilter>().sharedMesh,
17                      new Vector3(column * 2, row * 2, 0),
18                      Quaternion.identity,
19                      GetComponent<MeshRenderer>().sharedMaterial,
20                      0);
21              }
22          }
23      }
24  }
```

Fields

- A field is named data associated with an object. Also known as a "member variable".
- When declared as "public" exposed in the editor as a property.


```
using UnityEngine;

[ExecuteInEditMode]
public class CloneScript : MonoBehaviour
{
    public int NumRows = 5;
    public int NumColumns = 5;

    public void Update()
    {
        for (var column = 0; column < NumColumns; ++column)
        {
            for (var row = 0; row < NumRows; ++row)
            {
                Graphics.DrawMesh(
                    GetComponent<MeshFilter>().sharedMesh,
                    new Vector3(column * 2, row * 2, 0),
                    Quaternion.identity,
                    GetComponent<MeshRenderer>().sharedMaterial,
                    0);
            }
        }
    }
}
```

Variable Declaration

- A variable is a name associated with a value.
- The value associated with the name can change.
- However, the type of data (number, text, vector) associated can't be changed.
- You can use the variable name instead of expression.

```
1   using UnityEngine;
2
3   [ExecuteInEditMode]
4   public class CloneScript : MonoBehaviour
5   {
6       public int NumRows = 5;
7       public int NumColumns = 5;
8
9       public void Update()
10      {
11          for (var column = 0; column < NumColumns; ++column)
12          {
13              for (var row = 0; row < NumRows; ++row)
14              {
15                  Graphics.DrawMesh(
16                      GetComponent<MeshFilter>().sharedMesh,
17                      new Vector3(column * 2, row * 2, 0),
18                      Quaternion.identity,
19                      GetComponent<MeshRenderer>().sharedMaterial,
20                      0);
21              }
22          }
23      }
24  }
```

Variables are your friend

This is equivalent to the other script – which do you prefer?

```
using UnityEngine;

[ExecuteInEditMode]
public class CloneScriptWithVars : MonoBehaviour
{
    public int NumRows = 5;
    public int NumColumns = 5;

    public void Update()
    {
        var mesh = GetComponent<MeshFilter>().sharedMesh;
        var material = GetComponent<MaterialFilter>().sharedMaterial;

        for (var column = 0; column < NumColumns; ++column)
        {
            for (var row = 0; row < NumRows; ++row)
            {
                var position = new Vector3(column * 2, row * 2, 0);
                var rotation = Quaternion.identity;
                var layer = 0;
                Graphics.DrawMesh(mesh, position, rotation, material, layer);
            }
        }
    }
}
```

Expressions

A sequence of symbols (operators, numbers, variables) that represent computations. They are transformed into values (evaluated) when the program is executed.

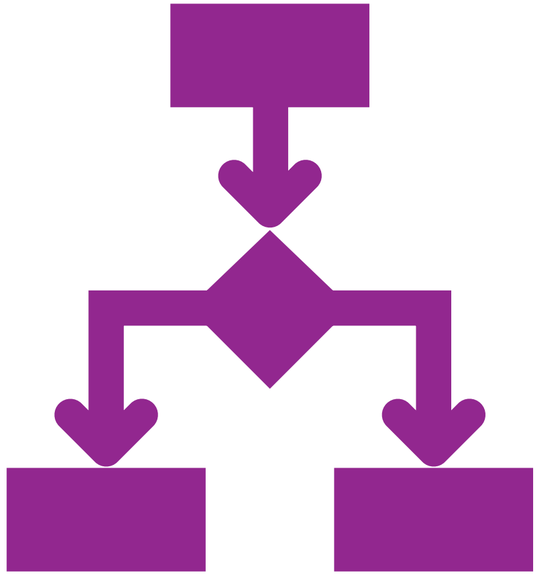
Some examples include:

- Numbers: `42`, `3.15`
- Booleans: `true`, `false`
- Variables: `x`, `MyVector`, `this_is_a_variable`
- An operation with operands: `x + 1`, `y >= 12`, `-z`
- Parenthesized expression: `3 * (y - 2)`
- A function call: `Math.Sqrt(16)`
- A member variable: `this.NumRows`

Expression Use Cases

Common uses of expressions are:

- operation input (operand)
- function input (argument)
- assigned to a variable
- return value of a function
- conditions for loop or branch statements



Practice: Find the Expressions

```
1   using UnityEngine;
2
3   [ExecuteInEditMode]
4   public class CloneScript : MonoBehaviour
5   {
6       public int NumRows = 5;
7       public int NumColumns = 5;
8
9       public void Update()
10      {
11          for (var column = 0; column < NumColumns; ++column)
12          {
13              for (var row = 0; row < NumRows; ++row)
14              {
15                  Graphics.DrawMesh(
16                      GetComponent<MeshFilter>().sharedMesh,
17                      new Vector3(column * 2, row * 2, 0),
18                      Quaternion.identity,
19                      GetComponent<MeshRenderer>().sharedMaterial,
20                      0);
21              }
22          }
23      }
24  }
```

Statements

Statements are sequences of instructions. They might declare something, execute a subroutine, or affect control flow.

```
1   using UnityEngine;
2
3   [ExecuteInEditMode]
4   public class CloneScript : MonoBehaviour
5   {
6       public int NumRows = 5;
7       public int NumColumns = 5;
8
9       public void DrawMesh(float x, float y)
10      {
11          var mesh = GetComponent<MeshFilter>().sharedMesh;
12          var material = GetComponent<MeshRenderer>().sharedMaterial;
13          Graphics.DrawMesh(mesh, new Vector3(x, y), Quaternion.identity, material, 0)
14      }
15
16      public void Update()
17      {
18          for (var column = 0; column < NumColumns; ++column)
19          {
20              for (var row = 0; row < NumRows; ++row)
21              {
22                  DrawMesh(column * 2, row * 2);
23              }
24          }
25      }
26  }
```

For Loop Statement

- Calls the next statement (the loop body) multiple times.
- Executes an initialization statement before starting.
- Only executes while the invariant is true.
- After each loop iteration, calls an iteration statement.

```
1   using UnityEngine;
2
3   [ExecuteInEditMode]
4   public class CloneScript : MonoBehaviour
5   {
6       public int NumRows = 5;
7       public int NumColumns = 5;
8
9       public void Update()
10      {
11          for (var column = 0; column < NumColumns; ++column)
12          {
13              for (var row = 0; row < NumRows; ++row)
14              {
15                  Graphics.DrawMesh(
16                      GetComponent<MeshFilter>().sharedMesh,
17                      new Vector3(column * 2, row * 2, 0),
18                      Quaternion.identity,
19                      GetComponent<MeshRenderer>().sharedMaterial,
20                      0);
21              }
22          }
23      }
24  }
```

Condition

- An expression (operation / function call / constant) with a value of type Boolean (true or false).
- In the context of a for loop is called the invariant.
- Loop is executed while condition is true.

```
1   using UnityEngine;
2
3   [ExecuteInEditMode]
4   public class CloneScript : MonoBehaviour
5   {
6       public int NumRows = 5;
7       public int NumColumns = 5;
8
9       public void Update()
10      {
11          for (var column = 0; column < NumColumns; ++column)
12          {
13              for (var row = 0; row < NumRows; ++row)
14              {
15                  Graphics.DrawMesh(
16                      GetComponent<MeshFilter>().sharedMesh,
17                      new Vector3(column * 2, row * 2, 0),
18                      Quaternion.identity,
19                      GetComponent<MeshRenderer>().sharedMaterial,
20                      0);
21              }
22          }
23      }
24  }
```

Increment Statement

- Adds one to a variable.
- Returns the value of a variable after adding one to the variable.
- The same as “x = x + 1”.
- In the context of a for loop, it is called after each loop iteration.

```
using UnityEngine;

[ExecuteInEditMode]
public class CloneScript : MonoBehaviour
{
    public int NumRows = 5;
    public int NumColumns = 5;

    public void Update()
    {
        for (var column = 0; column < NumColumns; ++column)
        {
            for (var row = 0; row < NumRows; ++row)
            {
                Graphics.DrawMesh(
                    GetComponent<MeshFilter>().sharedMesh,
                    new Vector3(column * 2, row * 2, 0),
                    Quaternion.identity,
                    GetComponent<MeshRenderer>().sharedMaterial,
                    0);
            }
        }
    }
}
```

Block Statements

- Also called a **compound statement**.
- Allows any number of statements (0 or more) to be treated like one statement.

```
1   using UnityEngine;
2
3   [ExecuteInEditMode]
4   public class CloneScript : MonoBehaviour
5   {
6       public int NumRows = 5;
7       public int NumColumns = 5;
8
9       public void Update()
10      {
11          for (var column = 0; column < NumColumns; ++column)
12          {
13              for (var row = 0; row < NumRows; ++row)
14              {
15                  Graphics.DrawMesh(
16                      GetComponent<MeshFilter>().sharedMesh,
17                      new Vector3(column * 2, row * 2, 0),
18                      Quaternion.identity,
19                      GetComponent<MeshRenderer>().sharedMaterial,
20                      0);
21              }
22          }
23      }
24  }
```

Why are Loops Important?

The computer doesn't care … you could write out a statement 25 times.

What could possibly go wrong with this approach?

An important thing to ask yourself frequently when programming.

```
using UnityEngine;

[ExecuteInEditMode]
public class CloneScript : MonoBehaviour
{
    public void Update()
    {
        var mesh = GetComponent<MeshFilter>().sharedMesh;
        var material = GetComponent<MeshRenderer>().sharedMaterial;

        // Row 1
        Graphics.DrawMesh(mesh, new Vector3(0, 0), Quaternion.identity, material, 0);
        Graphics.DrawMesh(mesh, new Vector3(0, 2), Quaternion.identity, material, 0);
        Graphics.DrawMesh(mesh, new Vector3(0, 4), Quaternion.identity, material, 0);
        Graphics.DrawMesh(mesh, new Vector3(0, 6), Quaternion.identity, material, 0);
        Graphics.DrawMesh(mesh, new Vector3(0, 8), Quaternion.identity, material, 0);

        // Row 2
        Graphics.DrawMesh(mesh, new Vector3(2, 0), Quaternion.identity, material, 0);
        Graphics.DrawMesh(mesh, new Vector3(2, 2), Quaternion.identity, material, 0);
        Graphics.DrawMesh(mesh, new Vector3(2, 4), Quaternion.identity, material, 0);
        Graphics.DrawMesh(mesh, new Vector3(2, 6), Quaternion.identity, material, 0);
        Graphics.DrawMesh(mesh, new Vector3(2, 8), Quaternion.identity, material, 0);

        // Row 3
        Graphics.DrawMesh(mesh, new Vector3(4, 0), Quaternion.identity, material, 0);
        Graphics.DrawMesh(mesh, new Vector3(4, 2), Quaternion.identity, material, 0);
        Graphics.DrawMesh(mesh, new Vector3(4, 4), Quaternion.identity, material, 0);
        Graphics.DrawMesh(mesh, new Vector3(4, 6), Quaternion.identity, material, 0);
        Graphics.DrawMesh(mesh, new Vector3(4, 8), Quaternion.identity, material, 0);

        // Row 4
        Graphics.DrawMesh(mesh, new Vector3(6, 0), Quaternion.identity, material, 0);
        Graphics.DrawMesh(mesh, new Vector3(6, 2), Quaternion.identity, material, 0);
        Graphics.DrawMesh(mesh, new Vector3(6, 4), Quaternion.identity, material, 0);
        Graphics.DrawMesh(mesh, new Vector3(6, 6), Quaternion.identity, material, 0);
        Graphics.DrawMesh(mesh, new Vector3(6, 8), Quaternion.identity, material, 0);

        // Row 5
        Graphics.DrawMesh(mesh, new Vector3(8, 0), Quaternion.identity, material, 0);
        Graphics.DrawMesh(mesh, new Vector3(8, 2), Quaternion.identity, material, 0);
        Graphics.DrawMesh(mesh, new Vector3(8, 4), Quaternion.identity, material, 0);
        Graphics.DrawMesh(mesh, new Vector3(8, 6), Quaternion.identity, material, 0);
        Graphics.DrawMesh(mesh, new Vector3(8, 8), Quaternion.identity, material, 0);
    }
}
```

Function Calls

- An expression or statement that executes a function.
- If it returns a value, can be used as an expression.
- Accepts types and/or expressions as inputs.

```
1   using UnityEngine;
2
3   [ExecuteInEditMode]
4   public class CloneScript : MonoBehaviour
5   {
6       public int NumRows = 5;
7       public int NumColumns = 5;
8
9       public void Update()
10      {
11          for (var column = 0; column < NumColumns; ++column)
12          {
13              for (var row = 0; row < NumRows; ++row)
14              {
15                  Graphics.DrawMesh(
16                      GetComponent<MeshFilter>().sharedMesh,
17                      new Vector3(column * 2, row * 2, 0),
18                      Quaternion.identity,
19                      GetComponent<MeshRenderer>().sharedMaterial,
20                      0);
21              }
22          }
23      }
24  }
```

Functions

- Functions are the fundamental building blocks of a computer program.
- Also known as procedures, subroutines, or methods.
- Let's write a function that draws the mesh at a specified location
- Consider other ways we could have written it

```
using UnityEngine;

[ExecuteInEditMode]
public class CloneScript : MonoBehaviour
{
    public int NumRows = 5;
    public int NumColumns = 5;

    public void DrawMesh(float x, float y)
    {
        var mesh = GetComponent<MeshFilter>().sharedMesh;
        var material = GetComponent<MeshRenderer>().sharedMaterial;
        Graphics.DrawMesh(mesh, new Vector3(x, y), Quaternion.identity, material, 0)
    }

    public void Update()
    {
        for (var column = 0; column < NumColumns; ++column)
        {
            for (var row = 0; row < NumRows; ++row)
            {
                DrawMesh(column * 2, row * 2);
            }
        }
    }
}
```