C# Beginners Course

C# vs .NET

- .NET is a framework for building apps on Windows and is not limited to C#

.NET

- CLR (Common Language Runtime)
- Class Library

CLR

- C# compiler translates code into `IL Code` (intermediate language code) which is independent of machine code is running on
- CLR translates `IL Code` into code native to machine code is running on in a process called `JIT` (Just-in-time Compilation)

Architecture of .NET Applications

- Consists of building blocks of classes that collaborate with each other at runtime

Class
Data
Methods

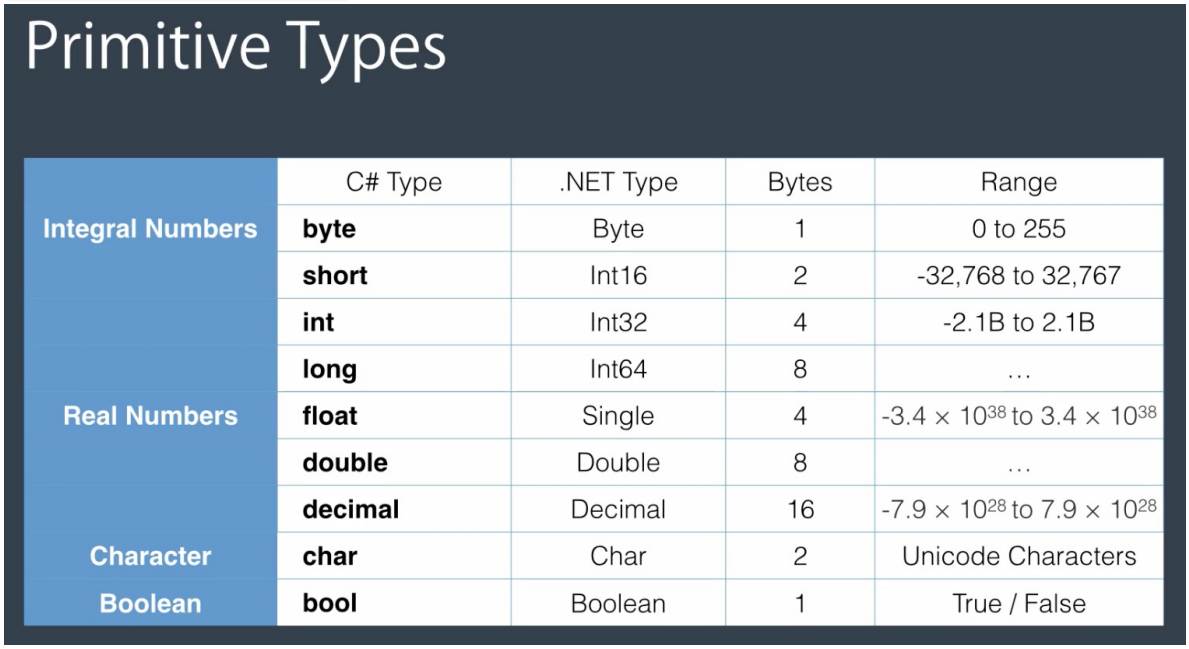
- A **Namespace** is a container of related classes
    - There are namespaces for working with databases, graphics, security, etc…
- An **Assembly** (DLL or EXE) is a container of related namespaces
- An **Application** is a collection of **Assemblies**

Variables and Constants

Primitives

- A **constant** is an immutable value

```
const float Pi = 3.14f;
```



- Add suffix `m` for decimals and `f` for floats

Non-Primitive Types

- String
- Array
- Enum
- Class

Overflowing

```
byte number = 255;
number = number + 1; // 0
```

- Since boundary of byte data type has been exceeded, it resets to `0` via overflowing
- C# does not perform overflow checking by default
- Use `checked` to check for overflowing:

```
checked
{
  byte number = 255;
```

```
  number = number + 1;
}
```

Scope

- Where a variable or constant has meaning and is accessible

Type Conversion

- Implicit type conversion
- Explicit type conversion (casting)
- Conversion between non-compatible types

Implicit Type Conversion

```
byte b = 1;   // 00000001
int i = b;    // 00000000 00000000 00000000 00000001
```

- Since no data is lost converting from a byte to int, **implicit** conversion can be used

Explicit Conversion

```
int i = 1;
byte b = (byte)i;
```

- Since data could be lost converting from `int` to `byte`, explicit conversion must be used

Operators

Postfix Increment

```
int a = 1;
int b = a++;

// a = 2 b = 1
```

Prefix Increment

```
int a = 1;
int b = ++a;

// a = 2 b = 2
```

Commenting

- Use comments to explain whys, hows, constraints, etc.
- Avoid using comments for 'Whats', as code should be self explanatory

Struct

- Similar to classes
- Rarely used
- Use a structure when wanting to define a small, light-weight object:

```
public struct RgbColor
{
  public int Red;
  public int Green;
  public int Blue;
}
```

Arrays

- A data structure used to store variables of the same type `int[] numbers = new int[3]`
  - Arrays have a fixed size in C#
  - Memory must be allocated for arrays
- Internally, arrays are just objects

Object Initialization Syntax

```
var person = new Person()
{
  FirstName = "Miller",
```

```
  LastName = "Anderson"
};

int numbers = new int[3] { 1, 2, 3 };
```

Strings

Using String Format

```
string name = string.Format("{0} {1}", firstName, lastName);
```

String Join

```
var numbers = new int[3] { 1, 2, 3 };

string list = string.Join(",", numbers);
```

String Elements

```
string name = "Miller";
char firstChar = name[0]; // M
```

Strings are Immutable

Verbatim Strings

```
string path = "projects\\project1\\folder1";

// OR

string path = @"c:\projects\project1\folder1"
```

string vs String

- String is a class within the .NET framework
- Similar to `Int32` vs `int`

Enums

- A data type representing a set of name/value pairs (constants)
- An `enum` is internally an integer

```
const int RegularAirMail = 1;
const int RegisteredAirMail = 1;
const int Express = 1;

// OR

public enum ShippingMethod
{
  RegularAirMail = 1,
  RegisteredAirMail = 2,
  Express = 3
}
```

- Enums can be created without explicitly declaring values:

```
public enum ShippingMethod
{
  RegularAirMail,
  RegisteredAirMail,
  Express
}
```

- First value is set to `0` automatically
- Best practice is to set values to id of records in database

Enums and Casting

```
var shipMethod = ShippingMethod.Express;
console.WriteLine(shipMethod); // Express
console.WriteLine((int)shipMethod); // 3
console.WriteLine(shipMethod.ToString()); // Express

var value = 2;
console.writeLine((ShippingMethod)value); // RegisteredAirMail

var methodName = "Express";
var shippingMethod = (ShippingMethod)Enum.Parse(typeof(ShippingMethod), methodName)
```

Reference Types and Value Types

- `Primitive` types are internally `Structures` in .NET
- `Arrays` and `Strings` are internally `Classes` in .NET
- `Structures` and `Classes` are treated differently in memory at runtime

Value Types

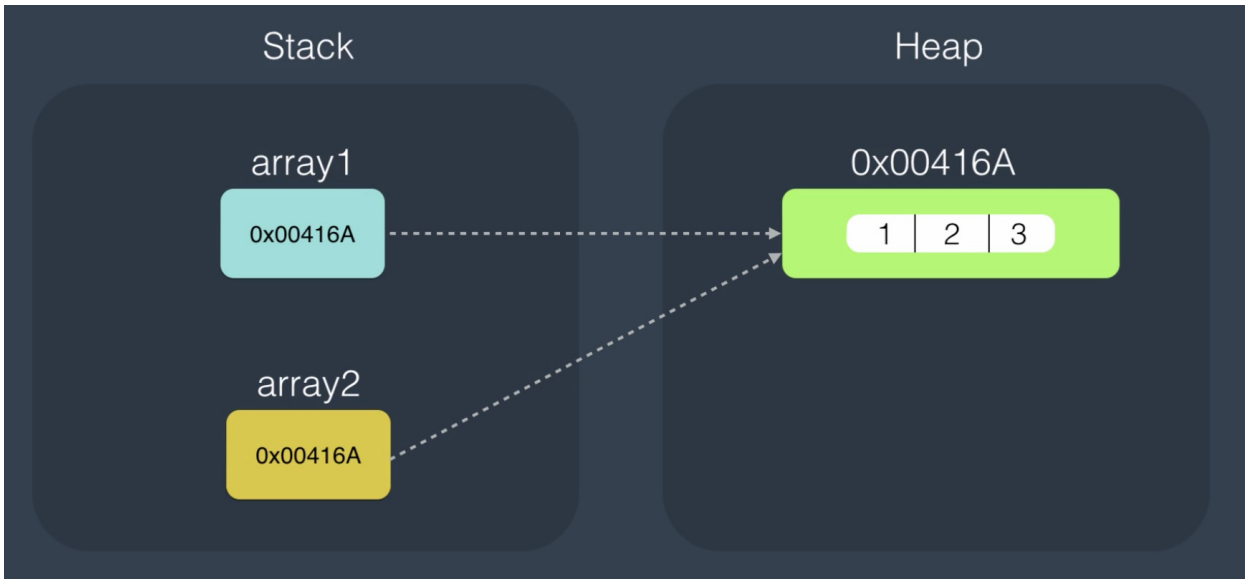
- Structures
- Allocated on `stack`
- Memory allocation done automatically
- Immediately removed when out of stack

Reference Types

- Classes
- Memory must be allocated manually (i.e. with `new` operator)
- Memory allocated on `heap`
- Garbage collected by `CLR` / runtime

```
var array1 = new int[3] {1, 2, 3};
var array2 = array1;
array2[0] = 4;

console.WriteLine(array1[0]); // 4
```



Control Flow

- `if / if else / else`
- `ternary operator`
- `switch / case`
  - `break`
- `for`
- `foreach`
- `while`
  - `continue`
  - `break`
- `do / while`

Random Class

- `Next()`
- `NextBytes()`
- `NextDouble()`

```
var random = new Random();
```

Arrays

- A fixed number of variables of a specific type

Multi Dimensional Arrays

Rectangular 2D

```
var matrix = new int[3, 5];
```

Rectangular 3D

```
var matrix = new int[3, 5, 2];
```

Jagged

```
var jaggedArr = new int[3][]
 {
     new int[4],
     new int[5],
     new int[3]
 };
```

Array Methods

- `Array.IndexOf()`
- `Array.Clear()`
- `Array.Copy()`
- `Array.sort()`
- `Array.Reverse()`

Lists

```
var numbers = new List<int>()
```

- Dynamically sized

Useful Methods

- `Add()`
- `AddRange()`
- `Remove()`
- `RemoveAt()`
- `IndexOf()`
- `LastIndexOf()`
- `Contains()`
- `Count`

DateTime

- Immutable
- Custom Format Strings

TimeSpan

```
var timeSpan = new TimeSpan(1, 0, 0); // OR var timeSpan = TimeSpan.FromHours(1);
```

Working with Text

- Strings are immutable

Useful Methods

- `ToLower()`
- `ToUpper()`
- `Trim()`
- `IndexOf()`
- `LastIndexOf()`
- `Substring()`
- `IsNullOrEmpty()`
- `IsNullOrWhiteSpace()`
- `Split()`
- `Replace()`

Format Strings

Format Specifier	Description	Example
c or **C**	Currency	123456 (C) -> $123,456
d or **D**	Decimal	1234 (D6) -> 001234
e or **E**	Exponential	1052.0329112756 (E) -> 1.052033E+003
f or **F**	Fixed Point	1234.567 (F1) -> 1234.5
x or **X**	Hexadecimal	255 (X) -> FF

String Builder

```
var builder = new StringBuilder();
```

- Defined in `System.Text`
- A mutable string
- Easy and fast to create and manipulate strings
- **NOT** for searching

Procedural Programming

- A programming paradigm based on procedure calls

Debugging

- `F5` -> Run in debug mode
- `F9` to add breakpoints
- `F10` to step over
- `F11` to step in

Defensive Programming

- `throw new ArgumentOutOfRangeException([parameter], [custom error message])`
- `throw new ArgumentNullException`