C#

(C Sharp)

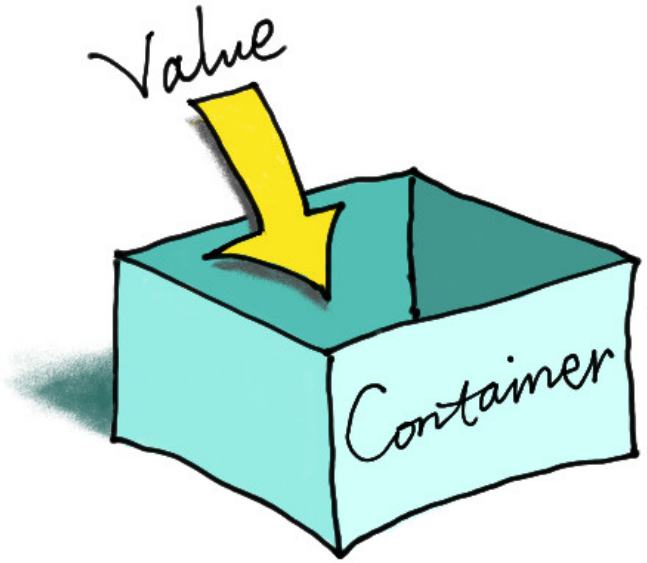
Checklist

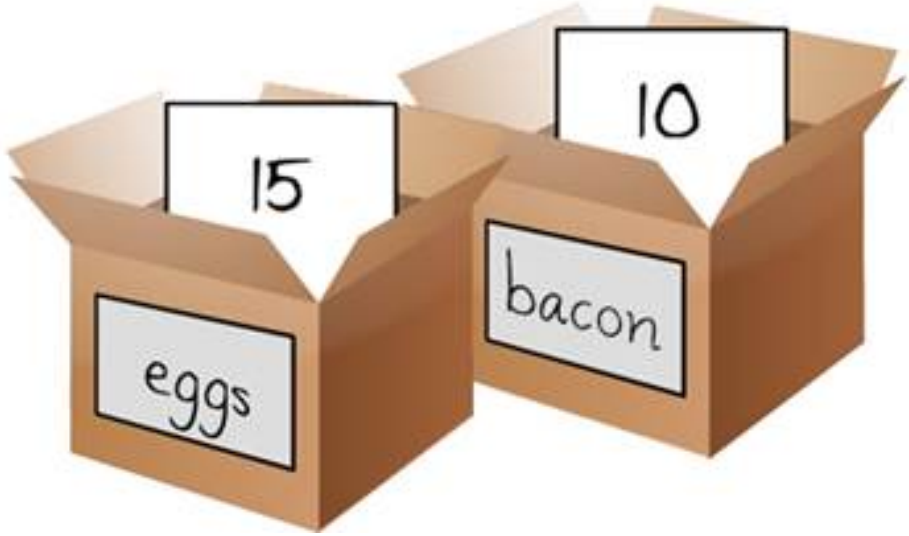
- Install VS Code
- Edit -> Preferences -> External Tools -> External Script Editor
  - Set to “Visual Studio Code”
- VS Code Extensions
  - “C#”
  - “Debugger for Unity”
  - (Optional) “Material-theme”

Variables

Storing Data

Named Boxes





VARIABLE NAME

VALUE

```
int numJupiterMoons = 67;
```

VARIABLE TYPE

ASSIGNMENT OPERATOR

```
// Camel Case

// Good - short, descriptive
numJupiterMoons
materialColor
playerSpeed

// Bad - long, ambiguous
thatFirstThing
theSuperImportantVariableThatMustNotBeNamed
```

Integral Types Table (C# Reference)

Visual Studio 2015 | Other Versions

The following table shows the sizes and ranges of the integral types, which constitute a subset of simple types.

Type	Range	Size
sbyte	-128 to 127	Signed 8-bit integer
byte	0 to 255	Unsigned 8-bit integer
char	U+0000 to U+ffff	Unicode 16-bit character
short	-32,768 to 32,767	Signed 16-bit integer
ushort	0 to 65,535	Unsigned 16-bit integer
int	-2,147,483,648 to 2,147,483,647	Signed 32-bit integer
uint	0 to 4,294,967,295	Unsigned 32-bit integer
long	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807	Signed 64-bit integer
ulong	0 to 18,446,744,073,709,551,615	Unsigned 64-bit integer

https://msdn.microsoft.com/en-us/library/exx3b86w.aspx

VALUE

```
float taxAmount = 0.07f;
```

SUFFIX

Floating-Point Types Table (C# Reference)

Visual Studio 2015 | Other Versions

The following table shows the precision and approximate ranges for the floating-point types.

Type	Approximate range	Precision
float	±1.5e−45 to ±3.4e38	7 digits
double	±5.0e−324 to ±1.7e308	15-16 digits

decimal (C# Reference)

Visual Studio 2015 | Other Versions

The **decimal** keyword indicates a 128-bit data type. Compared to floating-point types, the **decimal** type has more precision and a smaller range, which makes it appropriate for financial and monetary calculations. The approximate range and precision for the **decimal** type are shown in the following table.

Type	Approximate Range	Precision	.NET Framework type
decimal	$(-7.9 \times 10^{28}$ to $7.9 \times 10^{28}) / (10^{0}$ to $^{28})$	28-29 significant digits	System.Decimal

https://msdn.microsoft.com/en-us/library/9ahet949.aspx

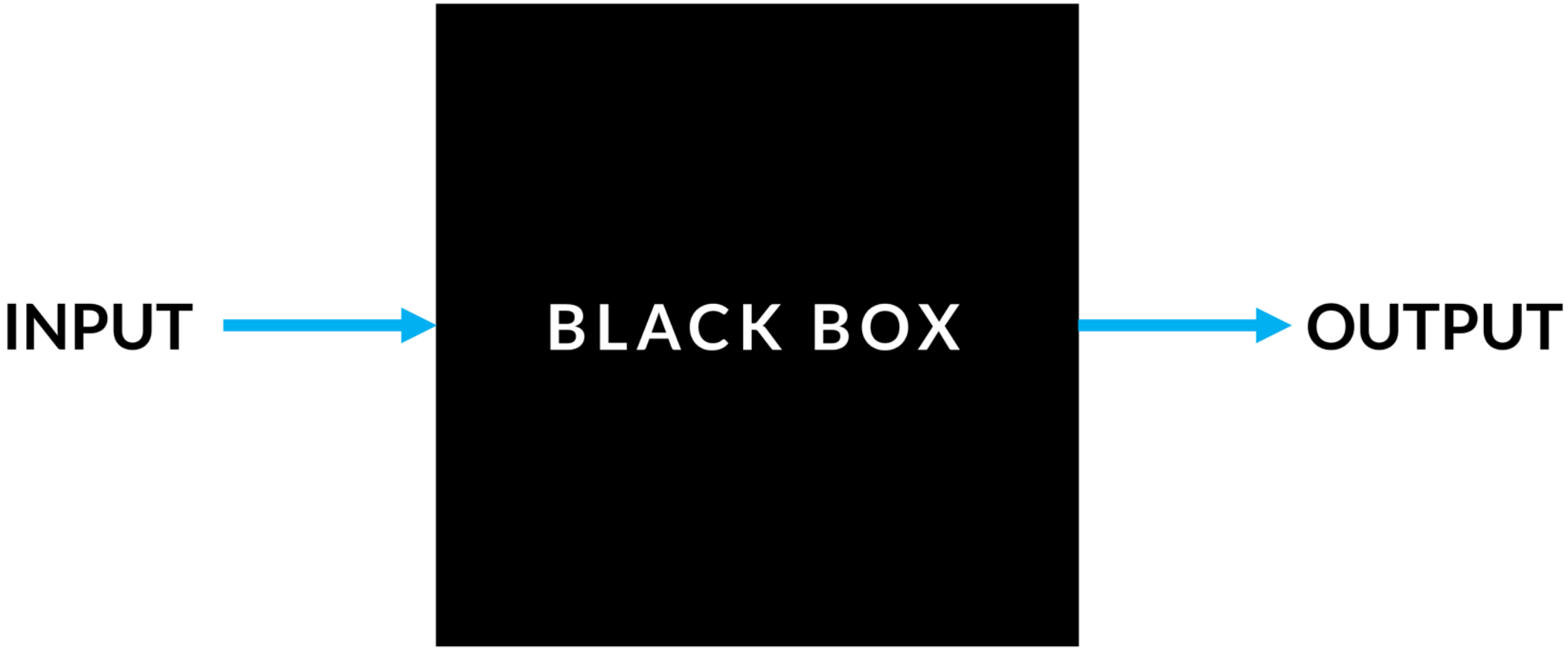
START END

```
string quoteOfDay = "Perfect is the enemy of good.";
```

```
// -- EXERCISES ----------------------------------------------------------

// 1. Create a variable that holds the total cost of three items:
//  - magnets, $25.75
//  - LEDs, $15.10
//  - tape, $3.10


// 2. Print the value of your total cost variable:


// 3. Now factor in a discount of 25% off and print out the new total:


// 4. Find the average of the following test scores: 100, 90, 85, 74, 82
```

Functions

Readability && Reusability


INPUT
BLACK BOX
OUTPUT

FUNCTION NAME

```
void PrintWelcomeMessage() {
    Debug.Log("Hello there. The console welcomes you.");
}
```

FUNCTION CONTENTS

```
void Start() {
    PrintWelcomeMessage();
}
```

FUNCTION INVOCATION

```
void PrintWelcomeMessage() {
    Debug.Log("Hello there. The console welcomes you.");
}
```

```
// Pascal Case

// Good - descriptive verb phrases
CalculateRectanglePerimeter
CreateExplosion

// Bad - long, ambiguous or not verb phrase
Health
IDoNotKnowWhatThisDoes
```

PARAMETER

```
void WelcomePlayer(string playerName) {
    Debug.Log("Hello there, " + playerName + ". Welcome!");
}
```

```
void Start() {
    WelcomePlayer("Mike");
}
```

(ARGUMENT)

(PARAMETER)

```
void WelcomePlayer(string playerName) {
    Debug.Log("Hello there, " + playerName + ". Welcome!");
}
```

RETURN TYPE

```
int CalculateRectanglePerimeter(int width, int height) {
    int perimeter = (2 * width) + (2 * height);
    return perimeter;
}
```

RETURN STATEMENT

```
void Start() {
    int perimeter1 = CalculateRectanglePerimeter(10, 20);
    Debug.Log(perimeter1);
}



int CalculateRectanglePerimeter(int width, int height) {
    int perimeter = (2 * width) + (2 * height);
    return perimeter;
}
```

Function Signatures

```
CalculateRectanglePerimeter(int width, int height)

CalculateRectanglePerimeter(float width, float height)
```

```
// Create a ComplimentPlayer function that takes one string parameter
// (playerName) and prints something nice about the player (e.g.
// "[Player name], you look nice today.")
// Test it by invoking the function with your name.

// Create a CalculateRectangleArea function that takes two float parameters
// (one for width and one for height) and returns the area of the rectangle.
// Test it by calculating the area of a 10.25 x 19.5 rectangle.

// Create a CalculateAverage function that takes three float parameters,
// averages them and returns the result.
// Test it by calculating the average of 10.5, 7.75 and 6.

// Create an ApplyDiscount function that takes two floats - a total cost
// and a discount fraction (e.g. .25) - and returns the final discounted
// price.
// Test it by calculating the discounted price of an item that is 19.99 and
// on sale for 30% off.
```

Manipulating the Transform

transform.Rotate

public void **Rotate**(float **xAngle**, float **yAngle**, float **zAngle**, Space **relativeTo** = Space.Self);

Parameters

xAngle	Degrees to rotate around the X axis.
yAngle	Degrees to rotate around the Y axis.
zAngle	Degrees to rotate around the Z axis.
relativeTo	Rotation is local to object or World.

Description

Applies a rotation of `zAngle` degrees around the z axis, `xAngle` degrees around the x axis, and `yAngle` degrees around the y axis (in that order).

If `relativeTo` is not specified or set to Space.Self the rotation is applied around the transform's local axes. If `relativeTo` is set to Space.World the rotation is applied around the world x, y, z axes.

```
using UnityEngine;

public class ExampleClass : MonoBehaviour
{
    void Update()
    {
        // Rotate the object around its local X axis at 1 degree per second
        transform.Rotate(Time.deltaTime, 0, 0);

        // ...also rotate around the World's Y axis
        transform.Rotate(0, Time.deltaTime, 0, Space.World);
    }
}
```

transform.Translate

public void **Translate**(float **x**, float **y**, float **z**, Space **relativeTo** = Space.Self);

Parameters

Description

Moves the transform by x along the x axis, y along the y axis, and z along the z axis.

If `relativeTo` is left out or set to Space.Self the movement is applied relative to the transform's local axes. (the x, y and z axes shown when selecting the object inside the Scene View.) If `relativeTo` is Space.World the movement is applied relative to the world coordinate system.

```
using UnityEngine;
using System.Collections;

public class ExampleClass : MonoBehaviour {
    void Update() {
        transform.Translate(0, 0, Time.deltaTime);
        transform.Translate(0, Time.deltaTime, 0, Space.World);
    }
}
```

gameObject.setActive

public void **SetActive**(bool **value**);

Parameters

value	Activate or deactivation the object.

Description

Activates/Deactivates the GameObject.

Note that a GameObject may be inactive because a parent is not active. In that case, calling SetActive() will not activate it, but only set the local state of the GameObject, which can be checked using GameObject.activeSelf. This state will then be used once all parents are active.
Making a GameObject inactive will disable every component, turning off any attached renderers, colliders, rigidbodies, scripts, etc... Any scripts that you have attached to the GameObject will no longer have `Update()` called, for example.

See Also: GameObject.activeSelf, GameObject.activeInHierarchy.

```
using UnityEngine;
using System.Collections;

public class ExampleClass : MonoBehaviour {
    void Example() {
        gameObject.SetActive(false);
    }
}
```

Events

(A.K.A. Messages)

Messages

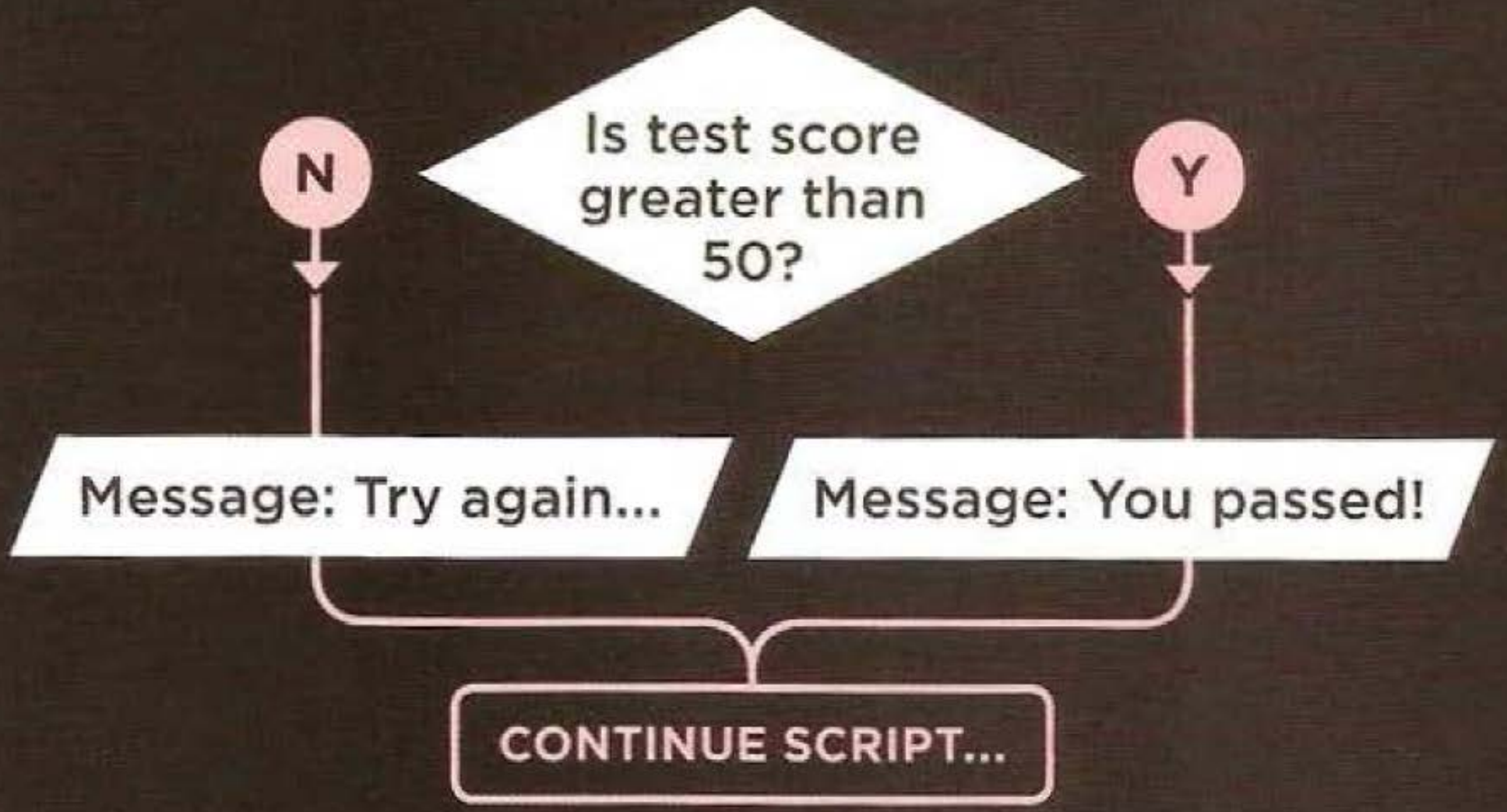
Start	Start is called on the frame when a script is enabled just before any of the Update methods is called the first time.
Update	Update is called every frame, if the MonoBehaviour is enabled.

OnMouseDown	OnMouseDown is called when the user has pressed the mouse button while over the GUIElement or Collider.
OnMouseDrag	OnMouseDrag is called when the user has clicked on a GUIElement or Collider and is still holding down the mouse.
OnMouseEnter	Called when the mouse enters the GUIElement or Collider.
OnMouseExit	Called when the mouse is not any longer over the GUIElement or Collider.
OnMouseOver	Called every frame while the mouse is over the GUIElement or Collider.
OnMouseUp	OnMouseUp is called when the user has released the mouse button.
OnMouseUpAsButton	OnMouseUpAsButton is only called when the mouse is released over the same GUIElement or Collider as it was pressed.

https://docs.unity3d.com/ScriptReference/MonoBehaviour.html

Conditionals

Making Decisions



From JavaScript & jQuery

```
float score = 100f;
```

CONDITION

```
if (score >= 50) {
    Debug.Log("The score was a passing grade :)");
}
```

RESULT

Comparison Operators

>

>=

<

<=

==

!=

Boolean Variables

```
bool isRaining = true;
bool hasPressedKey = false;
```

Boolean Variables

```
float score = 100f;
bool isPassing = score >= 50;
```

```
if (score >= 50) {
    Debug.Log("The score was a passing grade :)");
} else {
    Debug.Log("The score was a failure :(");
}
```

```
if (score >= 95) {
    Debug.Log("A+ bro.");
} else if (score >= 50) {
    Debug.Log("The score was a passing grade :)");
} else {
    Debug.Log("The score was a failure :(");
}
```

```
// Create a function that takes a (float) number and returns a string with
// the corresponding letter grade. Use this grade scale:
//
//  A = 90 to 100
//  B = 80 up to (but not including) 90
//  C = 70 up to (but not including) 80
//  D = 60 up to (but not including) 70
//  F = Anything below a 60
```

```
string determineGrade(float score) {
    if (score >= 90) {
        return "A";
    } else if (score >= 80) {
        return "B";
    } else if (score >= 70) {
        return "C";
    } else if (score >= 60) {
        return "D";
    } else {
        return "F";
    }
}
```

```
string determineGrade(float score) {
    string grade;
    if (score >= 90) {
        grade = "A";
    } else if (score >= 80) {
        grade = "B";
    } else if (score >= 70) {
        grade = "C";
    } else if (score >= 60) {
        grade = "D";
    } else {
        grade = "F";
    }
    return grade;
}
```

Getting Input

(Quick Way)

Input.GetKey

public static bool **GetKey**(string **name**);

Parameters

Description

Returns true while the user holds down the key identified by `name`. Think auto fire.

For the list of key identifiers see Input Manager. When dealing with input it is recommended to use Input.GetAxis and Input.GetButton instead since it allows end-users to configure the keys.

```
using UnityEngine;
using System.Collections;

public class ExampleClass : MonoBehaviour {
    void Update() {
        if (Input.GetKey("up"))
            print("up arrow key is held down");

        if (Input.GetKey("down"))
            print("down arrow key is held down");

    }
}
```

https://docs.unity3d.com/ScriptReference/Input.GetKey.html

Input.GetKeyDown

public static bool **GetKeyDown**(string **name**);

Parameters

Description

Returns true during the frame the user starts pressing down the key identified by `name`.

You need to call this function from the Update function, since the state gets reset each frame. It will not return true until the user has released the key and pressed it again.

For the list of key identifiers see Input Manager. When dealing with input it is recommended to use Input.GetAxis and Input.GetButton instead since it allows end-users to configure the keys.

```
using UnityEngine;
using System.Collections;

public class ExampleClass : MonoBehaviour {
    void Update() {
        if (Input.GetKeyDown("space"))
            print("space key was pressed");

    }
}
```

https://docs.unity3d.com/ScriptReference/Input.GetKeyDown.html

Input.GetKeyUp

public static bool **GetKeyUp**(string **name**);

Parameters

Description

Returns true during the frame the user releases the key identified by `name`.

You need to call this function from the Update function, since the state gets reset each frame. It will not return true until the user has pressed the key and released it again.

For the list of key identifiers see Input Manager. When dealing with input it is recommended to use Input.GetAxis and Input.GetButton instead since it allows end-users to configure the keys.

```
using UnityEngine;
using System.Collections;

public class ExampleClass : MonoBehaviour {
    void Update() {
        if (Input.GetKeyUp("space"))
            print("space key was released");

    }
}
```

https://docs.unity3d.com/ScriptReference/Input.GetKeyUp.html

Keys

The names of keys follow this convention:

- Normal keys: “a”, “b”, “c” ...
- Number keys: “1”, “2”, “3”, ...
- Arrow keys: “up”, “down”, “left”, “right”
- Keypad keys: “[1]”, “[2]”, “[3]”, “[+]”, “[equals]”
- Modifier keys: “right shift”, “left shift”, “right ctrl”, “left ctrl”, “right alt”, “left alt”, “right cmd”, “left cmd”
- Mouse Buttons: “mouse 0”, “mouse 1”, “mouse 2”, ...
- Joystick Buttons (from any joystick): “joystick button 0”, “joystick button 1”, “joystick button 2”, ...
- Joystick Buttons (from a specific joystick): “joystick 1 button 0”, “joystick 1 button 1”, “joystick 2 button 0”, ...
- Special keys: “backspace”, “tab”, “return”, “escape”, “space”, “delete”, “enter”, “insert”, “home”, “end”, “page up”, “page down”
- Function keys: “f1”, “f2”, “f3”, ...

The names used to identify the keys are the same in the scripting interface and the Inspector.

```
value = Input.GetKey ("a");
```

https://docs.unity3d.com/Manual/ConventionalGameInput.html

Input.GetAxis

public static float **GetAxis**(string **axisName**);

Parameters

Description

Returns the value of the virtual axis identified by `axisName`.

The value will be in the range -1...1 for keyboard and joystick input. If the axis is setup to be delta mouse movement, the mouse delta is multiplied by the axis sensitivity and the range is not -1...1.

This is frame-rate independent; you do not need to be concerned about varying frame-rates when using this value.

```
using UnityEngine;
using System.Collections;

public class ExampleClass : MonoBehaviour {
    public float horizontalSpeed = 2.0F;
    public float verticalSpeed = 2.0F;
    void Update() {
        float h = horizontalSpeed * Input.GetAxis("Mouse X");
        float v = verticalSpeed * Input.GetAxis("Mouse Y");
        transform.Rotate(v, h, 0);
    }
}
```

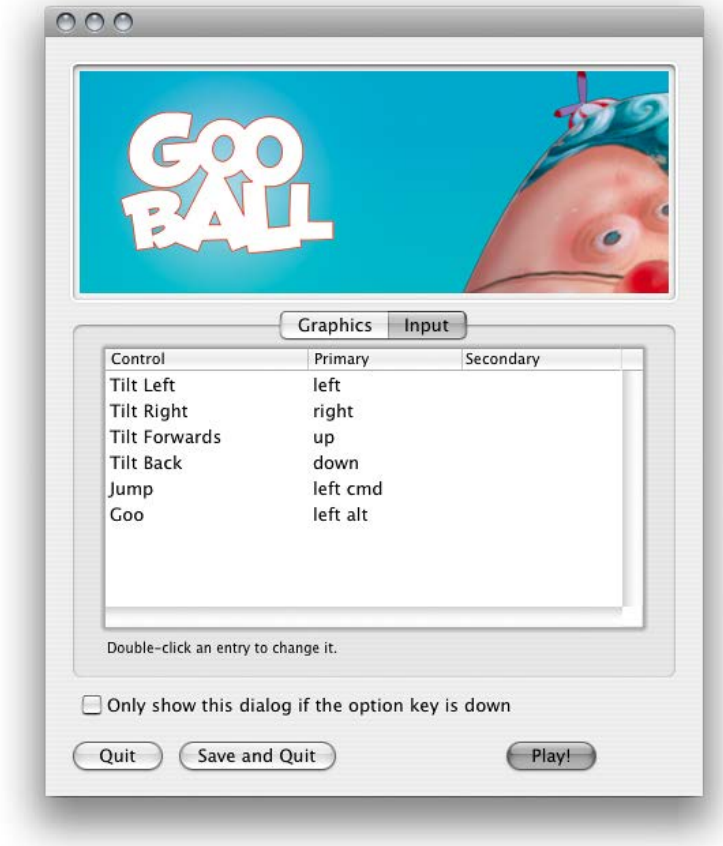
https://docs.unity3d.com/ScriptReference/Input.GetAxis.html

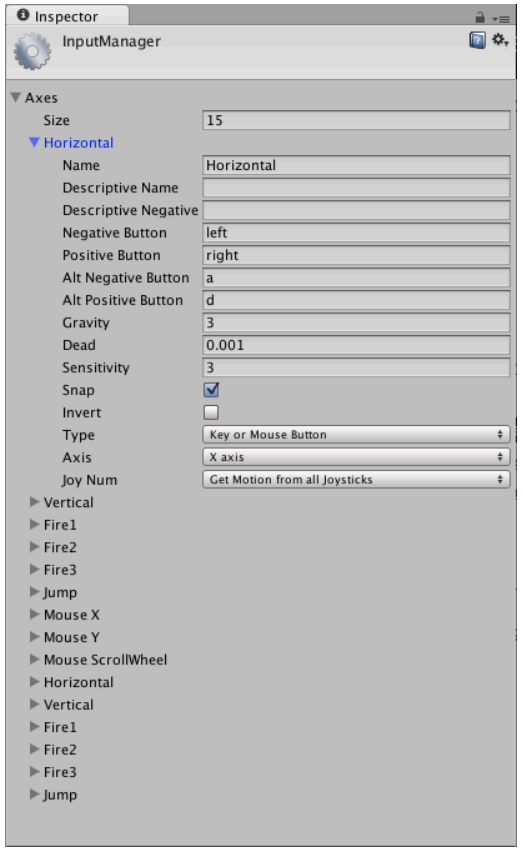
More Mouse Inputs

- Input.GetMouseButton
- Input.GetMouseButtonDown
- Input.GetMouseButtonUp

Customizable Input

See https://docs.unity3d.com/Manual/Input.html





Euler vs Quaternions

Euler Rotation

```
float horizontalMovement = Input.GetAxis("Mouse X");
float verticalMovement = Input.GetAxis("Mouse Y");

// Wrong way to rotate along two axes! Don't do this.
transform.Rotate(0, horizontalMovement, 0);
transform.Rotate(-verticalMovement, 0, 0);
```


Gimbal Lock
in 30 seconds

Quaternion.Euler

public static Quaternion **Euler**(float **x**, float **y**, float **z**);

Parameters

Description

Returns a rotation that rotates z degrees around the z axis, x degrees around the x axis, and y degrees around the y axis (in that order).

```
using UnityEngine;
using System.Collections;

public class ExampleClass : MonoBehaviour {
    public Quaternion rotation = Quaternion.Euler(0, 30, 0);
}
```

https://docs.unity3d.com/ScriptReference/Quaternion.Euler.html

Quaternion Rotation

```
// Rotating with quaternions - much better!
transform.localRotation = Quaternion.Euler(45f, 20f, 0f);
```