POWERING POTENTIAL

The C# Language

Contents

Objectives

To introduce the .NET framework and the C# language

Take at look around Visual Studio

Contents

Brief history of .NET

Key framework features

Basic C# code construction

Your first application

What is the .NET framework?

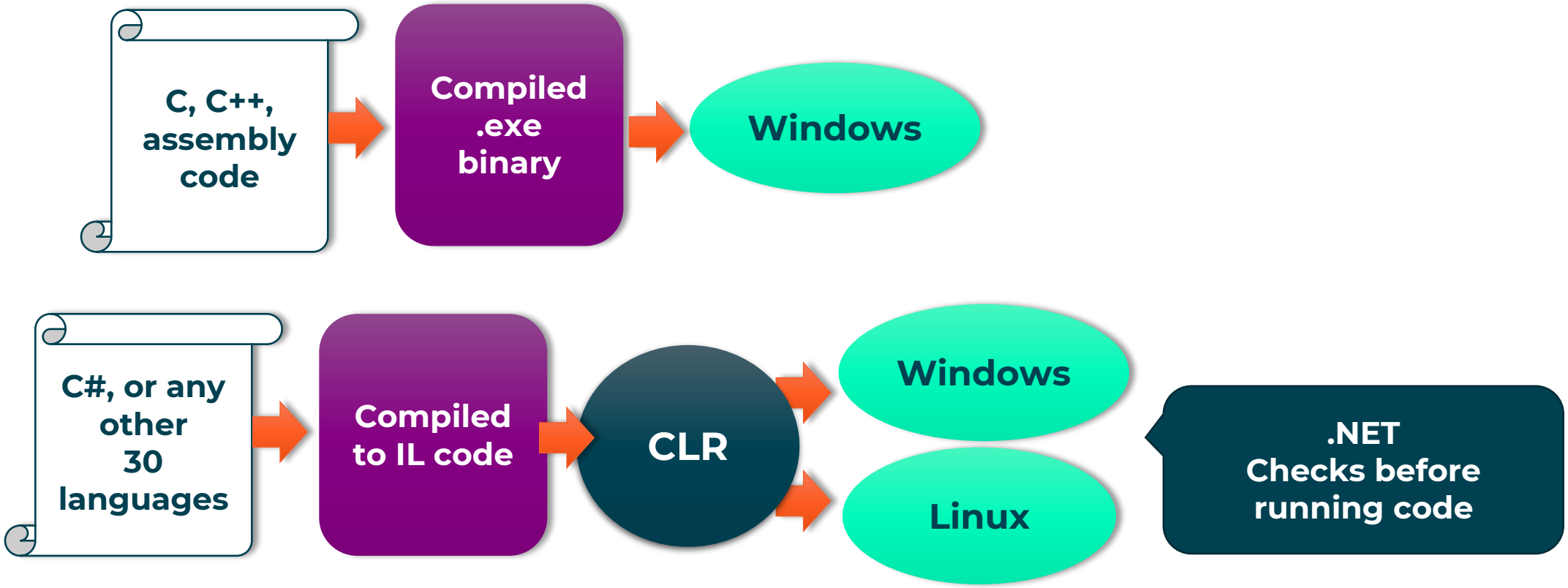
- Developed in the late 1990s, and originally called **Next Generation Windows Services** (NGWS).
  - By 2001 the first beta versions of .NET 1.0 were released.

- .NET supports many languages (e.g. C#, F#, Visual Basic, and dozens more)
- Compiled to Common Intermediate Language (**CIL**)
- **CIL** code is compiled for the target OS before running

- .NET introduced the idea of *managed code*.
- Code is managed and executed by the Common Language Runtime (**CLR**)
- But what is managed code?

What is managed code?

- Before .NET, there was no way of knowing what the compiled binary code would do inside the Windows OS
- The code was compiled for Windows OS only



POWERING POTENTIAL

C# Language basics

Namespaces - Used to group types and avoid naming conflicts

```
ŋăṇêṣřăçê QA.hr
{
  public class Timesheet
  {
      // code
  }
}
```

```
ŋăṇêṣřăçê QA.apprentice
{
  public class Timesheet
  {
      // code
  }
}
```

```
ŋăṇêṣřăçê ŘA

    řцčľîç çľăṣṣ Rsôĝsăṇ

        ṣʈăʈîç ŵôîđ Ňăîŋ ṣʈsîŋĝ   ăsĝṣ

            ŘA ḥs Ţîṇêṣḥêêʈ ʈ,   ŋêx ŘA ḥs Ţîṇêṣḥêêʈ
            ŘA ăřřsêŋʈîçê Ţîṇêṣḥêêʈ ʈ,   ŋêx ŘA ăřřsêŋʈîçê Ţîṇêṣḥêêʈ
```

The using statement

```
ụṣîŋĝ ŘA ăřřsêŋʈîçê

ŋăṇêṣřăçê ŘA


    řụčľîç çľăṣṣ Rsộĝsăṇ

        ṣʈăʈîç ŵộîđ Ňăîŋ ṣʈsîŋĝ   ăsĝṣ

              Ţîṇêṣḥêêʈ ʈ,   ŋêx Ţîṇêṣḥêêʈ
              ŘA ḥs Ţîṇêṣḥêêʈ ʈ,   ŋêx ŘA ḥs Ţîṇêṣḥêêʈ
```

```
ŘA ăřřsêŋʈîçê Ţîṇêṣḥêêʈ
```

File-scoped namespaces (C# 10+)

```
ụṣîŋĝ ŘA ǎřřsêŋʈîçê

ŋǎṇêṣřǎçê ŘA

řụčľîç çľǎṣṣ Rsộĝsǎṇ

        ṣʈǎʈîç ŵộîđ Ňǎîŋ ṣʈsîŋĝ   ǎsĝṣ

              Ţîṇêṣḥêêʈ ʈ,   ŋêx Ţîṇêṣḥêêʈ
              ŘA ḥs Ţîṇêṣḥêêʈ ʈ,   ŋêx ŘA ḥs Ţîṇêṣḥêêʈ
```

```
ŘA ǎřřsêŋʈîçê Ţîṇêṣḥêêʈ
```

Variables (Symbolic name for an address in memory)

- Must be declared with a type before use
- Variables in a method must be initialised before use

```
public void Main(string[] args)
{
  int myAge;
  bool answer = true;
  string myName = "Samantha";
  int i = 0, j;

  myAge = 21;
  Console.WriteLine(i);
  Console.WriteLine(j);     // not initialised
}
```

Pre-defined in-built primitive data types

```
byte        eightBit;

short       sixteenBit;

int         thirtyTwoBit;
long        sixtyFourBit;
```

`float`	**7** digits of precision
`double`	**16** digits of precision
`decimal`	**128-bit** floating-point number with a higher precision

```
char     initial;     // Unicode character (16 bits)

bool     isActive;    // can be set to true/false


initial = 'M';

isActive = true;
```

Standard mathematical operators

```
int x = 4;
x = x + 5;                    // x is 9

double d = 9.0;
d = d / 2;                    // d is 4.5


int y = ( x % 3 );            // y is  1
```

```
int x = 9;
x = x / 2;                    // x is 4!

double d = 9.0;
d = d / 2;                    // d is 4.5
```

+	addition
-	subtraction
*	multiplication
/	division
%	modulus division

Compound operators

- Each mathematical operator can be combined with “=”

```
int x = 4, y = 6;
x = x + y;          // x = 10
```

```
// Can be written as
x += y;
```

```
int x = 8;
x *= 2;             // x = 16
```

```
int x = 8;
Cộŋṣộľê ẄsîŧŷêĽîŋê(x % 5);     // displays 3 but x = 8

x %= 5;                         // x = 3
```

Pre & post-fix ++ and -- operators

```
int x = 0;
x++;                // x = 1
++x;                // x = 2
```

```
int x = 0;
int y = ++x;        // x=1, y=1
```

```
int x = 0;
int y = x++;        // x=1, y=0
```

```
int x = 1;
Cộnsộlê ẄsîŧêĽînê(x++);     // displays 1 and then x=2
Cộnsộlê ẄsîŧêĽînê(x);       // displays 2
```

Casting – implicit casting

```
int  x = 4;

long no = x;

double d = x;
```

```
char c = 'A';

int x = c;
```

65

Casting - Explicitly cast any numeric type to another type

```
double d = 4.5;
int  x = d;
```

✘

```
double d = 4.5;
int  x = (int) d;
```

```
long lng = 5;
int  x = lng;
```

✘

```
long lng = 5;
int  x = (int)lng;
```

```
float f = 4.5;
```

✘

```
float f = (float) 4.5;
      f = 6.75f;
```

```
bool b = 1;
```

✘

```
bool b = true;
```

Casting strings using the Parse and TryParse methods

```
string no = "123";
int x = (int)no;
```

```
string no = "123.45";
double d = (double)no;
```

```
int x    = int.Parse(no);

double d = double.Parse(no);

float f  = float.Parse(no);
```

```
ştsîŋĝ ŋộ   Cộŋşộĺê ŖêăđĹîŋê
îŋţ ŋ

îğ îŋţ ŢsỳRăsşê ŋộ  ộụţ ŋ
       Cộŋşộĺê ẄsîţêĹîŋê ŋ ,
```

If the cast is valid

POWERING POTENTIAL

Conditionals

Introducing ‘if’

```
int age = GetAge();

if ( age > 18 )
{
 // code for when over 18 years old
}
```

>	greater than
>=	greater than or equal to
<	less than
<=	less than or equal to
==	equal to
!=	not equal to

```
int age = GetAge();

if ( age < 18)
{
  // code for when under 18 years old
}
else
{
 // code for when 18 or over
}
```

Introducing ‘else if’

```
int mark; . . .

if ( mark > 80 )
{
   Cȩŋşȩľê ẄsîŧȩĽîŋê("Distinction");
}
else if( mark > 70 )
{
   Cȩŋşȩľê ẄsîŧȩĽîŋê("Merit");
}
else if( mark > 60 )
{
   Cȩŋşȩľê ẄsîŧȩĽîŋê("Pass");
}
else
{
   Cȩŋşȩľê ẄsîŧȩĽîŋê("Try again!");
}
```

'if' must come first

as many ‘else if'(s) as needed

'else’
(if needed) comes last

The ternary conditional operator (? :)

- Produces less code for simple if statements resulting in a value
- These two examples produce the same result.

```
double salary = GetSalary();

double rate = (salary < 21000) ? 0.2 : 0.4;
```

```
double salary = GetSalary();

if(salary < 21000)
{
    rate = 0.2;
}
else
{
    rate = 0.4;
}
```

Logical operators AND and OR

A

```
int var1 = 4, var2 = 2, var3 = 0;
if ((var1 > var2) && (var3 == 0))
{
        Cộŋşộľê ẄsîŧêĽîŋê( "Will we see this?" );
}
```

B

```
int var1 = 4, var2 = 6, var3 = 0;
if ((var1 > var2) || (var3 == 0))
{
        Cộŋşộľê ẄsîŧêĽîŋê( "Will we see this?" );
}
```

C

```
int var1 = 1, var2 = 2, var3 = 3;
if ((var1 == 1) || (var2 == 2) && (var3 == 1))
{
        Cộŋşộľê ẄsîŧêĽîŋê( "Will we see this?" );
}
```

&&	AND
||	OR
!	NOT

The switch statement

- Tests an integer, enum, char or String
- Statements may be in any order
- Often elegant alternative to if...else if... else
- Each case part must have a break

```
int no = 1, res;

switch ( no )
{
    case 0:
     res = 23;
     break;

    case 2:
     res = 8;
     break;

    case 1: case 2:
     res = 51;
     break;

    default:
     res = -1;
     break;
}
```

Methods

Example of a method with **no parameters and no return value**

```
public class Program
{
    public static void Main(string[] args)
    {
        UpdateAllSalaries();
    }

    private static void UpdateAllSalaries()
    {
        // Code to update all salaries
    }
}
```

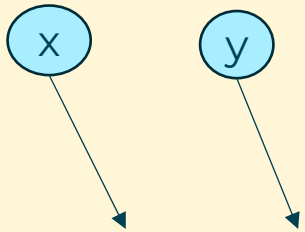
The caller just calls the method and let it do its work!

Example of a void method with two parameters

```
public class Program
{
    public static void Main(string[] args)
    {
        int x = 1, y = 2;

        Add(x, y);
        Add(3, 7);
    }

    private static void Add(int a, int b)
    {
         Cộŋṣộľê ẄsîŧŷêĽîŋê(a + b);
    }
}
```



Example of method returning a value

```
public class Program
{
    public static void Main(string[] args)
    {
        int x = 1, y = 2;
        int result = GetSum(x, y);
        Cõŋşõľê Ẅsîţêĺîŋê("Sum is = ", result);
    }

    private static int GetSum(int a, int b)
    {
        return a + b;
    }
}
```

A method can only return one thing
The caller decides what to do with the returned value

Method overloading

```
public class Program
{
    public static void Main(string[] args)
    {
        double result1 = GetTax(2000);
        Cộŋṣộľê ẄsîʈêĽîŋê(result1);

        double result2 = GetTax(2000, 0.4);
        Cộŋṣộľê ẄsîʈêĽîŋê(result2);
    }

    private static double GetTax(double salary)
    {
         return salary * 0.25;
    }

    private static double GetTax(double salary, double rate)
    {
        return salary * rate;
    }
}
```

500

800

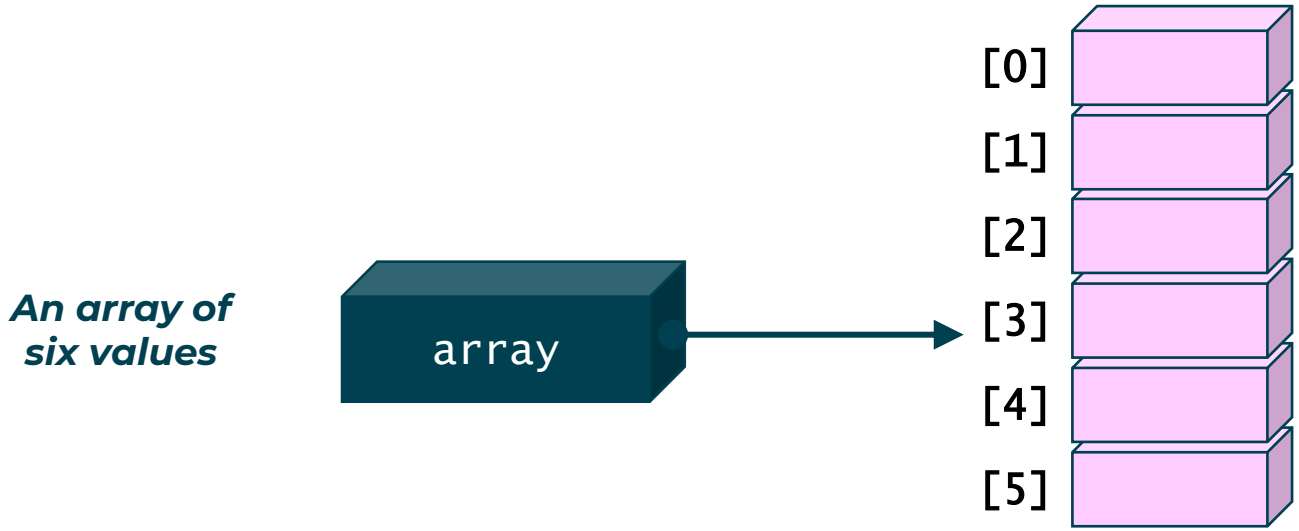
Same name
different parameters

Arrays and Loops

WHAT IS AN ARRAY?

- **An array can store a collection of variables all of the same type**
  - → Each array element can hold a single item (value/reference type)
  - → Array elements are accessed by index number, e.g. names[3]

- **Arrays are objects**
  - → Must be created before they can be used
  - → An array variable (of any type) is a reference type



Introduction to arrays

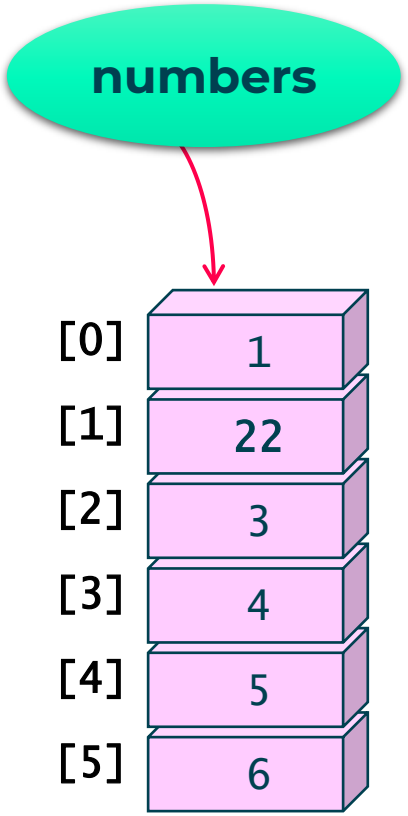
- You can change array elements
- Appending, inserting and removing elements is hard

```
int[] numbers = {1,2,3,4,5};

numbers[1] = 22;

Console.WriteLine(numbers[2]);
```

22



Create an empty array

```
int[] numbers = new int[5];
```

0
0
0
0
0

Initialised to default values for int

```
string[] names = new string[5];
```

null
null
null
null
null

Initialised to default values for String

LOOPS

- **Objectives**
  - To cover C#'s looping constructs

Iteration using while and for statements

```
while ( boolean_expression )
{
   statement(s);
}
```

```
do
{
   statement(s);
} while ( boolean_expression );
```

```
for ( init_expr; boolean_expr; update_expr )
{
   statement(s);
}
```

We'll see iteration using foreach loops later

Iteration using while loops

Declare/Init

Exit condition

```
int i = 0;
while (i < 10)
{
  Console.WriteLine( $"{i} squared = {i*i}" );
  i++;
}
```

Changing value

- or put the test condition at the end of the loop

```
int i = 0;
do
{
  Console.WriteLine( $"{i} squared = {i*i}" );
  i++;
} while (i < 10);
```

The for loop

Initialisation can include a declaration

- Declared variable is in scope only inside the loop

Initialisation | Exit condition | Changing

```
for ( int i = 0; i < 10; i++ )
{
   Console.WriteLine(i + "\n" );
}
```

what is the value of 'i' here?

Initialisation, exit condition, and update are a list of ';' separated expressions

```
for( int i = 0, j = 10; i < j; i++, j-- )
{
   Console.WriteLine( i * j + "\n");
}
```

Using break to exit any loop

```
double money = 50;          // £ pounds

for (int years = 1; money < 1000; years++)
{
        money += 100;

        Console.WriteLine($"Year {years}:\t {money}");

        double tax = money * 0.40;

        if (tax > 100)
        {
                Console.WriteLine("Tax is > 100");
                break;
        }
}
```

Break out of the current loop

```
Year 1:  150
Year 2:  250
Year 3:  350
Tax is > 100
```

continue

```
for (   ;   ;   )
{
   ...
   ...
        continue;
   ...

}
...
```

```
for ( int i = 0; i < 10; i++ )
{
  if ( i % 4 == 0)
  {
     // few statements
     continue;
  }
  // many statements
}
```

```
for ( int i = 0; i < 10; i++ )
{
  if ( i % 4 == 0)
  {
    // few statements
  }
  else
  {
    // many statements
  }
  // no code here!!
}
```

Equivalent

foreach loop - iterates over a collection or an array

```
public void ProcessNames( string[] names )
{
  foreach (string name in names )
  {
    Console.WriteLine( name );
  }
}
```

Read as: foreach String 'name' in the 'names' collection

```
string[] names = { "Bob", "Sasha“ };

foreach(string name in names)
{
        name += "x";
}
```

Compile time error

The elements are considered read-only

In this chapter we reviewed

Creating variables

Selection statements
if, else, else if and switch

Creating and using methods

Iteration statements
while, do while, for and foreach loops

LAB

Practice the basics of the C# language

Duration 1.5 hours

Operator Precedence

Order	Operators	Comments
1	( ) . f(x) [ ] x++ x- new	Primary
2	+ - ! ~ ++x --x (T)x	Unary
3	* / %	Multiplicative
4	+ -	Additive
5	<< >>	Bit Shift
6	< > <= >= instanceof	Relational
7	== !=	Equality
8	&	Bitwise AND
9	^	Bitwise exclusive OR
10	|	Bitwise inclusive OR
11	&&	Logical AND
12	||	Logical OR
13	?:	Conditional
14	= op=	Assignment

```
Vehicle v = ...;
((Car)v).OpenSunRoof();
```

Much more later ..

```
int p = x + (y * z);
int q = (x + y) * z;
bool b1, b2, b3;
if (b1 && b2 || b3) {
  //does this if b3 true
  //b1 and/or b2 might be
}
```

```
while((b = GetNextByte()) != -1) {...}
```