VARIABLES AND COMPONENTS

Topics

Topics

- Variables in C#

Topics

- **Variables in C#**
  - Declaring and defining variables

Topics

- Variables in C#
  - Declaring and defining variables
- Important C# Variable Types

Topics

- Variables in C#
  - Declaring and defining variables
- Important C# Variable Types
- Naming Conventions

Topics

- Variables in C#
  - Declaring and defining variables
- Important C# Variable Types
- Naming Conventions
- Important Unity Variable Types

Topics

- Variables in C#
  - Declaring and defining variables
- Important C# Variable Types
- Naming Conventions
- Important Unity Variable Types
- Unity GameObject Components

Variables in C#

Variables in C#

- Quick recap:

Variables in C#

- **Quick recap:**
  - **A *variable* is a named container for data**

Variables in C#

- **Quick recap:**
  - **A *variable* is a named container for data**
  - **Variables in C# are *typed*, so they can only hold one type of data (e.g., an integer, a float, a string)**

Variables in C#

- **Quick recap:**
  - A *variable* is a named container for data
  - Variables in C# are *typed*, so they can only hold one type of data (e.g., an integer, a float, a string)
  - Variables need to be *declared* to be used

Variables in C#

- **Quick recap:**
  - **A *variable* is a named container for data**
  - **Variables in C# are *typed*, so they can only hold one type of data (e.g., an integer, a float, a string)**
  - **Variables need to be *declared* to be used**
    - `int x;`

Variables in C#

- **Quick recap:**
  - A *variable* is a named container for data
  - Variables in C# are *typed*, so they can only hold one type of data (e.g., an integer, a float, a string)
  - Variables need to be *declared* to be used
    - `int x;`
  - Assigning a value to a variable is called *defining* the variable

Variables in C#

- **Quick recap:**
  - A *variable* is a named container for data
  - Variables in C# are *typed*, so they can only hold one type of data (e.g., an integer, a float, a string)
  - Variables need to be *declared* to be used
    - `int x;`
  - Assigning a value to a variable is called *defining* the variable
    - `x = 5;`

Variables in C#

- **Quick recap:**
  - A *variable* is a named container for data
  - Variables in C# are *typed*, so they can only hold one type of data (e.g., an integer, a float, a string)
  - Variables need to be *declared* to be used
    - `int x;`
  - Assigning a value to a variable is called *defining* the variable
    - `x = 5;`
  - A *literal* is a value that is entered into your code and can be assigned to a variable

Variables in C#

- **Quick recap:**
  - **A *variable* is a named container for data**
  - **Variables in C# are *typed*, so they can only hold one type of data (e.g., an integer, a float, a string)**
  - **Variables need to be *declared* to be used**
    - `int x;`
  - **Assigning a value to a variable is called *defining* the variable**
    - `x = 5;`
  - **A *literal* is a value that is entered into your code and can be assigned to a variable**
    - The 5 above is an integer literal

Variables in C#

- **Quick recap:**
  - **A *variable* is a named container for data**
  - **Variables in C# are *typed*, so they can only hold one type of data (e.g., an integer, a float, a string)**
  - **Variables need to be *declared* to be used**
    - `int x;`
  - **Assigning a value to a variable is called *defining* the variable**
    - `x = 5;`
  - **A *literal* is a value that is entered into your code and can be assigned to a variable**
    - The 5 above is an integer literal
    - string literals are surrounded by double quotes: `"Hello World!"`

Variables in C#

- **Quick recap:**
  - **A *variable* is a named container for data**
  - **Variables in C# are *typed*, so they can only hold one type of data (e.g., an integer, a float, a string)**
  - **Variables need to be *declared* to be used**
    - `int x;`
  - **Assigning a value to a variable is called *defining* the variable**
    - `x = 5;`
  - **A *literal* is a value that is entered into your code and can be assigned to a variable**
    - The 5 above is an integer literal
    - string literals are surrounded by double quotes: `"Hello World!"`
    - float literals are followed by an f: `3.14f`

Important C# Variable Types

Important C# Variable Types

- Core C# variable types start with a lowercase character

Important C# Variable Types

- Core C# variable types start with a lowercase character
  - bool

Important C# Variable Types

- Core C# variable types start with a lowercase character
  - bool
  - int

Important C# Variable Types

- Core C# variable types start with a lowercase character
  - bool
  - int
  - float

Important C# Variable Types

- Core C# variable types start with a lowercase character
  - bool
  - int
  - float
  - char

Important C# Variable Types

- Core C# variable types start with a lowercase character
  - bool
  - int
  - float
  - char
  - string

Important C# Variable Types

- Core C# variable types start with a lowercase character
  - bool
  - int
  - float
  - char
  - string
  - class

Important C# Variable Types

Important C# Variable Types

- bool – A 1-bit True or False Value

Important C# Variable Types

- **bool – A 1-bit True or False Value**
  - **Short for Boolean**

Important C# Variable Types

- **bool – A 1-bit True or False Value**
  - **Short for Boolean**
  - **Named after George Boole** (an English mathematician)

Important C# Variable Types

- **bool – A 1-bit True or False Value**
  - **Short for Boolean**
  - **Named after George Boole** (an English mathematician)
  - **bools in C# actually use more than 1-bit of space**

Important C# Variable Types

- **bool – A 1-bit True or False Value**
  - **Short for Boolean**
  - **Named after George Boole** (an English mathematician)
  - **bools in C# actually use more than 1-bit of space**
    - The smallest addressable memory chunk on a 32-bit system is 32 bits.

Important C# Variable Types

- **bool – A 1-bit True or False Value**
  - **Short for Boolean**
  - **Named after George Boole** (an English mathematician)
  - **bools in C# actually use more than 1-bit of space**
    - The smallest addressable memory chunk on a 32-bit system is 32 bits.
    - The smallest on a 64-bit system is 64 bits.

Important C# Variable Types

- **bool – A 1-bit True or False Value**
  - **Short for Boolean**
  - **Named after George Boole** (an English mathematician)
  - **bools in C# actually use more than 1-bit of space**
    - The smallest addressable memory chunk on a 32-bit system is 32 bits.
    - The smallest on a 64-bit system is 64 bits.
  - **Literal examples:** `true` `false`

Important C# Variable Types

- **bool – A 1-bit True or False Value**
  - **Short for Boolean**
  - **Named after George Boole** (an English mathematician)
  - **bools in C# actually use more than 1-bit of space**
    - The smallest addressable memory chunk on a 32-bit system is 32 bits.
    - The smallest on a 64-bit system is 64 bits.
  - **Literal examples:** `true` `false`
  - `bool verified = true;`

Important C# Variable Types

Important C# Variable Types

- int – A 32-bit Integer

Important C# Variable Types

- **int – A 32-bit Integer**
  - Stores a single integer number

Important C# Variable Types

- **int – A 32-bit Integer**
  - **Stores a single integer number**
    - Integers are numbers with no fractional or decimal element

Important C# Variable Types

- **int – A 32-bit Integer**
  - **Stores a single integer number**
    - Integers are numbers with no fractional or decimal element
  - **int math is very fast and accurate**

Important C# Variable Types

- **int – A 32-bit Integer**
  - **Stores a single integer number**
    - Integers are numbers with no fractional or decimal element
  - **int math is very fast and accurate**
  - **Can store numbers between –2,147,483,648 and 2,147,483,647**

Important C# Variable Types

- **int – A 32-bit Integer**
  - **Stores a single integer number**
    - Integers are numbers with no fractional or decimal element
  - **int math is very fast and accurate**
  - **Can store numbers between –2,147,483,648 and 2,147,483,647**
  - **31 bits used for number and 1 bit used for sign**

Important C# Variable Types

- **int – A 32-bit Integer**
  - **Stores a single integer number**
    - Integers are numbers with no fractional or decimal element
  - **int math is very fast and accurate**
  - **Can store numbers between –2,147,483,648 and 2,147,483,647**
  - **31 bits used for number and 1 bit used for sign**
  - **Literal examples:** `1` `34567` `-48198`

Important C# Variable Types

- **int – A 32-bit Integer**
  - **Stores a single integer number**
    - Integers are numbers with no fractional or decimal element
  - **int math is very fast and accurate**
  - **Can store numbers between –2,147,483,648 and 2,147,483,647**
  - **31 bits used for number and 1 bit used for sign**
  - **Literal examples:** `1` `34567` `-48198`
  - `int nonFractionalNumber = 12345;`

Important C# Variable Types

Important C# Variable Types

- float – A 32-bit Decimal Number

Important C# Variable Types

- **float – A 32-bit Decimal Number**
  - **Stores a floating-point number with a decimal element**

Important C# Variable Types

- **float – A 32-bit Decimal Number**
  - **Stores a floating-point number with a decimal element**
    - A floating-point number is stored in something like *scientific notation*

Important C# Variable Types

- **float – A 32-bit Decimal Number**
  - **Stores a floating-point number with a decimal element**
    - A floating-point number is stored in something like *scientific notation*
    - Scientific notation is numbers in the format **a*10^{b}**: 300 is 3*10^{2}

Important C# Variable Types

- **float – A 32-bit Decimal Number**
  - **Stores a floating-point number with a decimal element**
    - A floating-point number is stored in something like *scientific notation*
    - Scientific notation is numbers in the format **a*10^{b}**: 300 is 3*10^{2}
  - **Floating-point numbers are stored in the format a*2^{b}**

Important C# Variable Types

- **float – A 32-bit Decimal Number**
  - **Stores a floating-point number with a decimal element**
    - A floating-point number is stored in something like *scientific notation*
    - Scientific notation is numbers in the format **a*10^{b}**: 300 is 3*10^{2}
  - **Floating-point numbers are stored in the format a*2^{b}**
    - 23 bits are used for the significand (the **a** part)

Important C# Variable Types

- **float – A 32-bit Decimal Number**
  - **Stores a floating-point number with a decimal element**
    - A floating-point number is stored in something like *scientific notation*
    - Scientific notation is numbers in the format **a*10^{b}**: 300 is 3*10^{2}
  - **Floating-point numbers are stored in the format a*2^{b}**
    - 23 bits are used for the significand (the **a** part)
    - 8 bits are used for the exponent (the ^{b} part)

Important C# Variable Types

- **float – A 32-bit Decimal Number**
  - **Stores a floating-point number with a decimal element**
    - A floating-point number is stored in something like *scientific notation*
    - Scientific notation is numbers in the format **a*10^{b}**: 300 is 3*10^{2}
  - **Floating-point numbers are stored in the format a*2^{b}**
    - 23 bits are used for the significand (the **a** part)
    - 8 bits are used for the exponent (the ^{b} part)
    - 1 bit determines whether the number is positive or negative

Important C# Variable Types

- **float – A 32-bit Decimal Number**
  - **Stores a floating-point number with a decimal element**
    - A floating-point number is stored in something like *scientific notation*
    - Scientific notation is numbers in the format **a*10^{b}**: 300 is 3*10^{2}
  - **Floating-point numbers are stored in the format a*2^{b}**
    - 23 bits are used for the significand (the **a** part)
    - 8 bits are used for the exponent (the ^{b} part)
    - 1 bit determines whether the number is positive or negative
  - **Floats are *inaccurate* for large numbers and for numbers between -1 and 1**

Important C# Variable Types

- **float – A 32-bit Decimal Number**
  - **Stores a floating-point number with a decimal element**
    - A floating-point number is stored in something like *scientific notation*
    - Scientific notation is numbers in the format **a*10^{b}**: 300 is 3*10^{2}
  - **Floating-point numbers are stored in the format a*2^{b}**
    - 23 bits are used for the significand (the **a** part)
    - 8 bits are used for the exponent (the ^{b} part)
    - 1 bit determines whether the number is positive or negative
  - **Floats are *inaccurate* for large numbers and for numbers between -1 and 1**
    - There is no accurate float representation for **1 / 3**

Important C# Variable Types

- **float – A 32-bit Decimal Number**
  - **Stores a floating-point number with a decimal element**
    - A floating-point number is stored in something like *scientific notation*
    - Scientific notation is numbers in the format **a*10^{b}**: 300 is 3*10^{2}
  - **Floating-point numbers are stored in the format a*2^{b}**
    - 23 bits are used for the significand (the **a** part)
    - 8 bits are used for the exponent (the ^{b} part)
    - 1 bit determines whether the number is positive or negative
  - **Floats are *inaccurate* for large numbers and for numbers between -1 and 1**
    - There is no accurate float representation for **1 / 3**
  - **Literal examples:** `3.14f` `123f` `123.456f`

Important C# Variable Types

- **float – A 32-bit Decimal Number**
  - **Stores a floating-point number with a decimal element**
    - A floating-point number is stored in something like *scientific notation*
    - Scientific notation is numbers in the format **a*10^{b}**: 300 is 3*10^{2}
  - **Floating-point numbers are stored in the format a*2^{b}**
    - 23 bits are used for the significand (the **a** part)
    - 8 bits are used for the exponent (the ^{b} part)
    - 1 bit determines whether the number is positive or negative
  - **Floats are *inaccurate* for large numbers and for numbers between -1 and 1**
    - There is no accurate float representation for **1 / 3**
  - **Literal examples:** `3.14f` `123f` `123.456f`
  - `float notPreciselyOneThird = 1.0f / 3.0f;`

Important C# Variable Types

Important C# Variable Types

- char – A 16-bit Character

Important C# Variable Types

- **char – A 16-bit Character**
  - Single character represented by 16 bits of information

Important C# Variable Types

- **char – A 16-bit Character**
  - Single character represented by 16 bits of information
  - Uses Unicode values for the characters

Important C# Variable Types

- **char – A 16-bit Character**
  - **Single character represented by 16 bits of information**
  - **Uses Unicode values for the characters**
    - Unicode represents 110,000 different characters from over 100 different character sets and languages

Important C# Variable Types

- **char – A 16-bit Character**
  - **Single character represented by 16 bits of information**
  - **Uses Unicode values for the characters**
    - Unicode represents 110,000 different characters from over 100 different character sets and languages
  - **Floats are *inaccurate* for large numbers and for numbers between -1 and 1**

Important C# Variable Types

- **char – A 16-bit Character**
  - **Single character represented by 16 bits of information**
  - **Uses Unicode values for the characters**
    - Unicode represents 110,000 different characters from over 100 different character sets and languages
  - **Floats are *inaccurate* for large numbers and for numbers between -1 and 1**
    - There is no accurate float representation for **1 / 3**

Important C# Variable Types

- **char – A 16-bit Character**
  - **Single character represented by 16 bits of information**
  - **Uses Unicode values for the characters**
    - Unicode represents 110,000 different characters from over 100 different character sets and languages
  - **Floats are *inaccurate* for large numbers and for numbers between -1 and 1**
    - There is no accurate float representation for **1 / 3**
  - **Uppercase and lowercase letters are different values!**

Important C# Variable Types

- **char – A 16-bit Character**
  - **Single character represented by 16 bits of information**
  - **Uses Unicode values for the characters**
    - Unicode represents 110,000 different characters from over 100 different character sets and languages
  - **Floats are *inaccurate* for large numbers and for numbers between -1 and 1**
    - There is no accurate float representation for **1 / 3**
  - **Uppercase and lowercase letters are different values!**
  - **char literals are surrounded by single quotes**

Important C# Variable Types

- **char – A 16-bit Character**
  - **Single character represented by 16 bits of information**
  - **Uses Unicode values for the characters**
    - Unicode represents 110,000 different characters from over 100 different character sets and languages
  - **Floats are *inaccurate* for large numbers and for numbers between -1 and 1**
    - There is no accurate float representation for **1 / 3**
  - **Uppercase and lowercase letters are different values!**
  - **char literals are surrounded by single quotes**
  - **Literal examples:** `'A'` `'a'` `'\t'`

Important C# Variable Types

- **char – A 16-bit Character**
  - **Single character represented by 16 bits of information**
  - **Uses Unicode values for the characters**
    - Unicode represents 110,000 different characters from over 100 different character sets and languages
  - **Floats are *inaccurate* for large numbers and for numbers between -1 and 1**
    - There is no accurate float representation for **1 / 3**
  - **Uppercase and lowercase letters are different values!**
  - **char literals are surrounded by single quotes**
  - **Literal examples:** `'A'` `'a'` `'\t'`
  - `char theLetterA = 'A';`

Important C# Variable Types

Important C# Variable Types

- string – A Series of 16-bit Characters

Important C# Variable Types

- **string – A Series of 16-bit Characters**
  - Stores from no characters ("") to an entire novel

Important C# Variable Types

- **string – A Series of 16-bit Characters**
  - **Stores from no characters ("") to an entire novel**
    - Max length is 2 billion chars; 12,000 times the length of Hamlet

Important C# Variable Types

- **string – A Series of 16-bit Characters**
  - **Stores from no characters ("") to an entire novel**
    - Max length is 2 billion chars; 12,000 times the length of Hamlet
  - **string literals are surrounded by double quotes**

Important C# Variable Types

- **string – A Series of 16-bit Characters**
  - **Stores from no characters ("") to an entire novel**
    - Max length is 2 billion chars; 12,000 times the length of Hamlet
  - **string literals are surrounded by double quotes**
  - **Literal examples:** `"Hello"` `""` `"\tTab"`

Important C# Variable Types

- **string – A Series of 16-bit Characters**
  - **Stores from no characters ("") to an entire novel**
    - Max length is 2 billion chars; 12,000 times the length of Hamlet
  - **string literals are surrounded by double quotes**
  - **Literal examples:** `"Hello"` `""` `"\tTab"`
  - `string theFirstLineOfHamlet = "Who's there?";`

Important C# Variable Types

- **string – A Series of 16-bit Characters**
  - **Stores from no characters ("") to an entire novel**
    - Max length is 2 billion chars; 12,000 times the length of Hamlet
  - **string literals are surrounded by double quotes**
  - **Literal examples:** `"Hello"` `""` `"\tTab"`
  - `string theFirstLineOfHamlet = "Who's there?";`
  - **You can access individual characters via *bracket access***

Important C# Variable Types

- **string – A Series of 16-bit Characters**
  - **Stores from no characters ("") to an entire novel**
    - Max length is 2 billion chars; 12,000 times the length of Hamlet
  - **string literals are surrounded by double quotes**
  - **Literal examples:** `"Hello"` `""` `"\tTab"`
  - `string theFirstLineOfHamlet = "Who's there?";`
  - **You can access individual characters via *bracket access***
    - `char theCharW = theFirstLineOfHamlet[0];`

Important C# Variable Types

- **string – A Series of 16-bit Characters**
  - **Stores from no characters ("") to an entire novel**
    - Max length is 2 billion chars; 12,000 times the length of Hamlet
  - **string literals are surrounded by double quotes**
  - **Literal examples:** `"Hello"` `""` `"\tTab"`
  - `string theFirstLineOfHamlet = "Who's there?";`
  - **You can access individual characters via *bracket access***
    - `char theCharW = theFirstLineOfHamlet[0];`
    - `char questionMark = theFirstLineOfHamlet[11];`

Important C# Variable Types

- **string – A Series of 16-bit Characters**
  - **Stores from no characters ("") to an entire novel**
    - Max length is 2 billion chars; 12,000 times the length of Hamlet
  - **string literals are surrounded by double quotes**
  - **Literal examples:** `"Hello"` `""` `"\tTab"`
  - `string theFirstLineOfHamlet = "Who's there?";`
  - **You can access individual characters via *bracket access***
    - `char theCharW = theFirstLineOfHamlet[0];`
    - `char questionMark = theFirstLineOfHamlet[11];`
  - **The length of a string is accessed via .Length**

Important C# Variable Types

- **string – A Series of 16-bit Characters**
  - **Stores from no characters ("") to an entire novel**
    - Max length is 2 billion chars; 12,000 times the length of Hamlet
  - **string literals are surrounded by double quotes**
  - **Literal examples:** `"Hello"` `""` `"\tTab"`
  - `string theFirstLineOfHamlet = "Who's there?";`
  - **You can access individual characters via *bracket access***
    - `char theCharW = theFirstLineOfHamlet[0];`
    - `char questionMark = theFirstLineOfHamlet[11];`
  - **The length of a string is accessed via .Length**
    - `int len = theFirstLineOfHamlet.Length;`

Important C# Variable Types

- **string – A Series of 16-bit Characters**
  - **Stores from no characters ("") to an entire novel**
    - Max length is 2 billion chars; 12,000 times the length of Hamlet
  - **string literals are surrounded by double quotes**
  - **Literal examples:** `"Hello"` `""` `"\tTab"`
  - `string theFirstLineOfHamlet = "Who's there?";`
  - **You can access individual characters via *bracket access***
    - `char theCharW = theFirstLineOfHamlet[0];`
    - `char questionMark = theFirstLineOfHamlet[11];`
  - **The length of a string is accessed via .Length**
    - `int len = theFirstLineOfHamlet.Length;`
      - Sets `len` to `12`

Important C# Variable Types

Important C# Variable Types

- class – A Collection of Functions and Data

Important C# Variable Types

- class – A Collection of Functions and Data
  - A class creates a new variable type

Important C# Variable Types

- **class – A Collection of Functions and Data**
  - A class creates a new variable type
  - Covered extensively in Chapter 25, "Classes"

Important C# Variable Types

- **class – A Collection of Functions and Data**
  - **A class creates a new variable type**
  - **Covered extensively in Chapter 25, "Classes"**
  - **Already used in the HelloWorld project**

```
public class HelloWorld : MonoBehaviour {
        void Start() {
                print("Hello World!");
        }
}
```

Important C# Variable Types

- **class – A Collection of Functions and Data**
  - **A class creates a new variable type**
  - **Covered extensively in Chapter 25, "Classes"**
  - **Already used in the HelloWorld project**

```
public class HelloWorld : MonoBehaviour {
        void Start() {
                print("Hello World!");
        }
}
```

  - **Everything between the braces { } is part of the class**

C# Naming Conventions

C# Naming Conventions

- Use camelCase for almost everything

C# Naming Conventions

- Use camelCase for almost everything
- Variable names start with lowercase:

C# Naming Conventions

- Use camelCase for almost everything
- Variable names start with lowercase:
  - `thisVariable` `anotherVariable` `bob`

C# Naming Conventions

- Use camelCase for almost everything
- Variable names start with lowercase:
  - `thisVariable` `anotherVariable` `bob`
- Function names start with uppercase:

C# Naming Conventions

- Use camelCase for almost everything
- Variable names start with lowercase:
  - `thisVariable` `anotherVariable` `bob`
- Function names start with uppercase:
  - `ThatFunction()` `Start()` `Update()`

C# Naming Conventions

- Use camelCase for almost everything
- Variable names start with lowercase:
  - `thisVariable` `anotherVariable` `bob`
- Function names start with uppercase:
  - `ThatFunction()` `Start()` `Update()`
- Class names start with uppercase:

C# Naming Conventions

- Use camelCase for almost everything
- Variable names start with lowercase:
  - `thisVariable` `anotherVariable` `bob`
- Function names start with uppercase:
  - `ThatFunction()` `Start()` `Update()`
- Class names start with uppercase:
  - `SomeClass` `GameObject` `HeroShip`

C# Naming Conventions

- Use camelCase for almost everything
- Variable names start with lowercase:
  - `thisVariable` `anotherVariable` `bob`
- Function names start with uppercase:
  - `ThatFunction()` `Start()` `Update()`
- Class names start with uppercase:
  - `SomeClass` `GameObject` `HeroShip`
- Private variables start with underscore:

C# Naming Conventions

- Use camelCase for almost everything
- Variable names start with lowercase:
  - `thisVariable` `anotherVariable` `bob`
- Function names start with uppercase:
  - `ThatFunction()` `Start()` `Update()`
- Class names start with uppercase:
  - `SomeClass` `GameObject` `HeroShip`
- Private variables start with underscore:
  - `_privateVariable` `_hiddenVariable`

C# Naming Conventions

- Use camelCase for almost everything
- Variable names start with lowercase:
  - `thisVariable` `anotherVariable` `bob`
- Function names start with uppercase:
  - `ThatFunction()` `Start()` `Update()`
- Class names start with uppercase:
  - `SomeClass` `GameObject` `HeroShip`
- Private variables start with underscore:
  - `_privateVariable` `_hiddenVariable`
- Static variables use SNAKE_CASE:

C# Naming Conventions

- Use camelCase for almost everything
- Variable names start with lowercase:
  - `thisVariable` `anotherVariable` `bob`
- Function names start with uppercase:
  - `ThatFunction()` `Start()` `Update()`
- Class names start with uppercase:
  - `SomeClass` `GameObject` `HeroShip`
- Private variables start with underscore:
  - `_privateVariable` `_hiddenVariable`
- Static variables use SNAKE_CASE:
  - `STATIC_VAR` `NUM_INSTANCES`

Important Unity Variable Types

Important Unity Variable Types

- Because they are classes, important Unity variable types all start with an uppercase character

Important Unity Variable Types

- Because they are classes, important Unity variable types all start with an uppercase character
  - Vector3

Important Unity Variable Types

- Because they are classes, important Unity variable types all start with an uppercase character
  - Vector3
  - Color

Important Unity Variable Types

- Because they are classes, important Unity variable types all start with an uppercase character
  - Vector3
  - Color
  - Quaternion

Important Unity Variable Types

- Because they are classes, important Unity variable types all start with an uppercase character
  - Vector3
  - Color
  - Quaternion
  - Mathf

Important Unity Variable Types

- Because they are classes, important Unity variable types all start with an uppercase character
  - Vector3
  - Color
  - Quaternion
  - Mathf
  - Screen

Important Unity Variable Types

- Because they are classes, important Unity variable types all start with an uppercase character
  - Vector3
  - Color
  - Quaternion
  - Mathf
  - Screen
  - SystemInfo

Important Unity Variable Types

- Because they are classes, important Unity variable types all start with an uppercase character
  - Vector3
  - Color
  - Quaternion
  - Mathf
  - Screen
  - SystemInfo
  - GameObject

Important Unity Variable Types

Important Unity Variable Types

- Vector3 – A collection of 3 floats

Important Unity Variable Types

- **Vector3 – A collection of 3 floats**
  - **Used for position of objects in 3D**

Important Unity Variable Types

- **Vector3 – A collection of 3 floats**
  - **Used for position of objects in 3D**

```
Vector3 vec = new Vector3( 3, 4, 0 );
```

Important Unity Variable Types

- **Vector3 – A collection of 3 floats**
  - **Used for position of objects in 3D**

```
Vector3 vec = new Vector3( 3, 4, 0 );
```

  - **Instance variables and functions**

`vec.x` – The x component of the vector

`vec.y` – The y component of the vector

`vec.z` – The z component of the vector

`vec.magnitude` – The length of the vector

`vec.Normalize()` – New Vector3 in the same direction at unit length

Important Unity Variable Types

- **Vector3 – A collection of 3 floats**
  - **Used for position of objects in 3D**

```
Vector3 vec = new Vector3( 3, 4, 0 );
```

  - **Instance variables and functions**

`vec.x` – The x component of the vector

`vec.y` – The y component of the vector

`vec.z` – The z component of the vector

`vec.magnitude` – The length of the vector

`vec.Normalize()` – New Vector3 in the same direction at unit length

  - **Static class variables and functions**

Important Unity Variable Types

- **Vector3 – A collection of 3 floats**
  - **Used for position of objects in 3D**

```
Vector3 vec = new Vector3( 3, 4, 0 );
```

  - **Instance variables and functions**

`vec.x` – The x component of the vector

`vec.y` – The y component of the vector

`vec.z` – The z component of the vector

`vec.magnitude` – The length of the vector

`vec.Normalize()` – New Vector3 in the same direction at unit length

  - **Static class variables and functions**

`Vector3.zero` – Shorthand for `new Vector3( 0, 0, 0 );`

Important Unity Variable Types

- **Vector3 – A collection of 3 floats**
  - **Used for position of objects in 3D**

    ```
    Vector3 vec = new Vector3( 3, 4, 0 );
    ```

  - **Instance variables and functions**

    `vec.x` – The x component of the vector

    `vec.y` – The y component of the vector

    `vec.z` – The z component of the vector

    `vec.magnitude` – The length of the vector

    `vec.Normalize()` – New Vector3 in the same direction at unit length

  - **Static class variables and functions**

    `Vector3.zero` – Shorthand for `new Vector3( 0, 0, 0 );`

    `Vector3.Dot( vA, vB );` – Dot product of vA and vB

Important Unity Variable Types

Important Unity Variable Types

- Color – A color with transparency information

Important Unity Variable Types

- **Color – A color with transparency information**
  - 4 floats for red, green, blue, and alpha (all between 0 and 1)

Important Unity Variable Types

- **Color – A color with transparency information**
  - **4 floats for red, green, blue, and alpha (all between 0 and 1)**

```
Color col = new Color( 0.5f, 0.5f, 0, 1f );
```

Important Unity Variable Types

- **Color – A color with transparency information**
  - **4 floats for red, green, blue, and alpha (all between 0 and 1)**

```
Color col = new Color( 0.5f, 0.5f, 0, 1f );
Color col = new Color( 1f, 0f, 0f ); // Alpha is optional
```

  - **In the Unity color picker, the RGBA values are in the range 0–255. These are then mapped to 0–1f.**
  - **Instance variables and functions**

`col.r` – The red component of the vector

`col.g` – The green component of the vector

`col.b` – The blue component of the vector

Important Unity Variable Types

- **Color – A color with transparency information**
  - **4 floats for red, green, blue, and alpha (all between 0 and 1)**

```
Color col = new Color( 0.5f, 0.5f, 0, 1f );
Color col = new Color( 1f, 0f, 0f ); // Alpha is optional
```

  - **In the Unity color picker, the RGBA values are in the range 0–255. These are then mapped to 0–1f.**
  - **Instance variables and functions**

`col.r` – The red component of the vector

`col.g` – The green component of the vector

`col.b` – The blue component of the vector

`col.a` – The alpha component of the vector

Important Unity Variable Types

Important Unity Variable Types

- Color – A color with transparency information

Important Unity Variable Types

- Color – A color with transparency information
  - Static class variables and functions

Important Unity Variable Types

- **Color – A color with transparency information**
  - **Static class variables and functions**

```
// Primary Colors: Red, Green, and Blue
```

Important Unity Variable Types

- **Color – A color with transparency information**
  - **Static class variables and functions**

```
// Primary Colors: Red, Green, and Blue
Color.red     = new Color(1, 0, 0, 1); // Solid red
```

Important Unity Variable Types

- **Color – A color with transparency information**
  - **Static class variables and functions**

```
// Primary Colors: Red, Green, and Blue
Color.red     = new Color(1, 0, 0, 1); // Solid red
Color.green   = new Color(0, 1, 0, 1); // Solid green
```

Important Unity Variable Types

- **Color – A color with transparency information**
  - **Static class variables and functions**

```
// Primary Colors: Red, Green, and Blue
Color.red     = new Color(1, 0, 0, 1); // Solid red
Color.green   = new Color(0, 1, 0, 1); // Solid green
Color.blue    = new Color(0, 0, 1, 1); // Solid blue
```

Important Unity Variable Types

- **Color – A color with transparency information**
  - **Static class variables and functions**

```
// Primary Colors: Red, Green, and Blue
Color.red     = new Color(1, 0, 0, 1); // Solid red
Color.green   = new Color(0, 1, 0, 1); // Solid green
Color.blue    = new Color(0, 0, 1, 1); // Solid blue
```

Important Unity Variable Types

- **Color – A color with transparency information**
  - **Static class variables and functions**

```
// Primary Colors: Red, Green, and Blue
Color.red     = new Color(1, 0, 0, 1); // Solid red
Color.green   = new Color(0, 1, 0, 1); // Solid green
Color.blue    = new Color(0, 0, 1, 1); // Solid blue

// Secondary Colors: Cyan, Magenta, and Yellow
```

Important Unity Variable Types

- **Color – A color with transparency information**
  - **Static class variables and functions**

```
// Primary Colors: Red, Green, and Blue
Color.red     = new Color(1, 0, 0, 1); // Solid red
Color.green   = new Color(0, 1, 0, 1); // Solid green
Color.blue    = new Color(0, 0, 1, 1); // Solid blue

// Secondary Colors: Cyan, Magenta, and Yellow
Color.cyan    = new Color(0, 1, 1, 1); // Cyan, a bright greenish blue
```

Important Unity Variable Types

- **Color – A color with transparency information**
  - **Static class variables and functions**

```
// Primary Colors: Red, Green, and Blue
Color.red     = new Color(1, 0, 0, 1); // Solid red
Color.green   = new Color(0, 1, 0, 1); // Solid green
Color.blue    = new Color(0, 0, 1, 1); // Solid blue

// Secondary Colors: Cyan, Magenta, and Yellow
Color.cyan    = new Color(0, 1, 1, 1); // Cyan, a bright greenish blue
Color.magenta = new Color(1, 0, 1, 1); // Magenta, a pinkish purple
```

Important Unity Variable Types

- **Color – A color with transparency information**
  - **Static class variables and functions**

```
// Primary Colors: Red, Green, and Blue
Color.red     = new Color(1, 0, 0, 1); // Solid red
Color.green   = new Color(0, 1, 0, 1); // Solid green
Color.blue    = new Color(0, 0, 1, 1); // Solid blue

// Secondary Colors: Cyan, Magenta, and Yellow
Color.cyan    = new Color(0, 1, 1, 1); // Cyan, a bright greenish blue
Color.magenta = new Color(1, 0, 1, 1); // Magenta, a pinkish purple
Color.yellow  = new Color(1, 0.92f, 0.016f, 1); // A nice-looking yellow
```

Important Unity Variable Types

- **Color – A color with transparency information**
  - **Static class variables and functions**

```
// Primary Colors: Red, Green, and Blue
Color.red     = new Color(1, 0, 0, 1); // Solid red
Color.green   = new Color(0, 1, 0, 1); // Solid green
Color.blue    = new Color(0, 0, 1, 1); // Solid blue

// Secondary Colors: Cyan, Magenta, and Yellow
Color.cyan    = new Color(0, 1, 1, 1); // Cyan, a bright greenish blue
Color.magenta = new Color(1, 0, 1, 1); // Magenta, a pinkish purple
Color.yellow  = new Color(1, 0.92f, 0.016f, 1); // A nice-looking yellow
// As you can imagine, a standard yellow would be new Color(1,1,0,1), but
```

Important Unity Variable Types

- **Color – A color with transparency information**
  - **Static class variables and functions**

```
// Primary Colors: Red, Green, and Blue
Color.red     = new Color(1, 0, 0, 1); // Solid red
Color.green   = new Color(0, 1, 0, 1); // Solid green
Color.blue    = new Color(0, 0, 1, 1); // Solid blue

// Secondary Colors: Cyan, Magenta, and Yellow
Color.cyan    = new Color(0, 1, 1, 1); // Cyan, a bright greenish blue
Color.magenta = new Color(1, 0, 1, 1); // Magenta, a pinkish purple
Color.yellow  = new Color(1, 0.92f, 0.016f, 1); // A nice-looking yellow
// As you can imagine, a standard yellow would be new Color(1,1,0,1), but
//  in Unity's opinion, this color looks better.
```

Important Unity Variable Types

- **Color – A color with transparency information**
  - **Static class variables and functions**

```
// Primary Colors: Red, Green, and Blue
Color.red     = new Color(1, 0, 0, 1); // Solid red
Color.green   = new Color(0, 1, 0, 1); // Solid green
Color.blue    = new Color(0, 0, 1, 1); // Solid blue

// Secondary Colors: Cyan, Magenta, and Yellow
Color.cyan    = new Color(0, 1, 1, 1); // Cyan, a bright greenish blue
Color.magenta = new Color(1, 0, 1, 1); // Magenta, a pinkish purple
Color.yellow  = new Color(1, 0.92f, 0.016f, 1); // A nice-looking yellow
// As you can imagine, a standard yellow would be new Color(1,1,0,1), but
//  in Unity's opinion, this color looks better.
```

Important Unity Variable Types

- **Color – A color with transparency information**
  - **Static class variables and functions**

```
// Primary Colors: Red, Green, and Blue
Color.red     = new Color(1, 0, 0, 1); // Solid red
Color.green   = new Color(0, 1, 0, 1); // Solid green
Color.blue    = new Color(0, 0, 1, 1); // Solid blue

// Secondary Colors: Cyan, Magenta, and Yellow
Color.cyan    = new Color(0, 1, 1, 1); // Cyan, a bright greenish blue
Color.magenta = new Color(1, 0, 1, 1); // Magenta, a pinkish purple
Color.yellow  = new Color(1, 0.92f, 0.016f, 1); // A nice-looking yellow
// As you can imagine, a standard yellow would be new Color(1,1,0,1), but
//  in Unity's opinion, this color looks better.

// Black, White, and Clear
```

Important Unity Variable Types

- **Color – A color with transparency information**
  - **Static class variables and functions**

```
// Primary Colors: Red, Green, and Blue
Color.red     = new Color(1, 0, 0, 1); // Solid red
Color.green   = new Color(0, 1, 0, 1); // Solid green
Color.blue    = new Color(0, 0, 1, 1); // Solid blue

// Secondary Colors: Cyan, Magenta, and Yellow
Color.cyan    = new Color(0, 1, 1, 1); // Cyan, a bright greenish blue
Color.magenta = new Color(1, 0, 1, 1); // Magenta, a pinkish purple
Color.yellow  = new Color(1, 0.92f, 0.016f, 1); // A nice-looking yellow
// As you can imagine, a standard yellow would be new Color(1,1,0,1), but
//  in Unity's opinion, this color looks better.

// Black, White, and Clear
Color.black   = new Color(0, 0, 0, 1); // Solid black
```

Important Unity Variable Types

- **Color – A color with transparency information**
  - **Static class variables and functions**

```
// Primary Colors: Red, Green, and Blue
Color.red     = new Color(1, 0, 0, 1); // Solid red
Color.green   = new Color(0, 1, 0, 1); // Solid green
Color.blue    = new Color(0, 0, 1, 1); // Solid blue

// Secondary Colors: Cyan, Magenta, and Yellow
Color.cyan    = new Color(0, 1, 1, 1); // Cyan, a bright greenish blue
Color.magenta = new Color(1, 0, 1, 1); // Magenta, a pinkish purple
Color.yellow  = new Color(1, 0.92f, 0.016f, 1); // A nice-looking yellow
// As you can imagine, a standard yellow would be new Color(1,1,0,1), but
//  in Unity's opinion, this color looks better.

// Black, White, and Clear
Color.black   = new Color(0, 0, 0, 1); // Solid black
Color.white   = new Color(1, 1, 1, 1); // Solid white
```

Important Unity Variable Types

- **Color – A color with transparency information**
  - **Static class variables and functions**

```
// Primary Colors: Red, Green, and Blue
Color.red     = new Color(1, 0, 0, 1); // Solid red
Color.green   = new Color(0, 1, 0, 1); // Solid green
Color.blue    = new Color(0, 0, 1, 1); // Solid blue

// Secondary Colors: Cyan, Magenta, and Yellow
Color.cyan    = new Color(0, 1, 1, 1); // Cyan, a bright greenish blue
Color.magenta = new Color(1, 0, 1, 1); // Magenta, a pinkish purple
Color.yellow  = new Color(1, 0.92f, 0.016f, 1); // A nice-looking yellow
// As you can imagine, a standard yellow would be new Color(1,1,0,1), but
//  in Unity's opinion, this color looks better.

// Black, White, and Clear
Color.black   = new Color(0, 0, 0, 1); // Solid black
Color.white   = new Color(1, 1, 1, 1); // Solid white
Color.gray    = new Color(0.5f, 0.5f, 0.5f, 1) // Gray
```

Important Unity Variable Types

- **Color – A color with transparency information**
  - **Static class variables and functions**

```
// Primary Colors: Red, Green, and Blue
Color.red     = new Color(1, 0, 0, 1); // Solid red
Color.green   = new Color(0, 1, 0, 1); // Solid green
Color.blue    = new Color(0, 0, 1, 1); // Solid blue

// Secondary Colors: Cyan, Magenta, and Yellow
Color.cyan    = new Color(0, 1, 1, 1); // Cyan, a bright greenish blue
Color.magenta = new Color(1, 0, 1, 1); // Magenta, a pinkish purple
Color.yellow  = new Color(1, 0.92f, 0.016f, 1); // A nice-looking yellow
// As you can imagine, a standard yellow would be new Color(1,1,0,1), but
//  in Unity's opinion, this color looks better.

// Black, White, and Clear
Color.black   = new Color(0, 0, 0, 1); // Solid black
Color.white   = new Color(1, 1, 1, 1); // Solid white
Color.gray    = new Color(0.5f, 0.5f, 0.5f, 1) // Gray
Color.grey    = new Color(0.5f, 0.5f, 0.5f, 1) // British spelling of gray
```

Important Unity Variable Types

- **Color – A color with transparency information**
  - **Static class variables and functions**

```
// Primary Colors: Red, Green, and Blue
Color.red     = new Color(1, 0, 0, 1); // Solid red
Color.green   = new Color(0, 1, 0, 1); // Solid green
Color.blue    = new Color(0, 0, 1, 1); // Solid blue

// Secondary Colors: Cyan, Magenta, and Yellow
Color.cyan    = new Color(0, 1, 1, 1); // Cyan, a bright greenish blue
Color.magenta = new Color(1, 0, 1, 1); // Magenta, a pinkish purple
Color.yellow  = new Color(1, 0.92f, 0.016f, 1); // A nice-looking yellow
// As you can imagine, a standard yellow would be new Color(1,1,0,1), but
//  in Unity's opinion, this color looks better.

// Black, White, and Clear
Color.black   = new Color(0, 0, 0, 1); // Solid black
Color.white   = new Color(1, 1, 1, 1); // Solid white
Color.gray    = new Color(0.5f, 0.5f, 0.5f, 1) // Gray
Color.grey    = new Color(0.5f, 0.5f, 0.5f, 1) // British spelling of gray
Color.clear   = new Color(0, 0, 0, 0); // Completely transparent
```

Important Unity Variable Types

Important Unity Variable Types

- Quaternion – Rotation information

Important Unity Variable Types

- **Quaternion – Rotation information**
  - Based on three imaginary numbers and a scalar

Important Unity Variable Types

- **Quaternion – Rotation information**
  - Based on three imaginary numbers and a scalar
  - So, everyone uses Euler angles (e.g., x, y, z) to input rotation

Important Unity Variable Types

- **Quaternion – Rotation information**
  - **Based on three imaginary numbers and a scalar**
  - **So, everyone uses Euler angles (e.g., x, y, z) to input rotation**

```
Quaternion up45Deg = Quaternion.Euler( -45, 0, 0 );
```

  - **In Euler (pronounced "oiler") angles, x, y, & z are rotations about those respective axes**
  - **Quaternions are much better for interpolation and calculations than Euler angles**
    - They also avoid Gimbal Lock (where two Euler axes align)
  - **Instance variables and functions**

Important Unity Variable Types

- **Quaternion – Rotation information**
  - **Based on three imaginary numbers and a scalar**
  - **So, everyone uses Euler angles (e.g., x, y, z) to input rotation**

```
Quaternion up45Deg = Quaternion.Euler( -45, 0, 0 );
```

  - **In Euler (pronounced "oiler") angles, x, y, & z are rotations about those respective axes**
  - **Quaternions are much better for interpolation and calculations than Euler angles**
    - They also avoid Gimbal Lock (where two Euler axes align)
  - **Instance variables and functions**

`up45Deg.eulerAngles` – A Vector3 of the Euler rotations

Important Unity Variable Types

Important Unity Variable Types

- Mathf – A collection of static math functions

Important Unity Variable Types

- **Mathf – A collection of static math functions**
  - **Static class variables and functions**

Important Unity Variable Types

- **Mathf – A collection of static math functions**
  - **Static class variables and functions**

```
Mathf.Sin(x);          // Computes the sine of x
```

Important Unity Variable Types

- **Mathf – A collection of static math functions**
  - **Static class variables and functions**

```
Mathf.Sin(x);          // Computes the sine of x

Mathf.Cos(x);          // .Tan(), .Asin(), .Acos(), & .Atan() also available
```

Important Unity Variable Types

- **Mathf – A collection of static math functions**
  - **Static class variables and functions**

```
Mathf.Sin(x);          // Computes the sine of x

Mathf.Cos(x);          // .Tan(), .Asin(), .Acos(), & .Atan() also available

Mathf.Atan2( y, x ); // Gives you the angle to rotate around the z-axis to
                       //  change something facing along the x-axis to face
                       //  instead toward the point x, y.
```

Important Unity Variable Types

- **Mathf – A collection of static math functions**
  - **Static class variables and functions**

```
Mathf.Sin(x);          // Computes the sine of x

Mathf.Cos(x);          // .Tan(), .Asin(), .Acos(), & .Atan() also available

Mathf.Atan2( y, x ); // Gives you the angle to rotate around the z-axis to
                       //  change something facing along the x-axis to face
                       //  instead toward the point x, y.

print(Mathf.PI);       // 3.141593; the ratio of circumference to diameter
```

Important Unity Variable Types

- **Mathf – A collection of static math functions**
  - **Static class variables and functions**

```
Mathf.Sin(x);          // Computes the sine of x

Mathf.Cos(x);          // .Tan(), .Asin(), .Acos(), & .Atan() also available

Mathf.Atan2( y, x );   // Gives you the angle to rotate around the z-axis to
                       //  change something facing along the x-axis to face
                       //  instead toward the point x, y.

print(Mathf.PI);       // 3.141593; the ratio of circumference to diameter

Mathf.Min( 2, 3, 1 );// 1, the smallest of the numbers (float or int)
```

Important Unity Variable Types

- **Mathf – A collection of static math functions**
  - **Static class variables and functions**

```
Mathf.Sin(x);          // Computes the sine of x

Mathf.Cos(x);          // .Tan(), .Asin(), .Acos(), & .Atan() also available

Mathf.Atan2( y, x );   // Gives you the angle to rotate around the z-axis to
                       //  change something facing along the x-axis to face
                       //  instead toward the point x, y.

print(Mathf.PI);       // 3.141593; the ratio of circumference to diameter

Mathf.Min( 2, 3, 1 );// 1, the smallest of the numbers (float or int)

Mathf.Max( 2, 3, 1 );// 3, the largest of the numbers (float or int)
```

Important Unity Variable Types

- **Mathf – A collection of static math functions**
  - **Static class variables and functions**

```
Mathf.Sin(x);          // Computes the sine of x

Mathf.Cos(x);          // .Tan(), .Asin(), .Acos(), & .Atan() also available

Mathf.Atan2( y, x );   // Gives you the angle to rotate around the z-axis to
                       //  change something facing along the x-axis to face
                       //  instead toward the point x, y.

print(Mathf.PI);       // 3.141593; the ratio of circumference to diameter

Mathf.Min( 2, 3, 1 );// 1, the smallest of the numbers (float or int)

Mathf.Max( 2, 3, 1 );// 3, the largest of the numbers (float or int)

Mathf.Round( 1.75f );// 2, rounds up or down to the nearest number
```

Important Unity Variable Types

- **Mathf – A collection of static math functions**
  - **Static class variables and functions**

```
Mathf.Sin(x);          // Computes the sine of x

Mathf.Cos(x);          // .Tan(), .Asin(), .Acos(), & .Atan() also available

Mathf.Atan2( y, x ); // Gives you the angle to rotate around the z-axis to
                       //  change something facing along the x-axis to face
                       //  instead toward the point x, y.

print(Mathf.PI);       // 3.141593; the ratio of circumference to diameter

Mathf.Min( 2, 3, 1 );// 1, the smallest of the numbers (float or int)

Mathf.Max( 2, 3, 1 );// 3, the largest of the numbers (float or int)

Mathf.Round( 1.75f );// 2, rounds up or down to the nearest number

Mathf.Ceil( 1.75f ); // 2, rounds up to the next highest integer number
```

Important Unity Variable Types

- **Mathf – A collection of static math functions**
  - **Static class variables and functions**

```
Mathf.Sin(x);          // Computes the sine of x

Mathf.Cos(x);          // .Tan(), .Asin(), .Acos(), & .Atan() also available

Mathf.Atan2( y, x );   // Gives you the angle to rotate around the z-axis to
                       //  change something facing along the x-axis to face
                       //  instead toward the point x, y.

print(Mathf.PI);       // 3.141593; the ratio of circumference to diameter

Mathf.Min( 2, 3, 1 );// 1, the smallest of the numbers (float or int)

Mathf.Max( 2, 3, 1 );// 3, the largest of the numbers (float or int)

Mathf.Round( 1.75f );// 2, rounds up or down to the nearest number

Mathf.Ceil( 1.75f );  // 2, rounds up to the next highest integer number

Mathf.Floor( 1.75f );// 1, rounds down to the next lowest integer number
```

Important Unity Variable Types

- **Mathf – A collection of static math functions**
  - **Static class variables and functions**

```
Mathf.Sin(x);          // Computes the sine of x

Mathf.Cos(x);          // .Tan(), .Asin(), .Acos(), & .Atan() also available

Mathf.Atan2( y, x );   // Gives you the angle to rotate around the z-axis to
                       //  change something facing along the x-axis to face
                       //  instead toward the point x, y.

print(Mathf.PI);       // 3.141593; the ratio of circumference to diameter

Mathf.Min( 2, 3, 1 );// 1, the smallest of the numbers (float or int)

Mathf.Max( 2, 3, 1 );// 3, the largest of the numbers (float or int)

Mathf.Round( 1.75f );// 2, rounds up or down to the nearest number

Mathf.Ceil( 1.75f );  // 2, rounds up to the next highest integer number

Mathf.Floor( 1.75f );// 1, rounds down to the next lowest integer number

Mathf.Abs( -25 );      // 25, the absolute value of -25
```

Important Unity Variable Types

Important Unity Variable Types

- Screen – Information about the display

Important Unity Variable Types

- Screen – Information about the display
  - Static class variables and functions

Important Unity Variable Types

- **Screen – Information about the display**
  - **Static class variables and functions**

```
Screen.width            // The width of the screen in pixels
```

Important Unity Variable Types

- **Screen – Information about the display**
  - **Static class variables and functions**

```
Screen.width          // The width of the screen in pixels
Screen.height         // The height of the screen in pixels
```

Important Unity Variable Types

- **Screen – Information about the display**
  - **Static class variables and functions**

```
Screen.width                    // The width of the screen in pixels
Screen.height                   // The height of the screen in pixels
Screen.showCursor = false;      // Hide the cursor
```

Important Unity Variable Types

Important Unity Variable Types

- SystemInfo – Information about the device/computer

Important Unity Variable Types

- SystemInfo – Information about the device/computer
  - Static class variables and functions

Important Unity Variable Types

- **SystemInfo – Information about the device/computer**
  - **Static class variables and functions**

```
SystemInfo.operatingSystem // The width of the screen in pixels
                           //  e.g., Mac OS X 10.9.3
```

Important Unity Variable Types

- **SystemInfo – Information about the device/computer**
  - **Static class variables and functions**

```
SystemInfo.operatingSystem // The width of the screen in pixels
                           //  e.g., Mac OS X 10.9.3

SystemInfo.systemMemorySize          // Amount of RAM
```

Important Unity Variable Types

- **SystemInfo – Information about the device/computer**
  - **Static class variables and functions**

```
SystemInfo.operatingSystem // The width of the screen in pixels
                           //  e.g., Mac OS X 10.9.3
SystemInfo.systemMemorySize        // Amount of RAM
SystemInfo.supportsAccelerometer   // Has accelerometer
```

Important Unity Variable Types

- **SystemInfo – Information about the device/computer**
  - **Static class variables and functions**

```
SystemInfo.operatingSystem // The width of the screen in pixels
                           //  e.g., Mac OS X 10.9.3
SystemInfo.systemMemorySize        // Amount of RAM
SystemInfo.supportsAccelerometer   // Has accelerometer
SystemInfo.supportsGyroscope       // Has gyroscope
```

Important Unity Variable Types

Important Unity Variable Types

- GameObject – Base class for all objects in scenes

Important Unity Variable Types

- GameObject – Base class for all objects in scenes
  - Composed of *components*

Important Unity Variable Types

- **GameObject – Base class for all objects in scenes**
  - **Composed of *components***

```
GameObject go = new GameObject("MyGO");
```

Important Unity Variable Types

- **GameObject – Base class for all objects in scenes**
  - **Composed of *components***

```
GameObject go = new GameObject("MyGO");
```

  - **Always has a Transform component**
  - **Instance variables and functions**

```
go.name                    // The name of the GameObject ("MyGO")
go.GetComponent<Transform>()    // The Transform component
go.transform               // A shortcut to the Transform component
go.SetActive(false)        // Make this GameObject inactive
```

Important Unity Variable Types

- **GameObject – Base class for all objects in scenes**
  - **Composed of *components***

```
GameObject go = new GameObject("MyGO");
```

  - **Always has a Transform component**
  - **Instance variables and functions**

```
go.name          // The name of the GameObject ("MyGO")
go.GetComponent<Transform>()     // The Transform component
go.transform                     // A shortcut to the Transform component
go.SetActive(false)     // Make this GameObject inactive
go.name          // The name of the GameObject ("MyGO")
```

Important Unity Variable Types

- **GameObject – Base class for all objects in scenes**
  - **Composed of *components***

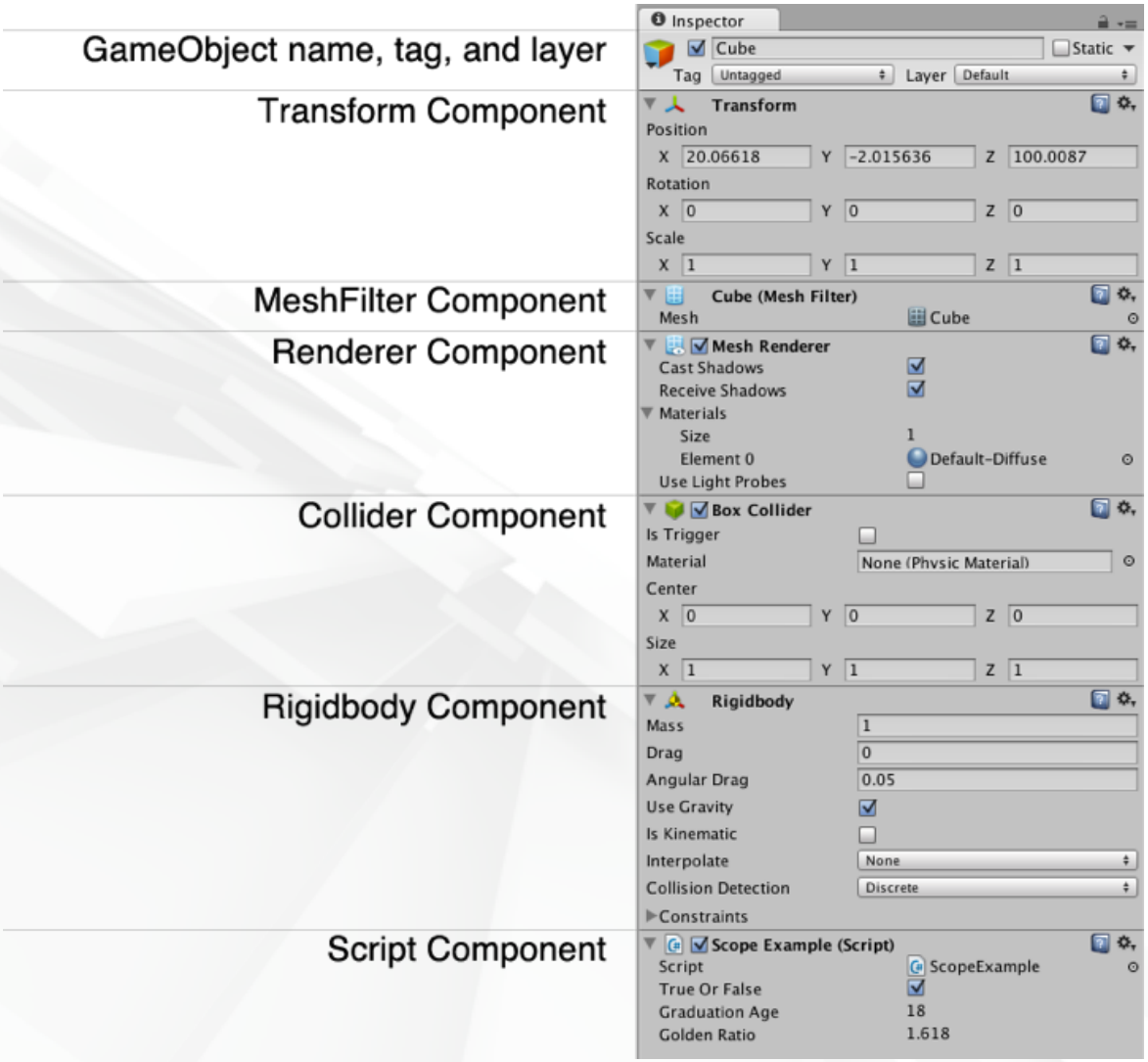
```
GameObject go = new GameObject("MyGO");
```

  - **Always has a Transform component**
  - **Instance variables and functions**

```
go.name          // The name of the GameObject ("MyGO")
go.GetComponent<Transform>()      // The Transform component
go.transform                      // A shortcut to the Transform component
go.SetActive(false)      // Make this GameObject inactive
go.name          // The name of the GameObject ("MyGO")
```

  - **GetComponent<>() is a generic method that can be used to access *any* component attached to a GameObject**

Unity GameObject Components



GameObjects are composed of Components

Unity GameObject Components

Unity GameObject Components

- Transform component

Unity GameObject Components

- **Transform component**
  - Controls position, rotation, and scale

Unity GameObject Components

- **Transform component**
  - **Controls position, rotation, and scale**

```
Transform tr = go.GetComponent<Transform>();
```

Unity GameObject Components

- **Transform component**
  - **Controls position, rotation, and scale**

```
Transform tr = go.GetComponent<Transform>();
```

  - **Also controls hierarchy of objects in the scene**

```
tr.parent      // The parent of this transform in the hierarchy
```

  - **Children can be iterated over with a foreach loop**

```
foreach (Transform tChild in tr) {…}
```

  - **Instance variables and functions**

```
tr.position            // The position in world coordinates
```

Unity GameObject Components

- **Transform component**
  - **Controls position, rotation, and scale**

```
Transform tr = go.GetComponent<Transform>();
```

  - **Also controls hierarchy of objects in the scene**

```
tr.parent      // The parent of this transform in the hierarchy
```

  - **Children can be iterated over with a foreach loop**

```
foreach (Transform tChild in tr) {…}
```

  - **Instance variables and functions**

```
tr.position          // The position in world coordinates
tr.localPosition     // The position relative to its parent
```

Unity GameObject Components

- **Transform component**
  - **Controls position, rotation, and scale**

```
Transform tr = go.GetComponent<Transform>();
```

  - **Also controls hierarchy of objects in the scene**

```
tr.parent      // The parent of this transform in the hierarchy
```

  - **Children can be iterated over with a foreach loop**

```
foreach (Transform tChild in tr) {…}
```

  - **Instance variables and functions**

```
tr.position          // The position in world coordinates
tr.localPosition     // The position relative to its parent
tr.rotation          // The rotation in world coordinates
```

Unity GameObject Components

- **Transform component**
  - **Controls position, rotation, and scale**

```
Transform tr = go.GetComponent<Transform>();
```

  - **Also controls hierarchy of objects in the scene**

```
tr.parent      // The parent of this transform in the hierarchy
```

  - **Children can be iterated over with a foreach loop**

```
foreach (Transform tChild in tr) {…}
```

  - **Instance variables and functions**

```
tr.position          // The position in world coordinates
tr.localPosition     // The position relative to its parent
tr.rotation          // The rotation in world coordinates
tr.localScale        // The scale (always in local coordinates)
```

Unity GameObject Components

Unity GameObject Components

- MeshFilter component

Unity GameObject Components

- **MeshFilter component**
  - The model that you see

Unity GameObject Components

- **MeshFilter component**
  - **The model that you see**

```
MeshFilter mf = go.GetComponent<MeshFilter>();
```

Unity GameObject Components

- **MeshFilter component**
  - **The model that you see**

```
MeshFilter mf = go.GetComponent<MeshFilter>();
```

  - **Attaches a 3D model to a GameObject**
  - **Is actually a 3D shell of the object (3D objects in games are hollow inside**
  - **This MeshFilter is rendered on screen by a MeshRenderer component**

Unity GameObject Components

Unity GameObject Components

- Renderer component

Unity GameObject Components

- **Renderer component**
  - Draws the GameObject on screen

Unity GameObject Components

- **Renderer component**
  - **Draws the GameObject on screen**

```
Renderer rend = go.GetComponent<Renderer>();
```

Unity GameObject Components

- **Renderer component**
  - **Draws the GameObject on screen**

```
Renderer rend = go.GetComponent<Renderer>();
```

  - **Usually, this is a MeshRenderer**
    - Renderer is the superclass for MeshRenderer
    - So, Renderer is almost always used in code

Unity GameObject Components

- **Renderer component**
  - **Draws the GameObject on screen**

```
Renderer rend = go.GetComponent<Renderer>();
```

  - **Usually, this is a MeshRenderer**
    - Renderer is the superclass for MeshRenderer
    - So, Renderer is almost always used in code
  - **Combines the MeshFilter with a Material (which contains various Textures and a Shader)**

Unity GameObject Components

Unity GameObject Components

- Collider component

Unity GameObject Components

- **Collider component**
  - **The physical presence of the GameObejct**

Unity GameObject Components

- **Collider component**
  - **The physical presence of the GameObejct**

```
Collider coll = go.GetComponent<Collider>();
```

Unity GameObject Components

- **Collider component**
  - **The physical presence of the GameObejct**

```
Collider coll = go.GetComponent<Collider>();
```

  - **There are four types of collider (in order of complexity)**
    - Sphere Collider – The fastest type. A ball or sphere.
    - Capsule Collider – A pipe with spheres at each end. 2nd fastest.

Unity GameObject Components

- **Collider component**
  - **The physical presence of the GameObejct**

```
Collider coll = go.GetComponent<Collider>();
```

  - **There are four types of collider (in order of complexity)**
    - Sphere Collider – The fastest type. A ball or sphere.
    - Capsule Collider – A pipe with spheres at each end. 2nd fastest.
    - Box Collider – A rectangular solid. Useful for crates, cars, torsos, etc.

Unity GameObject Components

- **Collider component**
  - **The physical presence of the GameObejct**

```
Collider coll = go.GetComponent<Collider>();
```

  - **There are four types of collider (in order of complexity)**
    - Sphere Collider – The fastest type. A ball or sphere.
    - Capsule Collider – A pipe with spheres at each end. 2nd fastest.
    - Box Collider – A rectangular solid. Useful for crates, cars, torsos, etc.
    - Mesh Collider – Collider formed from a MeshFilter. Much slower!

Unity GameObject Components

- **Collider component**
  - **The physical presence of the GameObejct**

```
Collider coll = go.GetComponent<Collider>();
```

  - **There are four types of collider (in order of complexity)**
    - Sphere Collider – The fastest type. A ball or sphere.
    - Capsule Collider – A pipe with spheres at each end. 2nd fastest.
    - Box Collider – A rectangular solid. Useful for crates, cars, torsos, etc.
    - Mesh Collider – Collider formed from a MeshFilter. Much slower!
      - Only convex Mesh Collider can collide with other Mesh Colliders

Unity GameObject Components

- **Collider component**
  - **The physical presence of the GameObejct**

```
Collider coll = go.GetComponent<Collider>();
```

  - **There are four types of collider (in order of complexity)**
    - Sphere Collider – The fastest type. A ball or sphere.
    - Capsule Collider – A pipe with spheres at each end. 2nd fastest.
    - Box Collider – A rectangular solid. Useful for crates, cars, torsos, etc.
    - Mesh Collider – Collider formed from a MeshFilter. Much slower!
      - Only convex Mesh Collider can collide with other Mesh Colliders
      - Much, much slower than the other three types

Unity GameObject Components

- **Collider component**
  - **The physical presence of the GameObejct**

```
Collider coll = go.GetComponent<Collider>();
```

  - **There are four types of collider (in order of complexity)**
    - Sphere Collider – The fastest type. A ball or sphere.
    - Capsule Collider – A pipe with spheres at each end. 2nd fastest.
    - Box Collider – A rectangular solid. Useful for crates, cars, torsos, etc.
    - Mesh Collider – Collider formed from a MeshFilter. Much slower!
      - Only convex Mesh Collider can collide with other Mesh Colliders
      - Much, much slower than the other three types
  - **Unity physics are performed by the NVIDIA PhysX engine**

Unity GameObject Components

- **Collider component**
  - **The physical presence of the GameObejct**

```
Collider coll = go.GetComponent<Collider>();
```

  - **There are four types of collider (in order of complexity)**
    - Sphere Collider – The fastest type. A ball or sphere.
    - Capsule Collider – A pipe with spheres at each end. 2nd fastest.
    - Box Collider – A rectangular solid. Useful for crates, cars, torsos, etc.
    - Mesh Collider – Collider formed from a MeshFilter. Much slower!
      - Only convex Mesh Collider can collide with other Mesh Colliders
      - Much, much slower than the other three types
  - **Unity physics are performed by the NVIDIA PhysX engine**
  - **Colliders *will not move* without a Rigidbody component**

Unity GameObject Components

Unity GameObject Components

- Rigidbody component

Unity GameObject Components

- **Rigidbody component**
  - **The physical simulation of the GameObject**

Unity GameObject Components

- **Rigidbody component**
  - **The physical simulation of the GameObject**

```
Rigidbody rigid = go.GetComponent<Rigidbody>();
```

Unity GameObject Components

- **Rigidbody component**
  - **The physical simulation of the GameObject**

```
Rigidbody rigid = go.GetComponent<Rigidbody>();
```

  - **Handles velocity, bounciness, friction, gravity, etc.**
  - **Updates every FixedUpdate()**
    - This is exactly 50 times per second

Unity GameObject Components

- **Rigidbody component**
  - **The physical simulation of the GameObject**

```
Rigidbody rigid = go.GetComponent<Rigidbody>();
```

  - **Handles velocity, bounciness, friction, gravity, etc.**
  - **Updates every FixedUpdate()**
    - This is exactly 50 times per second
  - **If Rigidbody isKinematic == true, the collider will move, but position will not change automatically due to velocity**

Unity GameObject Components

- **Rigidbody component**
  - **The physical simulation of the GameObject**

```
Rigidbody rigid = go.GetComponent<Rigidbody>();
```

  - **Handles velocity, bounciness, friction, gravity, etc.**
  - **Updates every FixedUpdate()**
    - This is exactly 50 times per second
  - **If Rigidbody isKinematic == true, the collider will move, but position will not change automatically due to velocity**

```
rigid.isKinematic = true;  // rigid will not move on its own
```

Unity GameObject Components

- **Rigidbody component**
  - **The physical simulation of the GameObject**

```
Rigidbody rigid = go.GetComponent<Rigidbody>();
```

  - **Handles velocity, bounciness, friction, gravity, etc.**
  - **Updates every FixedUpdate()**
    - This is exactly 50 times per second
  - **If Rigidbody isKinematic == true, the collider will move, but position will not change automatically due to velocity**

```
rigid.isKinematic = true;  // rigid will not move on its own
```

  - **Colliders *will not move* without a Rigidbody component**

Unity GameObject Components

Unity GameObject Components

- (Script) components

Unity GameObject Components

- (Script) components
  - Any C# class that you write

Unity GameObject Components

- (Script) components
  - Any C# class that you write

```
HelloWorld hw = go.GetComponent<HelloWorld>();
```

Unity GameObject Components

- **(Script) components**
  - **Any C# class that you write**

```
HelloWorld hw = go.GetComponent<HelloWorld>();
```

  - **Because C# scripts are handled as components, several can be attached to the same GameObject**
    - This enables more object-oriented programming
    - You'll see several examples throughout the book

Unity GameObject Components

- **(Script) components**
  - **Any C# class that you write**

```
HelloWorld hw = go.GetComponent<HelloWorld>();
```

  - **Because C# scripts are handled as components, several can be attached to the same GameObject**
    - This enables more object-oriented programming
    - You'll see several examples throughout the book
  - **Public fields in your scripts will appear as editable fields in the Unity Inspector**

Unity GameObject Components

- **(Script) components**
  - **Any C# class that you write**

```
HelloWorld hw = go.GetComponent<HelloWorld>();
```

  - **Because C# scripts are handled as components, several can be attached to the same GameObject**
    - This enables more object-oriented programming
    - You'll see several examples throughout the book
  - **Public fields in your scripts will appear as editable fields in the Unity Inspector**
    - However, Unity will often alter the names of these fields a bit

Unity GameObject Components

- **(Script) components**
  - **Any C# class that you write**

```
HelloWorld hw = go.GetComponent<HelloWorld>();
```

  - **Because C# scripts are handled as components, several can be attached to the same GameObject**
    - This enables more object-oriented programming
    - You'll see several examples throughout the book
  - **Public fields in your scripts will appear as editable fields in the Unity Inspector**
    - However, Unity will often alter the names of these fields a bit
      - The class name **`ScopeExample`** becomes **Scope Example (Script)**.

Unity GameObject Components

- **(Script) components**
  - **Any C# class that you write**

```
HelloWorld hw = go.GetComponent<HelloWorld>();
```

  - **Because C# scripts are handled as components, several can be attached to the same GameObject**
    - This enables more object-oriented programming
    - You'll see several examples throughout the book
  - **Public fields in your scripts will appear as editable fields in the Unity Inspector**
    - However, Unity will often alter the names of these fields a bit
      - The class name **`ScopeExample`** becomes **Scope Example (Script)**.
      - The variable **`trueOrFalse`** becomes **True Or False**.

Unity GameObject Components

- **(Script) components**
  - **Any C# class that you write**

```
HelloWorld hw = go.GetComponent<HelloWorld>();
```

  - **Because C# scripts are handled as components, several can be attached to the same GameObject**
    - This enables more object-oriented programming
    - You'll see several examples throughout the book
  - **Public fields in your scripts will appear as editable fields in the Unity Inspector**
    - However, Unity will often alter the names of these fields a bit
      - The class name **`ScopeExample`** becomes **Scope Example (Script)**.
      - The variable **`trueOrFalse`** becomes **True Or False**.
      - The variable **`graduationAge`** becomes **Graduation Age**.

Unity GameObject Components

- **(Script) components**
  - **Any C# class that you write**

```
HelloWorld hw = go.GetComponent<HelloWorld>();
```

  - **Because C# scripts are handled as components, several can be attached to the same GameObject**
    - This enables more object-oriented programming
    - You'll see several examples throughout the book
  - **Public fields in your scripts will appear as editable fields in the Unity Inspector**
    - However, Unity will often alter the names of these fields a bit
      - The class name **`ScopeExample`** becomes **Scope Example (Script)**.
      - The variable **`trueOrFalse`** becomes **True Or False**.
      - The variable **`graduationAge`** becomes **Graduation Age**.
      - The variable **`goldenRatio`** becomes **Golden Ratio**.

Chapter 19 – Summary

Chapter 19 – Summary

- Learned about declaring and defining C# variables

Chapter 19 – Summary

- Learned about declaring and defining C# variables
- Learned several important C# variable types

Chapter 19 – Summary

- Learned about declaring and defining C# variables
- Learned several important C# variable types
  - These all start with lowercase letters

Chapter 19 – Summary

- Learned about declaring and defining C# variables
- Learned several important C# variable types
  - These all start with lowercase letters
- Learned naming conventions used in this book

Chapter 19 – Summary

- Learned about declaring and defining C# variables
- Learned several important C# variable types
  - These all start with lowercase letters
- Learned naming conventions used in this book
- Important Unity Variable Types

Chapter 19 – Summary

- Learned about declaring and defining C# variables
- Learned several important C# variable types
  - These all start with lowercase letters
- Learned naming conventions used in this book
- Important Unity Variable Types
  - These all start with uppercase letters

Chapter 19 – Summary

- Learned about declaring and defining C# variables
- Learned several important C# variable types
  - These all start with lowercase letters
- Learned naming conventions used in this book
- Important Unity Variable Types
  - These all start with uppercase letters
- Learned several Unity GameObject components

Chapter 19 – Summary

- Learned about declaring and defining C# variables
- Learned several important C# variable types
  - These all start with lowercase letters
- Learned naming conventions used in this book
- Important Unity Variable Types
  - These all start with uppercase letters
- Learned several Unity GameObject components
- Next chapter will introduce you to Boolean operations and the conditionals used to control C# code