

Welcome to Computer Science

Please fill out the Google form before the end of the week so that I can learn a little about you!

The team



Tim Roden
(he/him)



**Isabel
Culmer**
(she/her)



Alex Colville
(he/him)



Chris Palmer
(he/him)



Collin Espeseth
(he/him)



Tristan White
(he/him)
*Subject Leader of
Computer Science*



Matt Armnor
(he/him)
*Director of Maths and
Computer Science*

What we expect of you

Punctuality

- Arrive **two minutes early** for lessons
 - Mobile phone in holder; headphones away
 - Get a whiteboard and pen
 - Log in to the computer
 - Be ready with paper and pen for notes

This ensures no time is wasted for you or anyone else.

What we expect of you

Notes

You are expected to take **some** notes in all lessons. Unless you have an access arrangement that allows the use of a word processor, we expect you to do this on paper.

But your first priority should be engaging with the lesson.

What we expect of you

Independent work

Every subject requires **4.5 hours** of independent study.

We will set assignments to complete each week, but more on this next lesson.

100% submission rate is expected.

What we expect of you

Responsibility for your learning

- You will not get a top grade without participating in class discussions
- If you have questions, **ask** – don't just sit there!
 - During the lesson
 - After the lesson
 - At lunchtime workshops

What we expect of you

Responsibility for your learning

- You will not get a top grade without participating in class discussions
- If you have questions, **ask** – don't just sit there!
 - During the lesson
 - After the lesson
 - At lunchtime workshops
- Getting things wrong is the best way to learn

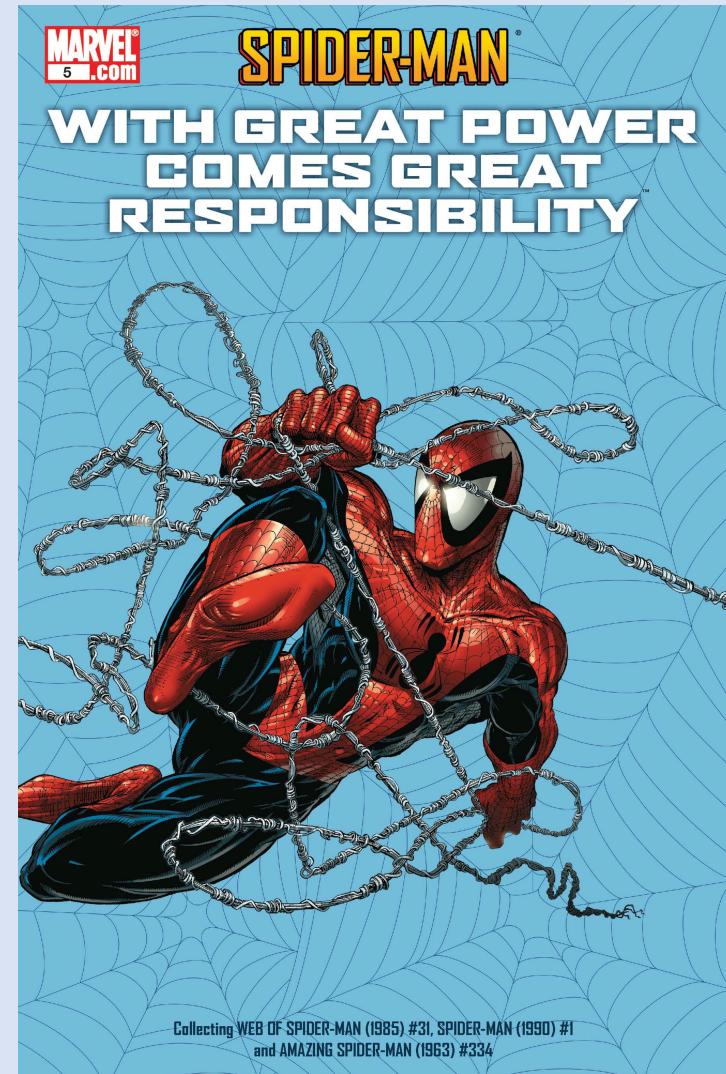
These are useful study skills for the future.

What we expect of you

Screens

- You should only use the computers for what we are working on
 - They're not for doing homework
 - They're not for checking your emails
- When we're doing theory, don't use the computers for programming

This is so nobody is distracted.



What we expect of you

Missing a lesson

If you miss a lesson (for illness or otherwise):

- Notify your teacher by email
- Get a parent to authorise your absence online
- Catch up with the missed content – your teacher will not teach the same lesson again
- If you're still confused, use Ada Computer Science to search for more information
 - Avoid Google; it doesn't know the specification
- If you're still confused, attend a lunchtime workshop

Resources

There are masses of resources available to you.

- BPCompSci
- Ada Computer Science
- Lunchtime workshops
- Brainscape sets

The course

Computer science is a linear course: all exams are at the end of the two years.

Our exam board is **AQA**.

Paper 1	+	Paper 2	+	Non-exam assessment
What's assessed This paper tests a student's ability to program, as well as their theoretical knowledge of computer science from subject content 10–13 above and the skills required from section 22 above.		What's assessed This paper tests a student's ability to answer questions from subject content 14–21 above.		What's assessed The non-exam assessment assesses student's ability to use the knowledge and skills gained through the course to solve or investigate a practical problem. Students will be expected to follow a systematic approach to problem solving, as shown in section 22 above.
Assessed - On-screen exam: 2 hours 30 minutes 40% of A-level		Assessed - Written exam: 2 hours 30 minutes 40% of A-level		Assessed 75 marks 20% of A-level
Questions Students answer a series of short questions and write/adapt/extend programs in an electronic answer document provided by us. We will issue preliminary material, a skeleton program (available in each of the programming languages) and, where appropriate, test data, for use in the exam.		Questions Compulsory short-answer and extended-answer questions.		

The course – overview (year 1)

C#

Basics, imperative,
procedural, recursive and
object-oriented
programming

The course – overview (year 1)

C#

Basics, imperative, procedural, recursive and object-oriented programming

Computer Architecture

How does a computer work internally? How do the components communicate?

The course – overview (year 1)

C#

Basics, imperative, procedural, recursive and object-oriented programming

Computer Architecture

How does a computer work internally? How do the components communicate?

SQL

Designing and accessing relational databases

The course – overview (year 1)

C#

Basics, imperative, procedural, recursive and object-oriented programming

Networking

How do computers and devices communicate with each other?

Computer Architecture

How does a computer work internally? How do the components communicate?

SQL

Designing and accessing relational databases

The course – overview (year 1)

C#

Basics, imperative, procedural, recursive and object-oriented programming

Networking

How do computers and devices communicate with each other?

Computer Architecture

How does a computer work internally? How do the components communicate?

Assembly

Low-level CPU programming

SQL

Designing and accessing relational databases

The course – overview (year 1)

C#

Basics, imperative, procedural, recursive and object-oriented programming

Networking

How do computers and devices communicate with each other?

Computer Architecture

How does a computer work internally? How do the components communicate?

Assembly

Low-level CPU programming

SQL

Designing and accessing relational databases

Data Structures & Algorithms

Encryption, graph theory, hashing, stacks, queues, searching and sorting

The course – overview (year 1)

C#

Basics, imperative, procedural, recursive and object-oriented programming

Networking

How do computers and devices communicate with each other?

Computer Systems

Logic, operating systems, software and hardware

Computer Architecture

How does a computer work internally? How do the components communicate?

Assembly

Low-level CPU programming

SQL

Designing and accessing relational databases

Data Structures & Algorithms

Encryption, graph theory, hashing, stacks, queues, searching and sorting

The course – extras

- We offer enrichment activities alongside the A-level content
 - British Informatics Olympiad
 - Annual competition in computer programming for sixth form students
 - Very elite!
 - Advent of Code
 - Daily programming challenges through December
 - Bebras Elite Computational Thinking Challenge
 - A computation challenge run by the University of Oxford
 - Cyber Discovery/CyberCenturion
 - Cyber security competitions

Starter



Evaluate

$$3 + 4 \times 5$$

Starter



Evaluate

$$3 + 4 \times 5$$

23

Starter



Evaluate
 $(3 + 4) \times 5$

Starter



Evaluate
 $(3 + 4) \times 5$

35

Starter



Evaluate

$$(3 + 4) \times (5 - 2)$$

Starter



Evaluate

$$(3 + 4) \times (5 - 2)$$

21

Starter



Evaluate

$$2 \times 7 - 10 + 4$$

Starter



Evaluate

$$2 \times 7 - 10 + 4$$

8

Starter



Evaluate

$$3 + 4 \times 5 - 8 / (2 + 2)$$

Starter



Evaluate

$$3 + 4 \times 5 - 8 / (2 + 2)$$

21

Starter



Evaluate

$$3 - 4 + 2$$

Starter



Evaluate

$$3 - 4 + 2$$

1

The problem with these expressions

- As humans, we have been trained to multiply before adding
 - BIDMAS
 - A computer would need to be programmed with these rules
- We use brackets to clarify exceptions
 - A computer would need to be programmed with these rules too

How can we write expressions in a way that is simpler for computers to understand?

Reverse Polish Notation

Introduction

$$3 + 4 * 5 - 8 / (2 + 2)$$

- This is called **infix** notation
 - The **operators** are placed between the **operands**

Introduction

$$3 + 4 * 5 - 8 / (2 + 2)$$

- This is called **infix** notation
 - The **operators** are placed between the **operands**

$$3\ 4\ 5\ *\ +8\ 2\ 2\ +\ /\ -$$

- This is called **postfix/reverse polish notation**

Evaluating RPN expressions

3 4 5 * +8 2 2 + / -

Starting at the leftmost part of the expression,

1. Move right until you reach an **operator**
2. Apply the operator to the previous two **operands**
3. Repeat until the expression is fully evaluated

Practice



Evaluate
6 2/

Practice



Evaluate

6 2 /

3

Practice



Evaluate

$2\ 9\ +\ 4\ *$

Practice



Evaluate

$2\ 9\ +\ 4\ *$

44

Practice



Evaluate
 $3\ 6\ 5\ -\ 2\ *\ +$

Practice



Evaluate
 $3\ 6\ 5\ -\ 2\ *\ +$

5

Exercise

1. 1 1 -

2. 1 1 - 2 +

3. 1 2 3 * 4 + +

4. 1 3 4 + 2 * 7 / +

5. 9 6 - 4 / 9 +

6. 1 3 3 7 - 2 + - +

7. 6 5 3 / 7 - 2 2 3 * -- 3 * -

Exercise

- | | |
|---------------------------------|----|
| 1. 1 1 - | 0 |
| 2. 1 1 - 2 + | 2 |
| 3. 1 2 3 * 4 + + | 11 |
| 4. 1 3 4 + 2 * 7 / + | |
| 5. 9 6 - 4 / 9 + | |
| 6. 1 3 3 7 - 2 + - + | |
| 7. 6 5 3 / 7 - 2 2 3 * -- 3 * - | |

Exercise

1. 1 1 -	0
2. 1 1 - 2 +	2
3. 1 2 3 * 4 + +	11
4. 1 3 4 + 2 * 7 / +	3
5. 9 6 - 4 / 9 +	9.75
6. 1 3 3 7 - 2 + - +	6
7. 6 5 3 / 7 - 2 2 3 * -- 3 * -	10

Notation vs. expression

- Using RPN does not change the expression; it only changes the way it is written
- Hence, we can convert between RPN and infix notation, but still be representing the same expression

Advantages of RPN

- There is no need for an order of precedence of operations (e.g. BIDMAS)
 - This is because postfix expressions can be evaluated left-to-right

Advantages of RPN

- There is no need for an order of precedence of operations (e.g. BIDMAS)
 - This is because postfix expressions can be evaluated left-to-right
- A computer can evaluate a postfix expression faster than the same expression in infix

Advantages of RPN

- There is no need for an order of precedence of operations (e.g. BIDMAS)
 - This is because postfix expressions can be evaluated left-to-right
- A computer can evaluate a postfix expression faster than the same expression in infix
- There is no need for brackets

C# Programming (Variables)

Getting started

1. Sign in to Google Drive for Desktop and create a folder called “CompSci”
2. Open Visual Studio 2022 Community Edition – not Visual Studio Code
3. Create a new Console App (.NET Framework) project called “L101 Variables”

Data types

Every piece of data that we store/use has a **type**

Data type	Description
int	A whole number between $\sim -2.1\text{bn}$ and $\sim 2.1\text{bn}$
byte	A whole number between 0 and 255 (inclusive)
float	A real number
double	A real number
char	A character (letter, digit, symbol etc.)
string	A sequence of characters
DateTime	A value storing a date and time
bool	true or false

Variables

- A **variable** is a location in memory that contains data
 - We'll look at how this works in future lessons
- A variable has:

Variables

- A **variable** is a location in memory that contains data
 - We'll look at how this works in future lessons
- A variable has:
 - Data type
 - We will only use `int` and `string` today

Variables

- A **variable** is a location in memory that contains data
 - We'll look at how this works in future lessons
- A variable has:
 - Data type
 - We will only use `int` and `string` today
 - Identifier

Variables

- A **variable** is a location in memory that contains data
 - We'll look at how this works in future lessons
- A variable has:
 - Data type
 - We will only use `int` and `string` today
 - Identifier
 - Value

Declaring a variable

- The first step to creating a variable is to **declare** it
- Variable declaration is as simple as stating the type and identifier of the variable

```
string name;  
int age;
```

Here, we have declared two variables: name and age. This is perfectly valid code. Behind the scenes, the program is reserving space in memory for us to store those values.

Variable assignment & initialisation

- We **assign** values to variables – this is known as variable **assignment**
 - e.g. we assign the value 16 to the variable age
- When we assign a variable its first value, we have **initialised** the variable

```
string name;  
int age;
```

```
name = "Alice"; // Initialisation  
age = 16; // Initialisation
```

```
age = 17; // Assignment
```

Variable assignment & initialisation

- We can declare and initialise a variable in one line
- This is more common, but it's useful to know both methods

```
string name = "Alice";
```

```
int age = 16;
```

```
age = 17; // Assignment
```

Identifiers

- Rules
 - Cannot start with a number
 - Can only contain letters (a-z, A-Z), numbers (0-9) and underscores
- Guidelines
 - Choose sensible identifiers
 - Avoid one-letter identifiers (except in for loops, but we'll see those later)
 - Avoid abbreviations
 - Describe what is stored
 - Identifiers are case-sensitive, so choose a sensible capitalisation convention and stick to it



camelCase

- For variable identifiers, it is common practice to use camelCase
- This is a capitalisation convention
- The first word starts with a lowercase letter, then all following words start with a capital letter
- Examples:
 - age
 - aReallyLongVariableIdentifier
 - birthYear
- Your code will work if you don't stick to this, but it'll be more readable (and nicer to look at) if you do
 - Any other convention will do too, but this typical for C#

Example 1

```
class Program
{
    static void Main(string[] args)
    {
        string myName = "";

        Console.WriteLine("Enter name: ");
        myName = Console.ReadLine();
        Console.WriteLine("Hello, " + myName);
        Console.ReadKey();
    }
}
```

1. Copy this program from Google Classroom
2. Amend it such that it also asks for the user's surname and displays it after the "Hello" line

Example 2

```
class Program
{
    static void Main(string[] args)
    {
        int num;
        int square;

        num = 5;
        square = num * num;

        Console.Write("The answer is: ");
        Console.WriteLine(square);
    }
}
```

Comments

- Comments allow us to include text in our code that isn't considered to be code
- They can be a form of documentation

```
static void Main()  
{  
    // This is a comment!  
    Console.WriteLine("Hello, world!");  
}
```


Constants

- Constants are like variables, except
 - They must be given a value when they are declared
 - Their value cannot be changed (they're constant!)

```
static void Main(string[] args)
{
    const double pi = 3.14;
}
```

Worksheets

- We have worksheets available for most programming lessons
 - Remember: the best way to learn to program is to practice!
 - Start the worksheets in class and complete them at home if you can – the more practice, the better
1. Create a folder in your Google Drive for computer science worksheets
 2. Share it with me (with editing permissions)
 3. Make a copy of the worksheet in the new folder
 4. Copy code solutions into the document

W100 - C# Variables