Scope and Access Modifiers

- Scope is a term in code that relates to the parameters given to a field or variable. In essence. It is the full region, or context, in which the variable can be referenced within the code.
- The scope of a local variable reaches throughout its code block, but not outside of it. This is known as local variable scope. Local variables can be called within sub, or nested, blocks, but they cannot be called outside of their block; the variable is invisible, and does not exist to the outside code block. Variables can go down, but not up.
- Accessibility relates to the visibility of variables and fields within their script, as well as their visibility to other scripts.
- Access modifiers are pieces of code that modify how accessible a block of code and its variables are to its script and other scripts.
- One of the most common modifiers is private. Private means the code can only be accessed within its type. It prevents other scripts from accessing, reading, or executing it. It also prevents any code outside of the block from reading it. While one can explicitly declare code private, it isn't necessary. That is because it is the default setting for code in C#. If you don't specifically put a modifier in front of a declaration, it will be assigned a private status.
- Two other common modifiers are public and protected. Public types can be seen by other editors and scripts. They are completely available and accessible by everything. Protected is slightly different. Protected code can be seen by its type or any nested code blocks, but not ones outside its type. It is private otherwise.

- The last two modifiers are internal and protected internal. Internal is only available to itself and associated assemblies. Non-nested code blocks default to this setting. Protected internal is essentially a combination of the protected and internal types. It is slightly more accessible than the internal or protected types by themselves.
- public class homeworkEffort : Monobehaviour {

```
public int guy = 5;
        public void regularEyes () {
                protected int scared = 8;
                int answer = (guy + scared);
                print(answer);
        }
}
```

- public class imTryin : Monobehaviour {

```
public string privateYo = “implied privacy”;
public void nest () {
        string privateYo = “private is the default modifier”;
        protected string wow = “I can be seen in this type and nested blocks, but not the root.”;
        void Inside () {
                wow = “new types can be declared within types. They are called nested types.”;
        }
}
```

}

- public class scope : Monobehaviour {

    public int guy = 5;

    public string truth;

        public void regularEyes () {

            int ninja = 3;

            protected int scared = 8;

            int answer = (guy + scared);

            print(answer);

        }

        public void learning () {

            truth = “I can’t call for scared or ninja because they are protected and private. Their scope limits them to their local type. I can call for guy because he is public.”

            print(truth);

        }

    }