**TEMASEK POLYTECHNIC**

**SCHOOL OF INFORMATICS & IT**

**DIPLOMA IN IMMERSIVE MEDIA & GAME DEVELOPMENT**

**AY2025/2026 APRIL SEMESTER**

**GADV (CGE2C25)**

**Introduction to Unity**

**Basic C# Programming**

To see the additional comments and resources, make sure you select **All Markup** in the **Review/Tracking** pane



**Objectives**

You’ve already covered Python and JavaScript in Year 1. Of course, it isn’t easy to learn programming in two semesters. You now have to learn another language, C#, which might seem intimidating.

So, we’ll take the opportunity to review the more fundamental programming concepts again, in the context of C#.

We’ve already covered the basics of scripting in Unity, so now we’ll explore C# in the context of game programming (which will hopefully make learning programming more interesting and relevant to you—it really isn’t that difficult!).

But as you become more confident, you’ll realise that the language you use isn’t so important. Each language has it strengths and weaknesses, and is generally used in a specific domain, e.g. Python for data science, JavaScript for web development, etc.

The more languages you use, the more you’ll see what they have in common. We’ll focus on these common aspects here, with the emphasis on C#.

**What is C#?**

C# is a strongly typed, object-oriented programming language developed by Microsoft to compete with Java.

It is the primary programming language used in Unity.

C# is similar to Java, C++, and Python, but it has its own rules and structure. Unity uses C# with MonoBehaviour, a special base class that allows scripts to work inside the Unity engine.

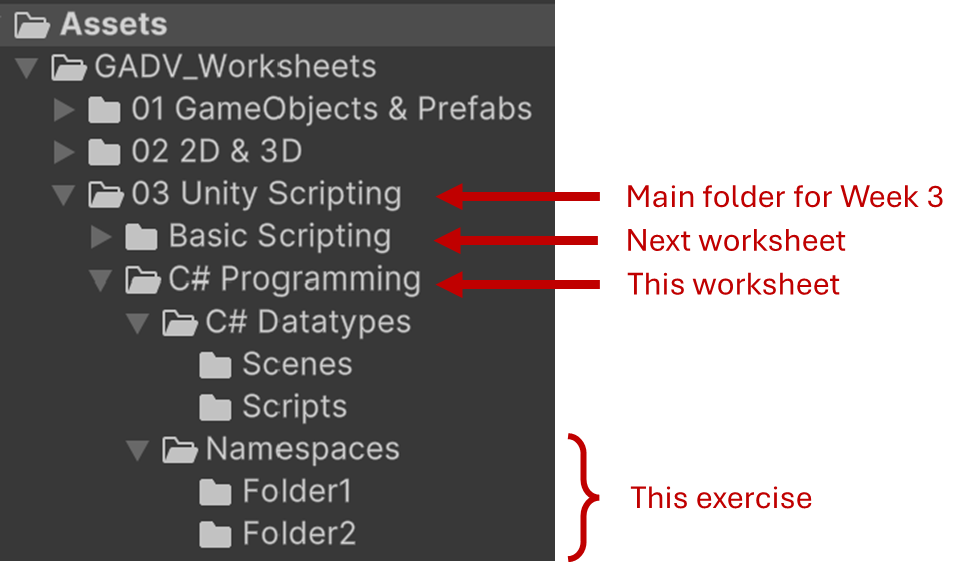
1. **Setup**
   1. **Project Setup**

We’ll use the **GADV\_Worksheets** GitHub repository you created in an earlier worksheet to keep track of your work. This means that all your worksheets will be in a single Unity project, with a separate folder for each worksheet.

But some of the scripts (i.e. classes) in these folders may have the same name. This will cause a complilation error.

We’ll see later that we can use **namespaces** to organise code and avoid name conflicts by grouping related classes together.

1. Create the folder structure shown bleow:



* 1. **External Tools: IDE Setup**

Make sure that Visual Studio (or VS Code) is set as your external IDE (Integrated Development Environment) for programming.

Select the menu option **Edit/Preferences**, then select **External Tools**. Set this to your version of Visual Studio or VS Code.

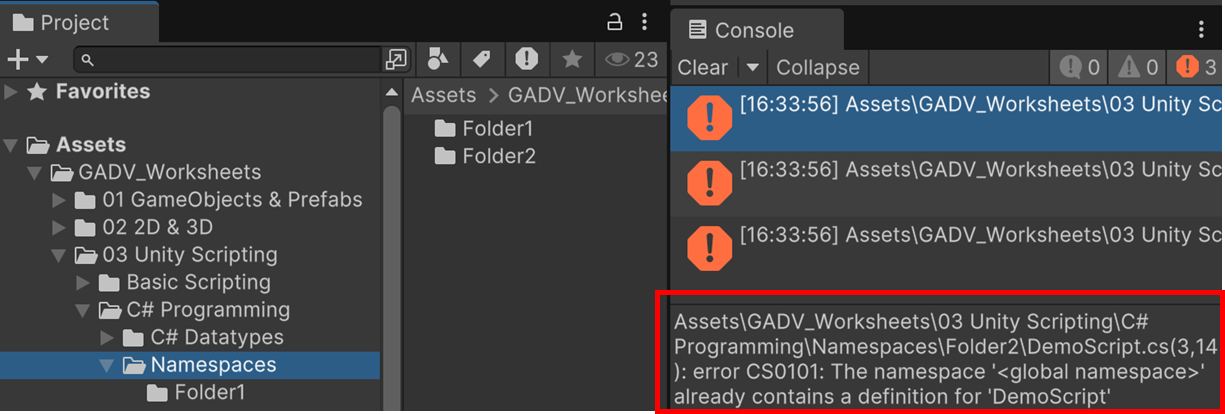
1. **Namespaces**

Before doing any programming, let’s look at the concept of a **namespace**.



1. Create a script called **DemoScript,cs** in the **Folder1**.
2. Create a script with the *same* name in **Folder2**.

What happens? **ERROR!**



This means that the Unity C# compiler sees *two* scripts in the project with the name **DemoScript**.

But it can’t create two classes with the same name!

To fix this, we can put each class into a different **namespace**.

1. Change **Folder1/DemoScript.cs** to this:

using UnityEngine;

**namespace ProjectNamespace1**

**{**

public class DemoScript : MonoBehaviour

{

// Other code

}

**}**

You’ve just wrapped the class inside a *namespace* called **ProjectNamespace1**.

Does the error go away?

1. In Chat GPT, run the prompt in the sidebar comment.
2. Now use the same namespace in **Folder2/DemoScript.cs**. Explain why the error returns.
3. Change the **Folder2/DemoScript.cs** namespace to **ProjectNamespace2**. What happens?

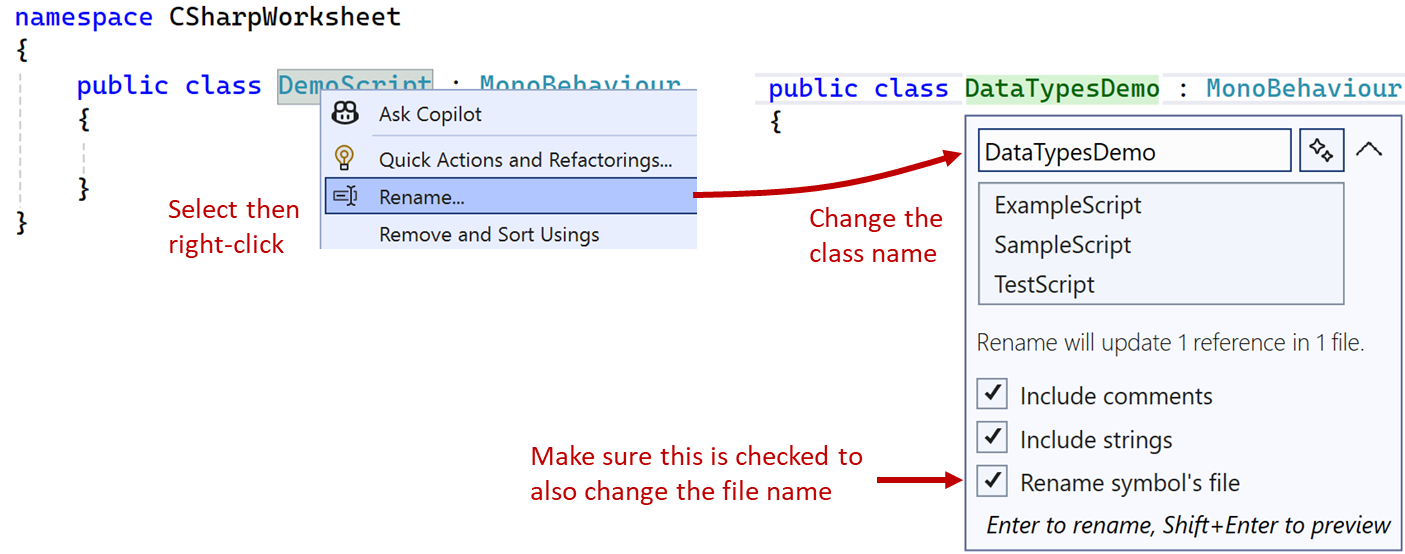
Namespaces are great for packaging code, but should be used wisely. Don’t use them to “correct” badly designed code. You should use namespaces to package related classes together, e.g. UI classes, utility classes, etc.

**Renaming a class**

But you might just decide to *rename* a class with a conflicting name. The problem is that you want the name of the script to also change.

Let’s rename DemoScript to something more meaningful.

1. Select the **class name** in Visual Studio.
2. Right click and select **Rename**.
3. Change the name in the box provided.
4. Make sure that **Rename Symbol’s File** is checked.
5. Press **Enter**.
6. Check in Unity that the file name has also been changed.



**Checkpoint!**

Answer the questions below, then before continuing raise your hand to check the answers with your tutor.

1. In Unity, why might using multiple namespaces across scripts introduce issues with GetComponent<T>() or script references in the Inspector?

Answer here : It might refer to the wrong script as it may not be able to differentiate between them.

1. How can namespaces affect code readability and maintenance in a team project with multiple developers?

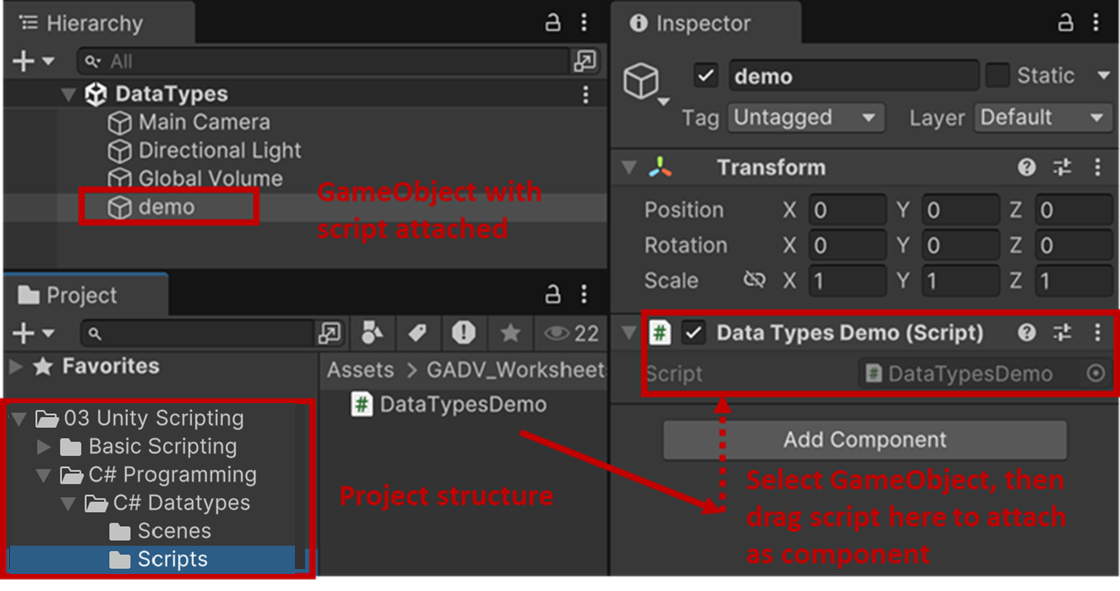
Answer here: Different namespaces can make it easier to identify what script does what and lets developers know easier which script does what.

1. Suppose two classes have the same name but exist in different namespaces. How would you explicitly refer to one of them in a script without using using statements?

Answer here …:idk



1. **Data Types in C#**
2. Drag the **DataTypesDemo.cs** script into **DataTypes/Scripts**.
3. Then set up the scene:
4. Create a new scene called **DataTypes** and save this in the **Scenes** folder.
5. In the Hierarchy, create an empty game object called **demo**.
6. Attach the **DataTypesDemo.cs** script as a component.



1. Add a debug statement and run the project to make sure it’s working:

private void Start()

{

Debug.Log("Data Types Demo");

}

1. **What is Data and Data Type?**

In programming, *data* refers to any information that a computer can store, process, and manipulate. Each piece of data has a data **type**. You should know what a **data type** is by now, but let’s make sure.

Look at each of these URLS:

* [Computer Hope: Data Types](https://www.computerhope.com/jargon/d/datatype.htm)
* [Study Smarter: Data Types in Programming](https://www.studysmarter.co.uk/explanations/computer-science/computer-programming/data-types-in-programming/)
* [Code 360: Data Types in C#](https://www.naukri.com/code360/library/data-types-in-c-sharp) (slightly more in-depth)

Run the prompt in this comment in Chat GPT.

Try to answer the questions Chat GPT creates, and then ask for the correct answers.

Now go to this [padlet](https://padlet.com/dougfinnmisc/gadv-c-data-types-questions-ikgzqncyu7g6j8ks) and copy/paste the two questions Chat GPT created. We’ll discuss these in class later. We’ll spend about 10 minutes in class trying to answer some of these questions.

1. **Loose vs Strong Typing**
   1. **Core concept**



Programming languages handle data types differently, and one key distinction is **loose** typing vs. **strong** typing.

Loosely typed languages, like Python and JavaScript, let variables hold different types of values at different times. A variable can start as a number and later be reassigned as a string without error. For example, in Python:

x = 10 # x is an int

x = "ten" # Now x is a string (no error)

print(x) # Output: ten

This flexibility makes loose typing convenient but can cause unexpected behavior, e.g. multiplying a string and an integer in Python ("5" \* 3) results in "555" instead of an error.

In contrast, strongly typed languages like C# enforce strict type rules. Every variable has a fixed type that cannot change. For example, this C# code would cause an error:

int x = 10;

x = "ten"; // ERROR: Cannot assign a string to an int!

**5.2 Type conversion**

C# requires explicit type conversion. Adding a string and an integer ("42" + 8) works in Python ("428") but causes an error in C#.

To fix this, convert the string to an integer:

string x = "42";

int y = 8;

int sum = int.Parse(x) + y;

Debug.Log(sum); // Output: 50

Loose typing enables faster prototyping with less code, while strong typing prevents bugs by keeping variables consistent.

**Exercise 1**

The exercises below use sprites, so make sure you’ve completed your earlier worksheet about 2D in Unity.

In this exercise, you’ll compare trying to change a sprite’s size using a **string** and a **float** value.

1. Create a new sprite in your scene at position (0, 0, 0). Either use the Asteroid image, or create a new one using any other image you like.
2. Attach this script to your sprite GameObject and run the project:

using UnityEngine;

namespace GADVDataTypes

{

public class SpriteScaler : MonoBehaviour

{

private Transform spriteTransform;

private string scale = "2.0";

void Start()

{

spriteTransform = GetComponent<Transform>();

spriteTransform.localScale = new Vector3(scale, scale, 1f);

}

}

}

What went wrong?

Explain the reason for the error.

Keep the **scale** variable as a string, and fix the error.

1. Now set the value of **scale** to “2.0a”.

Does the type conversion still work?

Look at the explanation in the comment, then answer this question.

Why doesn't the compiler catch the error when trying to convert the string "2.0a" to a float using float.Parse(), and what does this reveal about how C# handles type safety at compile time versus runtime?

1. You run your Unity project and the scene crashes with no visible result. How can you check what went wrong, and how do you find the exact line in your script that caused the error?

Answer here … : use Debug.Log to identify where the code reaches

1. Explain how this image might be thought to represent the concept of strict typing in C#.

****

Answer here …: It can make things easier to run through with less problems but can be annoying and difficult to identify a problem if it occurs



**Exercise 2**

Here, we’ll look at trying to animate a sprite using **int** and **float** values.

1. Create a new script called SpriteMover and attach it to your sprite GameObject:
2. Modify SpriteMover to:

using UnityEngine;

namespace GADVDataTypes

{

public class SpriteMover : MonoBehaviour

{

private int moveSpeed = 3;

private float timeElapsed = 2.5f;

void Update()

{

// Implicit conversion (int → float)

float totalMovement = moveSpeed + timeElapsed;

transform.position = new Vector3(totalMovement, 0, 0);

int roundedSpeed = totalMovement;

// Explicit conversion (truncates decimal)

roundedSpeed = (int)totalMovement;

}

}

}

1. Uncomment the **int roundedSpeed = totalMovement;** line and explain why it causes an error.



Answer here …

It is as totalMovement cannot be converted into char data

1. Why does the float value 2.5 have an “**f**” after it: **2.5f**?



Answer here …: is the type for float?

You should now have a basic understanding of how data typing works in C# vs. Python.

We just have one more topic to cover before moving on to more advanced C# language features.

1. **Using var in C#**

In C#, the **var** keyword allows the compiler to automatically [infer](https://www.dictionary.com/browse/infer) the data type of a variable based on its assigned value.

This makes the code cleaner and reduces redundancy while still maintaining strong typing.

* 1. **Why Use var?**

Instead of explicitly specifying types like this:

int score = 100;

float speed = 2.5f;

string playerName = "Alex";

You can write:

var score = 100; // Inferred as int

var speed = 2.5f; // Inferred as float

var playerName = "Alex"; // Inferred as string

But even though var makes declarations shorter, C# still enforces strong typing.

Once a type is assigned, it cannot change:

var lives = 3;

lives = "Game Over"; // ERROR: Cannot assign a string to an int!

IMPORTANT!

**var** is just a shortcut to save typing (on your keyboard), but C# still keeps everything strongly data typed. It doesn’t make C# a loosely-typed language like Python. A C# variable can’t *change* its type, as the **lives** variable above illustrates. The compiler sets **lives** to be an **int**, but doesn’t allow it to change its type to store a reference to a **string**!

Use **var** if the type is obvious and makes the code easier to read, for example:

var position = new Vector3(0, 1, 0); // clearly a Vector3

But avoid **var** when the type isn’t clear, especially when calling methods that return unknown values, for example:

var data = GetInfo();

Writing **string data = GetInfo();** makes the code clearer.

A good rule is: if you or a teammate might be confused about a variable’s type later, don’t use **var**.

If you’re still unsure, ask your tutor! 😊

**Exercise 3**

Let’s see **var** in action!

Add this script to your sprite GameObject:

using UnityEngine;

namespace GADVDataTypes

{

public class VarExample : MonoBehaviour

{

void Start()

{

// Inferred as SpriteRenderer

var sprite = GetComponent<SpriteRenderer>();

// Inferred as Color

var spriteColor = sprite.color;

// Inferred as Vector2

var spriteSize = sprite.bounds.size;

Debug.Log("Sprite Color: " + spriteColor);

Debug.Log("Sprite Size: " + spriteSize);

}

}

}

When you run the project, notice that Unity outputs the correct textual representation of the variable’s data type.

For example, spriteColor is output as: **RGBA(1.000, 1.000, 1.000, 1.000)**

The **RGBA** at the front shows that Unity knows this is a **Color** type of value.

**Exercise 4**

Let’s also make sure that C# is still type-safe.

1. Add this script to your sprite GameObject:

using UnityEngine;

public class VarMovement : MonoBehaviour

{

private void Update()

{

var speed = 5.0f; // Inferred as float

var direction = new Vector3(1, 0, 0); // Inferred as Vector3

transform.position += direction \* speed \* Time.deltaTime;

// speed = "fast";

}

}

What happens when you uncomment the last line?

1. **Checkpoint!**

Let your tutor know that you’ve completed the questions below.

If you aren’t sure how to answer, try to explain why here, e.g. you still don’t understand the difference between an int and a float. Then put up your hand and ask your tutor for guidance.

1. What does **var** do in C#?

Answer here …: it infers the data type of a variable

1. What’s the difference between **implicit** and **explicit** casting?

Answer here …: Implicit casting can convert data from one type to another where legal while Explicit casting states how the data should take place

1. Why does int x = "hello"; cause an error?

Answer here …: “hello” is not an int type data

1. How does C# differ from Python when assigning types?

Answer here …: C# creates a new variable to store a data of a different type

e.g. int x = 10

float y = float.Parse(x)

10 is now a float

Data types are more strict in C# than in Python where there are loose

1. **Summary**

You've now explored the key differences between loose typing (Python) and strong typing (C#).

In Python, variables can change types freely, making coding faster but sometimes leading to unexpected behavior—like "5" \* 3" producing "555". In C#, variables have fixed types, preventing errors but requiring explicit conversions.

* **Loose-typing**: A language where variables can hold different types of values at different times (e.g., x = 10; x = "ten"; no error in Python).
* **Strong-typing**: A language where variables have a fixed type that cannot change after being assigned (e.g., int x = 10; x = "ten"; an error in C#).

You’ve also looked at how a variable of one type is converted to a different type.

* **Implicit** *conversion*: The compiler automatically converts from one type to the other, where this is legal.
* **Explicit** *casting*: Your code states how the data type conversion should take place.

Of course, there's more to it than this. If you're curious, check out the sidebar comments for more information. These advanced topics won’t be covered in GADV, but they might be useful for your project work.

Rather than teaching fundamental concepts in isolation, it’s far more effective to explore them while working on real code. Future worksheets will introduce advanced data types and data type concepts as you tackle more complex programming challenges.

But by now, you should have a good grasp of how C# and Python handle data differently.

If you have any questions, ask your tutor.

Well done for getting so far.

Here’s a cute picture to reward you!😊

