CSE 490j
Unity Scripting Basics

Alright. Here we go with Scripting 101. This document can't cover absolutely everything that you can do when it comes to coding, but hopefully it will help you get started writing your own code. If anything is confusing, ask a TA and they will do there best to explain it to you. Also if you think something is missing, let us know as well.

Overview

Unity uses a programming language called C#, which is an object oriented programming language (if you are familiar with java then a lot of the core principles will feel familiar). As a general overview, we will discuss 3 main ideas: variables, functions, and classes.

Variables

```
public int number = 1;
public float decim = 1.5f;
public string word = "hello";
public char letter = 'h';
```

Variables are containers for information. It's how we store data so that we can use and manipulate it later it later.

To create a variable in C#, first you have to "declare it". From there you have access to what that variable stores and the ability to change it.

```
int a= 0; // declared a variable named "a", set it equal to zero
a = 1; // set "a" equal to 1
```

Each variable has a specific type, which determines the kind of information stored there, as well as a set of operations that can be applied to that variable. Variables come in a couple different flavors.

- integers represent whole numbers
- floats are decimal numbers
    - Note: you can do math with both floats and integers

```
int a = 1 + 1; // a = 2
int b = 1; // b = 1
int c = a + b; // c = 3
```

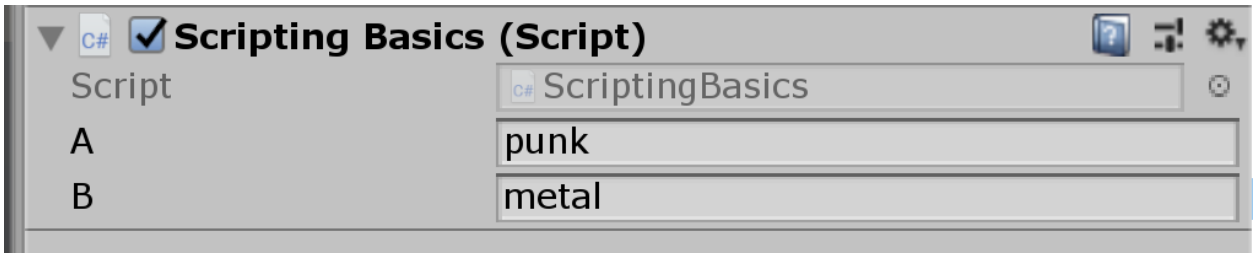
- char is a single character or letter
- Strings are words wrapped in in quotation marks
    - Note: there is a thing called string concatenation, that lets you add strings together, and even some other types.

```
string a = "punk";
string b = "metal";
int c = 2019;
string d = a + " " + b + " " + c; //d = "punk metal 2019"
```

public vs private:
Variables written outside of functions and functions themselves (more on those later) must be marked as private or public.

- private means that no other scripts can access this value or function
- Public means that any other script can access this value or function. Note that public variables actually end up in the inspector.

-



Unity specific component variables:
Any component or even custom script in Unity can also be stored as a variable so that we can access and change data in it later

```
// components
public Rigidbody rigidbody;
public BoxCollider boxCollider;
public OtherScript otherScript;
```

Arrays

Arrays are special variables that can hold multiple values. An array can be created for any type, but you have to give it a fixed size in order for it to work. When you initialize a array, it is usually a good idea to define a size.

```
    // arrays
    public int[] numArr = new int[5];
    public float[] floatArr = new float[1];
    public string[] wordArr = new string[1];
```

Each spot in an array is referred to as an index (starting at 0), and to set a value in the array, you have to specify a specific index;

```
    int[] numArr = new int[5];
    nymArr[2] = 1;
```

Array before:

0	0	0	0	0

Array After:

0	0	1	0	0

Note that in C#, an array can only hold one type. So an int array cannot hold strings or char variable types.

Functions

Functions are chunks of code where you perform operation. It is where “the action happens”. So if I need add numbers, run a calculation, move objects, this is where you will do so. The structure of a function is as such

```
    // this is a function. Functions are chunks where you can put your code
into. Functions can be called by other functions to be executed
    void functionExample() {
        // This function prints "do something" into the console menu in
Unity
        print("do something");
```

```
}
```

Any code that is between the curly braces, will be executed when the function is called. Functions can call other functions. And when you do this, you will execute the code that is in that function.

```
void functionExample() {
    print("do something");
}

void otherFunction() {
    // also will print "do something"
    functionExample();
}
```

Functions can “return” a variable type. This means, when it is called, the caller gets back a value specified by the function.

```
int functionExample() {
    print("do something");
    return 1;
}

void callerFunction() {
    // a = 1 and also print do something
    int a = functionExample();
}
```

functions can return any data type, as long as it is specified before the function name. The exception is void, which returns nothing.

```
// returns an int
int intFunction() {
    int a = 1;
    int b = 1;
    return a + b; // returns 2
}

// returns a string
string stringFunction() {
    return "punk metal";
```

```
    }

    // returns a float
    float floatFunction() {
        return 1.0f + 0.5f;
    }

    // returns nothing
    void voidFunction() {
        return;
    }
```

Functions can also have input values. These allow you to specify the values of variables used in the function. These are comma separated lists, in the parenthesis after the function name.

```
int add(int a, int b) {
      return a + b;
  }

  void addTest() {
      add(1, 1); // add will return 2

      // you can even use variables as input
      int x = 1;
      int y = 2;
      int z = add(x, y); // z = 3
  }
```

And same as before these values can be any variable type. However, if you establish input variables, they must be filled out every time you call that function.

If-Else Statements

If statements allow you to execute a certain chunk of code based on a condition. If the condition is not met, then that line of code is skipped out.

```
    if (1 + 1 == 2) {
```

```
        print("punk metal");
    }
```

If-else statements are one step further, allowing you to switch between two chunks of code based on a condition.

```
    if (1 + 1 == 2) {
        print("punk metal");
    } else {
        print("if you get here math is broken");
    }
```

Finally, else-ifs allow you to specify alternate conditions. And you can chain these indefinelty

```
    if (1 + 1 == 2) {
        print("punk metal");
    } else if (1 + 1 == 0){
        print("if you get here math is broken");
    } else {
        print("math is still broken");
    }
```

While and For loops

While loops will execute a certain block of code repeatedly until the condtion in the parenthses is false

```
   // will print "punk punk punk punk punk metal"
    int i = 0;
    while(i < 5) {
        i = int + 1;
        print("punk ");
    }
    print("metal");
```

For loops are a special kind of loop, that allow you to give a range, and a increment value automatically.

```
    // for(starting value, value you want to end at, value you want to
increment each time your execute code block inside)
    for(int i = 0; i < 5; i++) {
```

```
        print("punk ");
    }
    print("metal");
 // will also print "punk punk punk punk punk metal"
```

Classes

Classes, in a sense, are containers for functions and variables. Generally, you use this to organize data and instruction for a specific thing or set of tasks. For example, this is a student class. A student is defined by their name and age. And a student can print their name.

```
// "Student" is class name
// Monobehaviour is a class the we inheirt from.
//Essentially we get all of Monobehaviour's functions in this as well
public class Student : MonoBehaviour {

    // parameters of the class
    public string name;
    public int age;

    //Constructors are how you initilize the class. You give it starting
values.
    public Student(string name, int age) {
        this.name = name;
        this.age = age;
    }

    public string printStudent() {
        return "Hi I am " + name + " and I am " + age;
    }
}
```

You can refer to these functions and variables from others functions as such. You do this using this notion of dot notation. Where you take “variable name”.”parameter/function name”

```
public void createStudent() {
    //initializes the student with some values
    Student paul = new Student("paul", 12);

    // you can set and get a students variables with dot notation
```

```
    paul.age = 13;

    // you can call students functions with dot notation as well
    print(paul.printStudent());
}
```

Unity Specific Topics

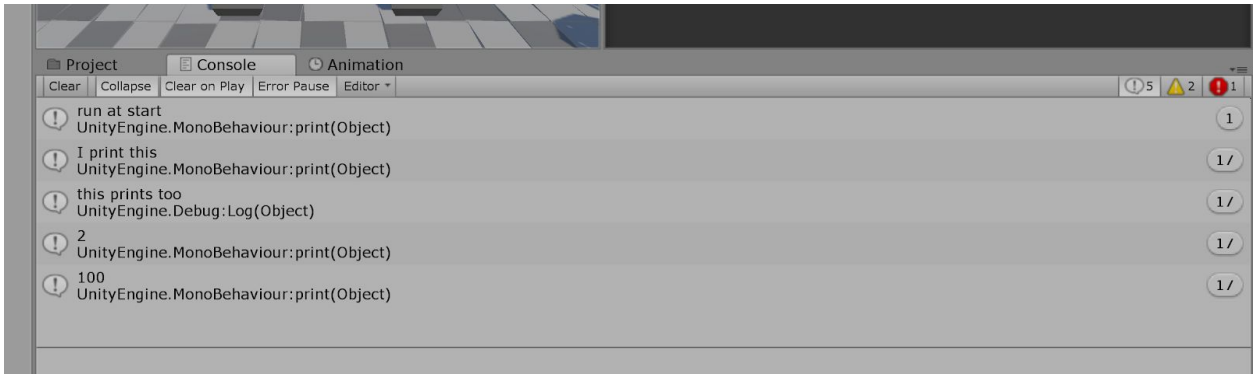
Debugging and Print statements in Unity

One of the most common methods for debugging are using print statements to print out information from your code/see if the code even gets to a certain point. The two main function to do that are print() and Debug.Log() which in essence function the same.

```
print("I print this");
Debug.Log("this prints too");
print(1 + 1);

float test = 100;
print(test);
```

By default the console window where this stuff prints out is in the console window next to the project tab.



Start and Update

Unity has a ton of functions that help make the programmers life easier. Tow of the most important ones are Start and Update.

```
        // the start function is a Unity function that is automatically executed at the beginning of
the scene
        void Start () {
        print("run at start");
        }


        // the Update function is a Unity function that is automatically called once per frame
    // this is where you will be doing a lot of your logic checking
        void Update () {

        }
```