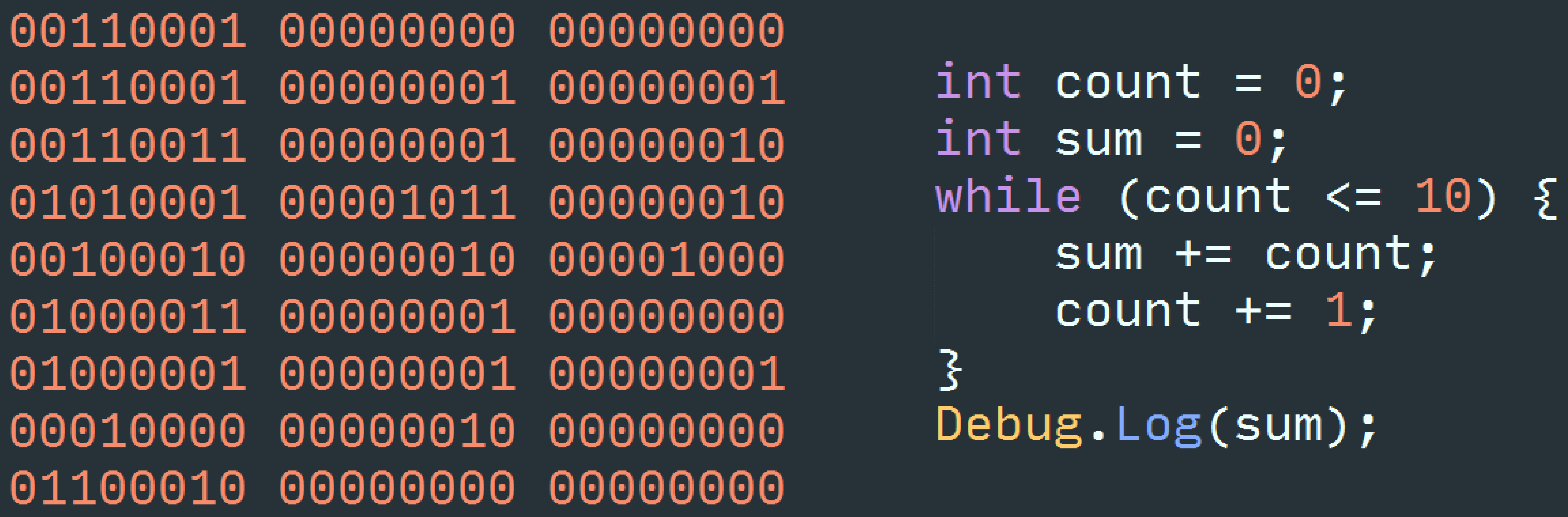
C#

(C Sharp)

Programming?

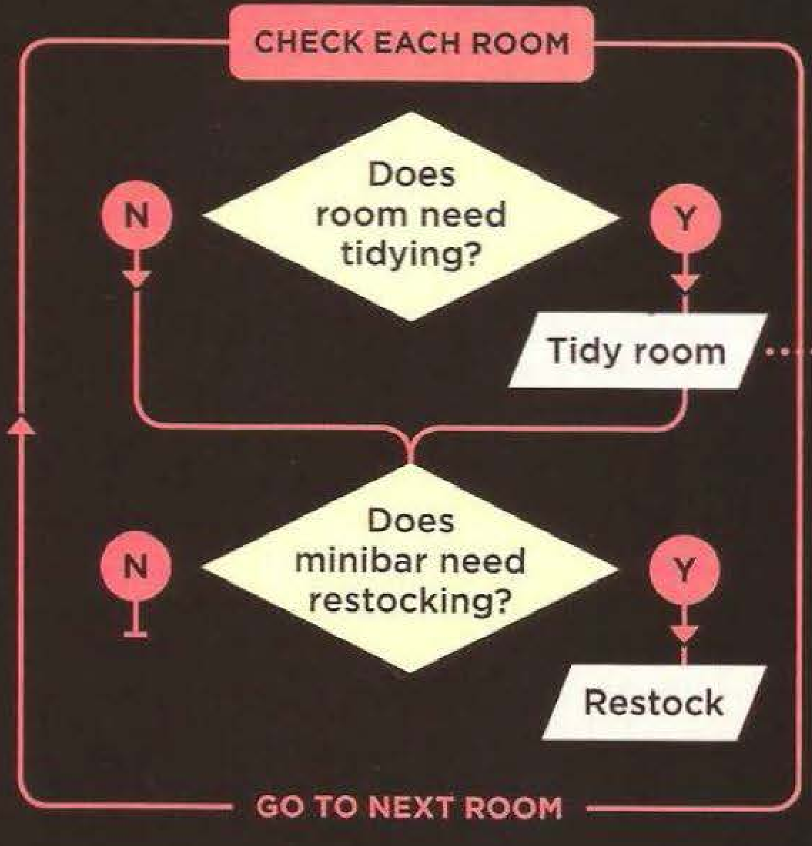


```
00110001 00000000 00000000
00110001 00000001 00000001
00110011 00000001 00000010
01010001 00001011 00000010
00100010 00000010 00001000
01000011 00000001 00000000
01000001 00000001 00000001
00010000 00000010 00000000
01100010 00000000 00000000
```

```
int count = 0;
int sum = 0;
while (count <= 10) {
    sum += count;
    count += 1;
}
Debug.Log(sum);
```

Modified from: http://eloquentjavascript.net/00_intro.html

FLOWCHART: TASKS OF A HOTEL CLEANER



CHECK EACH ROOM

Does room need tidying?

N

Y

Tidy room

Does minibar need restocking?

N

Y

Restock

GO TO NEXT ROOM

LIST: STEPS REQUIRED TO TIDY A ROOM

STEP 1 Remove used bedding

STEP 2 Wipe all surfaces

STEP 3 Vacuum floors

STEP 4 Fit new bedding

STEP 5 Remove used towels and soaps

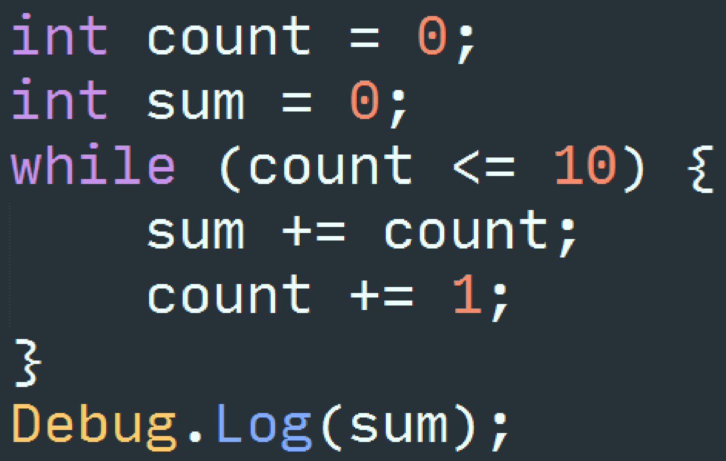
STEP 6 Clean toilet, bath, sink, surfaces

STEP 7 Place new towels and soaps

STEP 8 Wipe bathroom floor

From: JavaScript & jQuery by Jon Duckett

Compilation



```
int count = 0;
int sum = 0;
while (count <= 10) {
    sum += count;
    count += 1;
}
Debug.Log(sum);
```

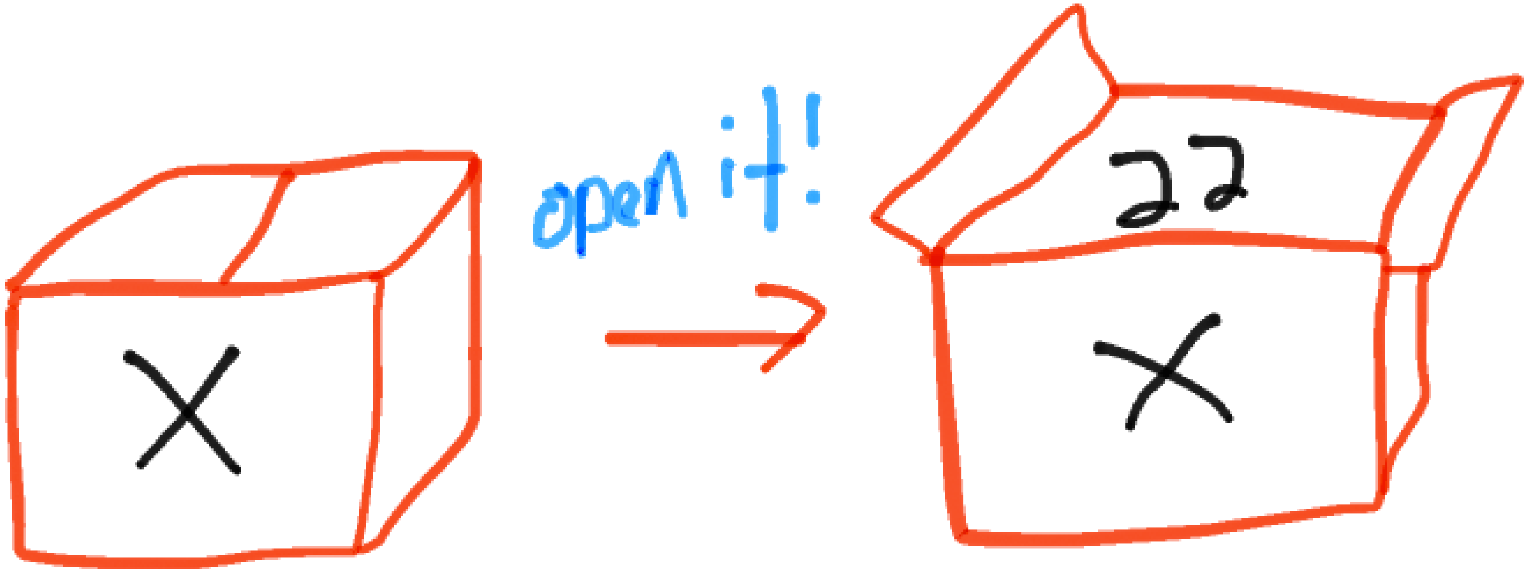
*Almost

```
00110001 00000000 00000000
00110001 00000001 00000001
00110011 00000001 00000010
01010001 00001011 00000010
00100010 00000010 00001000
01000011 00000001 00000000
01000001 00000001 00000001
00010000 00000010 00000000
01100010 00000000 00000000
```

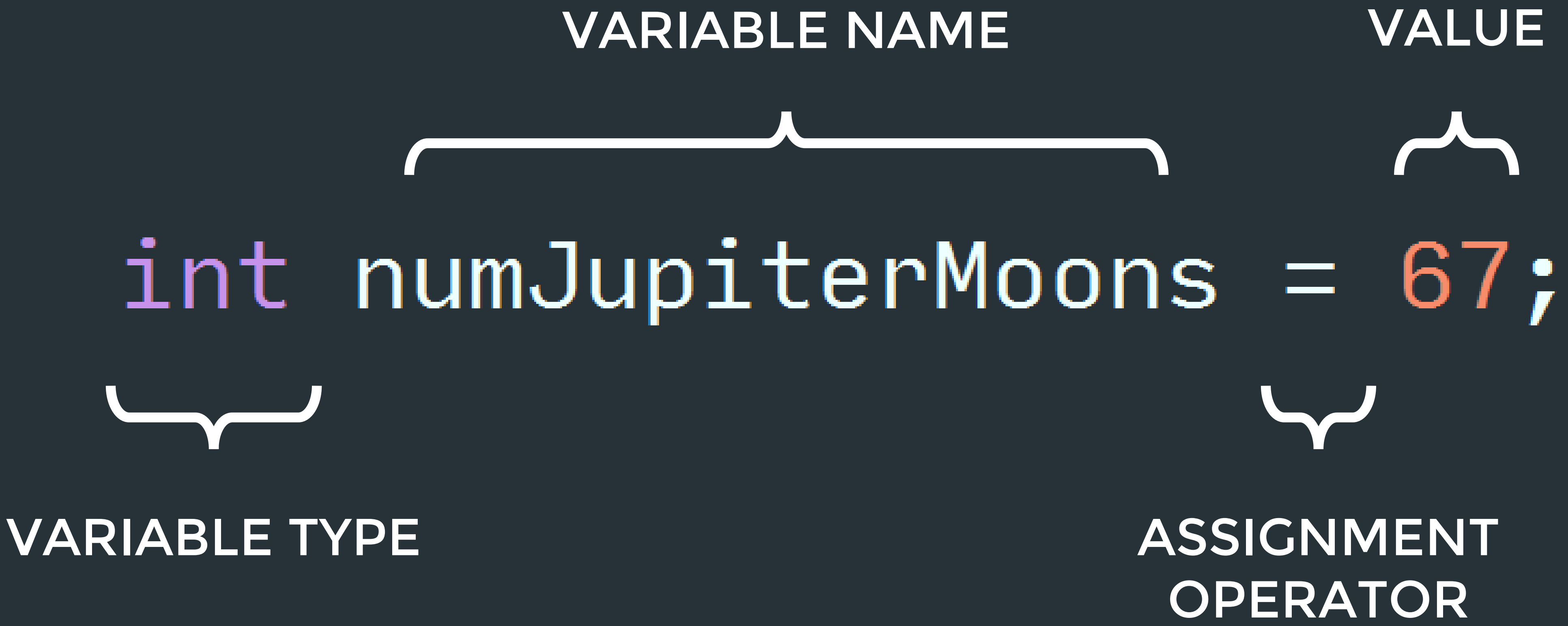
Variables

Storing Data

Named Boxes



http://cs-blog.khanacademy.org/2013/09/teaching-variables-analogies-and.html



VARIABLE NAME

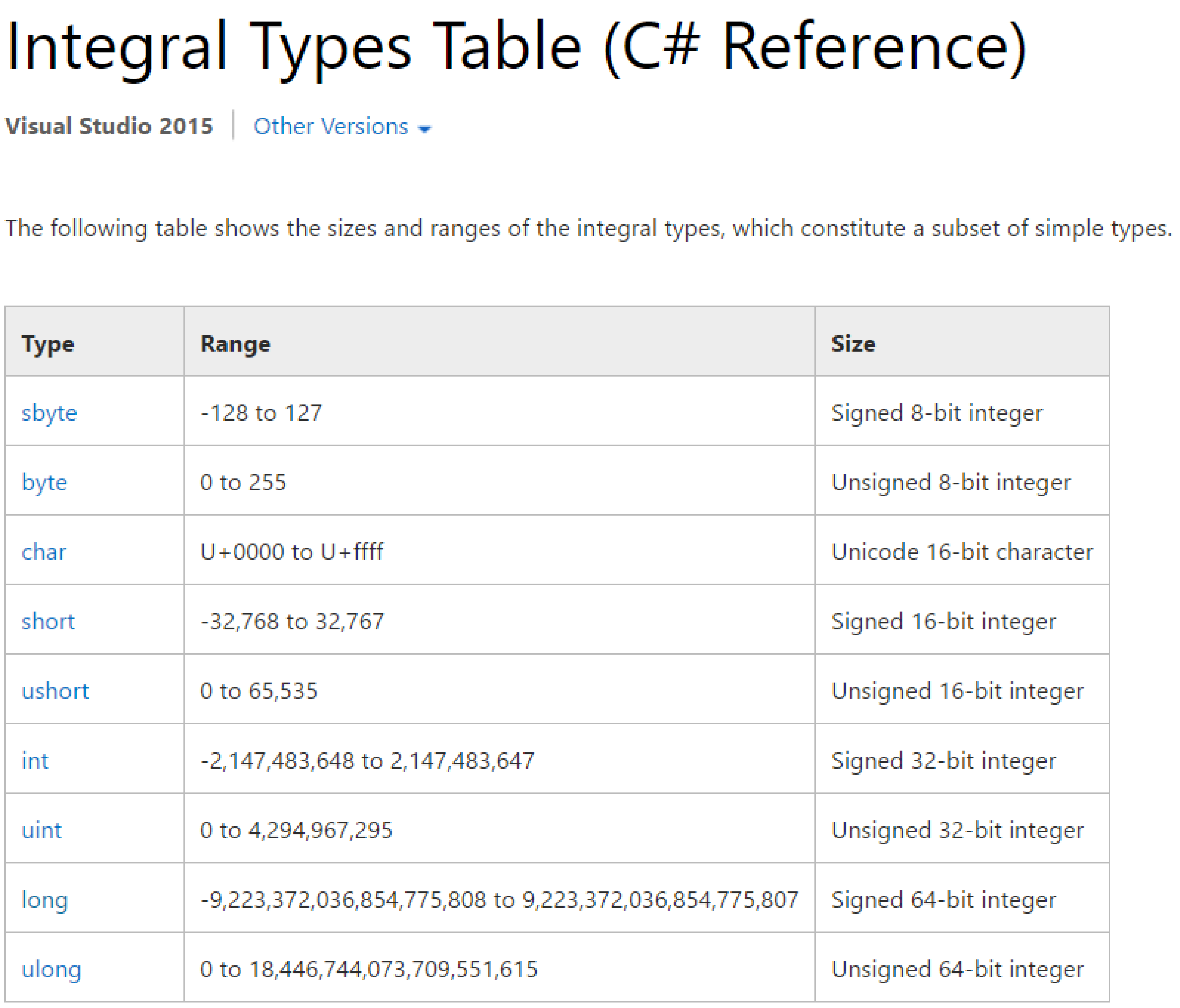
VALUE

```
int numJupiterMoons = 67;
```

VARIABLE TYPE

ASSIGNMENT OPERATOR

```
// Camel Case

// Good - short, descriptive
numJupiterMoons
materialColor
playerSpeed

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



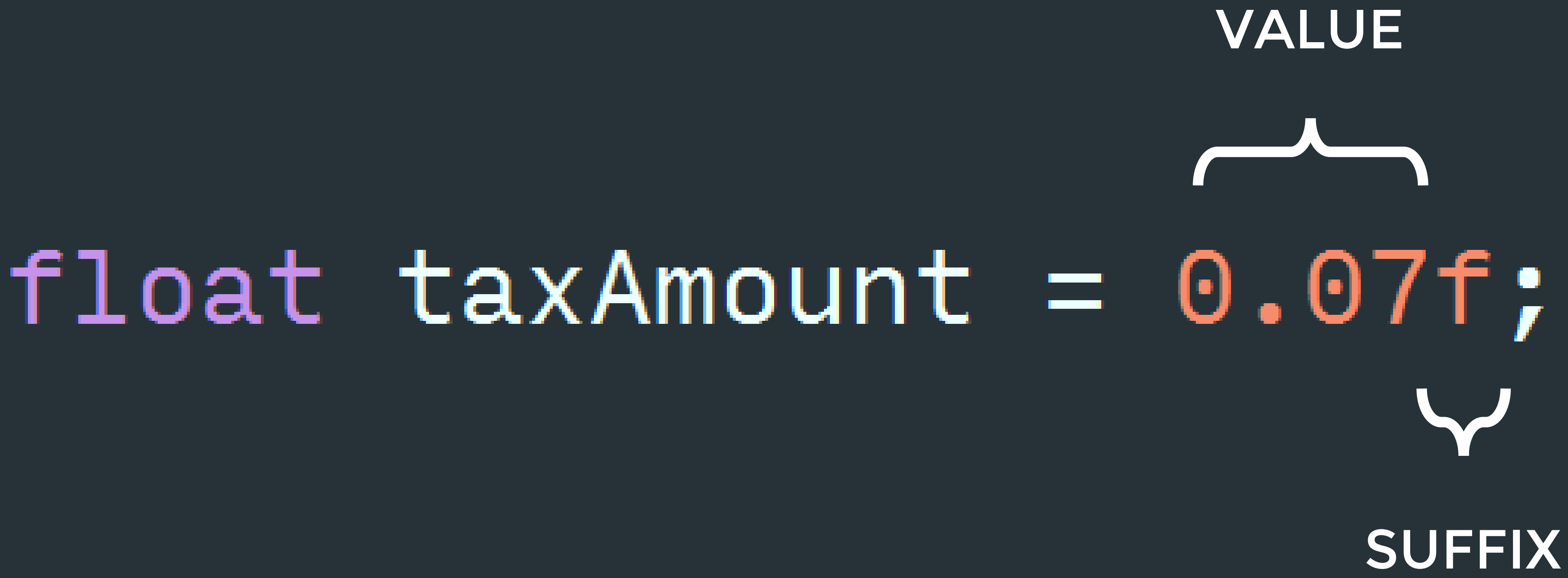
Integral Types Table (C# Reference)

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

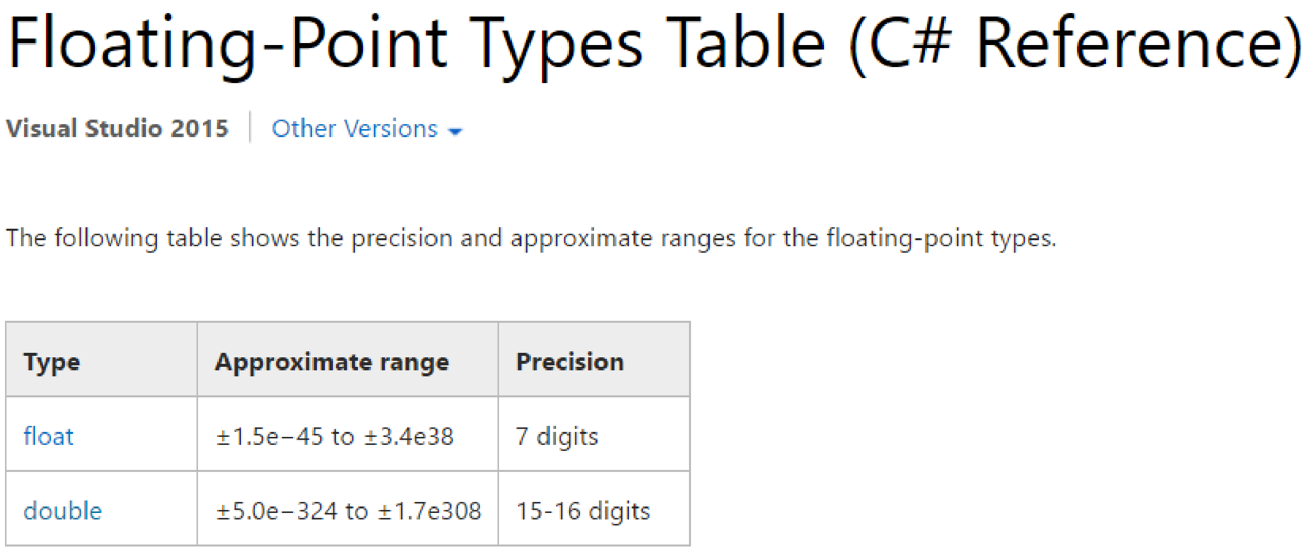
https://msdn.microsoft.com/en-us/library/exx3b86w.aspx



VALUE

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

SUFFIX

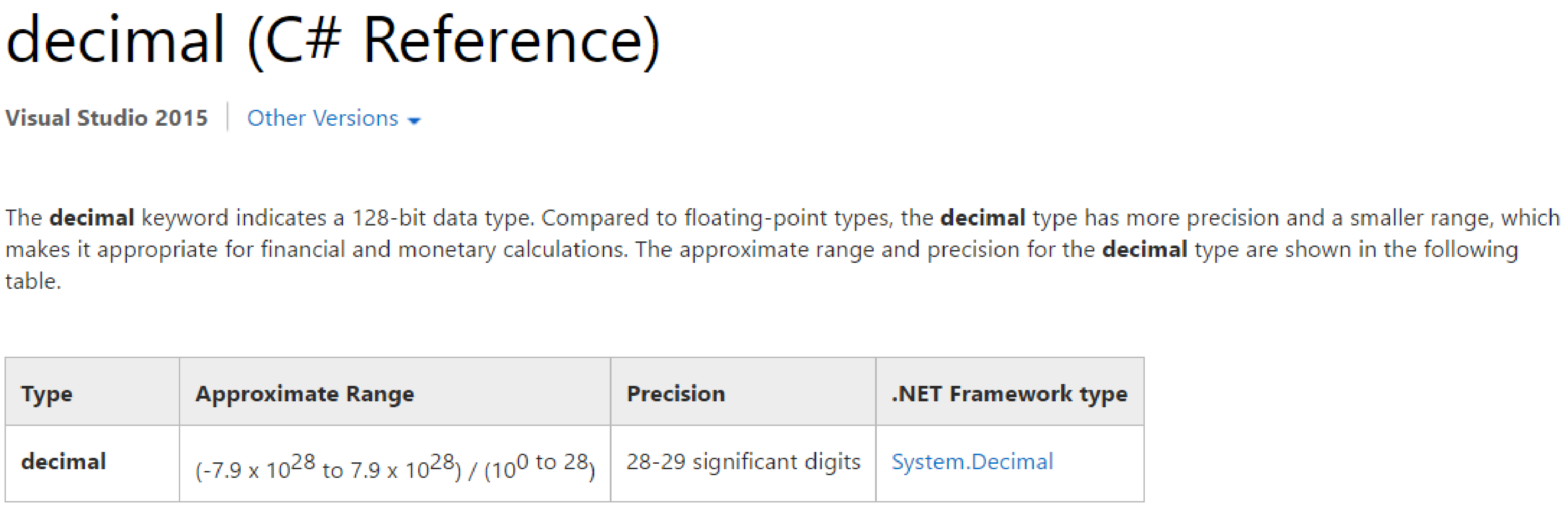


Floating-Point Types Table (C# Reference)

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 x 10^{28} to 7.9 x 10^{28}) / (10^{0} to 28)	28-29 significant digits	System.Decimal

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

START

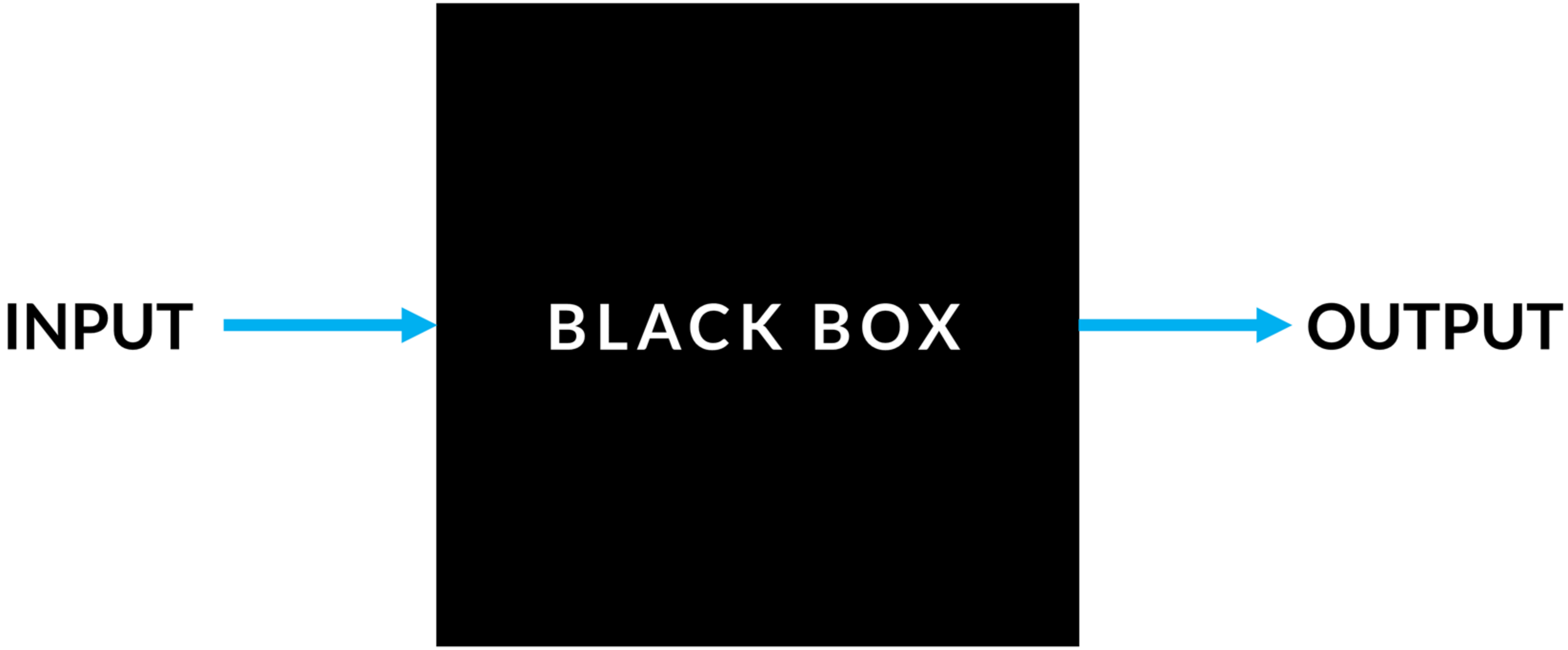
END

```
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

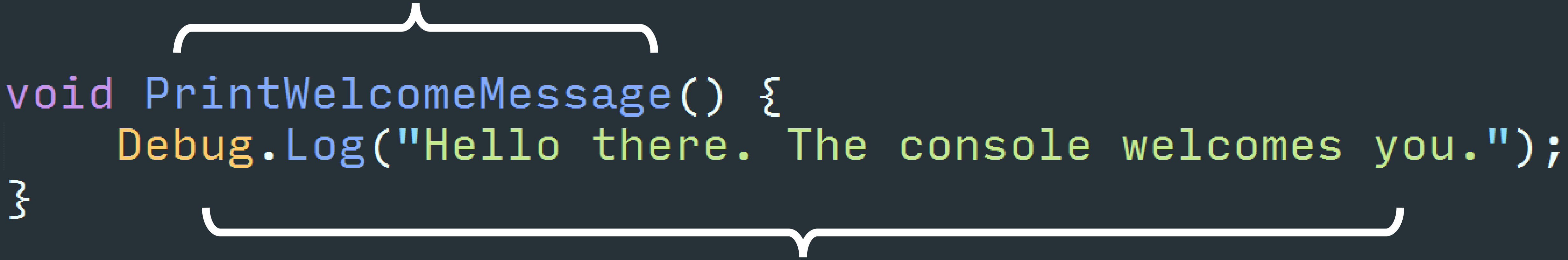


INPUT

BLACK BOX

OUTPUT

FUNCTION NAME



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

FUNCTION CONTENTS

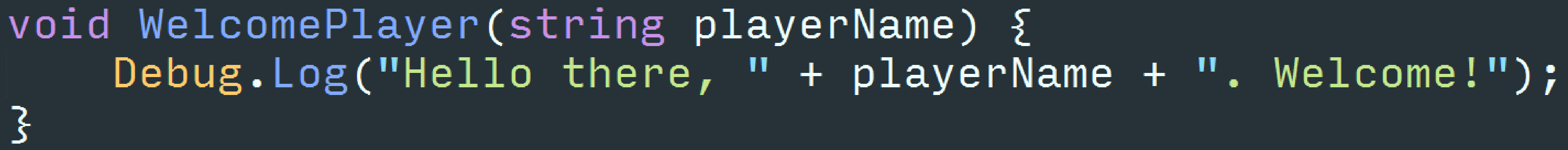
```
void Start() {
    PrintWelcomeMessage();
}
```

FUNCTION INVOCATION

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

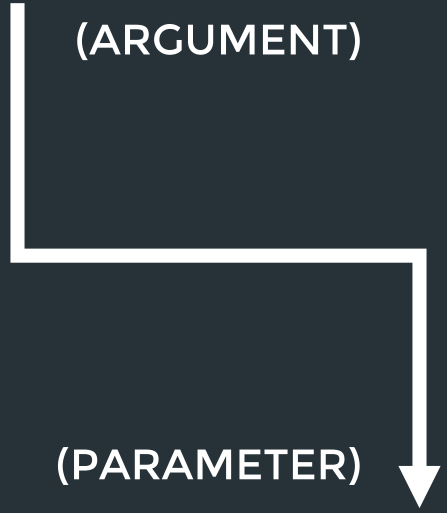
```
// Pascal Case

// 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!");
}
```

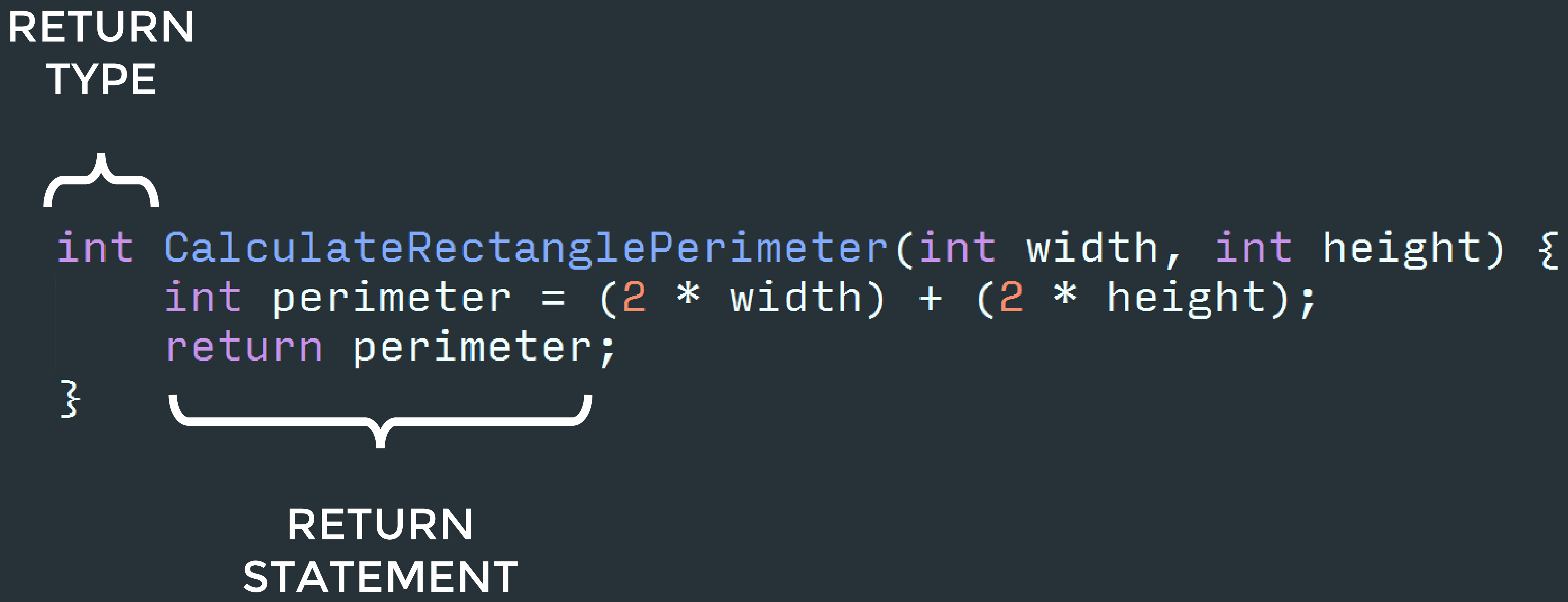
```
void Start() {
    WelcomePlayer("Mike");
}
```



(ARGUMENT)

(PARAMETER)

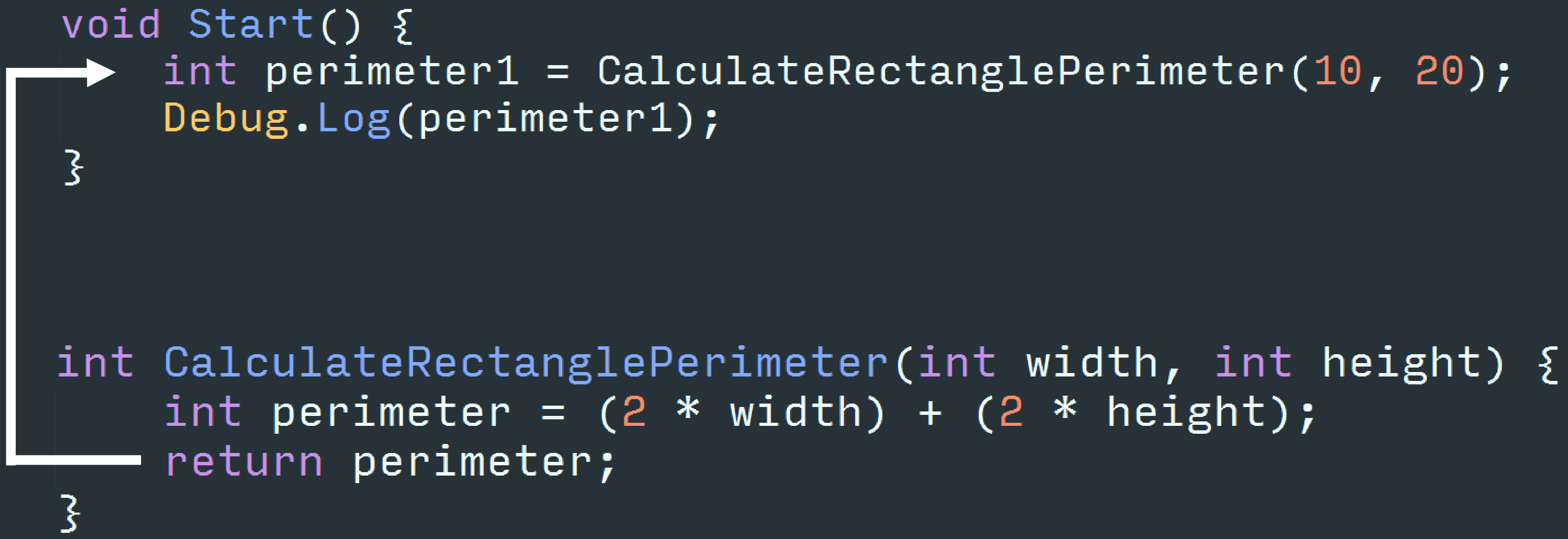
```
void WelcomePlayer(string playerName) {
    Debug.Log("Hello there, " + playerName + ". Welcome!");
}
```



RETURN
TYPE

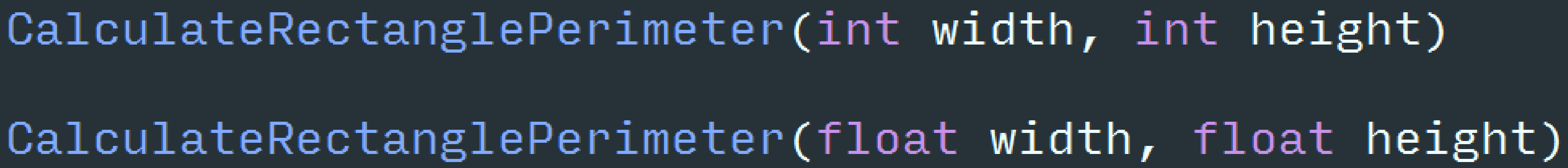
```
int CalculateRectanglePerimeter(int width, int height) {
    int perimeter = (2 * width) + (2 * height);
    return perimeter;
}
```

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



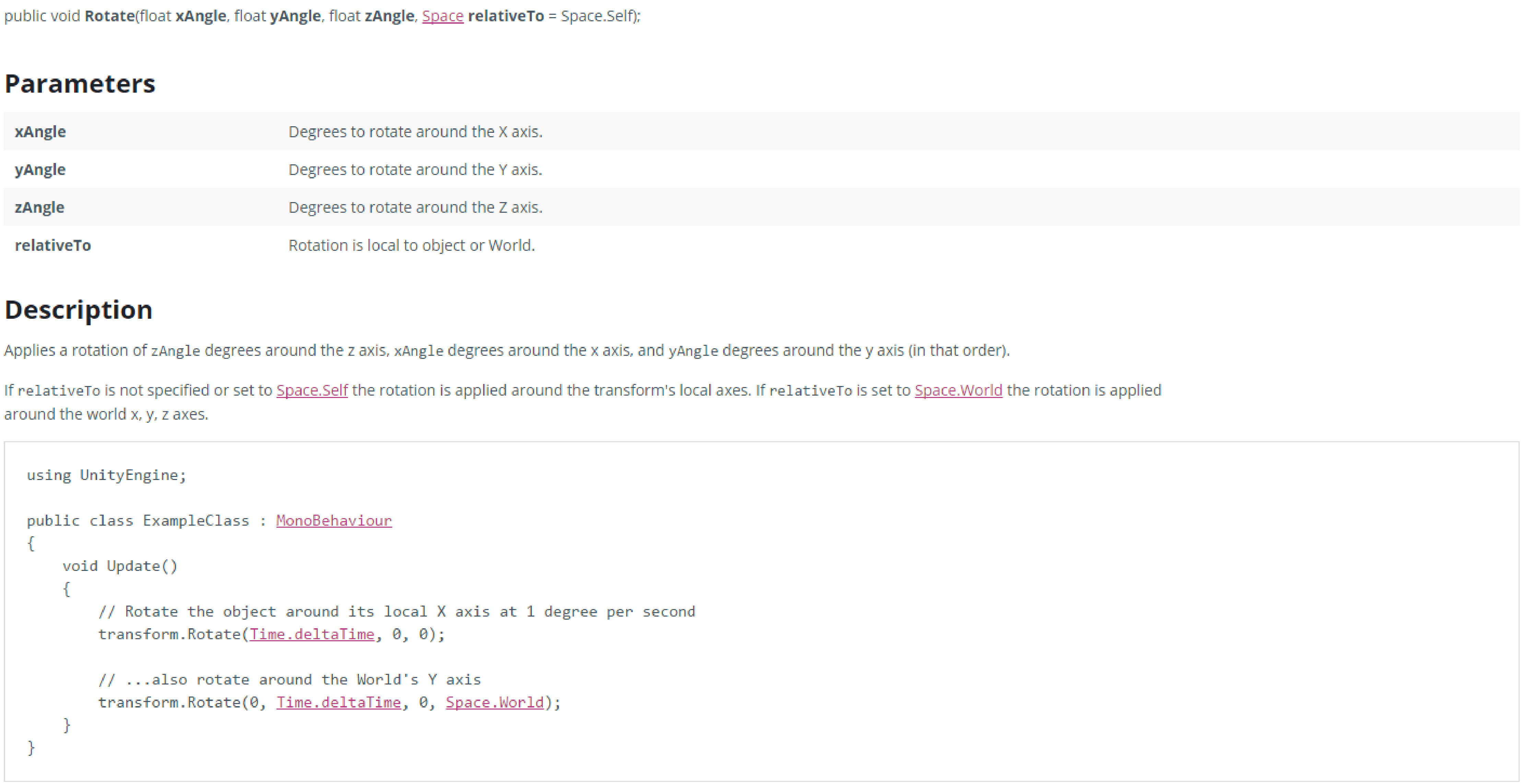
```
CalculateRectanglePerimeter(int width, int height)

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

(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.

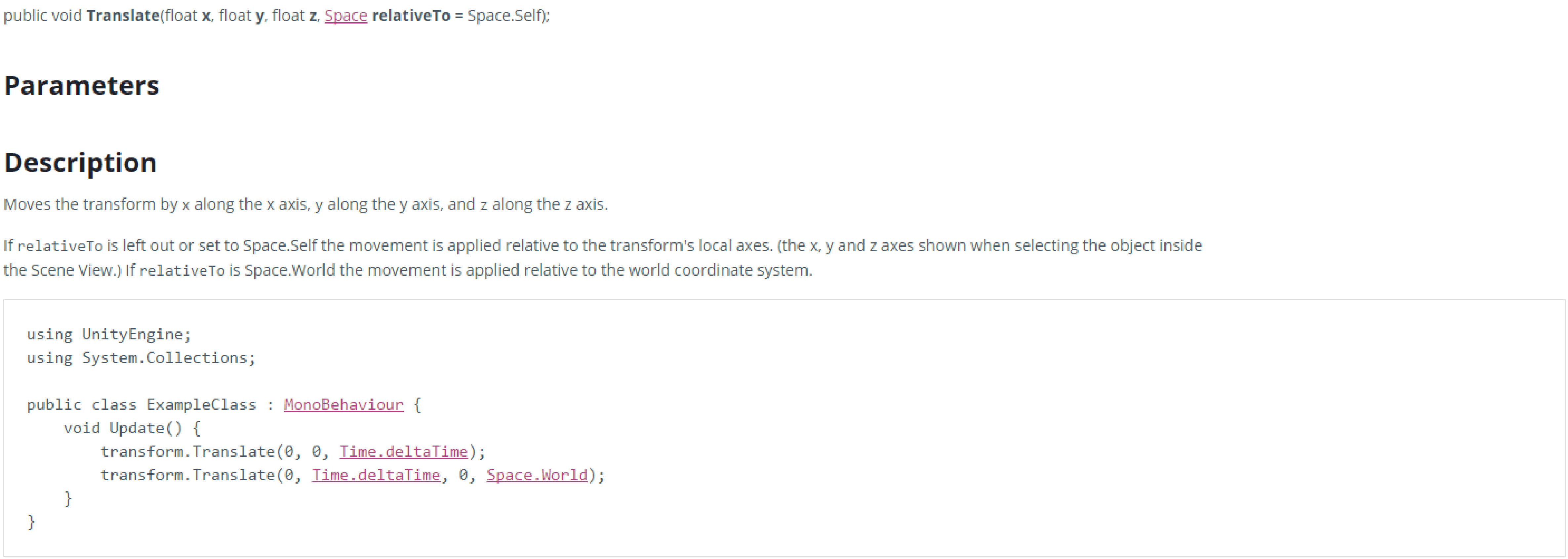
Description

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

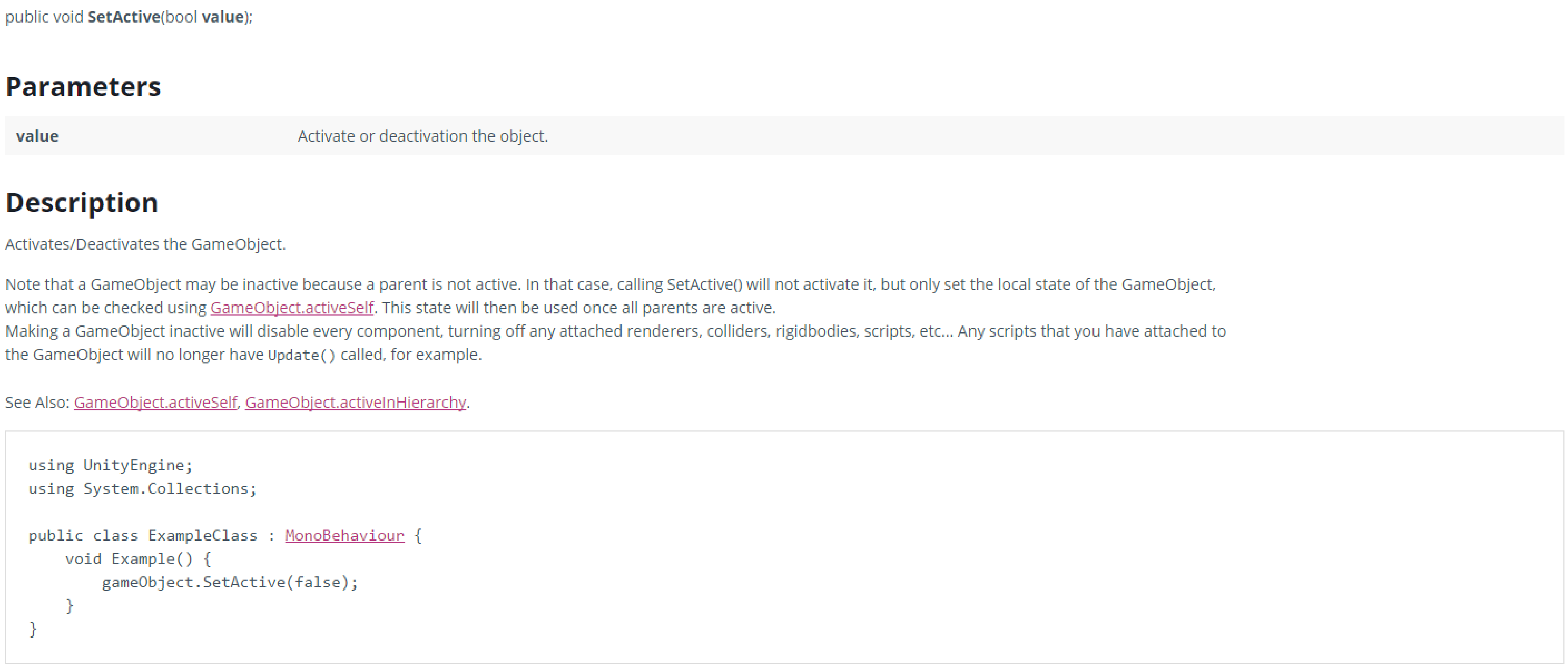
Description

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);
    }
}
```

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);
    }
}
```