Programming

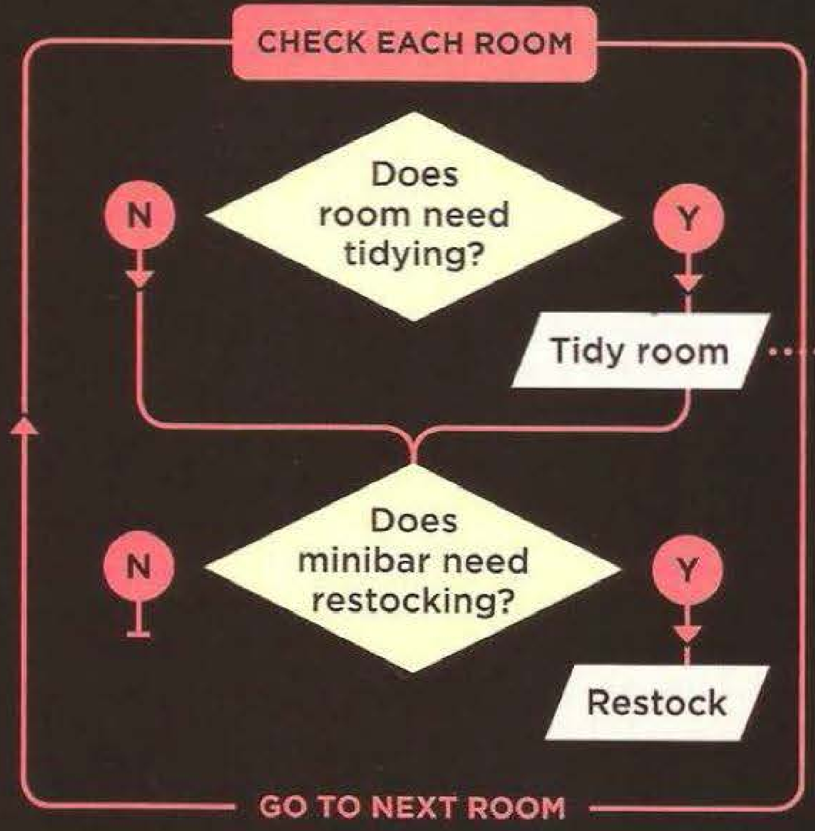
Programming?

```
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



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);
```

$\xrightarrow[\text{*Almost}]{}$

```
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
```

Programming Languages in Unity

C#

UnityScript

~~Boo~~ (~2014)

C#

~~UnityScript~~ (rumored?)

~~Boo~~ (~2014)

C#

(C Sharp)

Microsoft

Xamarin

IDE Setup: VS Code

Checklist

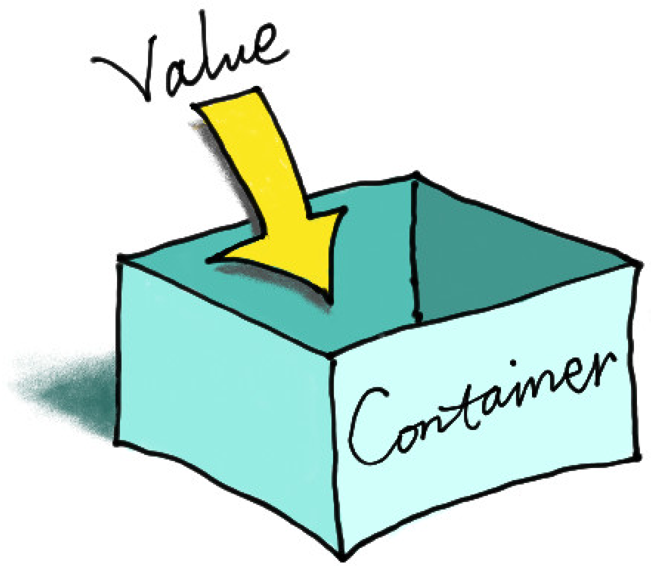
- Install VS Code
- Edit -> Preferences -> External Tools -> External Script Editor
  - Set to “Visual Studio Code”
- VS Code Extensions
  - “C#”
  - “Debugger for Unity”
  - (Optional) “Material-theme”

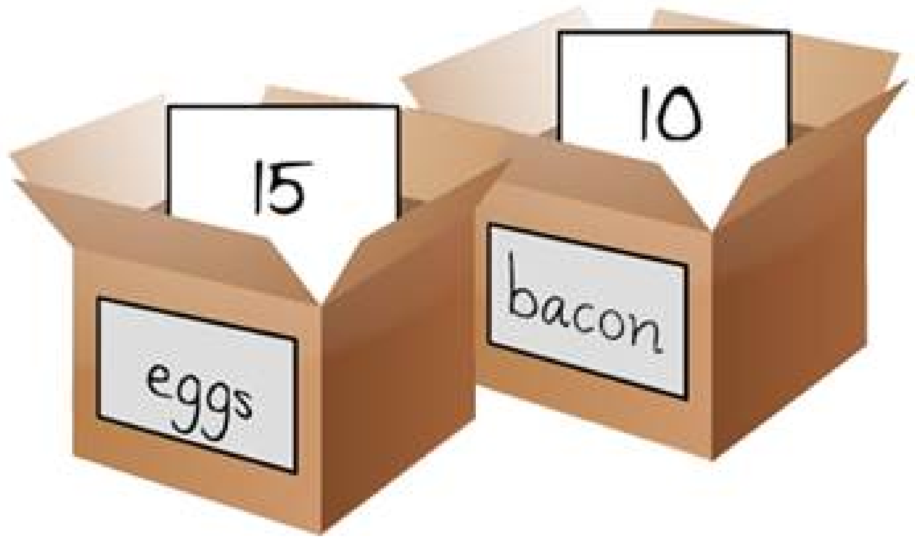
Logging Demo

Variables

Storing Data

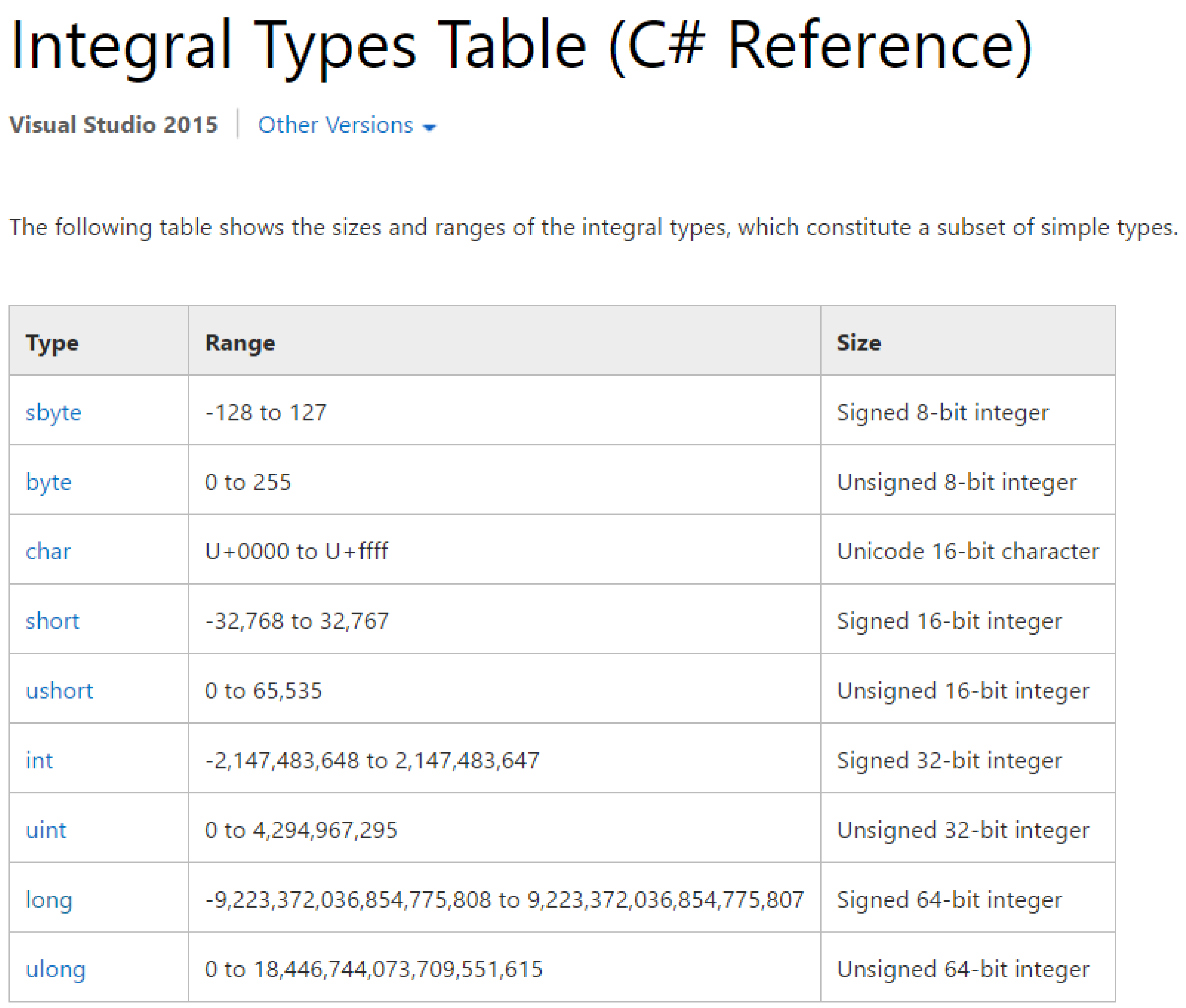
Named Boxes





$$\underbrace{\texttt{int}}_{\text{VARIABLE TYPE}}\ \overbrace{\texttt{numJupiterMoons}}^{\text{VARIABLE NAME}}\ \underbrace{\texttt{=}}_{\substack{\text{ASSIGNMENT}\\\text{OPERATOR}}}\ \overbrace{\texttt{67}}^{\text{VALUE}}\texttt{;}$$

```
// Camel Case

// Good - short, descriptive
numJupiterMoons
materialColor
playerSpeed

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



Integral Types Table (C# Reference)

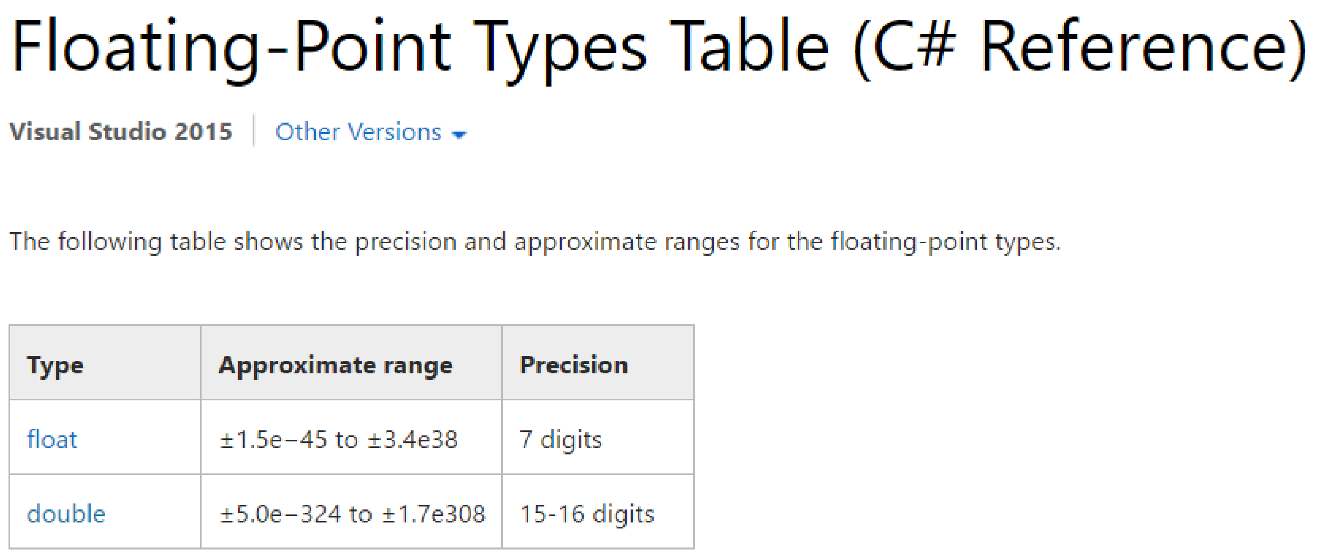
Visual Studio 2015 | Other Versions

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

$$\texttt{float taxAmount = }\overbrace{\texttt{0.07}}^{\text{VALUE}}\underbrace{\texttt{f}}_{\text{SUFFIX}}\texttt{;}$$

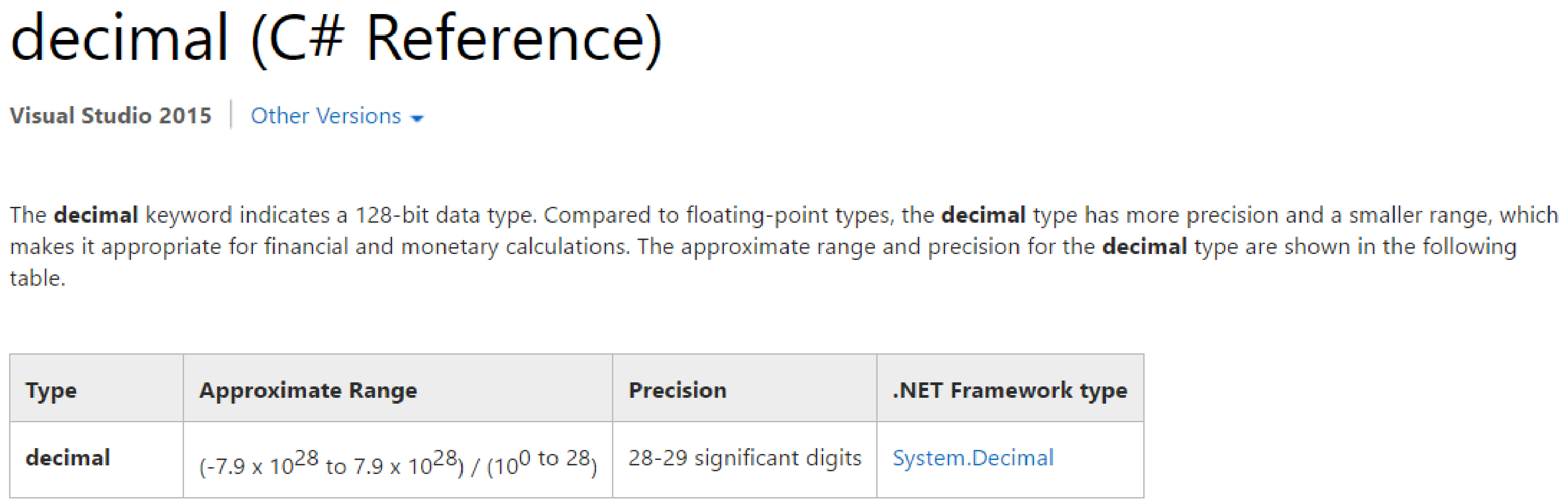


Floating-Point Types Table (C# Reference)

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

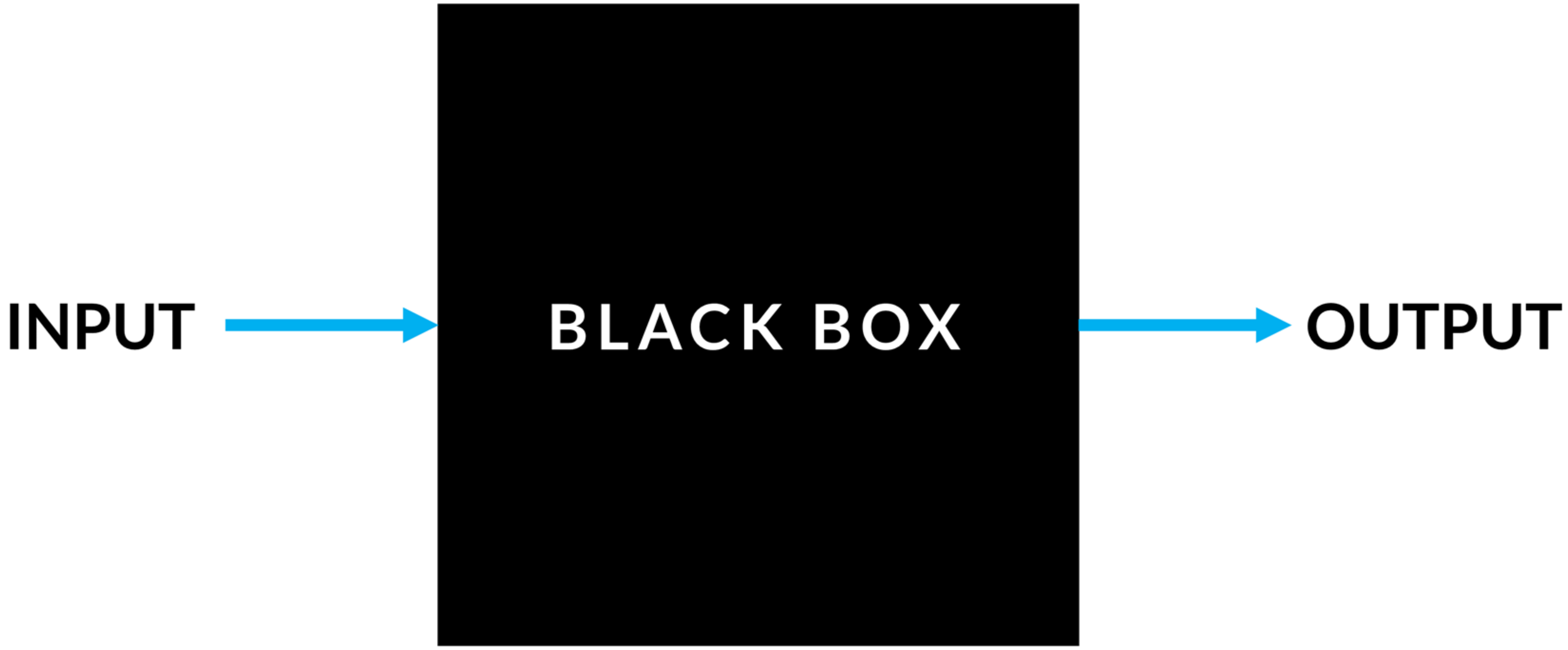
START END

```
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.
```