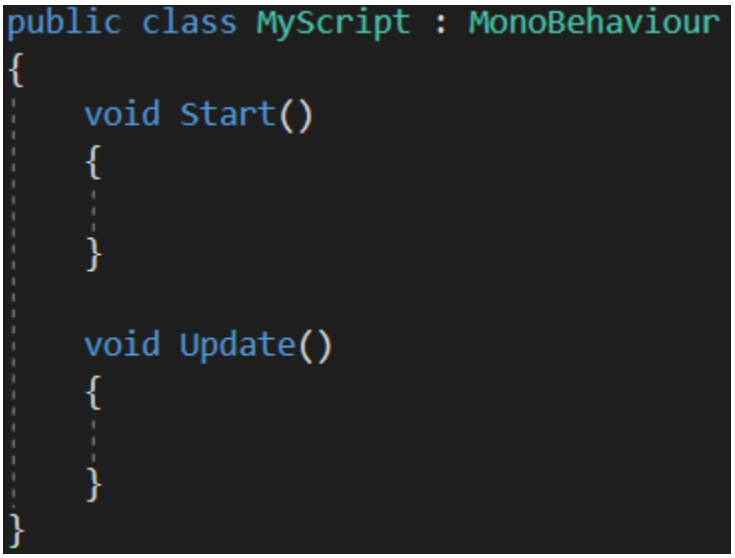
The Unity Documentation and The Accessor

Lesson Goals

- This lesson aims to look at how we can determine the functionality of a class.
- We will be looking at Unity's Documentation to find out the functionality of the classes we have been using throughout our project so far. We will also be taking a look at a few useful classes for game development.
- We will look both into Unity's Manual as well as the Scripting API.
- We will then learn how to use the accessor to access the functionality of a class.

Set Up

1. Before we start the lesson, like last time, let's clear out our MyScript script. We a going to remove any variables from the MyScript class as well as remove any code from the Start() and Update() function. We will start with an empty class.



Asking Questions

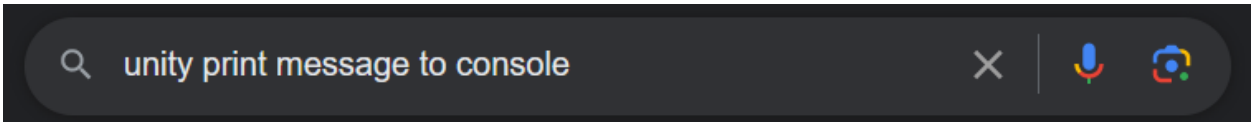
1. So up until this point in our lessons, while programming we have been calling functions from classes without actually discussing where these functions came from or how I knew they existed in the first place.

   For example we have been using the Log() function from the Debug class to output messages to the console.

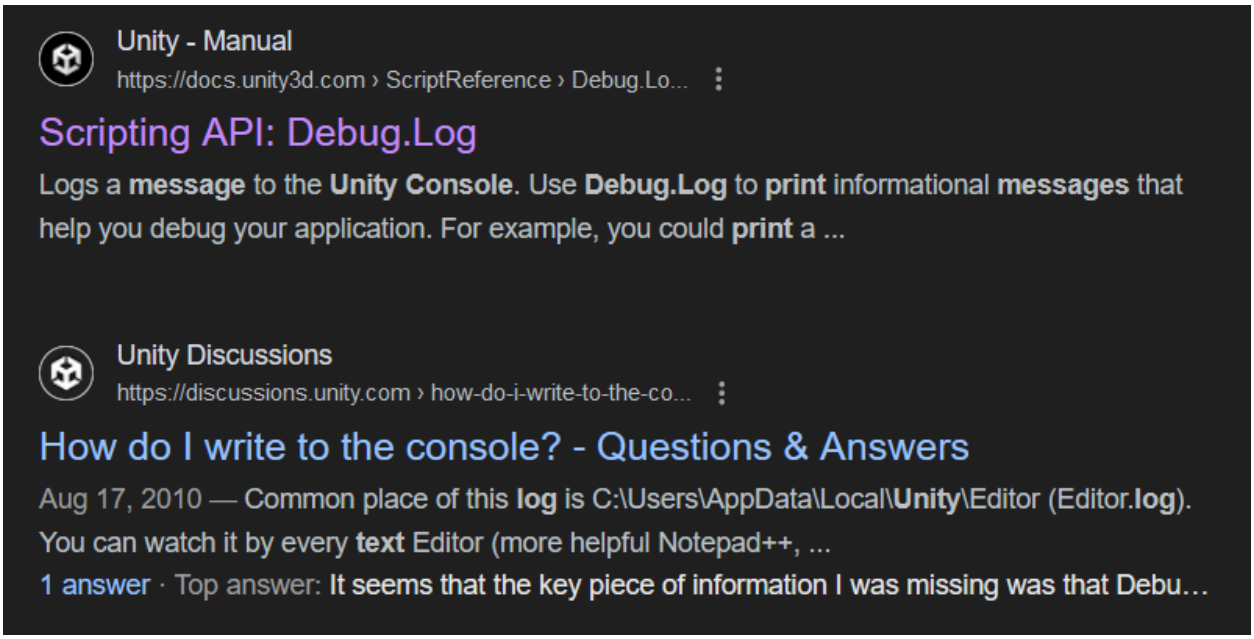
```
Debug.Log("Hello World");
```

But again the question is, how did I find this class and know what function to use to output the message?

2. Well, the answer is to google it. For this specific instance, my question was to output a message to the console in Unity. So, in google I typed:



The first two results are the best places to look for informtaion when trying to figure out a specific issue with Unity. These are the **Unity Documentation** and the **Unity Discussions** forum.



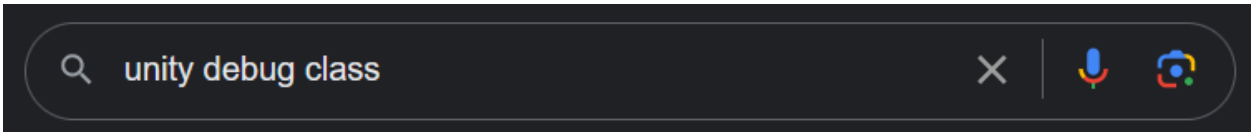
The **Unity Documentation** is broken up into two sections, the **Unity Manual** and the **Scripting API**. The Unity Manual gives you a description of what that class does and how you may use it for your project. The Scripting API lists out all of the variables and functions that the class has. It gives a brief description as to what each variable and function does.

The **Unity Discussion** forum will typically show a user posing a question. Other users can respond sharing their own answers to the user's question. I highly encourage you to post questions to the forum if you are having trouble.

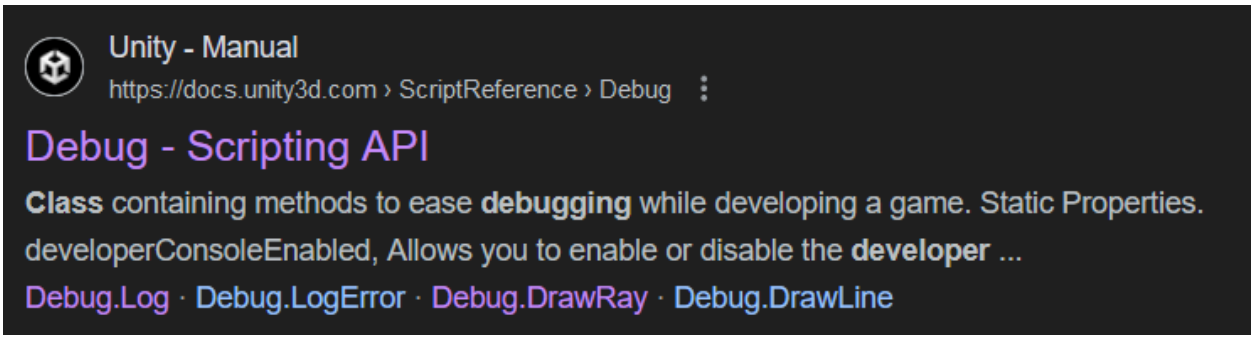
Unity Documentation

1. So, I know googling a question is a pretty obvious answer to how we should solve our unity problem. But with this lesson I want to highlight specifically how we can use the Unity Documentation to help deepen our knowledge of a class and its functions.

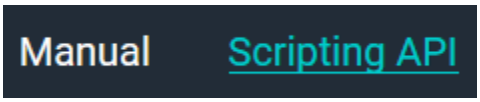
2. For starters, let's google search the Debug class for Unity.



We can click on the first link to be taken to the unity documentation.



3. Upon doing so we are taken to the **Scripting API** page for the Debug Class. We can tell that we are in the Scripting API because it is highlighted blue at the top of the page.



4. We will return to this page in a moment but for right now let's check out the manual page for the debug class. We can do this by clicking the **Switch To Manual** button near the top of the page.

5. On the manual page we should be able to see a description of what the debug class does.

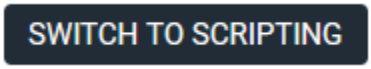
The Debug class allows you to visualise information in the Editor that may help you understand or investigate what is going on in your project while it is running. For example, you can use it to print messages into the **Console window**, draw visualization lines in the **Scene** view and Game view, and pause Play Mode in the Editor from script.

This page provides an overview of the Debug class and its common uses when scripting with it. For an exhaustive reference of every member of the Debug class, see the Debug script reference.

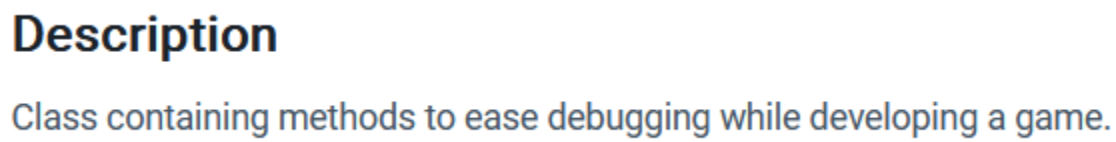
Just looking over the page in general we can see that this class is mainly used for logging errors and message but it also allows us to draw debug lines to our scene which my come in handy if we ever need to know which way an object in our game is facing.

If we ever want to know what a class does at a glance, the Unity manual page for the class is the way to go.

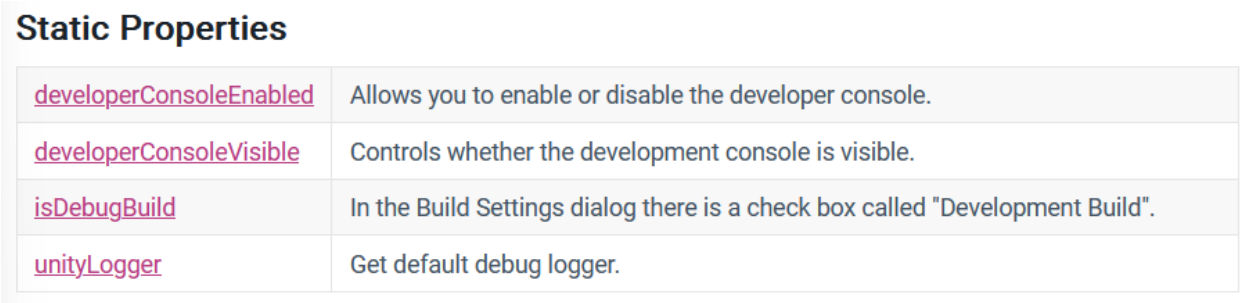
6. Let's switch back to the scripting API page. We can do this by hitting the **Switch To Scripting** button near the top of the page.



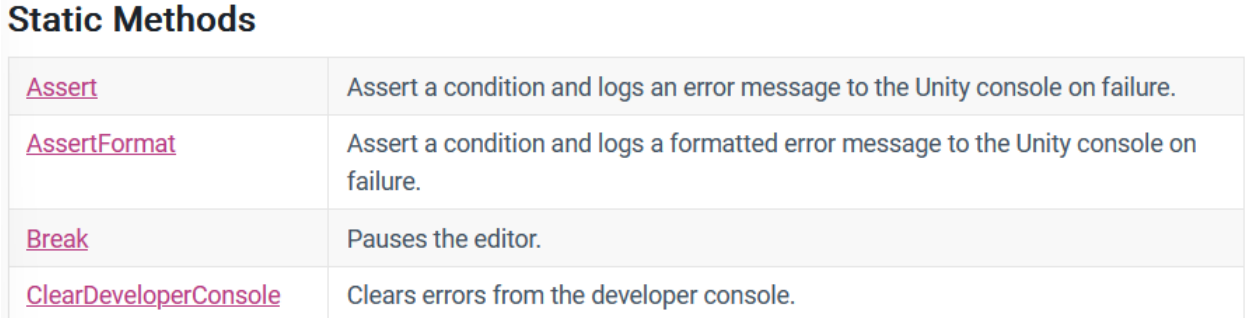
7. On the Scripting API page we will be given a brief description of the what the purpose of the class is.



We will also see a list of variables that the class has along with a description for each variable. Variables that are inside a class are referred to as **properties**.



Lastly, we will then see a list of functions the class has along with a description of what the function will do. Functions that are inside a class are referred to as **methods**.



8. If we look in the list of functions we can find our Log() function from before.

Log	Logs a message to the Unity Console.

9. Clicking on the purple Log text will take us to a page the further describes the function. At the top of the page, we can see the declaration of each function. These are the different ways that we are able to call the function.

Declaration

public static void **Log**(object **message**);

Declaration

public static void **Log**(object **message**, Object **context**);

The first declaration takes in one parameter. This is the declaration that we have been using for most of our project so far. The second declaration takes in two parameters. A **Parameter** is data that we can pass into a function for it to run. These might also be referred to as **Arguments**.

Underneath, we can look at the function's parameters as well as a description of what that parameter does.

Parameters

message	String or object to be converted to string representation for display.
context	Object to which the message applies.

Lastly, we get a description of what the function does.

Description

Logs a message to the Unity Console.

Use Debug.Log to print informational messages that help you debug your application. For example, you could print a message containing a GameObject.name and information about the object's current state.

You can format messages with string concatenation:

```
Debug.Log("Text: " + myText.text);
```

We are also supplied with code examples so that we can see how to implement the function in code.

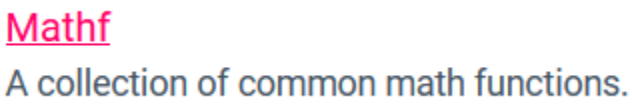
```
using UnityEngine;
using System.Collections;

public class MyGameClass : MonoBehaviour
{
    // A Light used in the Scene and needed by MyGameMethod().
    public Light light;

    void MyGameMethod()
    {
        // Message with a GameObject name.
        Debug.Log("Hello: " + gameObject.name);

        // Message with light type. This is an Object example.
        Debug.Log(light.type);

        // Message using rich text.
        Debug.Log("<color=red>Error: </color>AssetBundle not found");
    }
}
```

Mathf Class and the Accessor

1. So now that we understand the Unity documentation, let's look at a really useful class that comes up all the time in game development. The **Mathf** class.

2. Since we are in the documentation, we can search the Mathf class using the search bar at the top right-hand side of the page.

Once searched let's then click on the first result at the top of the page.



3. Like before we can switch over to the Manual to get a description of what this class is.

Mathf

SWITCH TO SCRIPTING

Unity's Mathf class provides a collection of common math functions, including trigonometric, logarithmic, and other functions commonly required in games and app development.

This page provides an overview of the Mathf class and its common uses when scripting with it. For an exhaustive reference of every member of the Mathf class, see the Mathf script reference.

But for this example, let's stay on the Scripting API page and take a look at a few functions that we will try out in our code.

We will be looking at the functions:

Abs	Returns the absolute value of f.
Clamp	Clamps the given value between the given minimum float and maximum float values. Returns the given value if it is within the minimum and maximum range.
Floor	Returns the largest integer smaller than or equal to f.
Ceil	Returns the smallest integer greater than or equal to f.
Round	Returns f rounded to the nearest integer.
Max	Returns the largest of two or more values. When comparing negative values, values closer to zero are considered larger.
Min	Returns the smallest of two or more values.

4. So now the question is how do we even use these functions in our code? Well to answer this, let's go ahead and open up our MyScript script.

We are going to be outputting a message to the console that shows of what each of these functions will do.

5. In the start function let's write the code:

```
private int number = -12;

void Start()
{
    number = Mathf.Abs(number);
    Debug.Log(number);
}
```

So let's break this down. We first declare a new integer variable called number that has a value of negative twelve.

In the start function we set number equal to the value of `Mathf.Abs(number);` and then output number as a message. But let's break down the `Mathf.Abs(number);` chunk of code.

First, we type **Mathf** to get a reference to the Mathf unity class. We are able to access this class in our script because of the **UnityEngine** library that is at the top of our script.

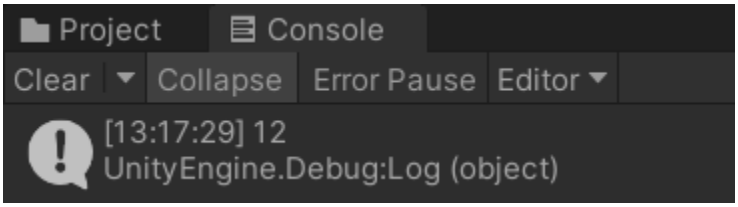
```
using UnityEngine;
```

We talked about this in a previous lesson, but this library is supplying our script with the knowledge of Unity's classes.

Next we type a period after Mathf. This period is something that is super important to programming and is what is known as the **Accessor**. This allows us to access the information in a class. If you remember back to our Fruit class, we used the Accessor to set the values of the name and price of the fruit.

Lastly because we have now accessed the Mathf class we can call a function from it. So the function we call is the absolute value function or Abs(). We then just supply it with a value, that value being our number variable.

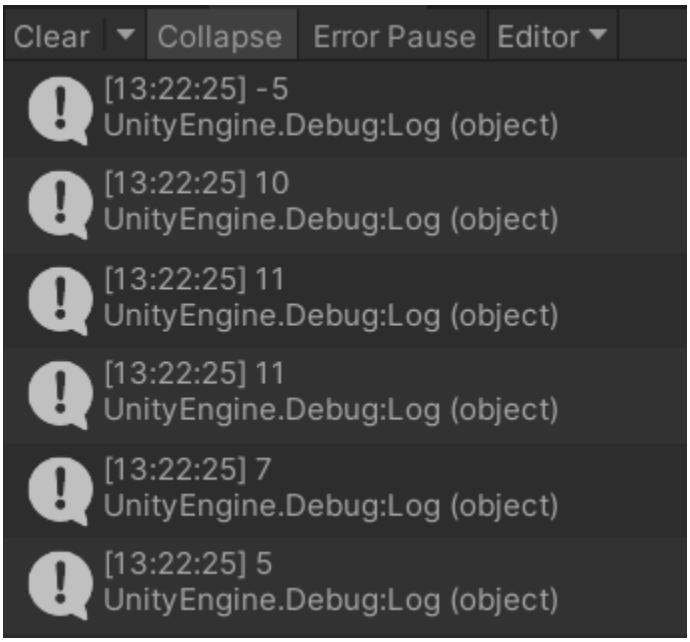
6. If we save the script and run the game. We should see the message:



7. Let's go ahead and as practice for accessing classes let's try out the other functions that were listed above. Let's write the code:

```
void Start()
{
    number = Mathf.Clamp(number,-5, 5);
    Debug.Log(number);

    Debug.Log(Mathf.Floor(10.7f));
    Debug.Log(Mathf.Ceil(10.7f));
    Debug.Log(Mathf.Ceil(10.7f));
    Debug.Log(Mathf.Max(5,7));
    Debug.Log(Mathf.Min(5, 7));
}
```

If we save the script and run the game we get the messages:



8. To cover one last thing that pertains to the accessor. Let's declare the variable:

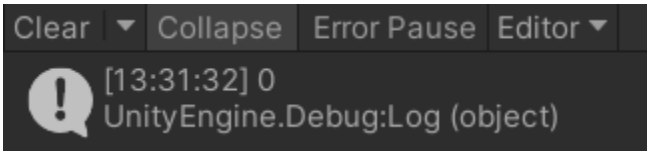
```
private Vector2 vec = new Vector2(0, 0);
```

Since this is a variable of the type Vector2 it is the class Vector2. This means that it has all of the variables and functions of the normal Vector2 class.

So instead of calling the Vector2 class directly, like we were doing for the Mathf class, we can call the name of the variable followed by the accessor to get the classes functionality.

```
Debug.Log(vec.magnitude);
```

This will output the message:



GitHub

1. Push the project to the GitHub repository.