OXFORD
INTERNATIONAL
AQA EXAMINATIONS

INTERNATIONAL GCSE COMPUTER SCIENCE

(9210)

Teaching guidance

For teaching from September 2017 onwards

For International GCSE exams September 2019 onwards

Our specification is published on our website **oxfordaqaexams.org.uk** We will let centres know in writing about any changes to the specification. We will also publish changes on our website. The definitive version of our specification will always be the one on our website; this may differ from printed versions.

Contents

Introduction

The Oxford AQA International GCSE Computer Science has been developed in consultation with teachers by experts with many years' experience of designing and running computer science qualifications in England.

We have taken the best aspects of AQA's GCSE Computer Science specification that is currently delivered by schools in England, and improved this to make it more appropriate for teaching in an International setting. These improvements include even greater focus on practical work, both in the subject content and in the assessments together with the use of an end-of-course programming exam instead of a coursework project to assess programming proficiency, freeing up more teaching time for students to develop their skills.

Whilst programming and computer science are not the same thing, programming is the mechanism through which the principles of computer science are put into practice and through the wording and weighting of the course assessment objectives, the selection of the subject content and the design of the assessments we have placed programming at the heart of our specification.

This is a modern specification, which in addition to programming in a high-level language such as Python or Visual Basic also include sections on database programming in SQL and developing web pages using HTML and CSS.

Students who successfully complete this course will be in an excellent position to go on to further study, such as an A-level in Computer Science.

When writing the specification, we ensured that we have made clear exactly what we will expect students to know about each topic so that students can go into exams knowing that they have learnt the material to an appropriate level of depth. We believe that the Oxford AQA International specification is the clearest, most detailed International GCSE Computer Science specification available.

In addition to providing a very detailed specification, Oxford AQA Exams has provided a range of other resources to help teachers. These include sample question papers and mark schemes, switching guide, Scheme of Work with resource list, teacher guide that includes pseudocode guide, programming language comparison and command words, teacher notes, exemplar student answers with examiner commentaries, past AQA exam questions mapped to the new International qualifications, training and an endorsed textbook.

Aims and learning outcomes

The Oxford AQA International GCSE in Computer Science will encourage students to:

- analyse problems in computational terms
- design and write computer programs to solve problems
- test and debug programs
- build simple web pages and work with relational databases
- demonstrate and apply knowledge and understanding of the key concepts and principles of computer science
- progress to employment or further courses of study.

Assessment Objectives (AOs)

There are three assessment objectives for the International GCSE in Computer Science:

AO1	Demonstrate knowledge and understanding of the key concepts and principles of computer science.
AO2	Apply knowledge and understanding of key concepts and principles of computer science.
AO3	Analyse problems in computational terms in order to develop and test programmed solutions and demonstrate an understanding of programming concepts.

Weighting of Assessment Objectives

The table below shows the approximate weighting of each of the assessment objectives in the International GCSE Computer Science components:

Assessment Objectives (AOs)	**Component weightings (approx %)**		**Overall weighting (approx %)**
	Paper 1	**Paper 2**	
AO1	0	20–25	20–25
AO2	0–5	20–25	20–30
AO3	45–50	0–5	45–55
Overall weighting of components	50	50	100

The assessment objectives put the assessment of programming and practical skills at the heart of the specification, allocating approximately 75% of the marks for the course to the application of knowledge and understanding of the concepts of computer science and the development, testing and understanding of programming concepts.

Specification at a glance

The specification is divided into eight sections, which are:

1	**Algorithms**	Covers what algorithms are and how they can be represented using flowcharts or pseudocode, standard searching and sorting algorithms and algorithmic efficiency.
2	**Programming**	Covers the key programming concepts that students will be expected to be able to use by the end of the course, such as data types, arrays and records, programming structures such as iteration and selection, input, output and file handling, and the use of subroutines and parameters for structured programming. This section also covers different levels of language and program translation.
3	**Data representation**	Covers how important types of data including positive whole numbers, text, images and sound are represented by computers together with data compression and simple binary arithmetic.
4	**Computer systems**	Covers the hardware and software components that make up a computer system. This includes design of logic circuits, the architecture of a standard Von Neumann computer, secondary storage, cloud storage, embedded systems and types of software including the operating system.
5	**Computer networks**	Covers LANs and WANS, wired and wireless networking, physical topologies, standard protocols such as HTTP and IMAP, the TCP/IP stack and network security.
6	**Cyber security**	Covers cyber security threats, social engineering and methods to protect computer systems.
7	**Relational databases and SQL**	Covers the design and use of relational databases including up to three tables and the programmatic use of these through SQL.
8	**Web page design**	Covers the creation of web pages using HTML and CSS.

Assessment in closer detail

Paper 1: Programming

This paper assesses students' understanding of programming concepts and their ability to test and design programs.

The assessment is based around a pre-released program, known as the skeleton program which students will have had access to from three months prior to examination. In the exam, students will be asked questions about how the skeleton program works. They will also be expected to modify and improve the skeleton program.

During the exam, students will use a computer and will put their answers into a word processor document. They will have access to their programming language and will be able to modify and test changes made to the skeleton program in this, before copying and pasting the program code that they have written into the word processor document for submission.

The developers of this specification have considerable experience of running this type of assessment successfully in England. We believe that a practical programming paper is a more realistic and effective method of assessing programming skills than a paper-based algorithmic thinking exam that uses methodologies such as pseudocode and flowcharts. Having a skeleton program reduces some of the anxiety that students might feel going into a programming exam that is completely unseen and also makes it possible for us to assess higher-order skills in a shorter exam as students to do not have to start writing programs from scratch during the exam time.

This paper will be available for the Python, Visual Basic and C# languages. The overwhelming majority of students taking the AQA Computer Science qualifications in England use one of these languages.

Paper 1: Programming
Assessed • 2 hours • 80 marks • 50% of qualification • closed book • completed on a computer using a programming language, with answers put into a word-processed document for submission.
Questions **Section A:** questions about programming that test programming concepts and some non-programming aspects of the skeleton program. **Section B:** short programming questions which require students to make small modifications to the skeleton program, eg correcting errors or writing up to several lines of program code. **Section C:** longer programming questions which require students to make major modifications to the skeleton program and will typically require more analysis than Section B questions, eg writing new subroutines, adding significant new functionality to the skeleton program.

A **Section A** question might, for example, ask students to:

- explain the purpose of a particular variable or section of code in the skeleton program
- explain why a part of the skeleton program has been written in a certain way, possibly comparing it to an alternative
- explain how a data structure is used in the Skeleton program
- explain why decomposing a program such as the skeleton program into subroutines is a good idea.

A **Section B** question might, for example, ask students to:

- improve the input and output of the skeleton program
- make the skeleton program more robust by implementing validation checks on input
- make small changes to the functionality of the skeleton program by editing up to several lines of code.

A **Section C** question might, for example, ask students to:

- make significant changes to the functionality of the skeleton program
- create new subroutines.

Paper 2: Concepts and principles of computer science

This paper assesses the theoretical aspects of the course together with the use of SQL and web page development using HTML and CSS. Whilst the focus is on the theoretical aspects of the course, where possible questions will be practical in nature, requiring students to demonstrate and apply the skills that they have learnt rather than to simply recall knowledge.

Paper 2: Concepts and principles of computer science
Assessed • 2 hours • 80 marks • 50% of qualification • closed book • paper examination.
Questions A mixture of question types including multiple choice, short and longer answer questions.

Questions might, for example, ask students to:

- complete a truth table for a logic circuit that it is given on the question paper
- convert numbers from one base (eg hexadecimal) to another base (eg binary)
- describe what happens during the fetch-execute cycle
- complete a trace table by stepping through an algorithm
- show how an image might be represented as a bitmap
- design the structure of a database
- compress some data by using a Huffman coding tree
- explain how a particular storage device works
- write SQL code to query or update a database
- explain the difference between high and low-level languages and the relative merits of using interpreters and compilers
- write HTML and CSS code to generate a webpage or identify errors in HTML or CSS code on the question paper.

Complete example question papers and mark schemes for both Paper 1 and Paper 2 are available.

Course planning

The course is designed to be as practical as possible, with much of the emphasis placed on students learning to write computer programs in a high-level programming language.

The Paper 1 assessment will be available in three languages: Python, Visual Basic and C#. The first choice to be made is which of these languages to use. Factors that might influence this include:

- prior experience of the teacher
- prior experience of the students
- availability of teaching materials in the language, for example, textbooks, web tutorials
- availability of training courses for the language
- what language(s) will be used if students go on to further courses of study.

All three of the supported programming languages are currently available at no cost for use in schools and by students at home.

It is expected that 70-80% of the course time will be spent programming. This time should include both learning to program in the selected high-level programming language together with using SQL and carrying out web development using HTML and CSS.

Programming skills should be built up gradually as it is important that students are given the time to practice using concepts that they have studied before more concepts are introduced. Alongside learning the technical ability to write syntactically correct program code, it is essential that students are given the opportunity to develop their logical reasoning skills so that when given a problem to solve they can see how this might be decomposed and turned into a computer programmed solution.

SQL and database design can be taught using a range of packages, including MySQL which is available for free and Microsoft Access. In industry, computer programs often interact with databases and whilst writing programs that can do this is not a requirement of the course, more able students may wish to attempt this.

No special software is required for web page design, all that is needed is a text editor and a web browser. Use of a syntax aware text editor such as Notepad++ is helpful to display the code in a clearer structured form. Commercial packages such as Adobe Dreamweaver can also be used. The subset of HTML and CSS commands that students will be expected to know are detailed in the specification.

With regard to the more theoretical topics in the specification, many of these can still be taught practically, for example software or websites can be used to teach about topics such as logic circuit design and representing sound on a computer. However these topics are taught, it is important that students are given the opportunity to regularly practice their skills by answering questions and receiving feedback of their responses.

A possible scheme of work for the course is available.

Appendices

Appendix 1 – Pseudocode command set

Introduction

The pseudocode described below is provided to assist students preparing for their Oxford AQA International GCSE Computer Science examinations.

In all assessment material, Oxford AQA Exams will use a consistent style of pseudocode, which is detailed in this document. This will ensure that, given sufficient preparation, students will understand the syntax of the pseudocode easily. It is not the intention that students must use this style of pseudocode in their own work, including responses to examination questions, although they are free to do so. The only direction to students when answering questions or describing algorithms in pseudocode is that their code is clear and consistent.

The document is not confidential and can be freely shared with students.

General Syntax

- IntExp, RealExp, BoolExp, CharExp and StringExp means any expression which can be evaluated to an integer, real, Boolean, character or string respectively.
- Exp means any expression.
- Emboldened pseudocode is used to indicate the keywords/operators.
- Exam paper questions will assume that indexing for arrays and strings starts at 0 unless specifically stated otherwise.

Variables and constants

Variable assignment	Identifier ← Exp	a ← 3
		b ← a + 1
		c ← c – 2

Constant assignment	**constant** IDENTIFIER ← Exp	constant PI ← 3.141 constant CLASS_SIZE ← 23

Arithmetic operations

Standard arithmetic operations	**+** **-** ***** **/**	Standard use using brackets to make precedence obvious. The / symbol is used instead of ÷ for division (for integer division use DIV.)
Integer division	IntExp **DIV** IntExp	9 DIV 5 evaluates to 1 5 DIV 2 evaluates to 2 8 DIV 4 evaluates to 2
Modulus operator	IntExp **MOD** IntExp	9 MOD 5 evaluates to 4 5 MOD 2 evaluates to 1 8 MOD 4 evaluates to 0

Relational operators for types that can be clearly ordered

Less than	Exp **<** Exp	4 < 6
Greater than	Exp **>** Exp	4.1 > 4.0
Equal to	Exp **=** Exp	3 = 3
Not equal to	Exp **≠** Exp	True ≠ False
Less than or equal to	Exp **≤** Exp	3 ≤ 4 4 ≤ 4
Greater than or equal to	Exp **≥** Exp	4 ≥ 3 4.5 ≥ 4.5

Boolean operations

Logical AND	BoolExp **AND** BoolExp	(3 = 3) AND (3 ≤ 4)
Logical OR	BoolExp **OR** BoolExp	(x < 1) OR (x > 9)
Logical NOT	**NOT** BoolExp	NOT (another_go = False)

Condition-controlled iteration

Repeat-until (repeat the statements until the Boolean expression is True)	**REPEAT**
	    # statements here
	UNTIL BoolExp
		a ← 1
		REPEAT
		    OUTPUT a
		    a ← a + 1
		UNTIL a = 4
		# will output 1, 2, 3

While (while the Boolean expression is True, repeat the statements)	**WHILE** BoolExp
	    # statements here
	ENDWHILE
		a ← 1
		WHILE a < 4
		    OUTPUT a
		    a ← a + 1
		ENDWHILE
		# will output 1, 2, 3

Count-controlled iteration

For	**FOR** Identifier ← IntExp **TO** IntExp
	    # statements here
	ENDFOR
		FOR a ← 1 TO 3
		    OUTPUT a
		ENDFOR
		# will output 1, 2, 3

Selection

If	**IF** BoolExp **THEN**
	  # statements here
	ENDIF
		a ← 1
		IF (a MOD 2) = 0 THEN
		  OUTPUT 'even'
		ENDIF

If-else	**IF** BoolExp **THEN**
	  # statements here
	ELSE
	  # statements here
	ENDIF
		a ← 1
		IF (a MOD 2) = 0 THEN
		  OUTPUT 'even'
		ELSE
		  OUTPUT 'odd'
		ENDIF

Else-if	**IF** BoolExp **THEN**
	  # statements here
	ELSE IF BoolExp **THEN**
	  # statements here
	# possibly more ELSE IFs
	ELSE
	  # statements here
	ENDIF
		a ← 1
		IF (a MOD 4) = 0 THEN
		  OUTPUT 'multiple of 4'
		ELSE IF (a MOD 4) = 1 THEN
		  OUTPUT 'leaves a remainder of 1'
		ELSE IF (a MOD 4) = 2 THEN
		  OUTPUT 'leaves a remainder of 2'
		ELSE
		  OUTPUT 'leaves a remainder of 3'
		ENDIF

Arrays

Assignment	Identifier ← **[**Exp**,** Exp**,**…**,** Exp**]**	primes ← [2, 3, 5, 7, 11, 13]
Accessing an element	Identifier**[**IntExp**]**	primes[0] # evaluates to 2 (questions on exam # papers will start indexing at # 0 unless specifically stated # otherwise)
Updating an element	Identifier**[**IntExp**]** ← Exp	primes[5] ← 17 # array is now [2,3,5,7,11,17]
Accessing an element in a two-dimensional array	Identifier**[**IntExp**][**IntExp**]**	tables ← [[1, 2, 3], [2, 4, 6], [3, 6, 9], [4, 8, 12]] tables[3][1] # evaluates to 8 as second element # (with index 1) of fourth array # (with index 3) in tables is 8
Updating an element in a two-dimensional array	Identifier**[**IntExp**][**IntExp**]** ← Exp	tables[3][1] ← 16 # tables is now #[[1, 2, 3], # [2, 4, 6], # [3, 6, 9], # [4, 16, 12]]

Arrays (continued)

Array length	**LEN(**Identifier**)**	LEN(primes)
		# evaluates to 6 using example above
		
		LEN(tables)
		# evaluates to 4 using example above
		
		LEN(tables[0])
		# evaluates to 3 using example above

Subroutines

Subroutine definition	**SUBROUTINE** Identifier**(**parameters**)**
	  # statements here
	ENDSUBROUTINE
		SUBROUTINE show_add(a, b)
		  result ← a + b
		  OUTPUT result
		ENDSUBROUTINE
		
		SUBROUTINE say_hi()
		  OUTPUT 'hi'
		ENDSUBROUTINE

Subroutine return value	**RETURN** Exp	SUBROUTINE add(a, b)
		  result ← a + b
		  RETURN result
		ENDSUBROUTINE

Calling a subroutine	Identifier**(**parameters**)**	show_add(2, 3)
		answer ← add(2, 3)

String handling

String length	**LEN(**StringExp**)**	LEN('computer science') # evaluates to 16 (including space)
Position of a character	**POSITION(**StringExp**,** CharExp**)**	POSITION('computer science', 'm') # evaluates to 2 (as with arrays, # exam papers will start indexing # at 0 unless specifically stated # otherwise)
Substring (the substring is created by the first parameter indicating the start position within the string, the second parameter indicating the final position within the string and the third parameter being the string itself)	**SUBSTRING(**IntExp**,** IntExp**,** StringExp**)**	SUBSTRING(2, 9, 'computer science') # evaluates to 'mputer s'
Concatenation	StringExp **+** StringExp	'computer' + 'science' # evaluates to 'computerscience'

String and character conversion

Converting string to integer	**STRING_TO_INT(**StringExp**)**	STRING_TO_INT('16') # evaluates to the integer 16
Converting string to real	**STRING_TO_REAL(**StringExp**)**	STRING_TO_REAL('16.3') # evaluates to the real 16.3
Converting integer to string	**INT_TO_STRING(**IntExp**)**	INT_TO_STRING(16) # evaluates to the string '16'
Converting real to string	**REAL_TO_STRING(**RealExp**)**	REAL_TO_STRING(16.3) # evaluates to the string '16.3'
Converting character to character code	**CHAR_TO_CODE(**CharExp**)**	CHAR_TO_CODE('a') # evaluates to 97 using # ASCII/Unicode
Converting character code to character	**CODE_TO_CHAR(**IntExp**)**	CODE_TO_CHAR(97) # evaluates to 'a' using # ASCII/Unicode

Input/output

User input	**USERINPUT**	a ← USERINPUT
Output	**OUTPUT** StringExp	OUTPUT a

Random number generation

Random integer generation (between two integers inclusively)	**RANDOM_INT(**IntExp, IntExp**)**	RANDOM_INT(3, 5) # will randomly generate 3, 4 or 5

Comments

Single line comments	# comment	
Multi-line comments	# comment # comment and so on	

Appendix 2 – Command words

Introduction

Command words are the words and phrases used in exams and other assessment tasks that tell students how they should answer the question.

The following command words list words and their meanings that are relevant to this subject:

Analyse

Apply put into effect in a recognised way.

Argue

Present a reasoned case.

Assess

Make an informed judgement.

Calculate

Work out the value of something.

Consider

Review and respond to given information.

Comment

Present an informed opinion.

Compare

Identify similarities.

Complete

Finish a task by adding to given information.

Contrast

Identify differences.

Criticise

Assess worth against explicit expectations.

Define

Specify meaning.

Describe

Set out characteristics.

Develop

Take forward or build upon given information.

Discuss

Present key points.

Evaluate

Judge from available evidence.

Examine

Investigate closely.

Explain

Set out purposes or reasons.

Explore

Investigate without preconceptions about the outcome.

Give

Produce an answer from recall.

Identify

Name or otherwise characterise.

Illustrate

Present clarifying examples.

Interpret

Translate information into recognisable form.

Justify

Support a case with evidence.

Outline

Set out main characteristics.

Relate

Demonstrate connections between items.

Suggest

Present a possible case.

State

Express in clear terms.

Summarise

Present principal points without detail.

Appendix 3 – Comparability of languages, C#, Python, VB.Net

Introduction

This gives an overview, including example syntax and links to resources, of the languages used in the International GCSE on-screen examination: C#, Python, and VB.Net. Syntax examples attempt to show the same functionality in each language.

How to pick a language

The languages selected by Oxford AQA Exams for the on-screen examination were chosen as they are appropriate to the level of the course. There are many differences between languages, including quirks of syntax, speed of execution, features within available Integrated Developer Environments (IDEs), popularity in industry, popularity within academia, etc. The aim of the International GCSE is to give students a firm grasp of programming concepts and it is recommended that you consider four factors when picking a language:

- Availability of suitable development environment for your institution. Each language will have a variety of IDEs available, some might be specific to a particular operating system or have very high system requirements.
- Familiarity of the language amongst teaching staff. Students will need to be supported in learning and it is important teachers can handle questions to help all students from the least to most able.
- Availability of support material. Are resources available to help teach the course as well and support students outside lessons, including extension activities?
- Availability of IDEs for students outside lessons. Can students program in their lunch time and at home? Many students will want to go above and beyond what is covered in lesson and this requires significant practice.

C#

C# is a modern programming language from Microsoft. The syntax resembles C++ and Java. C# is an object-orientated language built on the .Net framework, meaning that you can create software projects that use both C# and VB.Net code. The language is more commonly used in industry and academia than VB.Net and there are plenty of support websites and books for learning the language. C# allows for rapid development of graphical user interfaces, websites and the ability to quickly integrate databases. It also allows for the creation of video games on Windows, Windows Phone and Xbox using the Microsoft XNA Game Studio.

C# is strict in how it treats the data types of variables; different data types cannot be combined without error: to add an integer to a decimal you first need to convert the integer into a decimal. C# is case sensitive, this means that declaring a variable called *name* and then assigning a value to *Name* would create an error, as it would treat *name* and *Name* as different variables. However, when using an IDE such as Visual Studio, auto correction would normally convert *Name* into *name*, fixing this issue. To define the end of a statement in C# requires a semi-colon (*;*), when using an IDE, omission of these will normally be highlighted for the user. When using a comparison operator such as *if(age == 18)* a double equals sign is needed as a single equals would be treated as an assignment. These features can prove frustrating for novice programmers, but they force the good practice that can make transitioning to other languages easier.

Development in C# is supported by the Windows only Microsoft Visual Studio IDE. This is an industry leading development environment offering runtime debugging, code tracing, auto correction and IntelliSense code completion tool. The latest versions of Visual Studio can be acquired by students and institutions free through Microsoft Dream Spark. Visual Studio Express also offers a free way to develop C# programs on the Windows platform. Other platforms are supported through the Mono project, including command line and GUI development.

Coding examples and syntax

Default program view with "Hello world"

The code of a default C# command line program in Visual Studio can appear quite daunting. The *using* statement provides extra functionality through different libraries, not all of which might be necessary for the code being written. In the example below only the first line: *using System;* is necessary.

```
using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;


namespace ConsoleApplication1

{

    class Program

    {

        static void Main(string[] args)

        {

            Console.WriteLine("Hello world");

        }

    }

}
```

Variables

Variables are defined by specifying the data type, followed by the name of the variable. Data type conversions are needed to when combining variables of different types:

```
static void Main(string[] args)

{

            int a;

            a = 3;

            Console.WriteLine(a.ToString() + " is the magic number");

}
```

Comments

C# allows for single line (//) and block comments (/* */). Note that comments don't need semicolons at the end of each line as they aren't executed.

```
static void Main(string[] args)

{

        //this is a comment



        /*

        *     so is this

        *     and this

        */

        Console.WriteLine("this isn't");

}
```

Arrays

C# uses square brackets for arrays. Arrays are based from 0 by default, however, when you declare an array you pass the size of the array in the brackets, meaning onedim below can hold 4 values, *onedim[0]*, *onedim[1]*, *onedim[2]* and *onedim[3]*, twodim 16 values (4 values, each of 4 values), and threedim below 64 values (4 values, each of 4 values, each of 4 values).

```
   string[] onedim = new string[4];

   string[,] twodim = new string[4,4];

   string[,,] threedim = new string[4,4,4];
```

Structuring code

The scope of statement blocks are defined using curly brackets, { } ,such as the code to execute when the *if* statement below is true. Semi-colons are used to end single statements. Indentation is by default a manual process in Visual Studio, but automatic indentation can be applied by selecting a block of code and pressing ctrl+e,d in Visual Studio.

```
if (a == 6)

{

        a = a + 1;

}

a = a + 4;
```

Online teaching resources

Wikibooks - C# Programming

Visual Studio – C# Programming Guide

Home and Learn - Visual C# .NET Programming

Downloads

Microsoft DreamSpark (for institutional and student licensing - Windows only).

Visual Studio Download (the community version is free - Windows only).

Mono software platform (free Open Source .NET command line tool - Multiplatform).

MonoDevelop (free Open Source IDE - Multiplatform).

Python 3.x

Python is a multi-platform programming language and is one of the most commonly used programming languages in education. There are plenty of support websites and student textbooks are easily sourced. When looking for resources, check the version of Python that they were written for. Python is also supported by online tutorial websites such as codecademy. Python libraries allow you to extend functionality, building games (through PyGame), websites and apps. There are some visual design tools to help you create GUIs, though they don’t always have the polish of longer established products such as Visual Studio and Delphi. For project work some students might want to look at the 3D modelling tool Blender, which includes Python as a scripting language.

Python is unusual amongst modern programming languages in using whitespace (indentation) to define code blocks. This allows you to easily see the structure of code. However, the length of an indentation isn’t universally defined, you might start at three space keys, others may start at four, which can cause compatibility issues when running someone else's code. There are two versions of Python, Python 2.x and Python 3.x. Only version 3.x is supported by Oxford AQA Exams. The differences between the versions are minor and tutorials and code written for one version can normally be converted into another quite quickly. Python does not have a built-in data type for arrays, using lists instead. Lists can be used in much the same way as arrays, including referencing items and implementing multiple dimensions.

Python IDLE provides a simple and free multiplatform IDE. It offers debugging and tracing, and a portable version is available that can be run off a USB. It is also possible to set up Python to run with major IDEs such as Eclipse, netbeans and Visual Studio, but the process to do this isn't always simple. Python comes pre-installed on the Raspberry Pi and there are plenty of Raspberry Pi and Python resources available.

Coding examples and syntax

Default program with "Hello world"

It is very simple in python to get started using the print command.

```
print("Hello world")
```

Variables

Variables do not need to have a data type defined, you can declare a variable by writing its name and assigning a value.

```
a = 3

print(a, "is the magic number")
```

Comments

Python uses the hash symbol (#) to comment out a line of code. For block comments you can use three speech marks (""") to begin and end the comment. IDEs such as IDLE will allow you to add and remove comments from blocks of text using shortcuts.

```
# this is a comment

"""

so is this

and this

"""

print("this isn't")
```

Arrays

Python doesn’t use arrays, but they can be emulated through the use of lists. Lists use square brackets for referencing, and are based from 0. There is no need to specify the size of a list and they can grow dynamically. The example below shows one way that lists could be used to emulate arrays.

```
onedim =      [0,0,0,0]

twodim =      [[0,0,0,0],[0,0,0,0],[0,0,0,0],[0,0,0,0]]

threedim =    [[[0,0,0,0],[0,0,0,0],[0,0,0,0],[0,0,0,0]],
               [[0,0,0,0],[0,0,0,0],[0,0,0,0],[0,0,0,0]],
               [[0,0,0,0],[0,0,0,0],[0,0,0,0],[0,0,0,0]],
               [[0,0,0,0],[0,0,0,0],[0,0,0,0],[0,0,0,0]]]
```

Structuring code

The scope of statement blocks are defined using indentation, such as the single line code: *a = a + 1* that would be executed only when the *if* statement below is true. As indentation is such a crucial part of the language in defining the functioning of code, it has to be manually applied.

```
if a == 6:
    a = a + 1


a = a + 4
```

Online teaching resources

The Python Tutorial - Python 3

Dive Into Python 3 (python 3.x)

Online Python Tutor (online programming tool that visualises code execution)

Wikibooks - A Beginner's Python Tutorial

Downloads

Python - downloads (free Open Source IDE - Multiplatform)

Python Tools for Visual Studio (Python coding in Visual Studio)

PyDev (python IDE for eclipse)

Visual Basic .Net

Visual Basic .Net (VB.Net) is a modern programming language from Microsoft. The syntax closely resembles VB 6.0, but the underlying functionality is built on the .NET framework and it has a lot in common with C#. VB 6.0 code and VB.Net code are not compatible. You can create software projects that use both C# and VB.Net code. There are plenty of support websites and books for learning the language. Most of them focus on GUI development rather than console mode. VB.Net allows for rapid development of graphical user interfaces, websites and the ability to quickly integrate databases. It also allows for the creation of video games on Windows, Windows Phone and Xbox using the Microsoft XNA Game Studio.

In some cases VB.Net allows variables of different data types to be combined without error, such as adding an integer to a decimal or concatenating a string with an integer. VB.Net is case insensitive, this means that declaring a variable called *name* and then assigning a value to *Name* would treat both as the same variable. However, when using an IDE such as Visual Studio, auto-correction would convert *Name* into *name* and you wouldn't notice this issue. Both of these features can be useful for novice programmers, making for a more forgiving coding experience, but it may breed bad habits.

Development in VB.Net is supported by the Windows only Microsoft Visual Studio IDE. This is an industry leading development environment offering runtime debugging, code tracing, auto-correction and IntelliSense code completion tool. The latest versions of Visual Studio can be acquired by students and institutions for free through Microsoft Dream Spark. Visual Studio Express also offers a free way to develop VB.Net programs on the Windows platform. Other platforms are supported through the Mono project, including command line and GUI development.

Coding examples and syntax

Default program view with "Hello world"

Visual basic command line programs require a *Sub main()* to execute code:

```
Module Module1

  Sub Main()

    Console.WriteLine("Hello world")

  End Sub

End Module
```

Variables

Variables are declared using the *Dim* command, the data type declaration *As Integer* is optional but recommended:

```
Sub Main()

      Dim a As Integer

      a = 3

      Console.WriteLine(a & " is the magic number")

End Sub
```

Comments

Comments are written using REM or ‘ (apostrophe). There are no block comments in VB.Net.

```
Sub Main()
    'this is a comment
    REM so is this
    Console.WriteLine("this isn't")
End Sub
```

Arrays

VB .Net uses round brackets for arrays. Arrays are based from 0, meaning onedim below can hold four values.

```
Dim onedim(3) As String
Dim twodim(3, 3) As String
Dim threedim(3, 3, 3) As String
```

Structuring code

VB.Net uses keywords to define blocks for programming concepts such as classes, subroutines, functions and if statements. If you are using an IDE such as Visual Studio then indentation will automatically occur.

```
If a = 6 Then
    a = a + 1
End If
a = a + 4
```

Online teaching resources

Wikibooks - Fundamentals of Programming: A program

Downloads

Microsoft DreamSpark (for institutional and student licensing - Windows only)

Visual Studio Download (the community version is free - Windows only)

Mono software platform (free Open Source .NET command line tool - Multiplatform)

MonoDevelop (free Open Source IDE - Multiplatform)

GET HELP AND SUPPORT

Visit our website for information, guidance, support and resources at **oxfordaqaexams.org.uk**

You can contact the computer science subject team directly;

E: **computerscience@oxfordaqaexams.org.uk**

OXFORD

INTERNATIONAL
AQA EXAMINATIONS

OXFORD INTERNATIONAL AQA EXAMINATIONS
LINACRE HOUSE, JORDAN HILL, OXFORD, OX2 8TA
UNITED KINGDOM
enquiries@oxfordaqaexams.org.uk
oxfordaqaexams.org.uk