unity

C# Scripting – Beginner

Hit **Ctrl = ’** (3 keys) while highlighting text in MonoDevelop for help

MonoDevelop Commands

Command	Description
Ctrl Tab	Go to next script
Ctrl Shift Tab	Go to previous script
Tab	Indent selected lines (no need to select all the characters on the lines)
Shift Tab	Reverses indent on selected lines
Ctrl = ’	Opens Unity Script Reference in browser and searches for selected text.
Ctrl F	Find (add shift to search all scripts)
Ctrl H	Replace (add shift to search all scripts)

Variable (Data) Types

Variable Type	Description
int	whole numbers i.e. 1
float	Fractional numbers (anything with a decimal) *i.e. 1.0 or 1.123*
bool	True or False
string	String of characters *i.e. “I am a string.”*
GameObject	Stores a Unity GameObject
ClassName	Any class declared (your classes too) can be used to create variables.

Declaring a Variable: Use ‘CamelCap’ to name variables. Capitalize the first letter of each word in the name (the humps on a camel). A **Variable’s** first word is **not capitalized** while **Functions and Classes/Scripts** have all words **capitalized** (helps differentiate between them)

Declaration	Description
int myInt;	Type and Name only, no starting value
int myInt = 0;	You can also have an initial value.
public int myInt;	Allows the variable to be edited in the inspector and by other scripts.

Conventions and Syntax

	Description
//	Starts a one line comment
/*	Open ended comment, useful for disabling code
***/**	Ends a multiple line comment

The Dot Operator: The dot operator (period ‘**.**’) allows you to access public sub-classes, functions, and variables within the parent *i.e. gameObject.transform* gets the transform from gameObject.

Semi-Colons: The semi-colon **;** is used to terminate statements. Everything not followed by a **{** needs a **;**

Brackets: Functions, Classes, Ifs, Loops use **{**. You should start a new line and then indent everything inside the bracket to visually show proper nesting. Anything declared inside **{}** has a smaller **scope** and is not known outside the **{}**. *i.e.*

```
Void FunctionName()
{
    string myString1 = “a string”;
    If()
    {
        string myString2 = “a second string”;
        //myString1 is known here (myString1 has a larger scope)
    }
    //myString2 is not known here (myString2 has a smaller scope)
}
```

Functions

Declaring a function: first declare what the function is expected to return which can be any of the data types listed earlier or if you don’t want it to return anything just type **void** *FunctionName***.** *i.e.*

```
string FunctionName()
{
    return “I am a string”;
    /* exits function and returns the string; Will not exit until it hits a
      return. no code after a return will be run*/
}
void FunctionName()
{
    //do something here, will automatically exit
    // when the end of the code has been reached
}
```

Parameters: Some functions take arguments / parameters. These are just local variables for the function to use. Declare variables within the **()** for the function. Use a comma **,** to declare multiple. *i.e.*

```
void FunctionName(bool myBool, string myString = “default value”)
{
    /*myBool is now a variable that can be used within this function.
      This variable can be passed in when calling a function.*/
    /*if the variable has a declared value then it does not need to be
      passed but the default value can be overridden if passed.*/
}
```

Calling a Function: To call a function, simply use the proper scope (use the Dot Operator as necessary) and then type in the function name followed by (); *i.e.*

```
MyFunction(); //no parameters
MyFunction(MyBoolean, MyString); // if it takes parameters
OtherComponent.MyFunction(); // if it resides in a different scope
```

unity

C# Scripting – Beginner

Hit **Ctrl = '** (3 keys) while highlighting text in MonoDevelop for help

IF() / ELSE Statements

Logical Operators	Description
A == B	Equal To
A != B	Not Equal To
A < B , A > B	Less than, greater than
A <= B, A >= B	Less than or equal, Greater or equal
&&	If **both** conditions are true *i.e. (A==B && B==C)*
||	If **One** condition is true *i.e. (A==B || B==C)*
Blank	If no condition is given then it tests if the bool is **true**
!	Place before a bool to test if **False** *!myBool same as myBool == false*

If statements set conditions in your code that come down to a Boolean value of **True** or **False**. *i.e.*

```
Void Update()
{
    if(Input.GetKeyDown(KeyCode.Space))
    {
        //do something only if Space was just hit
    }
    else if (myInt > 30 && myInt <= 100)
    {
        /* run only if previous was not true but both current
          arguments are true*/
    }
    else
    {
        //run only if all directly previous ifs were false
    }
}
```

Useful Functions

Name	Description
Start	Runs when instantiated.
Update	Runs once every frame.
FixedUpdate	Runs once every physics step.
OnCollisionEnter	Runs once this collider hits another.
OnTriggerEnter	Runs once this trigger hits a collider.

```
OnCollisionEnter(Collision other)
{
    // runs once every time 'other' touches this collider.
    // Similar functions:
    //OnCollisionStay – runs every frame they are colliding
    //OnCollisonExit – runs once when no longer colliding
}
OnTriggerEnter(Collider other)
{
    //same as OnCollision functions except it is Collider not Collision
}
```

Loops

ForLoop: for(declare/use integer here; what condition has to be true for this to stop; what to do after every iteration) *i.e.*

```
for (int i = 0; i < 20; i++) // this for loop will add 1 to i until i is 20
{
    // i = current loop iteration i.e. 1,2,3,4,5...
}
```

ForEach: iterated through an array or list of items. Foreach(Type/Class currentObjName in list/array) *i.e.*

```
foreach(GameObject currentObject in
GameObject.FindGameObjectsWithTag("Enemies"))
{
    //Do something to currentObject
}
```

While: While loops test the condition **before running** *i.e.*

```
myBool = false;
while(myBool == true)
{
    myBool = true;
     /*WARNING will continue running until condition is met so if
       that never happens you will be stuck in this loop forever
       (infinite loop)*/
}
```

DoWhile: tests the condition **after running** *i.e.*

```
do
{
    //run code here... automatically runs at least once
}
While(myBool == true);
```

Components and Instantiating:

Getting Components: *i.e.*

```
GetComponent<Light>(); // gets light component
GetComponent<Light>().color; // gets the color property from Light
GetComponent<Light>().enabled = false; // sets the Light to disabled
```

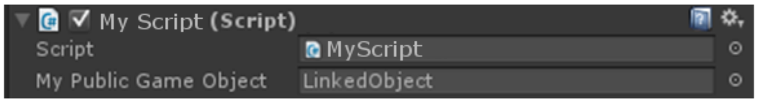
To instantiate a GameObject:

```
Instantiate(myPrefab);
```

If you want to set an instantiated object to a variable:

```
GameObject GO = Instantiate(myPrefab) as GameObject;
GO.transform.position = Vector3.one();   //now do stuff to GO
```

Public GameObject variables are important when instantiating. If you declare a public variable then you can drag, in the inspector, custom GameObjects from the scene or prefabs to use.


My Script (Script)
Script: MyScript
My Public Game Object: LinkedObject