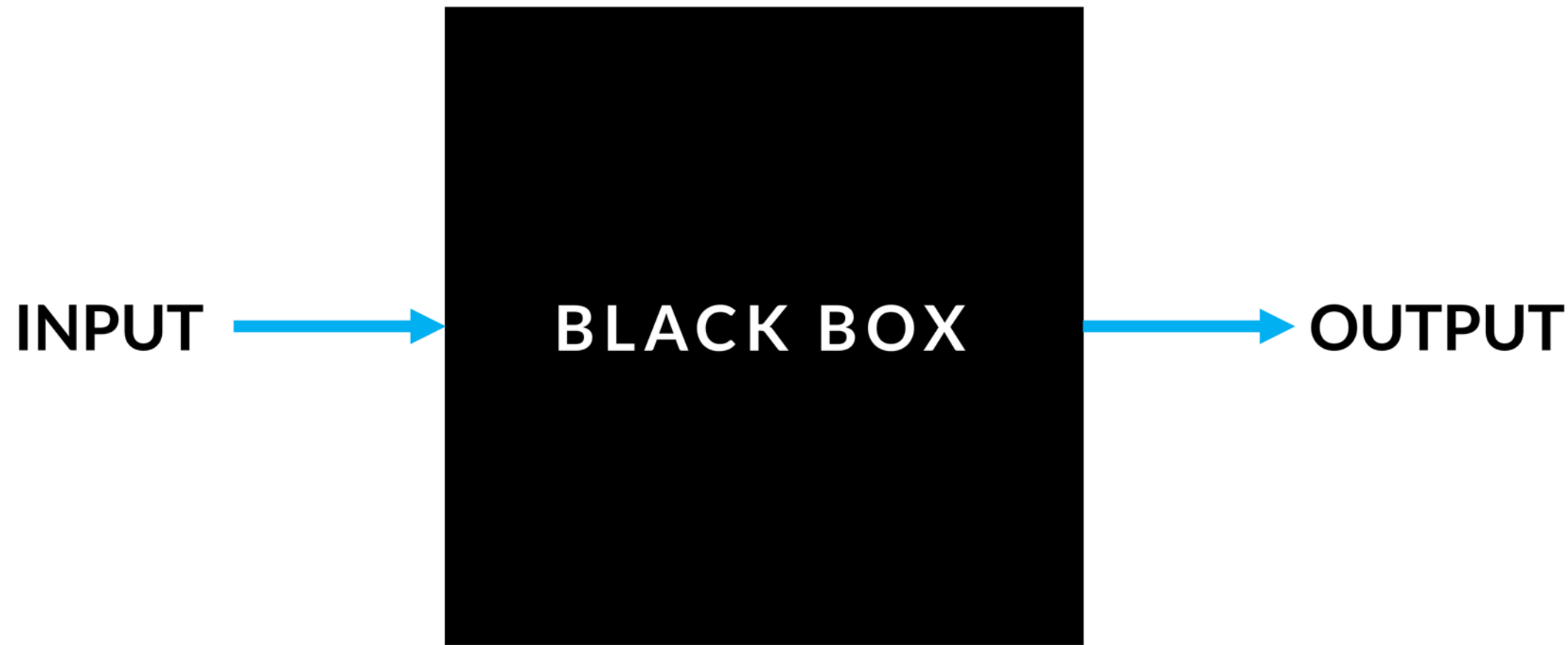




Functions

Readability && Reusability





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;  
}
```

A white arrow originates from the `CalculateRectanglePerimeter(10, 20);` line in the `Start()` function and points to the `CalculateRectanglePerimeter` function definition below. The arrow is composed of a horizontal segment pointing right, a vertical segment pointing down, and another horizontal segment pointing right to the function signature.



Function Signatures

```
CalculateRectanglePerimeter(int width, int height)
```

```
CalculateRectanglePerimeter(float width, float height)
```


Manipulating the Transform

(Easy Mode)



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);
    }
}
```

Events

(A.K.A. Messages)



Messages

<u>Start</u>	Start is called on the frame when a script is enabled just before any of the Update methods is called the first time.
<u>Update</u>	Update is called every frame, if the MonoBehaviour is enabled.
<u>OnMouseDown</u>	OnMouseDown is called when the user has pressed the mouse button while over the GUIElement or Collider.
<u>OnMouseDrag</u>	OnMouseDrag is called when the user has clicked on a GUIElement or Collider and is still holding down the mouse.
<u>OnMouseEnter</u>	Called when the mouse enters the GUIElement or Collider.
<u>OnMouseExit</u>	Called when the mouse is not any longer over the GUIElement or Collider.
<u>OnMouseOver</u>	Called every frame while the mouse is over the GUIElement or Collider.
<u>OnMouseUp</u>	OnMouseUp is called when the user has released the mouse button.
<u>OnMouseUpAsButton</u>	OnMouseUpAsButton is only called when the mouse is released over the same GUIElement or Collider as it was pressed.



Accessing Components

Components On The Same Object

```
public class LightColorSwitcher : MonoBehaviour {  
  
    private Light LightComponent;  
  
    // Use this for initialization  
    void Start () {  
        LightComponent = GetComponent<Light>();  
    }  
  
    // Update is called once per frame  
    void Update () {  
  
    }  
}
```

Generic Method

```
LightComponent = GetComponent<Light>();
```




TYPE OF
COMPONENT

MeshRenderer & Material

<http://docs.unity3d.com/ScriptReference/MeshRenderer.html>

<http://docs.unity3d.com/ScriptReference/Material.html>



▼  ☒ **Mesh Renderer**

Cast Shadows

Receive Shadows

Motion Vectors

▼ Materials

Size

Element 0

Light Probes

Reflection Probes

Anchor Override

On

☒

Per Object Motion



1

☒ Cereal

Blend Probes

Blend Probes

None (Transform)

↑

↓

↑


↓

↓

↓

↓

⦿



```
private Material Mat;
```

```
void Start () {
```

```
    // Get a reference to the MeshRenderer component
```

```
    MeshRenderer renderer = GetComponent<MeshRenderer>();
```

```
    // Store a reference to the (first) material
```

```
    Mat = renderer.material;
```

```
    // Change the material to red
```


```
    Mat.color = new Color(1f, 0f, 0f);
```

```
}
```

Conditionals

Making Decisions





```
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 :(");  
}
```




```
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");
    }
}
```

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");
    }
}
```

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");
    }
}
```

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");
```