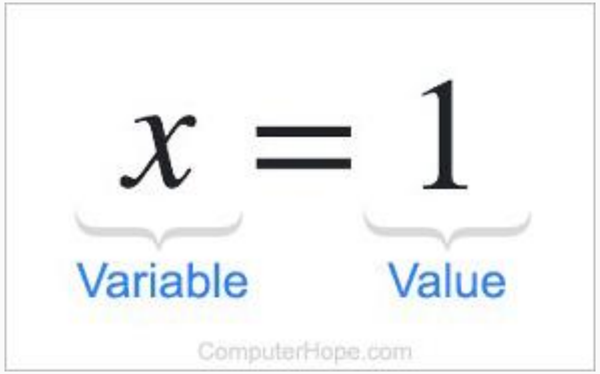
Value and Reference Types

Value and Reference Type Introduction

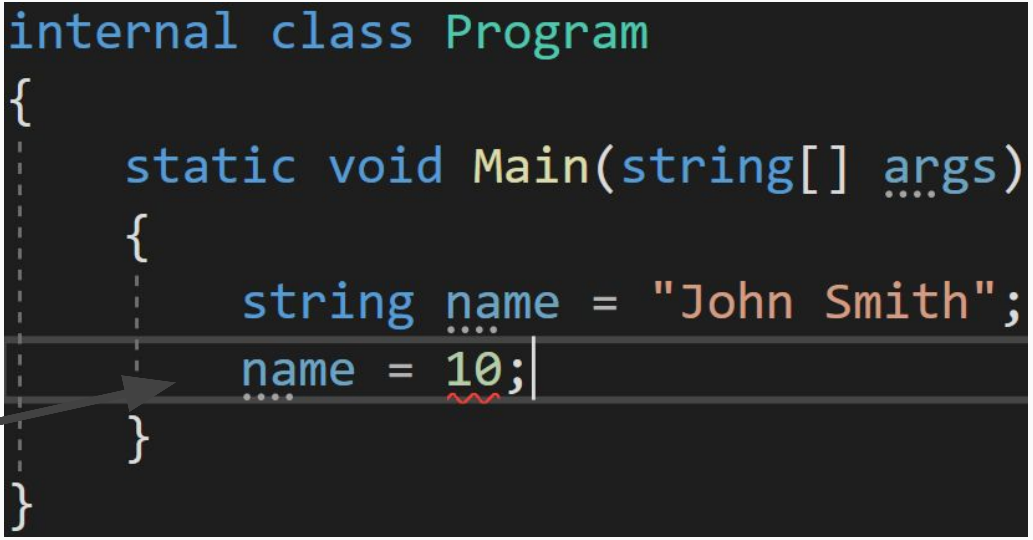
Before we can talk about **Value** and **Reference Types** we must first understand what is a **variable**.

In computer science, a **variable** is a named container that stores a value.



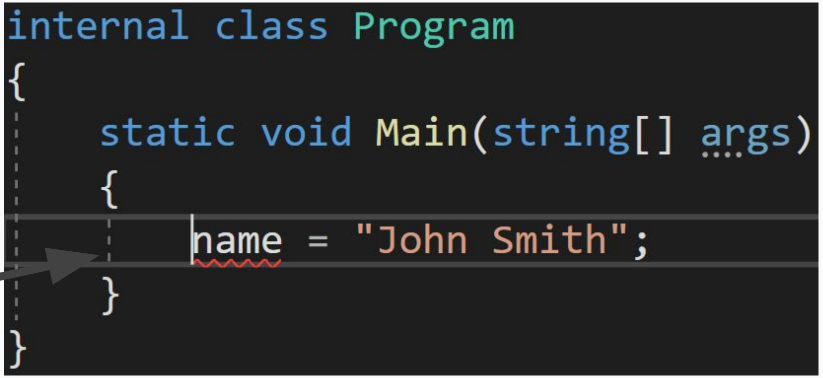
C# is a **strongly** and **statically** typed Object Oriented Programming Language.

- **Strongly**: Once a variable's type is declared it cannot change



```
internal class Program
{
    static void Main(string[] args)
    {
        string name = "John Smith";
        name = 10;
    }
}
```

- **Statically:** Every variable must have a type at Compile Time



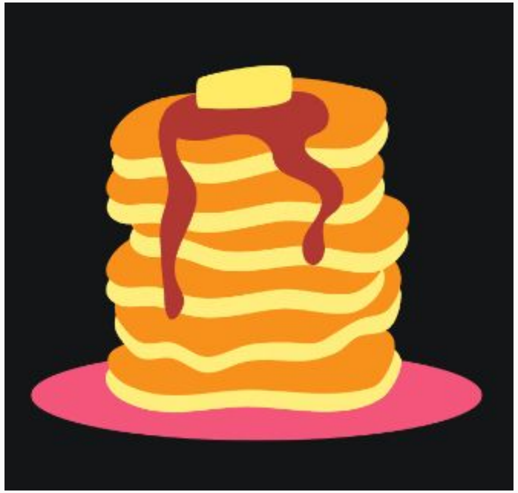
```
internal class Program
{
    static void Main(string[] args)
    {
        name = "John Smith";
    }
}
```

Two Different Types of Memory

1. **Stack**
2. **Heap**

Stack

Section of memory that grows and shrinks automatically (like a stack of plates).

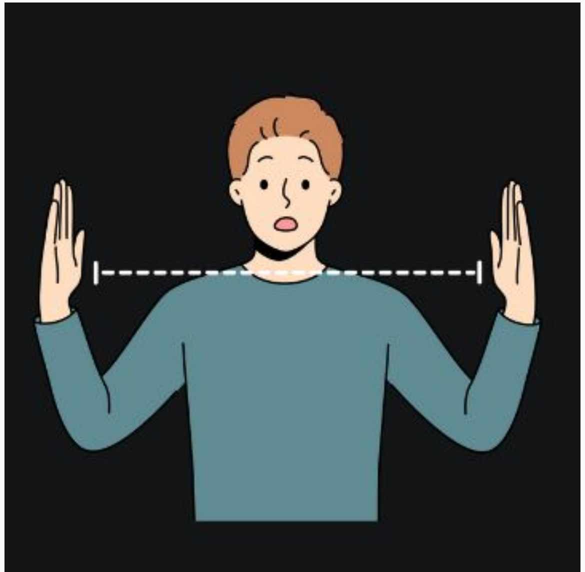


Heap

Section of memory where data is allocated dynamically.

Stack

Has a limited size, determined at the start of the program.



Heap

The heap can grow to the size of the available memory

So what do we mean by Value and Reference types?

In computer programming, data types can be divided into 2 categories: **value types** and **reference types**.

- A value of **value type** is the actual value.

- A value of **reference type** is a reference to another value.

Stack vs Heap

Ordered, on top of eachother!
No particular order!
Stack
Heap

Stack vs Heap

- **Value types** are stored on the **Stack**

Ordered, on top of eachother!
No particular order!
Stack
Heap

Stack vs Heap

- **Reference types** are stored on the **Heap**

Ordered, on top of eachother!
No particular order!
Stack
Heap

Stack vs Heap

- The **Stack** is a Last In First Out (LIFO) Abstract Data Structure

Ordered, on top of eachother!
No particular order!
Stack
Heap

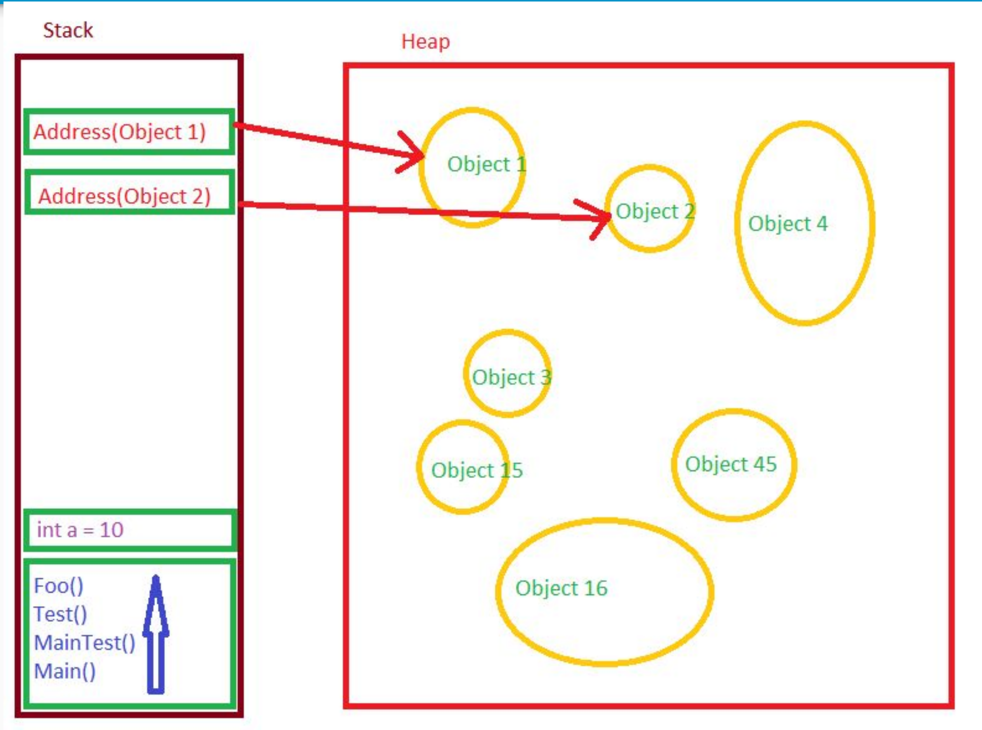
Stack vs Heap

- The **Heap** is a specialized tree-like data structure with no particular order for data retrieval

Ordered, on top of eachother!
No particular order!
Stack
Heap

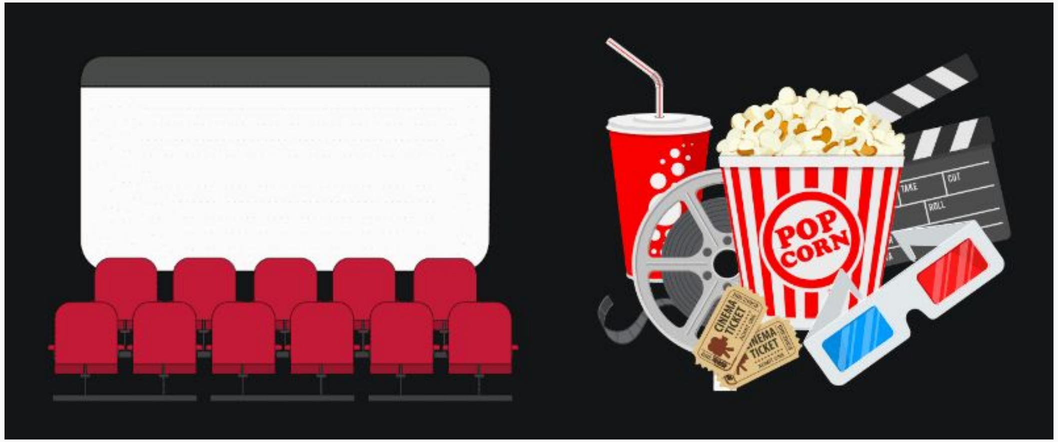
Value vs Reference Types

- Variables that store **value types** actually hold the values
- Variables that store **reference types** actually hold pointers to the value in memory



Queue

- A queue is a collection data structure that follows the First-In-First-Out (FIFO) principle.
- This means that the first element added to the queue will be the first one to be removed.



Value Types

Examples:

- structs
- enums
- bools
- chars
- and numeric types

TrueCoders

Value Types

Examples:

- **Structs -** A struct (short for "structure") is a composite data type that groups together variables of different data types under a single name.
- **Enums -** Enum (short for "enumeration") is a user-defined data type that consists of integral constants. An enumeration provides a way to assign symbolic names to a set of distinct integer values.
- **Bools -** A bool (short for "boolean") is a data type that can have one of two values, either `true` or `false`.
- **Chars -** A char (short for "character") represents a single character and is usually stored as a single byte in memory
- **and numeric types**

Reference Types

Examples:

- classes
- interfaces
- objects
- arrays
- and strings

TrueCoders

Reference Types

Examples:

- **Classes -** A class is a blueprint from which objects are created
- **Interfaces -** An interface acts as a contract that defines a set of abstract methods that the implementing class must define.
- **Objects -** An object is an instance of a class.
- **Arrays -** An array is a collection of items (elements) stored at contiguous memory locations.
- **and Strings -** A string is a sequence of characters.

char

```
char                  'A'
```

char

```
char              'A'
```

enum

```
public enum

```

enum

```
public enum

```

string

```
string            "Tim Corey"
```

string

```
string                  "Tim Corey"
```

int

```
int             30
```

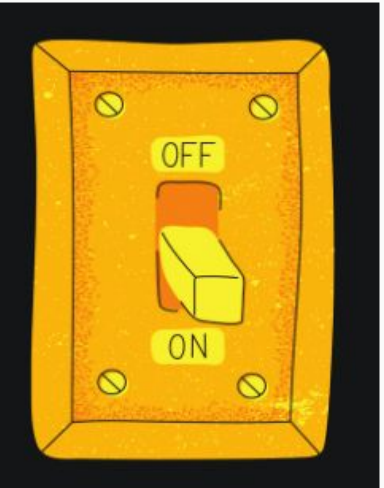
int

```
int          30
```

bool

```
bool                    false
```

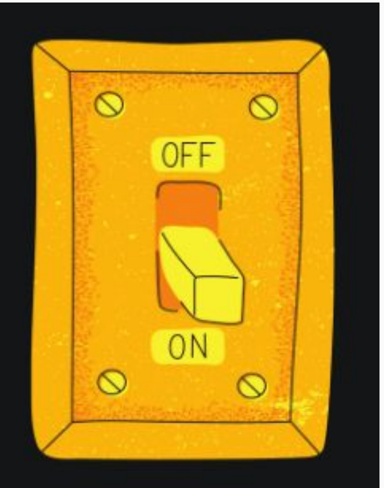
```
bool                    true
```



bool

```
bool                                 false
```

```
bool                                 true
```



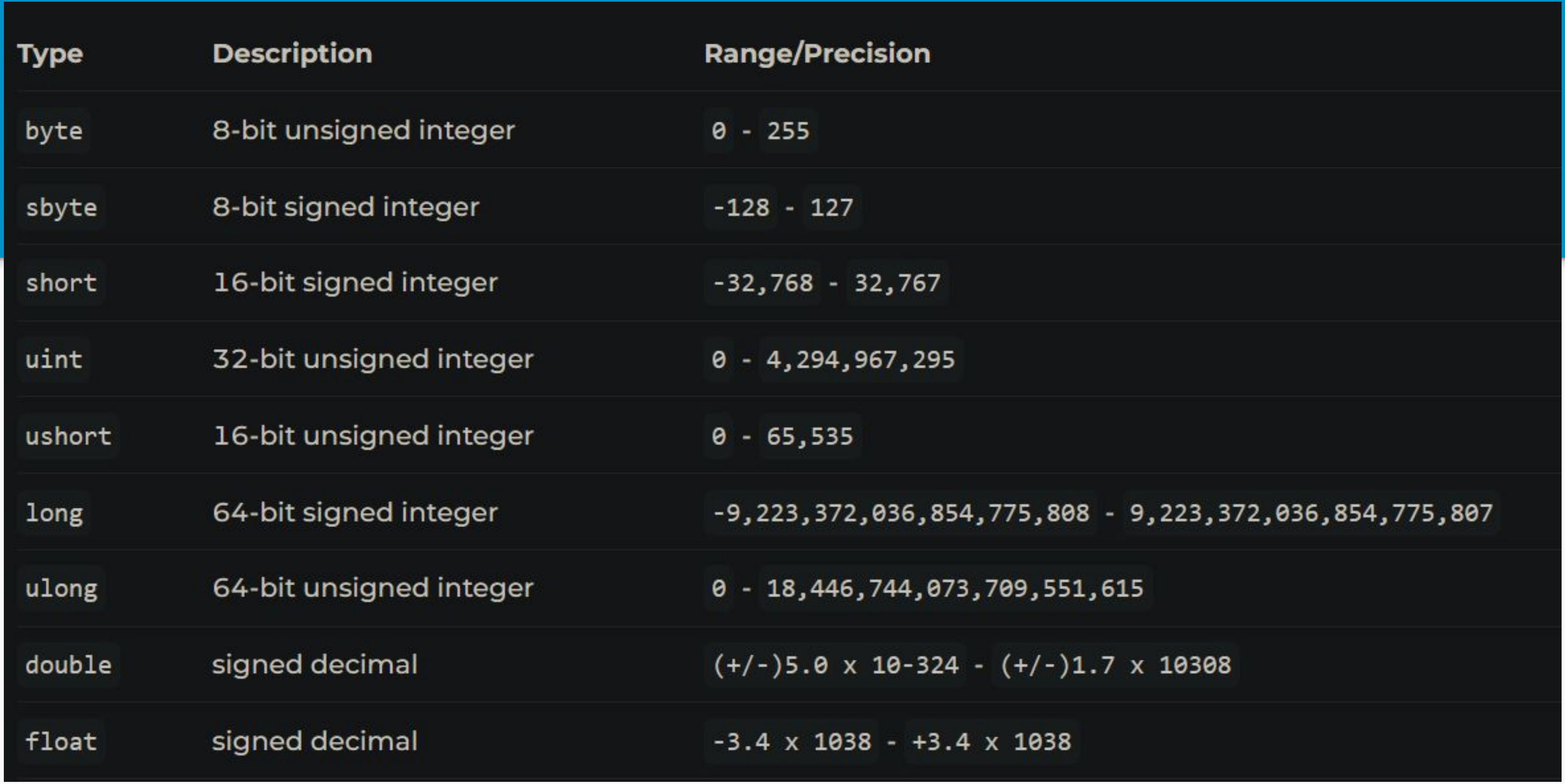
Numeric values

Range:

- The range of a data type indicates the minimum and maximum values that can be represented using that type.

Precision:

- Precision refers to the number of significant digits that a data type can represent reliably.



Type	Description	Range/Precision
`byte`	8-bit unsigned integer	`0` - `255`
`sbyte`	8-bit signed integer	`-128` - `127`
`short`	16-bit signed integer	`-32,768` - `32,767`
`uint`	32-bit unsigned integer	`0` - `4,294,967,295`
`ushort`	16-bit unsigned integer	`0` - `65,535`
`long`	64-bit signed integer	`-9,223,372,036,854,775,808` - `9,223,372,036,854,775,807`
`ulong`	64-bit unsigned integer	`0` - `18,446,744,073,709,551,615`
`double`	signed decimal	`(+/-)5.0 x 10-324` - `(+/-)1.7 x 10308`
`float`	signed decimal	`-3.4 x 1038` - `+3.4 x 1038`

Signed vs. Unsigned

Signed: A signed integer is one with either a plus or minus sign in front. That is it can be either positive or negative. **-7 , +7**

Unsigned: An unsigned integer is assumed to be positive. **7**

Important to know for memory

Whether a number is signed or unsigned determines how its bits are interpreted. A signed 8-bit number can represent values from -128 to 127, while an unsigned 8-bit number can represent values from 0 to 255.

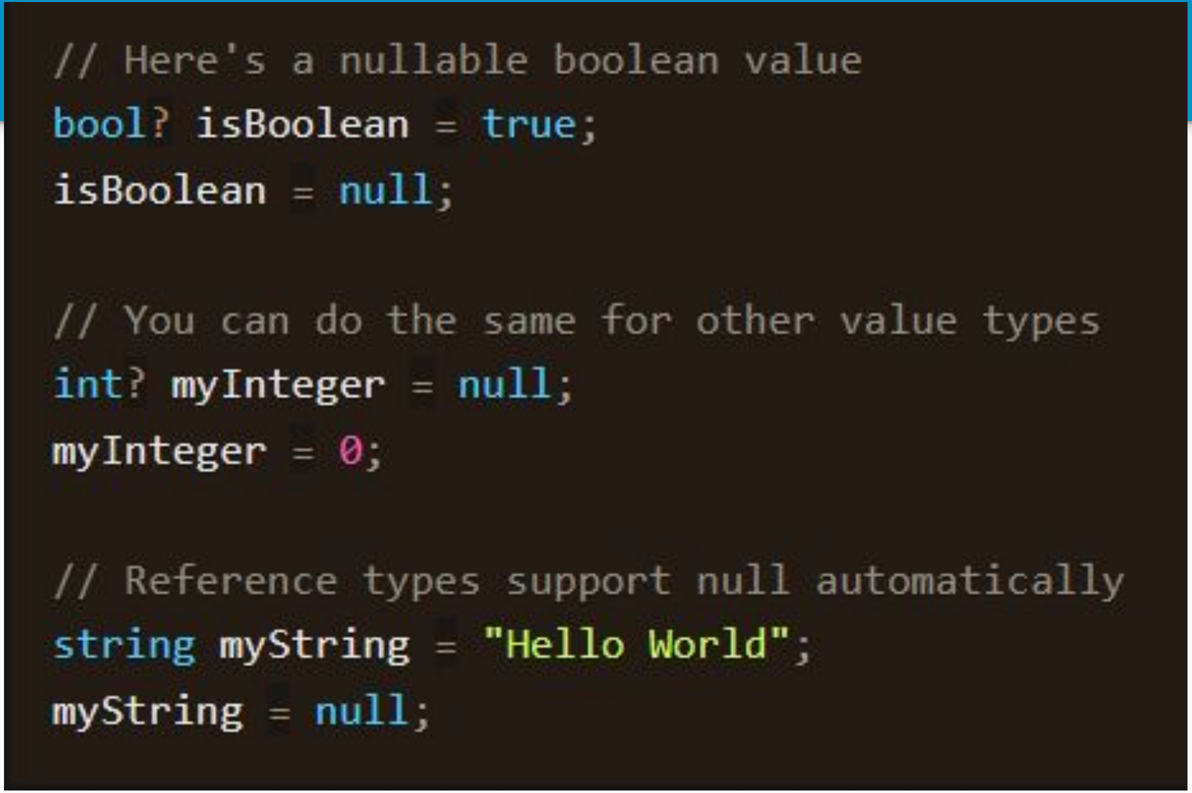
Null

In C#, the keyword `null` represents the absence of value.

Null

While reference types automatically support being set to `null`, value types require an actual value.

When you need to assign `null` to a value type, you employ the "nullable" of that type. A value type, followed by a `?` is shorthand syntax for nullable:



```
// Here's a nullable boolean value
bool? isBoolean = true;
isBoolean = null;

// You can do the same for other value types
int? myInteger = null;
myInteger = 0;

// Reference types support null automatically
string myString = "Hello World";
myString = null;
```

When you would need to make a value type nullable:

When dealing with databases. Maybe you have a column named Age to indicate the age of a dog, but the dog’s age is unknown. Instead of assigning it 0, you can assign the value as null.

Value and Reference Types Takeaways

- Variables are either a **Value Type** or a **Reference Type**
- C# is a **strongly** and **statically** typed Object Oriented Programming Language.
- Value types store the actual value
- Reference types store a reference to the value in memory
- Value types are stored on the **stack**
- Reference types are stored on the **heap**

Value and Reference Types Demo