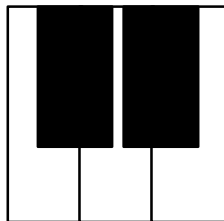


# Introduction to C#

# The C# Language

- ❖ C# was first announced in 2000, developed by Anders Heljsberg from Microsoft, alongside .NET and Visual Studio.
- ❖ Close to Java syntax, and the .NET framework enables the language to run on multiple OS' (like Java's JVM).
- ❖ C# gets its name from music; in music theory, the note C# is one semitone higher in pitch than the note C.



**Borland's Anders**

Version	.NET Framework	Visual Studio
C# 1.0	.NET Framework 1.0/1.1	Visual Studio .NET 2002
C# 2.0	.NET Framework 2.0	Visual Studio 2005
C# 3.0	.NET Framework 3.0/3.5	Visual Studio 2008
C# 4.0	.NET Framework 4.0	Visual Studio 2010
C# 5.0	.NET Framework 4.5	Visual Studio 2012/2013
C# 6.0	.NET Framework 4.6	Visual Studio 2013/2015
C# 7.0	.NET Core 2.0	Visual Studio 2017
C# 8.0	.NET Core 3.0	Visual Studio 2019
C# 9.0	.NET Core 5.0	Visual Studio 2019
C# 10.0	.NET Core 6.0	Visual Studio 2022
C# 11.0	.NET Core 7.0	Visual Studio 2022

<https://learn.microsoft.com/en-us/dotnet/csharp/whats-new/csharp-version-history>

# Build it with .NET



## Web

Build web apps and services for macOS, Windows, Linux, and Docker.



## Mobile and desktop

Use a single codebase to build native apps for Windows, macOS, iOS, and Android.



## Cloud

Build scalable and resilient cloud-native apps that run on all major cloud providers.



## Microservices

Create independently deployable microservices that run on Docker containers.

[Machine learning →](#)

[Game development →](#)

[Internet of Things →](#)

[Mobile →](#)

[Desktop →](#)

[Front-end web →](#)

[Back-end APIs →](#)

## Enhance your .NET experience

Explore further tools



### Visual Studio

Fully-featured integrated development environment (IDE) on Windows for building every type of .NET application.

[Download Visual Studio →](#)



### Visual Studio Code

Develop on Linux, macOS, or Windows to build cross-platform websites and services. Install the **C# extension** to get the best experience.



### Visual Studio for Mac

Build native Android, iOS, macOS, and Windows apps with .NET MAUI, plus websites and services with ASP.NET Core.

[Download Visual Studio for Mac →](#)

# From 'unmanaged' C++ to 'managed' C#

---

C++ was designed to be a low-level platform neutral language (that is also an object-oriented language). C# was designed to be a somewhat higher-level component-oriented language.

C# is about 'letting go' of precise control, and letting the framework help you focus on the 'big picture'.

<https://learn.microsoft.com/en-us/archive/msdn-magazine/2001/july/c-csharp-what-you-need-to-know-to-move-from-c-to-csharp>

# No access to memory addresses C#

---

For example, in C++ you have tremendous control over the creation and even the layout of objects. You can create an object on the stack, on the heap, or even in a particular place in memory using the placement operator 'new'.

With the managed environment of .NET, you give up that level of control. When you choose the type of your object, the choice of where the object will be created is implicit. Simple types (ints, doubles, and longs) are always created on the stack, and classes always on the heap. You cannot control where on the heap an object is created, you can't get its address, and you can't pin it down in a particular memory location.

<https://learn.microsoft.com/en-us/archive/msdn-magazine/2001/july/c-csharp-what-you-need-to-know-to-move-from-c-to-csharp>

# 'Garbage collection' in C#

---

You no longer truly control the lifetime of your object. C# has no destructor. The garbage collector will take your item's storage back sometime after there are no longer any references to it, but finalisation is nondeterministic.

C# did not have multiple inheritance nor templates. Multiple inheritance is hard to implement efficiently in a managed, garbage-collected environment, and because generics have not been implemented in the framework.

<https://learn.microsoft.com/en-us/archive/msdn-magazine/2001/july/c-csharp-what-you-need-to-know-to-move-from-c-to-csharp>

# Common Language Runtime (CLR) Types

---

The C# simple types map to the underlying common language runtime (CLR) types. For example, C# `int` maps to a `System.Int32`. The types in C# are determined not by the language, but by the common type system. In fact, if you want to preserve the ability to derive C# objects from Visual Basic objects, then restrict to the common language subset – those features shared by all .NET language.

<https://learn.microsoft.com/en-us/archive/msdn-magazine/2001/july/c-csharp-what-you-need-to-know-to-move-from-c-to-csharp>



# Common Language Runtime (CLR) Types

---

The managed environment and CLR bring a number of tangible benefits. In addition to garbage collection and a uniform type system across all .NET languages you get a greatly enhanced component-based language, which supports versioning and provides extensible metadata, available at runtime through reflection.

There is no need for special support for late binding; type discovery and late binding are built into the language. In C#, enums and properties are first-class members of the language, fully supported by the underlying engine, as are events and delegates (type-safe function pointers).

<https://learn.microsoft.com/en-us/archive/msdn-magazine/2001/july/c-sharp-what-you-need-to-know-to-move-from-c-to-csharp>

# Introduction to C#

---

Significant Syntax differences:

- In C# everything extends from the class 'Object'.
- C# has no destructor, no multiple inheritance, no templates but does have generics.
- C# has 'garbage collection', which automatically releases memory as opposed to C++ which gives users that 'control'.

# .NET Build



**.NET Release Candidate (RC)** Want to try out the latest RC release? .NET 8.0.0-rc.1 is available.

[Get .NET RC](#)

Free. Cross-platform. Open source.

# Download .NET

## For macOS

### .NET 7.0

Standard Term Support

Recommended

.NET SDK x64 (Intel)

.NET SDK Arm64 (Apple Silicon)

Version 7.0.11, released September 12, 2023

[All .NET 7.0 downloads](#) [All .NET versions](#)

### .NET 6.0

Long Term Support

.NET SDK x64 (Intel)

.NET SDK Arm64 (Apple Silicon)

Version 6.0.22, released September 12, 2023

[All .NET 6.0 downloads](#)

# Build process – Compiler or Interpreter?

---

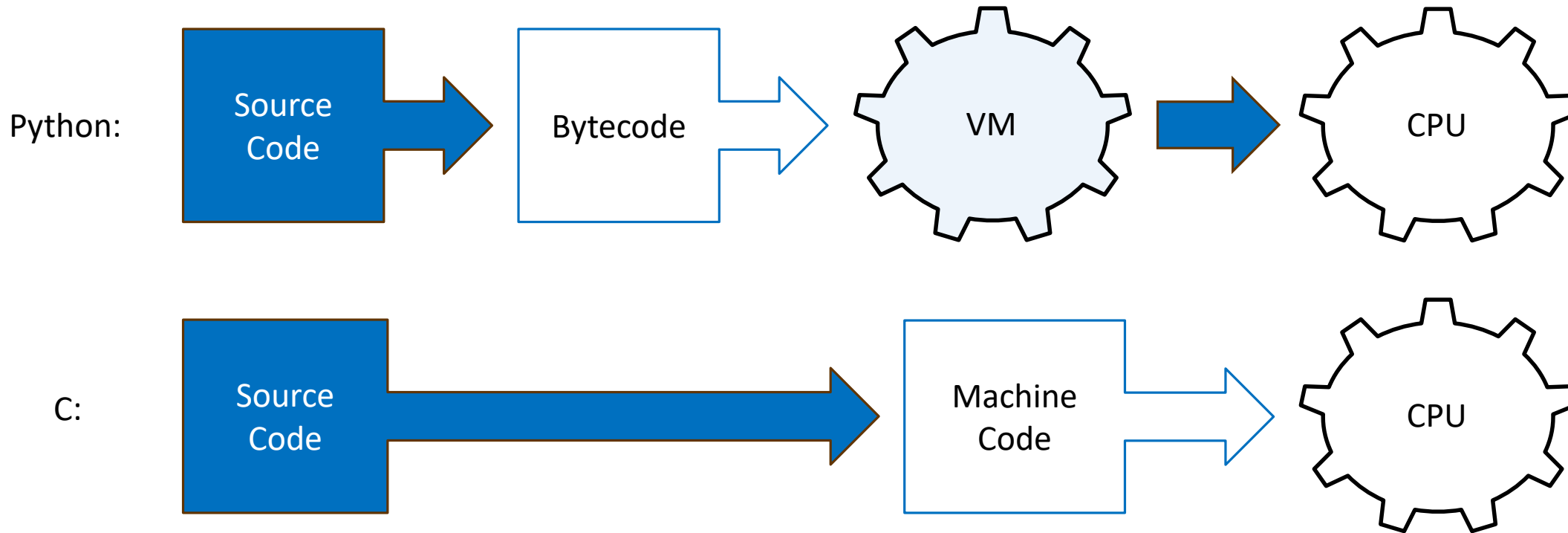
The .NET Framework was previously intended for Windows, but .NET Core is cross platform.

The .NET build process is different to the exclusive C/C++ compilation process (1970s/80s) and interpretation processes of Python (of the late 80s/90s).

The build process in .NET converts the source code and resources into an executable or library (via the CLR and JIT) that can be run by .NET runtime.

# Comparison: Python and C

---



# Build process

---

Source code – C#, F#, Visual Basic.NET

1 Compilation – the source code is compiled into Immediate Language (IL) code (which is 'platform-agnostic'). Achieved by the .NET compiler (csc.exe C Sharp Compiler) or vbc.exe (Visual Basic Compiler)

2 Just In Time (JIT) Compilation: IL is executed by the Common Language Runtime (CLR). This includes a Just In Time (JIT) compiler that converts the IL code into native machine code when the .NET application is run (hence 'just in time').

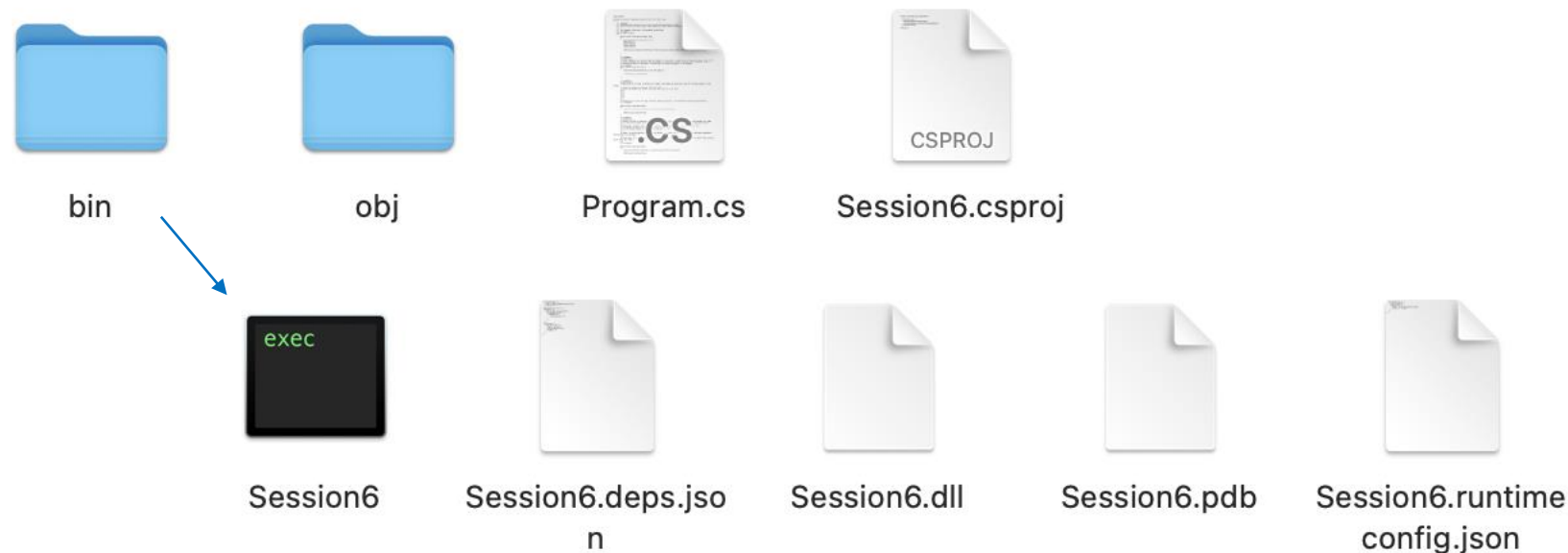
3 Execution: Once IL has been compiled by JIT into native code, it's then executed by the CPU. CLR does manage program execution, memory management (automatically – garbage collection), exceptions and security.

4 BCL Libraries - .NET also provides a set of libraries in the Base Class Library. Which contain pre-built classes for file I/O, networking, database access etc.

# Output Directory

---

Compiled assemblies are placed in a specified output directory. By default – the 'bin' folder with project directory.



Execution / Deployment - .exe in Powershell or .dll (dynamic link library) to refer to in other projects.



# C# language vs C/C++

# Exploration of the differences

---

In the following section we shall explore some of the differences between the C# and C/C++ languages.

It is not intended to be an exhaustive coverage of all C# language features, but highlight some of the key differences in approach.

The intention is to also reinforce the core programming concepts. Same concepts, different syntax/expression across language.

# Output

## Output (Hello World) in C# vs Python



```
using System;
namespace Project
{
    public class Program
    {
        public static void Main(string[] args)
        {
            Console.WriteLine("Hello World!");
        }
    }
}
```



```
def main():
    print("Hello, World!")

if __name__ == "__main__":
    main()
```

## Output (Hello World) in C# vs C



```
using System;
namespace Project
{
    public class Program
    {
        public static void Main(string[] args)
        {
            Console.WriteLine("Hello World!");
        }
    }
}
```



```
#include <stdio.h>

int main() {
    printf("Hello, World!\n");
    return 0;
}
```

## Output (Hello World) in C# vs Python



```
using System;
namespace Project
{
    public class Program
    {
        public static void Main(string[] args)
        {
            Console.WriteLine("Hello World!");
        }
    }
}
```



```
#include <iostream>
using namespace std;
int main() {
    cout << "Hello, World!" << endl;
    return 0;
}
```

# Input

## String input in C# vs C++



```
using System;
namespace Project
{
    public class Program
    {
        public static void Main(string[] args)
        {
            Console.Write("Enter input: ");
            string input = Console.ReadLine();
            Console.WriteLine("You entered " + input);
        }
    }
}
```



```
#include <iostream>
using namespace std;

int main() {
    string input;
    cout << "Enter input: ";
    cin >> input;
    cout << "You entered " << input << endl;
    return 0;
}
```



## Numerical input in C# vs C++



```
using System;
namespace Project
{
    public class Program
    {
        public static void Main(string[] args)
        {
            Console.Write("Enter number: ");
            string input = Console.ReadLine();
            int number = Convert.ToInt32(input);
            Console.WriteLine(number);
        }
    }
}
```



```
#include <iostream>
using namespace std;

int main() {
    int number;
    cout << "Enter number: ";
    cin >> number;
    cout << number << endl;
    return 0;
}
```



```
#include <stdio.h>
```

```
int main() {  
    int number;  
    printf("Enter an integer: ");  
    scanf("%d", &number);  
    printf("You entered: %d\n", number);  
    return 0;  
}
```



```
number = int(input("Enter an integer: "))  
print("You entered:", number)
```

# Naming Convention

# C# Language

C# - Pascal Case  
Java - Camel Case

```
/// <summary>
/// This is a comment describing the
/// main purpose of the class
/// </summary>
0 references
public class Car
{
    // Attributes

    private string make;

    private string model;

    private string colour;

    // Methods
```

**Encapsulation:**  
private attributes  
Public methods

```
// Methods

0 references
public void ChangeGear()
{
}

0 references
public void Accelerate()
{
}

0 references
public void Brake()
{
} // end of method

} // end of class
```

# Microsoft Naming Conventions 2008

General  
Naming  
Conventions

Names of  
Classes

Names of  
Properties &  
Methods

# C# Naming Convention

BNV 2021

C# is case sensitive

All names must start with a letter

Method names start with a capital letter

Class names start with a capital (Pascal)

Names can contain letters, digits and underscores

# Naming Continued



Object names start with lower case



Attribute (private variable) names start with lower case letter (camel)



Start with or contain **Nouns**

Classes  
Objects  
Attributes



Methods start with **Verbs**  
(or imply verbs)

# Use Block Comments

```
namespace C0453_DerekConsoleApps.App01
{
    /// <summary>
    /// This class offers methods for converting a given
    /// distance measured in miles to the equivalent
    /// distance measured in feet
    /// </summary>
    /// <author>
    /// Derek Peacock version 0.1
    /// </author>
    0 references
    public class DistanceConverter
    {
    }
}
```

- Block comments are multiline
- Generates XML API documentation
- Every class
- Every method

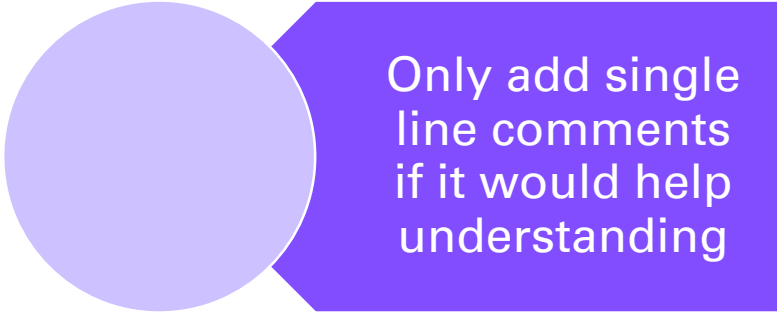
API documentation is for other programmers to read



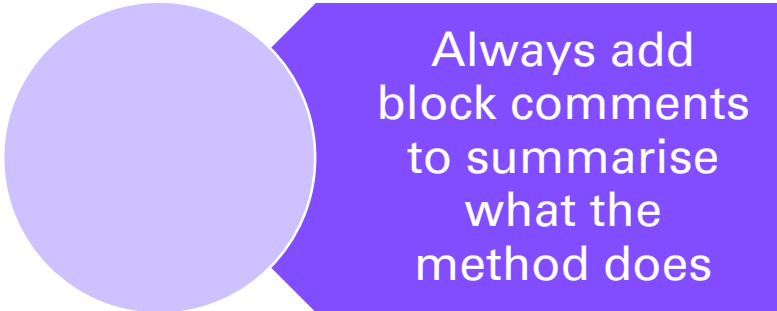
# Block and Single Line Comments

```
public class DistanceConverter
{
    // Distance measured in miles
    private double miles;
    // Distance measured in feet
    private double feet;

    /// <summary>
    /// This method will input the distance measured in miles
    /// calculate the same distance in feet, and output the
    /// distance in feet.
    /// </summary>
    0 references
    public void Run()
    {
        InputMiles();
        CalculateFeet();
        OutputFeet();
    }
}
```



Only add single line comments if it would help understanding



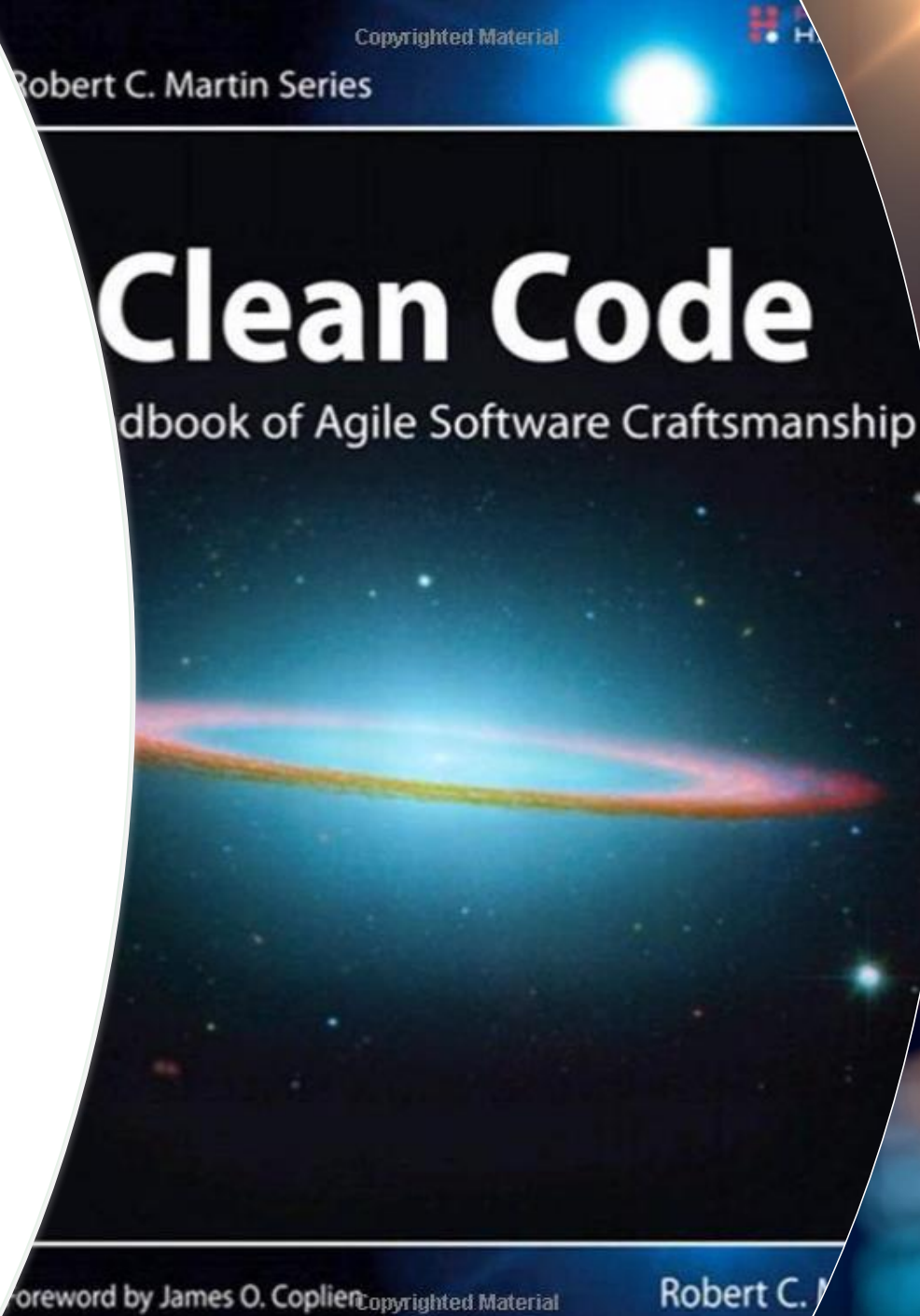
Always add block comments to summarise what the method does

# SOME C# DATA TYPES

Data Type	Description	Range
int	Whole numbers	-2.1 Billion to +2.1 Billion
Float (single)	Floating decimal point	6-7 Digits
double	Floating decimal point	15 Digits
bool	Logical value	true   false
char	Single character	Use single quotes marks 'A'
string	Sequence of chars	Use double quotes "Hello"
BNU 2020		34

# References

- [2008 Microsoft Naming Conventions](#)
- [2019 Microsoft Architectural Principles](#)
- [2011 Best Practice Readable Code](#)
- [Refactoring](#)
- [C# If Statement](#)
- [Switch Statement](#)
- [Agile Alliance](#)



```
mirror_mod = modifier_ob.  
#set mirror object to mirror  
mirror_mod.mirror_object =  
operation == "MIRROR_X":  
mirror_mod.use_x = True  
mirror_mod.use_y = False  
mirror_mod.use_z = False  
operation == "MIRROR_Y":  
mirror_mod.use_x = False  
mirror_mod.use_y = True  
mirror_mod.use_z = False  
operation == "MIRROR_Z":  
mirror_mod.use_x = False  
mirror_mod.use_y = False  
mirror_mod.use_z = True  
  
#selection at the end -add  
mirror_ob.select= 1  
modifier_ob.select=1  
context.scene.objects.active  
("Selected" + str(modifier_ob.  
mirror_ob.select = 0  
= bpy.context.selected_object  
data.objects[one.name].select  
  
print("please select exactly  
  
-- OPERATOR CLASSES --  
  
types.Operator):  
X mirror to the selected  
object.mirror_mirror_x"  
mirror X"  
  
context):  
context.active_object is not
```

# Constants

---

Unlike variables, constants are intended for values which do not 'vary' or 'change' throughout the lifetime of the program.

Constants can be defined with the prefix **const** keyword (which is the equivalent of **final** in Java).

```
const int HOURS_IN_DAY = 24;
```

```
const int MAX_MARK = 100;
```

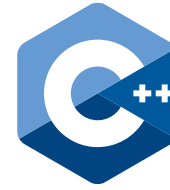
```
const int ADULT_AGE = 18;
```

# Selection

## If statements in C# vs C++



```
...  
int number = 5;  
if (number > 0)  
{  
    Console.WriteLine("The number is positive.");  
}  
else if (number < 0)  
{  
    Console.WriteLine("The number is negative.");  
}  
else  
{  
    Console.WriteLine("The number is 0.");  
}  
...
```



```
int main() {  
    int number = 5;  
    if (number > 0){  
        cout << "The number is positive." << endl;  
    }  
    else if (number < 0){  
        cout << "The number is negative." << endl;  
    }  
    else{  
        cout << "The number is 0." << endl;  
    }  
    return 0;  
}
```



```
int main() {  
    int number = 5;  
    if (number > 0){  
        printf("The number is positive.\n");  
    }  
    else if(number < 0){  
        printf("The number is negative.\n");  
    }  
    else{  
        printf("The number is 0.\n");  
    }  
    return 0;  
}
```



```
number = 5  
if number > 0:  
    print("The number is positive.")  
elif (number < 0):  
    print("The number is negative.")  
else:  
    print("The number is 0.")
```

# switch statement

---

The switch statement selects between cases

```
char grade = 'C';  
switch(grade) {  
    case 'A' : Console.WriteLine("You achieved an A grade"); break;  
    case 'B' : Console.WriteLine("You achieved a B grade"); break;  
    case 'C' : Console.WriteLine("You achieved a C grade"); break;  
    case 'D' : Console.WriteLine("You achieved a D grade"); break;  
    case 'F' : Console.WriteLine("You failed this course"); break;  
    default : Console.WriteLine("Invalid grade.");  
}
```



# Ternary operator / arithmetic if

---

The arithmetic if operator takes three arguments and is a more condensed way of selecting between a true and false option.

```
int a = 18;
```

```
int b = 7;
```

```
int highest;
```

```
highest = (a > b) ? a : b; //assign highest
```

```
Console.WriteLine("The highest number is " + highest);
```

# Iteration



```
using System;
namespace Project
{
    public class Program
    {
        public static void Main(string[] args)
        {
            for(int i = 0; i < 5; i++)
            {
                Console.WriteLine(i);
            }
        }
    }
}
```



```
#include <iostream>
using namespace std;

int main() {
    for(int i = 0; i < 5; i++) {
        cout << i << endl;
    }
    return 0;
}
```

for each in C# vs C++



```
using System;
namespace Project
{
    public class Program
    {
        public static void Main(string[] args)
        {
            int[] numbers = {1,2,3,4,5};
            foreach(int i in numbers)
            {
                Console.WriteLine(numbers[i]);
            }
        }
    }
}
```



```
#include <iostream>
using namespace std;

int main() {
    int numbers[5] = {1,2,3,4,5};
    for (int i : numbers){
        cout << numbers[i] << endl;
    }
    return 0;
}
```



```
#include <iostream>
using namespace std;

int main() {
    int numbers[5] = {1,2,3,4,5};
    for (int i : numbers){
        cout << numbers[i] << endl;
    }
    return 0;
}
```



```
for i in range(1,6):
    print(i)
```

# C# do while loop

---

The do while loop repeats whilst true, but the comparison takes place after one iteration of the loop. So loop repeats at least once.

```
char response;  
do  
{  
    // program instructions go here  
    Console.Write("another go (y/n)?");  
    response = Console.ReadLine();  
}  
while (response == 'y');
```

# Functions



```
...
public class Program
{
    public static void Main(string[] args)
    {
        SayHello();
    }
    public static void SayHello()
    {
        Console.WriteLine("Hello World");
    }
}
}
```



```
#include <stdio.h>
```

```
void sayHello(); //declaration
```

```
int main() {
    sayHello();
    return 0;
}
```

```
void sayHello(){ //definition
    printf("Hello, World!\n");
}
```





```
...
public class Program
{
    public static void Main(string[] args)
    {
        SayHello();
    }
    public static void SayHello()
    {
        Console.WriteLine("Hello World");
    }
}
}
```



```
def say_hello():
    print("Hello, World!")

say_hello()
```

# Arrays

for each in C# vs C++



```
using System;
namespace Project
{
    public class Program
    {
        public static void Main(string[] args)
        {
            int[] numbers = {1,2,3,4,5};
            foreach(int i in numbers)
            {
                Console.WriteLine(numbers[i]);
            }
        }
    }
}
```



```
#include <iostream>
using namespace std;

int main() {
    int numbers[5] = {1,2,3,4,5};
    for (int i : numbers){
        cout << numbers[i] << endl;
    }
    return 0;
}
```



```
using System;
namespace Project
{
    public class Program
    {
        public static void Main(string[] args)
        {
            int[] numbers = {1,2,3,4,5};
            foreach(int i in numbers)
            {
                Console.WriteLine(numbers[i]);
            }
        }
    }
}
```



```
myList = [1,2,3,4,5]

print(myList)
```



```
using System;
namespace Project
{
    public class Program
    {
        public static void Main(string[] args)
        {
            Object[] myObjArray = new Object[5] { ... };
            Array myArray = Array.CreateInstance(typeof(int), 5);
            myArray.SetValue(42, 0); //value, position
            int val1 = (int)myArray.GetValue(0);
        }
    }
}
```



```
myList = [1,2,3,4,5]

print(myList)
```

# Classes



## Classes in C# vs C++



```
class Student
{
    private int id;
    private string name;

    public Student(int id, string name)
    {
        this.id = id;
        this.name = name;
    }
}
```

```
class Student{
    private:
        int id;
        string name;

    public:
        Student(int id, string name){
            this.id = id;
            this.name = name;
        }
}
```



```
class Student
{
    private int id;
    private string name;

    public Student(int id, string name)
    {
        this.id = id;
        this.name = name;
    }
}
```



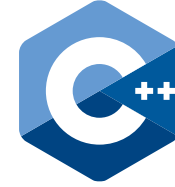
```
class Student:
    def __init__(self, id, name):
        self.id = id
        self.name = name
```



# Objects



```
using System;
namespace Project
{
    public class Program
    {
        public static void Main(string[] args)
        {
            Student nick = new Student(1234, "Nick");
            nick.print();
        }
    }
}
```



```
#include <stdio.h>
#include <Student.h>

int main() {
    Student *nick = new Student(1234, "Nick");
    nick->print();
    return 0;
}
```



## Objects in C# vs Python



```
using System;
namespace Project
{
    public class Program
    {
        public static void Main(string[] args)
        {
            Student nick = new Student(1234, "Nick");
            nick.print();
        }
    }
}
```

```
from student import Student
```

```
nick = Student(1234, "Nick")
nick.print()
```



```
#include <stdio.h>
#include <Student.h>

int main() {
    Student nick(1234, "Nick");
    nick.print();
    return 0;
}
```



```
from student import Student
```

```
nick = Student(1234, "Nick")
nick.print()
```

# Inheritance



```
class Child : Parent
{
    private int id;
    private string name;

    public Child(int id, string name)
    {
        base(id); //call parent constructor
        this.name = name;
    }
}
```

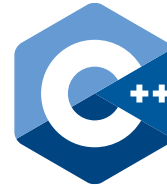


```
class Child(Parent):
    def __init__(self, id, name):
        super().__init__(id)
        self.name = name
```



```
class Child : Parent
{
    private int id;
    private string name;

    public Child(int id, string name)
    {
        base(id); //call parent constructor
        this.name = name;
    }
}
```



```
class Child : public Parent{
    private:
        int id;
        string name;

    public:
        Child(int id, string name): Parent(id){
            //explicit call Parent constructor
            this.name = name;
        }
}
```

# Summary



# Summary

---

C# was developed in throughout the 1990s by Anders Hejlsberg at Microsoft and was first announced in 2000. The language has evolved alongside the .NET Framework/Core and versions of Visual Studio.

Although Anders Hejlsberg doesn't agree, many would say that the syntax of C# is closer to Java, than the original C language and C++.

ASP.NET and .NET MAUI are recent and current ways to use C# in mobile application development.