**Unity Certification Preparation:**

**C# Programming**

**Orlando Unity3d Development Meetup**

**Date: 23-Sep-18**

**All Copyright PompaSetting Studios © 2018**

**Contents**

[**2.0** **So, What is Scripting ?** 3](#_Toc523933439)

[**2.1** **Double Check your IDE Settings** 3](#_Toc523933444)

[**2.2** **How to Create a Script ?** 3](#_Toc523933445)

[**2.3** **The Default Code : Namespaces, Inheritance** 3](#_Toc523933446)

[**2.4** **The Class Declaration Statement, introduction to the Inheritance principle** 4](#_Toc523933447)

# **So, What is Scripting ?**

# Reference to Unity, you will often hear people talking about scripting, what they really mean is computer programming => In other words, giving “instructions” to the game objects in the scene to do something. For example, if we create an object in the scene. Naturally we want to be able to interact and control that object. E.g. Let's say we want to move the object from point A in the scene to a point B. To do this in unity we write a “script” (**the source code**) then attach it to the object in the scene window that we want to move.

# Now, the correct technical definition of scripting means => **the program does not need to be “complied” before running it. Rather your instructions/the program are interpreted at runtime**. In effect you do not have the “hassle” of creating an executable .exe file before experiencing it. The source code is translated to the CPU at run-time (i.e. the source code is processed at the same time your program is being run and experiencing the application in real time) from any format to CPU machine language instructions.

# Whereas **compiling** means “all your source code” is executed directly by the computer's CPU. i.e. the **source code you have typed in your IDE is converted “one-time” to the CPU's native assembly language before it is actually run (before runtime)**.

# Please be aware effective from 01 Apr 2017 Unity **has deprecated** JavaScript (aka UnityScript) for programming, i.e. they now only officially support C# for programming.

# **Double Check your IDE Settings**

Now before we create the 1st script. First, double check your:

**1) IDE settin**gs are set up correctly. You should all have **visual studio 2015 or 2017** setup as your default IDE. Navigate to the ribbon/toolbar Top left.

|  |
| --- |
| **Double Check IDE Settings:-** |
| (Navigate to Toolbar Top Left tab to right)(Select)[Edit][Preferences][External Tools][External Script Editor](Check/Select) **< Visual Studio 2017 >** |
|  |

**2)Asset folders** **(\_MATERIALS, \_PREFABS, \_SCENES, \_SCRIPTS**) have been created.

|  |
| --- |
| **Create 3 New Asset Folders: 1 x\_MATERIALS, 1 x\_SCRIPTS, 1 x\_SCENES, 1 x PREFABS** |
| (Navigate to Project tab)[Assets](Right Click)(Select)[Create][Folder][F2](Rename) **< \_MATERIALS >** |
| (Navigate to Project tab)[Assets](Right Click)(Select)[Create][Folder][F2](Rename) **< \_SCRIPTS >** |
| (Navigate to Project tab)[Assets](Right Click)(Select)[Create][Folder][F2](Rename) **< \_SCENES >** |
| [Navigate to Project tab][Assets](Right Click)(Select)[Create][Folder][F2](Rename) **< \_PREFABS >** |
|  |

# **How to Create a Script ?**

Ok In unity there are **2 key ways** to create a script 1)From inside the **Project panel** via the asset folder, or 2)Adding a script as a **component** to the object inside of the inspector.

|  |
| --- |
| **CREATE A NEW 1 x SCRIPT** |
| (Navigate to)[Project tab] > (Select)[Assets][Scripts] > (Right Click)[Create][C# Script] > (F2 Rename or Name) < "NewScriptX" > |
| Drag N Drop NewScriptX onto one of Cube GameObject in Inspector |

If you double, click on the "NewScriptX" > It should open into VISUAL STUDIO > And you should see some default code.

# **The Default Code : Namespaces, Inheritance**

It is very important to understand what the default code is and does, before charging headlong into your game, because that will help you narrow down what code you need. Also, whenever you create a new script in unity. It automatically creates a new class with the same name. At the beginning of your code. In your script, the very first lines you see are:

|  |
| --- |
| using System.Collections; |
| using System.Collections.Generic; |
| **using UnityEngine;** |

In unity C# these are known as **namespaces** or **code libraries**. Anytime you see a statement with the command **"using"** --> it means you can access the **commands, classes, methods & objects** from those libraries for use in your code. **A namespace is used to avoid naming conflicts**. When you start to add libraries from other programmers it is highly likely you are going to use classes, methods & objects with the same names. Therefore, we create a **namespace** as a unique group name identifier for the collection of classes, methods & objects that it contains.

# **The Class Declaration Statement, introduction to the Inheritance principle**

In **OOP** there is a principle called **inheritance**. Which means that the class NewScript "inherits" all the **commands, classes, methods and data** from the Monobehaviour class. The **class declaration statement** is as follows:

|  |
| --- |
| **public class NewScript1 : MonoBehaviour** |
| **{** |
| **//Ditto** |
| **}** |

The **MonoBehaviour class** = **is aka the Parent (Base) class used to derive your C# script** -> It contains all the **commands, methods, classes** and components from the unity game engine to use in your C# script. MonoBehaviour is an **API.** For example, using JavaScript in Unity every script **automatically or implicitly** derives the code from the MonoBehaviour class.

However, when using C#, you must **explicitly** derive your code from the MonoBehaviour class. MonoBehaviour is also part of the UnityEngine namespace (i.e. using UnityEngine). So, if we delete the UnityEngine namespace => you will get an error. When you create a new script; by default, Unity creates 2 methods or functions (between the two curly braces inside the body of the class) => **void Start()** and **void Update()** aka **The Default Execution Order** **functions.** If you do not inherit from MonoBehaviour you “WILL NOT” be able to use these two functions as they are derived from the Monobehaviour class. **void start() function is called only once for the duration of your script, at the start of your application. Whereas Void update() is called repeatedly at each frame.** OK so how do we use the code. Let’s start by typing Debug.Log….

**Debug.Log** = Echo outputs or prints the string contents as a message to the Unity Console Window.

|  |
| --- |
| using System.Collections; |
| using System.Collections.Generic; |
| using UnityEngine; |
|  |
| **public class NewScript1 : MonoBehaviour** |
| { |
| // Use this for initialization |
| **void Start ()** |
| **{** |
| **Debug.Log("Start of program");** |
| **}** |
| // Update is called once per frame |
| **void Update ()** |
| **{** |
| **Debug.Log("Frame updated !");** |
| **}** |
| }//End of HelloScript |

/End