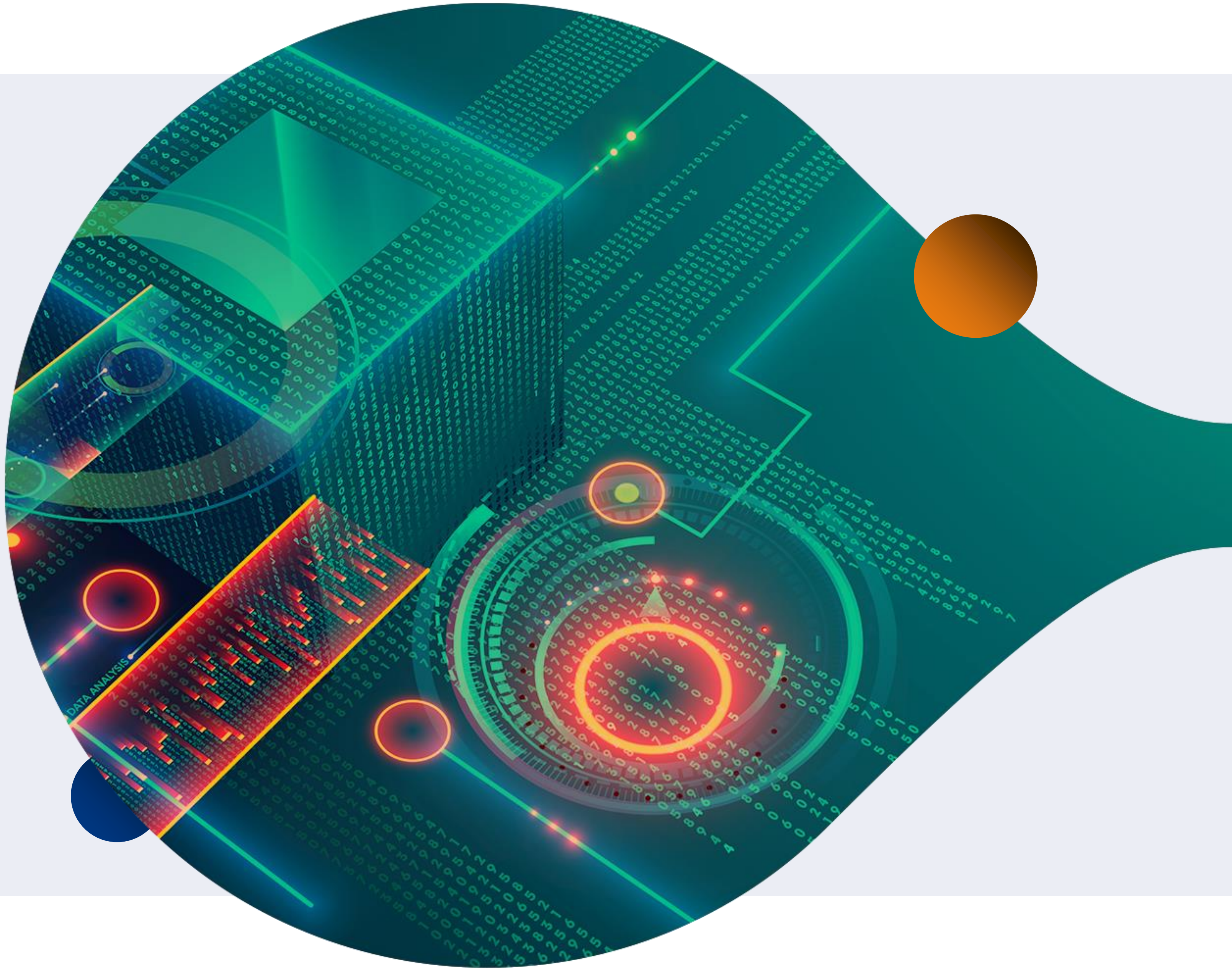
Digital Testing

Basic C#

An introduction to programming with C#



Introduction

Guidelines

- Welcome and introductions
- Course timings
- Breaks
- Phones / doorbell / distractions etc.
- Questions

Introduction

Course contents

- What is C# ?
- Installation and setup
- Syntax
- Variables and Data Types
- Operators
- Logic and Loops
- Methods
- Classes

Introduction

Aims and Objectives

- Understanding of core programming concepts
- Confidence to write your own code
- Hands on practice/experience

What is C# ?

Section 1

eurofins | Digital Testing

What is C# ?

- An object-oriented programming language created by Microsoft
- Can be used to develop Web, Desktop and Mobile applications and services
- Similar to C, C++ and Java
- Very popular programming language
- Easy to learn, simple to use
- Online community support

Installation and Setup

Section 2

eurofins | Digital Testing

Installation

We need to download and install an integrated development environment (IDE).

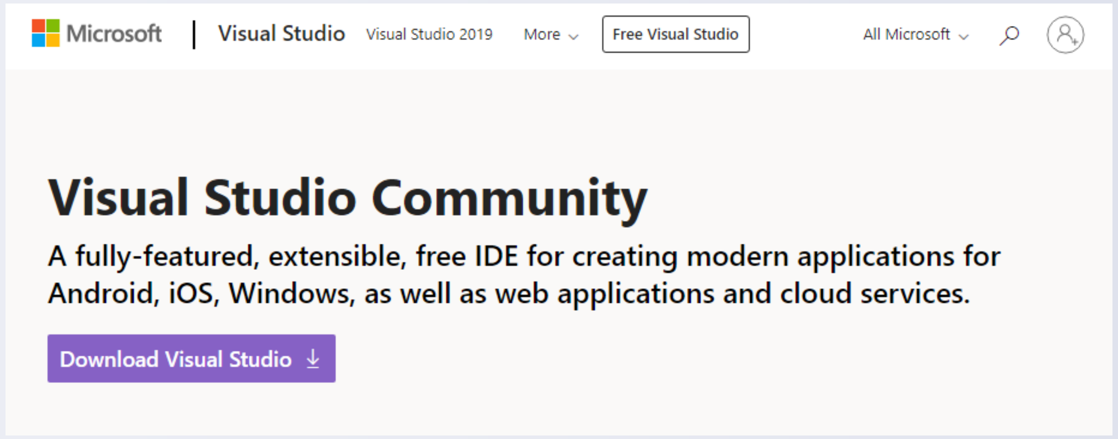
An IDE is used to write, edit and compile your code. We will be using Visual Studio Community edition.

We also need to install the .NET framework, but this can be done during the installation of Visual Studio.

Installation

Visual Studio can be downloaded from Microsoft here:

https://visualstudio.microsoft.com/vs/community/



Visual Studio Community

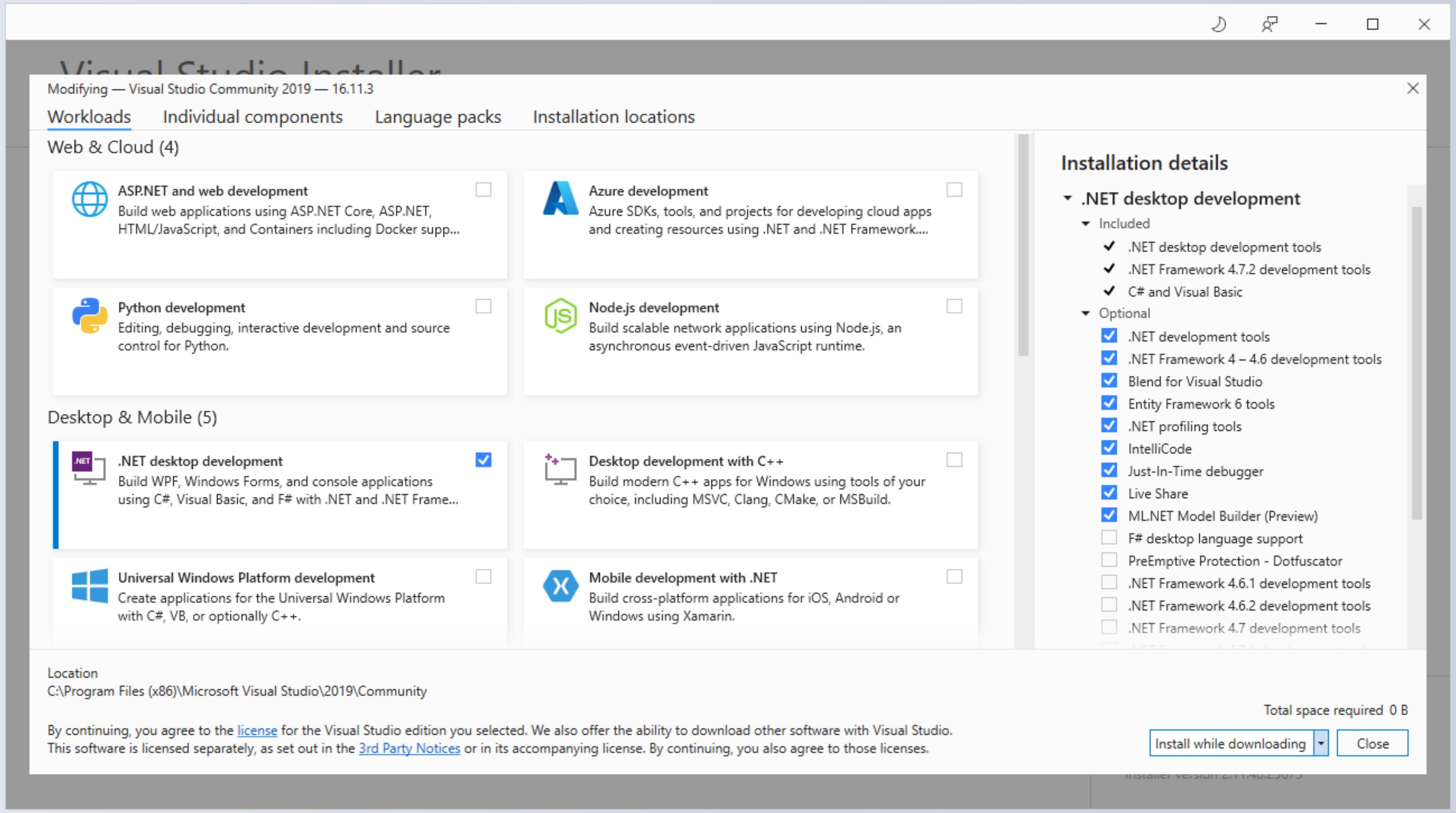
A fully-featured, extensible, free IDE for creating modern applications for Android, iOS, Windows, as well as web applications and cloud services.

Once downloaded, open the downloaded file to start the installation.

Installation

Installation will pause at this step to allow you to choose which components you want to install.

Choose ".NET desktop development" and click Install.



Visual Studio Installer

Modifying — Visual Studio Community 2019 — 16.11.3

Workloads | Individual components | Language packs | Installation locations

Web & Cloud (4)

- [ ] **ASP.NET and web development** — Build web applications using ASP.NET Core, ASP.NET, HTML/JavaScript, and Containers including Docker supp...
- [ ] **Azure development** — Azure SDKs, tools, and projects for developing cloud apps and creating resources using .NET and .NET Framework....
- [ ] **Python development** — Editing, debugging, interactive development and source control for Python.
- [ ] **Node.js development** — Build scalable network applications using Node.js, an asynchronous event-driven JavaScript runtime.

Desktop & Mobile (5)

- [✓] **.NET desktop development** — Build WPF, Windows Forms, and console applications using C#, Visual Basic, and F# with .NET and .NET Frame...
- [ ] **Desktop development with C++** — Build modern C++ apps for Windows using tools of your choice, including MSVC, Clang, CMake, or MSBuild.
- [ ] **Universal Windows Platform development** — Create applications for the Universal Windows Platform with C#, VB, or optionally C++.
- [ ] **Mobile development with .NET** — Build cross-platform applications for iOS, Android or Windows using Xamarin.

Installation details

.NET desktop development

Included
- ✓ .NET desktop development tools
- ✓ .NET Framework 4.7.2 development tools
- ✓ C# and Visual Basic

Optional
- [✓] .NET development tools
- [✓] .NET Framework 4 – 4.6 development tools
- [✓] Blend for Visual Studio
- [✓] Entity Framework 6 tools
- [✓] .NET profiling tools
- [✓] IntelliCode
- [✓] Just-In-Time debugger
- [✓] Live Share
- [✓] ML.NET Model Builder (Preview)
- [ ] F# desktop language support
- [ ] PreEmptive Protection - Dotfuscator
- [ ] .NET Framework 4.6.1 development tools
- [ ] .NET Framework 4.6.2 development tools
- [ ] .NET Framework 4.7 development tools

Location
C:\Program Files (x86)\Microsoft Visual Studio\2019\Community

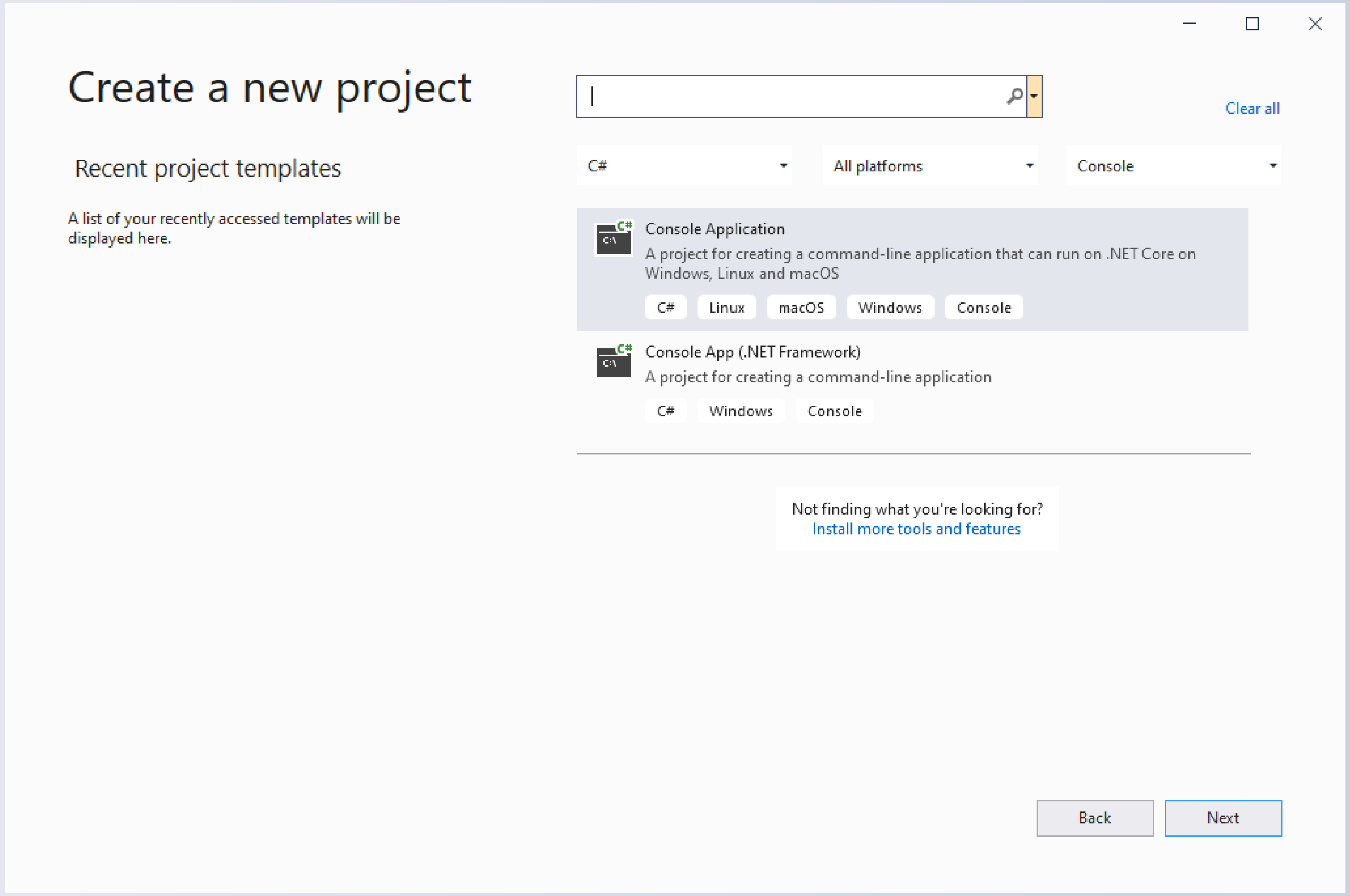
By continuing, you agree to the license for the Visual Studio edition you selected. We also offer the ability to download other software with Visual Studio. This software is licensed separately, as set out in the 3rd Party Notices or in its accompanying license. By continuing, you also agree to those licenses.

Total space required 0 B

Setup

Select “Console Application” and click Next.

Depending on your version the option might say “Console App (.NET Core)”.



A list of your recently accessed templates will be displayed here.

Console Application
A project for creating a command-line application that can run on .NET Core on Windows, Linux and macOS
C# Linux macOS Windows Console

Console App (.NET Framework)
A project for creating a command-line application
C# Windows Console

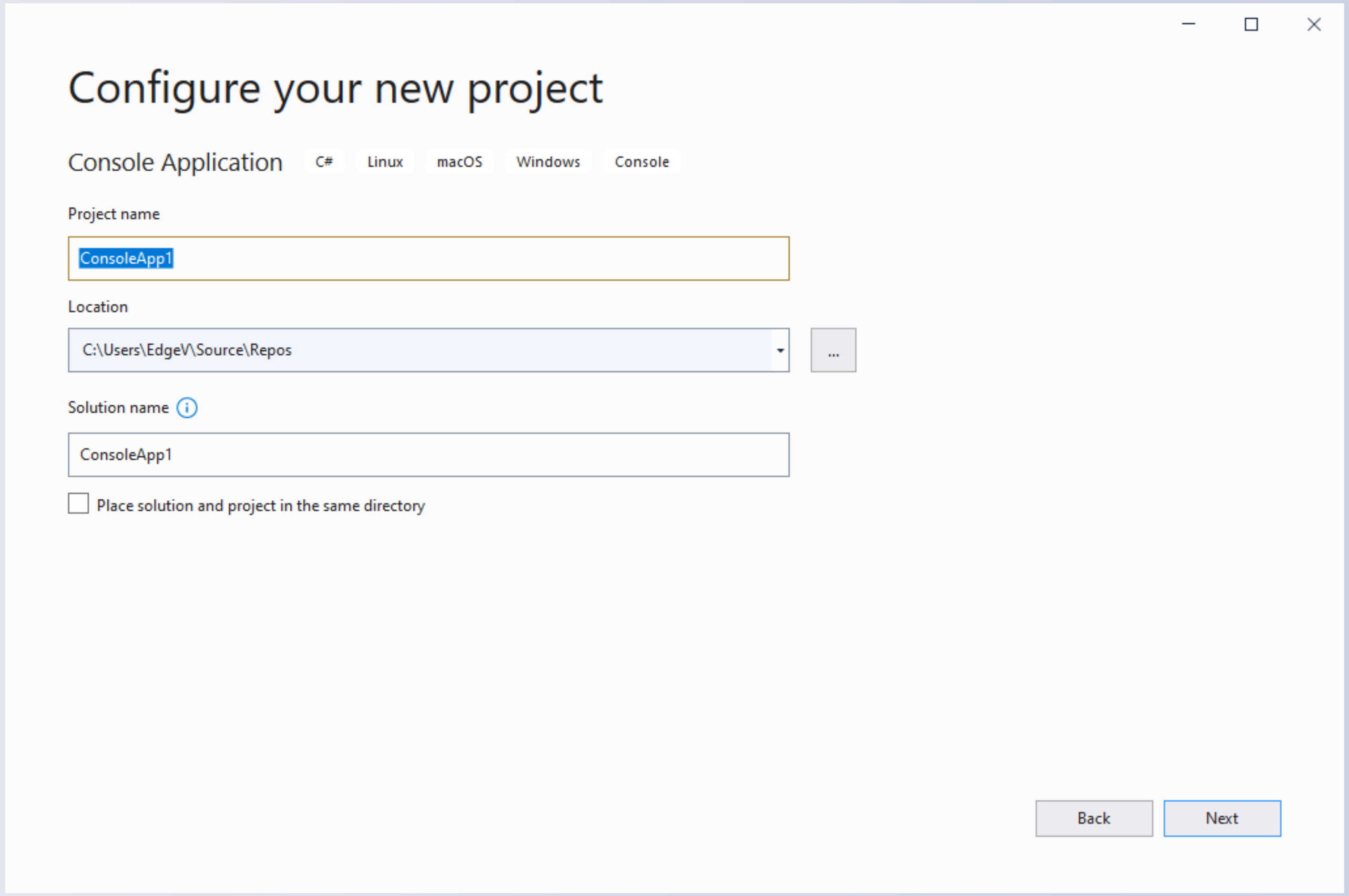
Not finding what you're looking for?
Install more tools and features

Setup

Give your project and solution a name e.g. “HelloWorld”.

Check the save location.

Click Next.


Configure your new project

Console Application C# Linux macOS Windows Console

Project name: ConsoleApp1

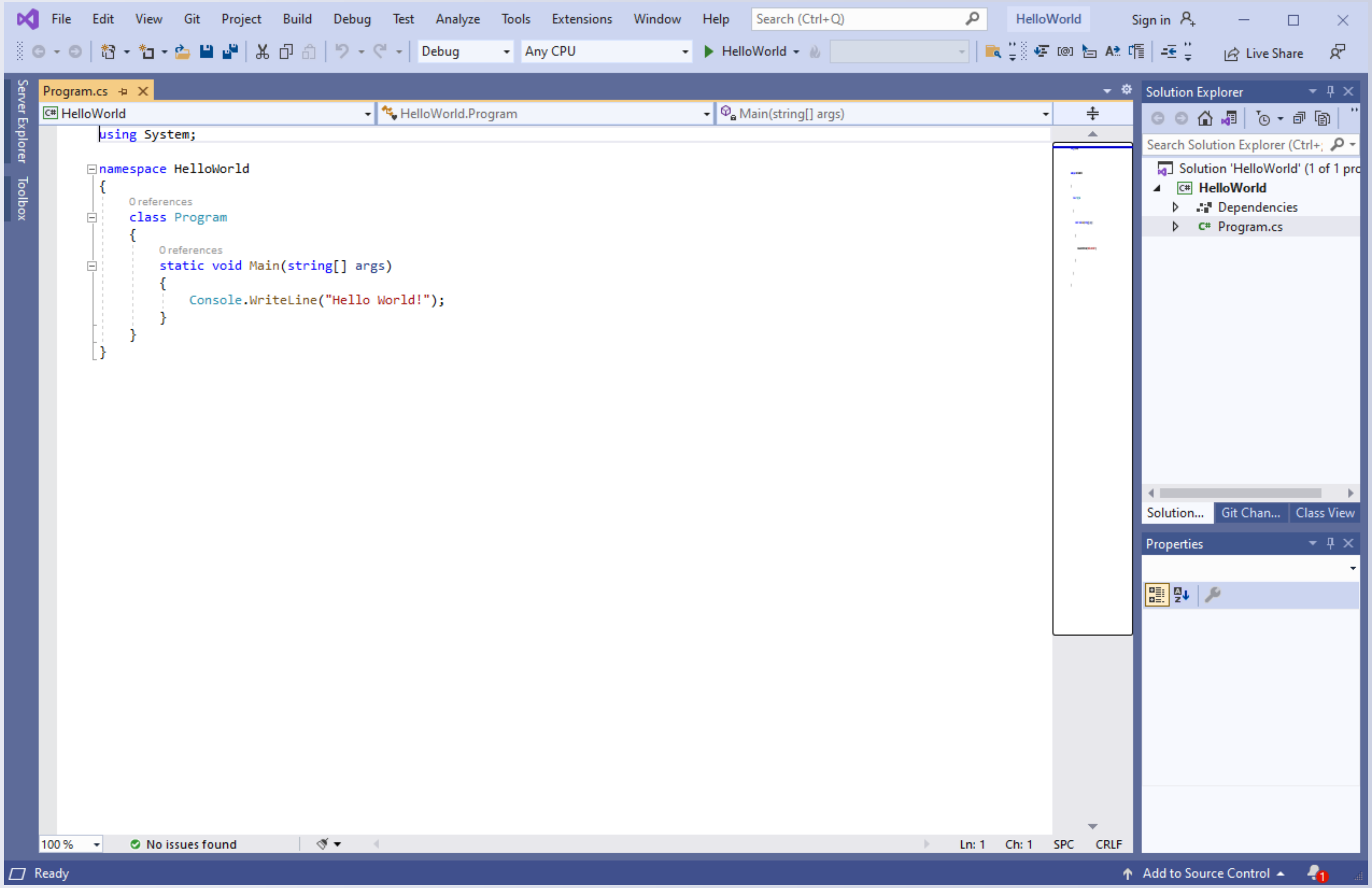
Location: C:\Users\EdgeV\Source\Repos

Solution name: ConsoleApp1

☐ Place solution and project in the same directory

Setup

Your project has been created and Visual Studio has generated some code for you.



```
using System;

namespace HelloWorld
{
    0 references
    class Program
    {
        0 references
        static void Main(string[] args)
        {
            Console.WriteLine("Hello World!");
        }
    }
}
```

Installation and Setup

Exercise 1

Syntax

Section 3

eurofins | Digital Testing

Syntax

During setup, Visual Studio created a C# file called Program.cs and populated it with this code. Executing this program will result in the text “Hello World!” being printed as output.

```
using System;

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

Syntax

The ‘using’ keyword

```
using System;

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

Line 1: using System

The ‘using’ keyword allows us to import namespaces into our program.

A namespace is a grouping of code that can be used as a reference, rather than having the actual code in our project.

‘System’ is a namespace that contains commonly used code such as Console.WriteLine();

Syntax

Whitespace

```
using System;

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

Line 2: blank line

Blank lines and white spaces are ignored by C#.

They are used to make the code look more structured and easier to read.

Syntax

The ‘namespace’ keyword

```
using System;

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

Line 3: `namespace HelloWorld`

A namespace is a way of organizing and grouping code.

There are many namespaces that come with .NET (like System), but we can also create our own.

In this case we have created a namespace called ‘HelloWorld’.

A namespace can contain classes and even other namespaces.

Syntax

Curly braces ‘{ }’

```
using System;

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

Line 4: curly braces

Curly braces mark the beginning ‘{‘ and the end ‘}’ of blocks of code.

Pairs of curly braces are usually indented to the same level for readability.

Some blocks of code that require curly braces are namespaces, classes, methods and loops.

Syntax

The ‘class’ keyword

```
using System;

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

Line 5: class Program

A class is a container for data and methods.

All code that we want to execute must be written inside of a class.

Syntax

The ‘Main’ method

```
using System;

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

Line 7: Main

The ‘Main’ method is something that will appear in every C# program.

When the program is executed, the code inside the ‘Main’ method is run.

The surrounding keywords will be discussed later.

Syntax

Console.WriteLine(“ ”)

```
using System;

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

Line 9: Console.WriteLine()

Console is a class in the System namespace.

WriteLine() is a method in the console class that is used to print (output) text from the program.

“Hello World!” is the text that will be printed.

Syntax

Semicolons ;

```
using System;

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

Line 9: ;

Semicolons are used in C#, and other languages, to end a statement (a single line of code).

Examples of statements are declaring variables, calling methods and printing text to the console.

Syntax

Comments

```
using System;

namespace HelloWorld
{
    class Program
    {
        static void Main(string[] args)
        {
            //This is a single line comment
            Console.WriteLine("Hello World!");

            /*
             This is a
             multi-line
             comment
            */
        }
    }
}
```

Comments are lines of code that will be ignored when the program is run.

They can be used to:

- describe lines of code
- set reminders (e.g. To do)
- block out lines of code during testing or debugging

Single line comments start with double forward slashes '//'.

Multi-line comments start with '/*' and end with '*/' .

Syntax

End of Section

Variables and Data Types

Section 4

eurofins | Digital Testing

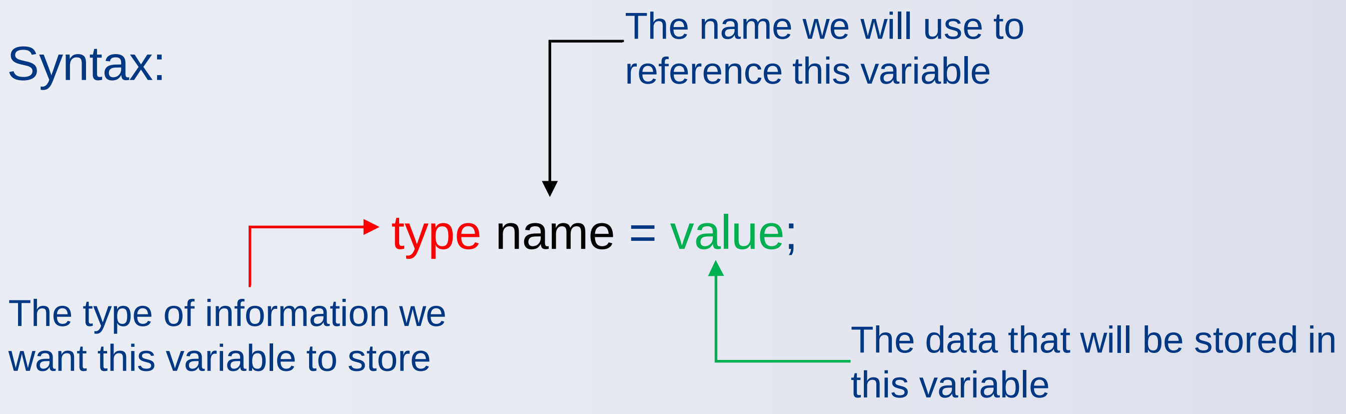
Variables

Declaring a variable

In programming, a variable is the name given to a storage area that our programs can use to store data.

In C#, when creating a variable, you must specify the type and name of the variable. This is called ‘declaring’ the variable.

Syntax:



```
type name = value;
```

The name we will use to reference this variable

The type of information we want this variable to store

The data that will be stored in this variable

The equals ‘=‘ operator is used to assign the value on the right, to the variable on the left.

Variables

Declaring a variable - example

Here is an example of a variable that is used to store a whole number:

```
int myNumber = 24;
```

Remember the syntax: type name = value

int is the data type – this allows the program to store a whole number, e.g. 57 or -119.

myNumber is the name of the variable.

24 is the value of ‘myNumber’

Variables

Using variables

Once a variable has been declared, the program can reference this variable in other areas of the code, e.g.

```
int myNumber = 24;
Console.WriteLine(myNumber);
```

This will output the text “24” when the program is run.

Variables

Assigning values

You can declare a variable without a value and assign the value later.

```
int myNumber;
myNumber = 24;
Console.WriteLine(myNumber);
```

Note: Once you have declared the variable, you don’t need to write the data type (‘int’) every time you use the variable. The program will remember the data type you specified.

This will output the text “24” when the program is run.

Variables

Assigning values

You can change the value of a variable at any time using the equals ‘=‘ operator.

```
int myNumber = 24;
myNumber = 101;
Console.WriteLine(myNumber);
```

This will output the text “101” when the program is run because the value of myNumber is changed before Console.WriteLine(myNumber) is executed.

Variables

Some guidelines

Variable names:

- must begin with a letter, or an underscore (_variableName)
- can contain letters, numbers and underscores
- cannot contain whitespace
- are case sensitive (meaning myNumber and mynumber are two different variable names)
- cannot be keywords such as ‘int’, ‘string’ or ‘class’
- you cannot declare multiple variables with the same name

Data Types

Common data types

These are the most common data types in C#

int – stores integers (whole numbers), including negative numbers, such as 32 and -109

double – stores decimal values such as 0.05 and -11.56

char – stores a single character such as ‘a’ or ‘!’. Char values are surrounded by single quotes

string – stores text such as “Hello World!”. String values are surrounded by double quotes

bool – stores a value that is either true or false (true and false are keywords in C#)

Data Types

Common data types - examples

```
int myNumber = 24;
  Console.WriteLine(myNumber);

double myDouble = 0.06;
  Console.WriteLine(myDouble);

char myCharacter = ‘Z’;
  Console.WriteLine(myCharacter);

bool myBool = true;
  Console.WriteLine(myBool);

string myText = "Hello World!";
  Console.WriteLine(myText);
```

Data Types

Arrays

Arrays are used to store multiple values in a single variable. You can think of them like a list of values.

To declare a variable as an array, use the following syntax:

```
dataType[ ] variableName;
```

The square brackets tell our program to create an array capable of holding multiple variables of the same data type.

Examples:

```
string[ ] names;

int[ ] numbers;

double[ ] myDecimals;

char[ ] characters;
```

Data Types

Creating an array

Note: Arrays are created with a fixed size.
E.g. if you create an array of size 3, it can only ever hold up to 3 values.

Arrays can be created multiple ways.
These are the most common:

With values

```
string[ ] names = { “john” , “jane” , “judy”};
```

This creates an array of strings which has 3 values.

Without values (which can be added later)

```
string[ ] names = new string[3];
```

This creates an empty array of strings which is capable of having 3 values.
The values will be ‘null’ until we set them.

Data Types

Using arrays

The position of a value in an array is known as an index of the array. In most programming languages, indexes start at 0 (zero).

E.g.

```
string[ ] names = { “John” , “Jane” , “Jill” , “Jack”};
```

Index:	**0**	**1**	**2**	**3**
Value:	“John”	“Jane”	“Jill”	“Jack”

Data Types

Using arrays

To get a particular value from the array you need to provide the index of the value you want.
You can do this by writing the name of the array followed by the index you want to retrieve.

E.g.

```
string[ ] names = { “John” , “Jane” , “Jill” , “Jack”};

Console.WriteLine( names[0] );   --------------- This will print out: “John”
Console.WriteLine( names[1] );   --------------- This will print out: “Jane”
Console.WriteLine( names[2] );   --------------- This will print out: “Jill”
Console.WriteLine( names[3] );   --------------- This will print out: “Jack”
Console.WriteLine( names[4] );   --------------- This will give an indexOutOfRangeException error message
```

Data Types

Using arrays

You can change the values in an array at any time.
E.g.

```
string[ ] names = { “John” , “Jane” , “Jill” , “Jack”};

names[2] = “Peter”;

Console.WriteLine( names[2] );    --------------- This will print out: “Peter”
```

Hint: you can find out the length of an array using the Length property. E.g.

```
Console.WriteLine( names.Length );       will output 3
```

User Input

Sometimes you will want a program to get input from the user while the program is running.

In C# this is very simple:

```
string userInput = Console.ReadLine();
// Program waits for the user to type something into the console and press the ‘Enter’ key
before continuing execution
Console.WriteLine(userInput);
```

This code will wait for some input from the user, and then store the input as a string variable.

Variables and Data Types

Exercise 2

Variables and Data Types

End of Section

Operators

Section 5

eurofins | Digital Testing

Operators

Operators are used to perform operations on variables and their values.

The most common operators are:

- arithmetic operators: +, -, *, /
- assignment operators: =
- comparison operators: >, <, >=, <=
- logical operators: &&, ||, !

Operators

Arithmetic operators

Arithmetic operators are used to perform common mathematical functions:

Operation	Symbol	Example	Description
Addition	+ (plus)	a + b	Adds a and b
Subtraction	- (hyphen)	a - b	Subtracts b from a
Multiplication	* (asterisk)	a * b	Multiplies a and b
Division	/ (forward slash)	a / b	Divides a by b
Modulus	% (percent)	a % b	Returns the remainder of a / b

Operators

Arithmetic operators - examples

```
int number1 = 1 + 4;
int number2 = 7 - number1;
int number3 = number1 * number2;
```

Arithmetic operators can go between 2 numbers (values)

or between a number and a variable

or between 2 variables

number1 is 5

number2 is 2

number3 is 10

Operators

Assignment operators

Assignment operators are used to assign values to variables:

Symbol	Example	Description
=	a = 5	Assigns the value 5 to the variable a

Symbol	Written shorthand	Written full	Description
+=	a += 5	a = a + 5	Adds 5 to the value of a, then assigns the result to a.
-=	a -= 5	a = a - 5	Subtracts 5 from the value of a, then assigns the result to a.
*=	a *= 5	a = a * 5	Multiplies a by 5, then assigns the result to a.
/=	a /= 5	a = a / 5	Divides a by 5, then assigns the result to a.

Operators

Assignment operators - examples

number1 = 4;	Assigns the value 4	number1 is 4
number1 += 6;	Same as number1 = number1 + 6	number1 is 10
number1 -= 1;	Same as number1 = number1 - 1	number1 is 9

Operators

Comparison operators

Comparison operators are used to compare values and give a true/false result:

Symbol	Comparison	Example	Description
==	equal to	a == b	Returns true if a is equal to b. Returns false otherwise.
!=	not equal to	a != b	Returns true if a is not equal to b. Returns false otherwise.
>	greater than	a > b	Returns true if a is greater b. Returns false otherwise.
<	less than	a < b	Returns true if a is less than b. Returns false otherwise.
>=	greater than or equal to	a >= b	Returns true if a is greater than or equal to b. Returns false otherwise.
<=	less than or equal to	a <= b	Returns true if a is less than or equal to b. Returns false otherwise.

Operators

Comparison operators - examples

```
int number1 = 4;
int number2 = 6;
```

`number1 == number2`	Is number1 equal to number2	Returns false
`number2 >= number1`	Is number2 greater than or equal to number1	Returns true

Operators

Logical operators

Logical operators are used to determine logic between multiple statements:

Symbol	Name	Example	Description
&&	And	a < b && b < c	Returns true if both statements are true. Otherwise false.
||	Or	a < b || a < c	Returns true if either statement is true. False if both statements are false.
!	Not	!(a > b && a > c)	Reverses the result of the logic inside the brackets.

Operators

Logical operators - examples

```
int number1 = 4;
int number2 = 6;
int number3 = 8;
```

`(number1 == number2) || (number2 < number3)`	Returns true because the second statement is true
`number2 >= number1 && number1 < number3`	Returns true because both statements are true
`!(number2 >= number1 && number1 < number3)`	Returns false because the logic is reversed

Operators

Concatenation

Concatenation is the process of joining two strings together.

It uses the + operator, just like the arithmetic + for addition. But when used with string variables, instead of adding values together, concatenation will join the values.

```
string firstname = “John";
string surname = “Smith";
string fullName = firstname + “ “ + surname;
```

fullName will be “John Smith”

```
string firstname = “John";
int age = 40;
string combined = firstname + age;
```

combined will be “John40”

Operators

End of Section

Logic and Loops

Section 6

eurofins | Digital Testing

Logic

Recap on Booleans

Booleans are used regularly in programming when you want to compare things and perform different actions depending on the outcome of the comparison.

Booleans have the value true or false.

You can assign the value `true` or `false` directly to a Boolean variable:

```
bool x = true;
bool y = false;
```

Or you can use an expression which will evaluate to `true` or `false`:

```
bool x = 1 < 0;
```

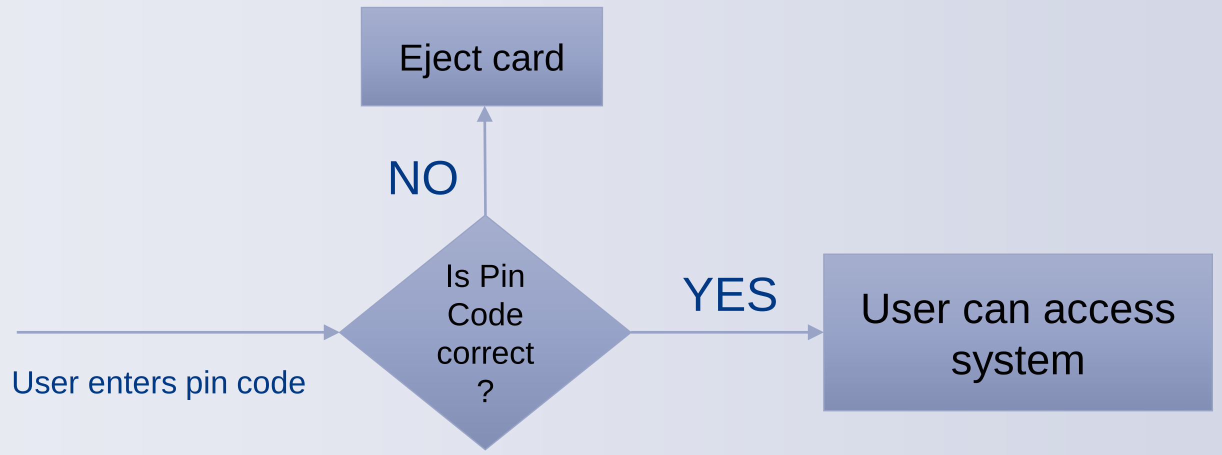
x will be false

Logic

Condition statements

Condition statements are used to control the flow of a program and perform different actions under different conditions.

For example an ATM might use this logic:



Logic

The if statement

The if statement is used to run a block of code only if a condition is True.

It looks like this:

```
if (condition){
   // this code will execute if the condition is True
}
```

Logic

The if statement - example

This is what the if statement looks like with a condition:

```
if ( 1 < 2 ){
   Console.WriteLine(“1 is less than 2");
}
```

```
int x = 7;
int y = 10;

if ( x > y ){
   Console.WriteLine(“x is greater than y");
}
```

```
string text = “Hello World!”;

if ( text == “Hello World!” ){
   Console.WriteLine(“the text is a match");
}
```

Logic

The else statement

The else statement can be used to run a block of code when the if statement is false.

It looks like this:

```
if (condition){
   // this code will execute if the condition is True
}
else {
   // this code will execute if the condition is False
}
```

Logic

The else statement - example

This is what the else statement looks like with a condition:

```
if ( 1 < 2 ){
  Console.WriteLine("1 is less than 2");
}
else {
  Console.WriteLine("1 is not less than 2");
}
```

```
int x = 7;
int y = 10;

if ( x > y ){
  Console.WriteLine("x is greater than y");
}
else {
  Console.WriteLine("x is not greater than y");
}
```

Logic

The else if statement

The else if statement can be used to include a second condition (or any number of conditions) to check if the previous condition is False.

It looks like this:

```
if (condition1){
   // this code will execute if condition1 is True
}
else if (condition2){
   // this code will execute if condition1 is False, and condition2 is True
}
else {
   // this code will execute if both condition1 and condition2 are False
}
```

Logic

The else if statement

You can have multiple ‘else if’ statements, but there can only be one ‘if’ at the start and one ‘else’ at the end.

```
if (condition1){
  // this code will execute if condition1 is True
}
else if (condition2){
  // this code will execute if condition1 is False, and condition2 is True
}
else if (condition3){
  // this code will execute if conditions 1 and 2 are False, and condition3 is True
}
else {
  // this code will execute if all conditions are False
}
```

Logic

The else if statement - example

This is what the else if statement looks like with a condition:

```
if ( 1 < 2 ){
  Console.WriteLine(“1 is less than 2");
}
else if ( 1 > 2 ){
  Console.WriteLine(“1 is greater than 2”);
}
else {
  Console.WriteLine(“1 is equal to 2”);
}
```

```
int x = 7;
int y = 10;

if ( x > y ){
  Console.WriteLine(“x is greater than y");
}
else if ( x < y ){
  Console.WriteLine(“x is less than y”);
}
else {
  Console.WriteLine(“x is equal to y”);
}
```

Loops

Loops allow you to repeat blocks of code without having to duplicate code.

Loops will repeat over and over again until a specified condition is met.

We will look at the ‘while’ loop and the ‘for’ loop.

Loops

While loops

The While loop repeats a block of code as long as the condition is True.

It looks like this:

```
while (condition){
   // code block to be executed
}
```

x++ is shorthand for x = x + 1.
This is called an increment.

```
int x = 1;
while ( x < 10){
   Console.WriteLine(x);
   x++;
}
```

This will print out the numbers from 1 to 9.

Loops

For loops

The For loop repeats a block of code until a condition is met. Use it when you know exactly how many times you want the loop to repeat.

It looks like this:

```
for (index; condition; counter){
   // code block to be executed
}
```

index - this is executed 1 time, before the first loop is executed

condition - the condition is checked before each loop begins

counter - this is executed after every loop

Loops

For loops - example

x++ is shorthand for x = x + 1.
This is called an increment.

```
for (int x = 1; x < 10; x++){
   Console.WriteLine(x);
}
```

index - the variable x is created and value is set to 1

condition - the loop will repeat as long as x is less than 10

counter - x is incremented by 1 after each loop

This will print out the numbers from 1 to 9.

Logic and Loops

Exercise 3

Logic and Loops

End of Section

Methods

Section 7

Methods

A method is a block of code containing statements that are only executed when the method is called.

‘Making a method call’ or ‘Calling a method’ just means telling the method to run.

Methods are used to perform particular actions, can have data passed to them, and can return data.

They are used as a way of organising and reusing code.

Methods

Creating / defining a method

This is what a method declaration looks like:

```
public void Print()
{
  Console.WriteLine(“Hello World!”);
}
```

When this method is called, the code inside will execute and the text “Hello World!” will be printed to the output.

Methods are declared inside a class, and have the following:

- an access modifier - e.g. public (will learn more later)
- a return type - e.g. void
- a name - e.g. Print
- parameters - these are optional
- code body - inside the curly braces

Methods

Calling a method

To call (execute) a method, just type the name of the method followed by parenthesis () and a semi-colon.

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

  public void Print() {
      Console.WriteLine("Hello World!");
  }
}
```

When this program is run, the Main method is executed first (as it always is).

The Main method has a single line of code, which calls the Print method.

The output of this program is the text “Hello World!”.

Methods

Calling a method

To call (execute) a method, just type the name of the method followed by parenthesis () and a semi-colon.

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

  public void Print() {
      Console.WriteLine(“Hello World!”);
  }
}
```

You can call a method multiple times.

The output of this program is the text “Hello World!” printed 3 times.

Methods

Method parameters

Data can be passed to a method as parameters. This is done during the method call. Parameters then act as variables inside the method.

```
class Program
{
  static void Main(string[] args) {
      Print("Hello World!");
  }

  public void Print(string text) {
      Console.WriteLine(text);
  }
}
```

Parameters are specified as part of the method declaration inside the parenthesis ().

You must include the data type and give a name for the parameter e.g. string text

The parameter can then be used just like any other variable inside the method.

Methods

Method parameters

```
class Program
{
  static void Main(string[] args) {
      Print(“Hello World!”);
      Print(“Something”);
      Print(“Something else”);
  }

  public void Print(string text) {
      Console.WriteLine(text);
  }
}
```

Using parameters, you can reuse methods and change the data passed to it each time.

The output of this program is:
“Hello World!”
“Something”
“Something else” all printed to the output.

Methods

Method parameters

```
class Program
{
  static void Main(string[] args) {
      Print(“Hello”, “World!”);
  }

  public void Print(string text1, string text2) {
      Console.WriteLine(text1 + “ “ + text2);
  }
}
```

You can declare multiple parameters for a method by separating them with a comma.

When you call the method, you must send the same number or parameters as the method expects.

This program will output the phrase “Hello World!”.

Methods

Return types

Methods can return a value when they have finished executing.

```
class Program
{
  static void Main(string[] args) {
      int answer = Add(5, 10);
      Console.WriteLine(answer);
  }

  public int Add(int num1, int num2) {
      return num1 + num2;
  }
}
```

The return keyword tells the method to return a value back to where the method was called.

Void means there is no return value.

In the method declaration, you need to declare what data type you want to return e.g. int, string, bool etc.

The Add method accepts 2 parameters and will return an integer.

When this program runs, the value 15 will be stored in the variable ‘answer’.

Methods

Exercise 4

Methods

End of Section

Classes

Section 8

Classes

A class is one of the main components of object-oriented programming.

A class is a template, or a blueprint, used to define objects. An object is an instance of a class.

Everything in C# is associated with classes and objects, along with its attributes and methods.

This example shows the difference between a Class and an Object.

Class	Objects
Fruit	Banana
	Orange
	Apple
	Pear

Classes

A class describes the attributes (variables) and methods that its objects will have.

When an object is created, it inherits all the variables and methods from the class.

Class	Objects
Fruit	Banana
	Orange
	Apple
	Pear

Attributes - shape, colour, size
Methods - peel, eat, slice

Classes

Examples

Class	Objects
Car	Ford
	Renault
	BMW
	Mercedes

Attributes - make, model, colour
Methods - start, stop, soundHorn

Class	Objects
Animal	Dog
	Elephant
	Bear
	Kangaroo

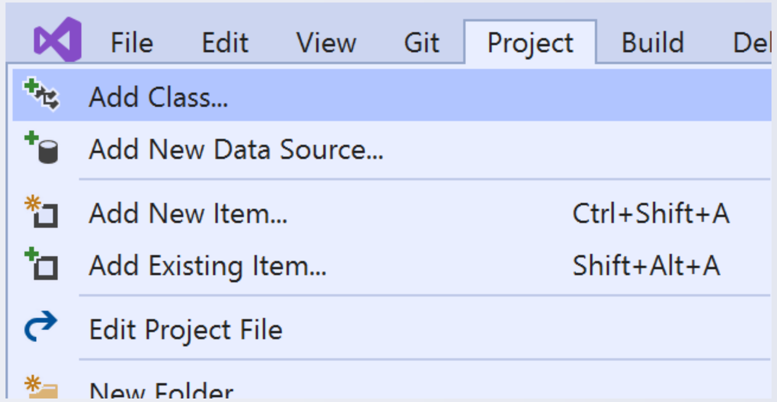
Attributes - species, numLegs
Methods - eat, sleep, walk

Class	Objects
Student	Student1
	Student2
	Student3
	Student4

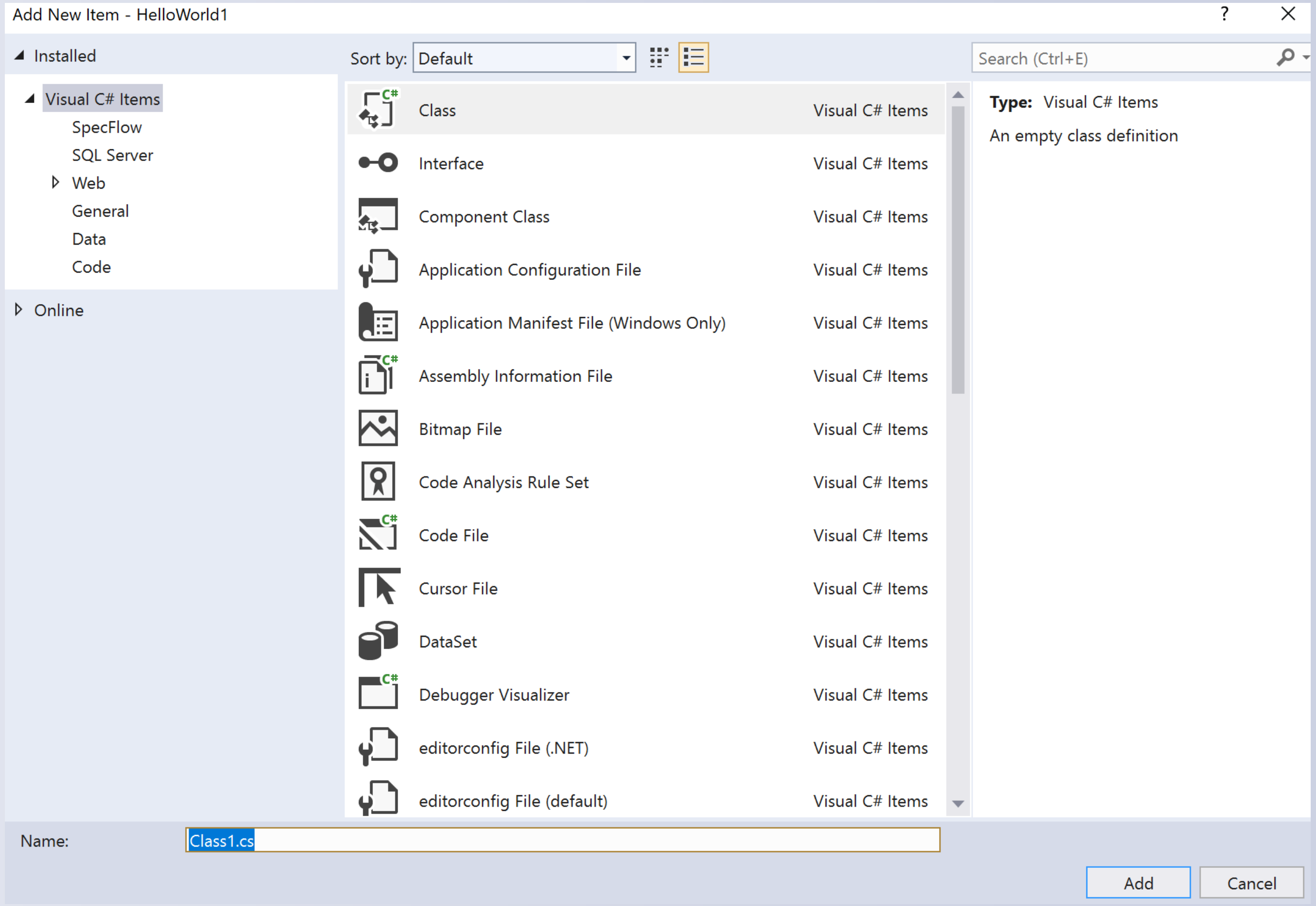
Attributes - subject, email, school
Methods - enrol, setExamResult

Classes

Creating a class in Visual Studio



Click Project in the top menu then select Add Class


Add New Item - HelloWorld1

Installed

- Visual C# Items
  - SpecFlow
  - SQL Server
  - Web
  - General
  - Data
  - Code
- Online

Sort by: Default

Search (Ctrl+E)

Class	Visual C# Items
Interface	Visual C# Items
Component Class	Visual C# Items
Application Configuration File	Visual C# Items
Application Manifest File (Windows Only)	Visual C# Items
Assembly Information File	Visual C# Items
Bitmap File	Visual C# Items
Code Analysis Rule Set	Visual C# Items
Code File	Visual C# Items
Cursor File	Visual C# Items
DataSet	Visual C# Items
Debugger Visualizer	Visual C# Items
editorconfig File (.NET)	Visual C# Items
editorconfig File (default)	Visual C# Items

Type: Visual C# Items

An empty class definition

Name: Class1.cs

Add Cancel

Give the class a name and click Add

Classes

Creating a class

To create a class, you use the ‘class’ keyword followed by the name of the class:

```
class Car
{
  public string colour = “black";

  public void soundHorn() {
    Console.WriteLine(“Beep! Beep!”);
  }
}
```

This is a class called Car that has a single attribute, colour, and a single method, soundHorn().

Classes

Creating an object

```
class Car
{
  public string colour = “black";

  public void soundHorn() {
    Console.WriteLine(“Beep! Beep!”);
  }
}
```

This is the Car class which we can use to create objects.

To create an object, use the class name of the object, e.g. Car, followed by a name for the object e.g. myCar.

Then use the keyword ‘new’ to create the new object.

```
Car myCar = new Car();

Console.WriteLine(myCar.colour);

myCar.soundHorn();
```

Using the dot ‘.’ , you can access the variables/attributes of the object.

Classes

Creating multiple objects

```
class Car
{
  public string colour = “black";

  public void soundHorn() {
    Console.WriteLine(“Beep! Beep!”);
  }
}
```

This is the Car class which we can use to create objects.

To create an object, use the class name of the object, e.g. Car, followed by a name for the object e.g. myCar.

Then use the keyword ‘new’ to create the new object.

```
Car myCar1 = new Car();
Car myCar2 = new Car();

Console.WriteLine(myCar1.colour);
Console.WriteLine(myCar2.colour);

myCar1.soundHorn();
myCar2.soundHorn();
```

You can create multiple objects from the same class. Just make sure you give them different names.

Classes

Constructors

A Constructor is a special method that can be used to initialise objects.

You can use them to set values for variables at the same time as creating the object.

```
class Car
{
  public string colour;

  public Car(string carColour)
  {
    colour = carColour;
  }

  public void soundHorn() {
    Console.WriteLine(“Beep! Beep!”);
  }
}
```

Classes

Constructors - example

```
class Car
{
  public string colour;

  public Car(string carColour)
  {
    colour = carColour;
  }

  public void soundHorn() {
    Console.WriteLine(“Beep! Beep!”);
  }
}
```

```
Car myCar1 = new Car(“black”);
Car myCar2 = new Car(“red”);

Console.WriteLine(myCar1.colour);
Console.WriteLine(myCar2.colour);
```

You can pass parameters in the constructor.

This means you can create each object with its own variable values.

Classes

Access modifiers

```
class Car
{
  public string colour;
  private double price;

  public void getPrice()
  {
    Console.WriteLine(price);
  }

  private void getColour() {
    Console.WriteLine(colour);
  }
}
```

```
Car myCar = new Car();

myCar.colour = “red”;
myCar.price = 2500.00;

myCar.getPrice();
myCar.getColour();
```

this will give an error

this will give an error

Public fields and methods are accessible for all classes.

Private fields and methods are only accessible within the same class.

Classes

Exercise 5

Classes

End of Section

Summary / Conclusions

Section 9

eurofins | Digital Testing

Thank you!

Any questions?