

Copyright

© 2016 PG Online Limited

The contents of this unit are protected by copyright.

This unit and all the worksheets, PowerPoint presentations, teaching guides and other associated files distributed with it are supplied to you by PG Online Limited under licence and may be used and copied by you only in accordance with the terms of the licence. Except as expressly permitted by the licence, no part of the materials distributed with this unit may be used, reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic or otherwise, without the prior written permission of PG Online Limited.

Licence agreement

This is a legal agreement between you, the end user, and PG Online Limited. This unit and all the worksheets, PowerPoint presentations, teaching guides and other associated files distributed with it is licensed, not sold, to you by PG Online Limited for use under the terms of the licence.

The materials distributed with this unit may be freely copied and used by members of a single institution on a single site only. You are not permitted to share in any way any of the materials or part of the materials with any third party, including users on another site or individuals who are members of a separate institution. You acknowledge that the materials must remain with you, the licensing institution, and no part of the materials may be transferred to another institution. You also agree not to procure, authorise, encourage, facilitate or enable any third party to reproduce these materials in whole or in part without the prior permission of PG Online Limited.



Teacher’s Guide

Introduction

This teacher’s guide contains a detailed lesson plan to accompany the set of PowerPoint slides and worksheets for each topic.

The lesson plans are designed to form a basis for ideas for the teacher and should be adapted to suit the teaching style and preferences of the individual teacher, and the resources and nature of the individual school or Computing / ICT department.

The material supplied for this unit includes:

* 6 PowerPoint presentations, each designed to cover one topic, which may take more than one lesson
* 6 worksheets
* 6 homework sheets
* approximately 40 programs written in Python, illustrating how the pseudocode answers to worksheets and homework translate into code
* the same number of programs written in Visual Basic
* An end-of-unit test for assessment purposes

Summary

This unit covers the fundamentals of programming, while recognising that some students may have had little previous experience of programming and others will already be seasoned programmers. It covers Section 3.1 of the specification, including the principles of structured programming in a procedural language such as Python or Visual Basic, arrays, subroutines, parameter passing and text and binary files. Each of the six topics may be spread over more than one lesson, especially if time is spent in the lessons coding solutions and going over homework tasks.

The unit does not teach any particular programming language, leaving this choice up to the teacher. These theory lessons could be run in parallel with practical programming sessions, and it is recommended that students code the pseudocode solutions that they write to give extra experience in practical programming.

Learning Outcomes for the unit

At the end of this Unit all students should be able to:

* explain the difference between a variable and a constant
* write a pseudocode solution for a simple problem involving iteration and selection
* use nested selection and iteration statements
* use arithmetic operations and Boolean operations NOT, AND and OR
* use functions and library subroutines including random number generation
* know how to define and call a subroutine in a program
* construct algorithms using one-dimensional arrays
* read from and write to a simple text file

Most students will be able to:

* write a pseudocode solution for a problem involving iteration and selection
* determine the output from a pseudocode program
* use structured programming techniques and write their own subroutines with parameters
* Construct algorithms using two-dimensional arrays
* read from and write to text and binary files
* Use exception handling routines

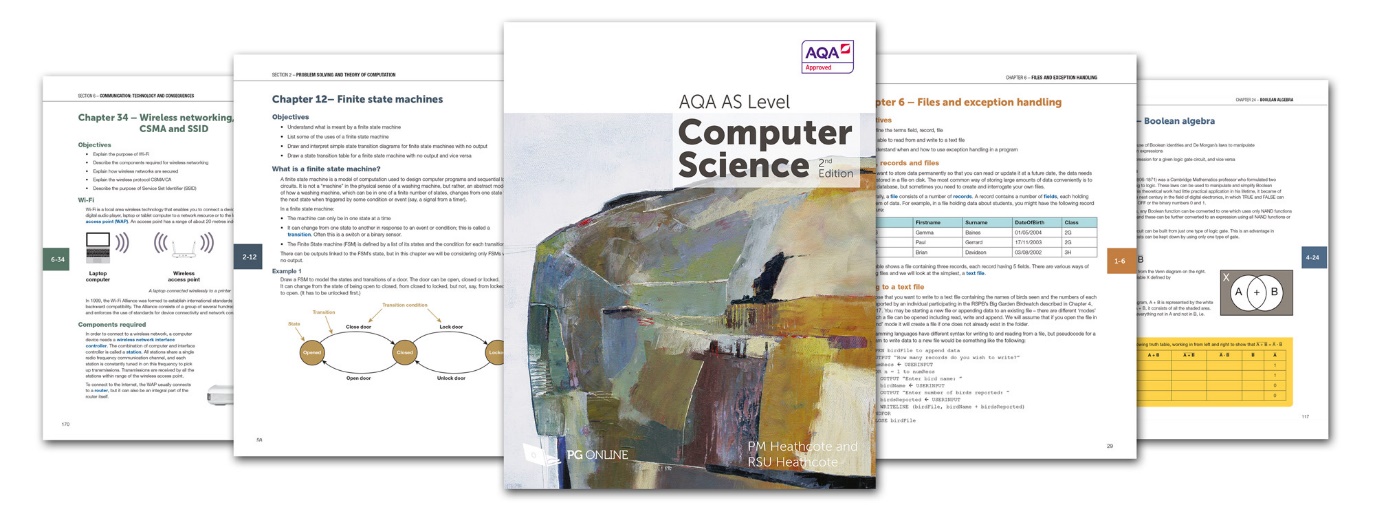
Some students will be able to:

* write complex algorithms involving data structures, subroutines and file-handling
* interpret complex algorithms and determine the output

Previous Learning

Some programming experience in a high level language will be useful for this unit. It is expected that practical programming lessons will take place in parallel with or prior to studying this unit. Students should have a basic understanding of computer systems from lessons delivered as part of the Key Stage 3 national curriculum and preferably also a GCSE course in Computer Science.

Suggested Resources



The textbook AQA AS Computer Science by PM Heathcote and RSU Heathcote, published May 2015 (192 pages) provides comprehensive coverage of all the theoretical topics in the AQA specification (7516).

Each of the six sections in the book corresponds to one of the teaching units in this series and will be extremely useful as a course textbook and also as a revision guide. Sample questions, many taken from past exam papers, are included at the end of each chapter and can be set as additional homework.

The book is in a printed and digital subscription edition. Please refer to www.pgonline.co.uk for ordering and pricing details.

Vocabulary

Vocabulary associated with this Unit, such as:

algorithm, structured programming, data type, variables, constants, assignment, arithmetic operations, Boolean operators, sequence, selection, definite and indefinite iteration, top down design, modular programming, subroutine, procedure, function, parameter, argument, exception handling, global and local variables, field, record, binary file, text file, data structure

Assessment

Students will sit an end-of-unit test.

A few points to note:

These are not live assessment questions. They have all been created from scratch for this scheme of work, and the format is different from that of A Level Paper 1 which relates to a large extent to preliminary material studied over a long period. This test requires students to apply the programming knowledge and skills they have learned while studying this Unit and has questions of varying levels of difficulty for a range of abilities.

Real exam papers go through a serious quality assurance process; feel free to use and adapt these questions as you see fit.

Lesson plans

|  |  |  |
| --- | --- | --- |
| Topic 1 | Programming Basics |  |
| Preparation:  Before each lesson you will need to make copies of the worksheets and homework sheets to give to students. These will serve as valuable notes and so you should encourage them to keep these in a ring binder as reference. If you are teaching Python or VB as the programming language, additional help is provided in the form of program solutions to most of the algorithms in every worksheet and homework.  Although it is not essential to code the programs for which pseudocode is produced, it would be an advantage and provide valuable programming practice.  Learning Objectives:   * Define what is meant by an algorithm and pseudocode * Learn how and when different data types are used * Learn the basic arithmetic operations available in a typical programming language * Become familiar with basic string handling operations * Distinguish between variables and constants * Write pseudocode solutions to simple problems | | |
| Content | | Resources |
| **Starter**  There may be a great deal of variation between students in the amount of programming experience they already have. For some it may be a new skill to be learned, others may already be capable programmers, and the worksheet tasks in this first lesson will help you discover how fast you can proceed.  Show the first slide and introduce the topic and key objectives. Explain that this first lesson is about developing skills and building blocks to allow development of effective algorithms to solve a wide range of tasks.  Remind the students that the focus is on key constructs that can be implemented across a number of different programming languages which share a common core.  Ask students to suggest instructions for getting a robot to carry out simple tasks, such as moving from a particular desk in the classroom to the door, drawing a number of concentric circles, calculating the total value of a bag of mixed coins.  **Main**  Ask for a definition of an algorithm, and then show the slide. Introduce the “calculate paint problem” and encourage the students consider **input**, **calculation**, **output** and simple arithmetic operations such as +, -, / and \*(multiply).  Hand out **Worksheet 1** and ask students to do Task 1. Discuss answers that students have written.  **Pseudocode**  Determine what programming statements students are familiar with and describe the need for pseudocode, which bridges the gap between English and coding syntax within a particular language. Remind students that in some instances problems may be developed by teams working in pseudocode without knowledge of the target programming language that the algorithms will be implemented in.  Introduce the assignment, input and output statements. Show how pseudocode relates to program code, using the example in Visual Basic. If you are using a different programming language, do a similar comparison.  Prompt the group to consider what will be done with data as it is entered and calculated. How will it be identified in an algorithm? Introduce the concept of the identifier (variable) that will point to a place in RAM holding the data. Ask students what happens to the data when the program finishes running (it is lost).  What rules and conventions are there for variable names in the programming language they are studying?  **Data Types**  Introduce the concept of data types and ask how the type of data might differ. Ask why it is necessary to know what type of data is being declared. Lead them to an understanding that different data types use different numbers of bytes and are stored in different formats.  Not all languages require data types to be declared.  Describe the difference between variables and constants, and why constants are sometimes useful.  Ask students to work through the questions in **Worksheet 1 Task 2**. Go over the answers when they are ready.  **Mod and Div**  Mod finds the remainder when dividing one integer by another. In Python a % symbol is used, e.g. 7%3 = 1  Div (// in Python) performs integer division, e.g. 7 div 3 = 2  **String Handling**  Ask students what a name is composed of – a list of characters. Introduce the notion that each letter in a name has an index or position starting from 0 or possibly 1 when implemented in a language. Once it has been established that elements have an index, consider how to access, say, the third element in a name.  In all pseudocode in this unit, it will be assumed that the index starts at 0.  Discuss the existence of a number of language-dependent functions that take strings and return data such as length, substrings and searches for characters.  Ask students to come up with an algorithm for writing a 5-character string in reverse order.  You could do this with:  Name = (“a”, “b”, “c”, “d”, “e”)  reverseName(0) = name(4)  reverseName(1) = name(3)  etc.  The sample Python program in the folder gives a more sophisticated solution, and if you have experienced programmers in the class, they could come up with a better algorithm and code it as an extra task. (They may find it can be done with a single statement using an appropriate string-handling function! It would be a good exercise for some students to look up string handling functions on the Internet and use some of them. Can they take a name like “Fred Jones” and print out “Jones F”? There is a Python solution in the folder).  Work through **Worksheet 1 Task 3**.  **Plenary**  Go over the Worksheet answers.  Assess students’ understanding of algorithms, pseudocode, assignment, variable, identifier, operator, sequence, data types and string handling.  Give out **Fundamentals Homework 1**. | | PowerPoint Guide: Fundamentals Topic 1 Programming Basics.ppt  Fundamentals Worksheet 1 Programming basics.docx  Fundamentals Worksheet 1 Answers.docx  Python program PP1 ReverseString.py  PP1 reverse string.py  PP1 Surname and initial.py  Fundamentals Homework 1 Programming basics.docx  Fundamentals Homework 1 Answers.docx  Python and VB coded programs in folders |

|  |  |  |
| --- | --- | --- |
| Topic 2 | Selection |  |
| Learning Objectives:   * be able to use relational operators * be able to use Boolean operations AND, OR * be able to use nested selection statements | | |
| Content | | Resources |
| **Starter**  Recall from the last lesson the idea of an algorithm as a set of instructions, and that instructions can be carried out in sequence one after another.  Consider how data can be input and assigned to temporary storage, and decisions might be made as to what to do next.  Ask how a decision is made that it is time to get ready for school.  e.g. “IF time = 7am THEN get ready”.  “IF time > 7am THEN running late equals True”.  Prompt the students to answer questions about knowing when to go to lesson 2 etc.  “IF time = 9.30 AND bell = ring THEN go next lesson”. How do we make decisions?  We use “IF Boolean expression is True THEN do something”. A **Boolean expression** evaluates to True or False.  **Main**  Using the Power Point presentation, work through the concepts of Boolean conditions, relational operators and the simple IF.. THEN.. construct.  **Complex Boolean expressions**  Consider the way in which Boolean operators AND, OR can be used to link relational expressions to create more complex rules. Understand that even the most complex rule evaluates to either true or false.  You may wish to let students work in small groups to evaluate the conditions for different values of x, y and z shown on the slide. (Answers on next slide).  **IF .. THEN.. ELSE.. END IF**  Now consider what might happen if the condition is not met, this will introduce the concept of **ELSE** which defines the actions to be taken when the condition is not met.  Give out **Worksheet 2** and ask students to complete the questions in **Task 1**.  Go through the answers.  **IF .. THEN .. ELSE IF .. ELSE**  Next, ask students how one might deal with multiple conditions, (for example, 3 or more different membership types or age ranges) as a way to introduce IF.. THEN.. ELSEIF.. ELSE..  **CASE statement**  Work through the examples on the PowerPoint slide with students recording their answers on paper or answering when called upon. Consider the CASE statement as a shortcut provided by the programming language and how it is logically equivalent to the multiple IF .. THEN.. ELSEIF.. ELSE.. ENDIF constructs.  Give out **Worksheet 2** and ask students to complete the questions in **Task 2**.    Next consider “nesting” an IF statement within another. IF condition A is TRUE THEN, IF condition B is TRUE THEN etc… Using slide 20 consider the difference and similarity between using nested IF statements and using AND in a complex IF statement.  Consolidate understanding by working through **Worksheet 2**, **Task 3**. There are more questions on this Worksheet than students will be able to complete in a lesson – use as you see fit.  Programs in Python and VB are provided.  **Plenary**  Work through the answers to the Worksheet Task.  Summarise the key points using the last slide.  Introduce the homework. | | PowerPoint Guide: Fundamentals Topic 2 Selection.ppt  Fundamentals Worksheet 2 Selection.docx  Fundamentals Worksheet 2 Answers.docx  Fundamentals Homework 2 Selection.docx  Fundamentals Homework 2 Answers.docx  Python and VB coded programs in folders |

|  |  |  |
| --- | --- | --- |
| Topic 3 | Iteration |  |
| Learning Objectives:   * Understand and use three different types of iterative statement: WHILE, REPEAT and FOR * Be familiar with, and be able to use, random number generation | | |
| Content | | Resources |
| **Starter**  Recap what we understand thus far;   * algorithms are instructions to carry out a task * data values can be assigned to temporary memory locations that use an identifying name * instructions can be carried out in sequence, one after another, OR * branching statements such as IF.. THEN which use simple and complex expressions evaluating to TRUE or FALSE can be used to control which statements are performed   At this point, do we have all we need to write reasonably comprehensive instructions? One might argue that with a few exceptions such as a way of permanently storing and retrieving data, we now have all we need.  What instructions might be used to calculate a letter grade based on a test mark?  e.g. IF mark > 90 THEN grade = A  But what if the calculation is required for 10, 20 or 100 students? **Iteration** or **Loop** constructs are primarily about efficiency.  **Main**  Using the Power Point slide headed **Iteration**, work through a definition and understanding of iteration.  Recall how a Boolean expression was used in **selection** statements to determine the sequence of instructions. How does this differ from **iteration**?  They are essentially the same thing, with one exception; when the condition is tested, one branch of the loop goes back to a previous line in the code (the start of the loop) and one or more instructions are repeated before the condition is again evaluated.  This is done repeatedly until the condition is met – otherwise we have an infinite loop.  **WHILE … ENDWHILE loop**  Prompt students to consider that there are two key points in the code where the loop condition can be evaluated; **before** the instructions within the loop, or **after** the instructions within the loop.  These correspond **to entry condition** and **exit condition** loops.  Ask students work through **Worksheet 3**, **Task 1**.  **REPEAT .. UNTIL loop**  Now consider the case of **exit condition** loops. How do these differ from **entry condition** loops?  Prompt students to understand that instructions within the loop may never be executed in an entry condition loop but they are always executed at least once in an exit condition loop.  Work through the questions and examples on the slides on the Repeat..Until loop with the group.  Python does not support the Repeat..Until loop. Show that an equivalent WHILE loop can be written for any REPEAT loop.  **Infinite loop**  Discuss the use of infinite loops in games and control applications. The loop will continue indefinitely.  Allow students to consolidate their understanding by working through **Worksheet 3, Task 2.**  When complete, work through the correct answers, either as a group or through observing each student’s work.  **FOR .. NEXT loop**  Ask students to consider what might be necessary if the loop is to be executed a set number of times.  We need a condition, a starting value and an ending value.  We also need a **counting variable** and a method of incrementing the variable within the loop, such as  count 🡨count + 1  We could do this effectively using the constructs we have already come across.  Programming languages provide a shortcut for executing instructions a set number of times, and this is the FOR loop. This is implemented in different ways, depending on the programming environment.  The pseudocode on the slides using the NEXT statement to end the FOR loop. Alternatively, you can use an END FOR statement.  Work through the examples on the slides, with students recording their answers to the trace table problems either individually or in pairs. Monitor students to ensure everyone understands the concepts  **Random number generators**  Now that we have considered repeating instructions that terminate upon meeting a set condition and we have considered repeating instructions a set number of times, we can consider the notion of randomness and when we might use it. Random numbers are frequently used in games and simulation problems, for example simulating the throw of a dice, the number of squares moved, or in the modelling of business and environmental problems for example.  To further consolidate the topics covered students should work through **Worksheet 3 Task 3**.  **Plenary**  Work through the answers to Task 3, either as a group or as part of a more formal assessment.  Recap the lesson objectives to be sure students understand the role of looping within algorithms and in particular how loops can provide efficiency. Consider again entry condition, exit condition and for loops.  Give out the Homework for this section. | | PowerPoint Guide: Fundamentals Topic 3 Iteration.ppt  Fundamentals Worksheet 3 Iteration  Fundamentals Worksheet 3 Answers  (sample Python and VB programs in relevant folders)  Fundamentals Homework 3 Iteration  Fundamentals Homework 3 Answers  Python and VB coded programs in folders |

|  |  |  |
| --- | --- | --- |
| Topic 4 | Arrays |  |
| Learning Objectives:   * Be familiar with the concept of a data structure * Use 1- and 2-dimensional arrays in the design of solutions to simple problems | | |
| Content | | Resources |
| **Starter**  Recap what students have learned about instructions that include sequence, selection, iteration and assignment.  Ask students to describe what a variable declaration accomplishes; it creates an identifier, a name that points to a temporary memory location, it also specifies a data type and this sets aside an amount of memory, perhaps a byte for a character or four bytes for an integer.  By declaring the data type, we also enable the programming environment to give us early warning of errors in source code such as attempting to assign a character value to an integer variable.  However, some languages such as Python have no explicit variable declaration.  Use the revision of elementary data types to lead on to an understanding that many situations such as storing several students’ names and marks might require something more complex.  **Main**  **Arrays**  Ask, what is the purpose of an array, as with loops it is possible to solve the problem using lots of variables such as name1, name2… name100?  Clearly, using an identifier with an index, **name[index],** is more efficient and cuts out potentially hundreds of lines of repetitive code, and this is the essential purpose served by the array.  Point out that arrays are often used with FOR loops which will loop from the start to the end of the array and that the counting variable or **index** is used to access each element in the array.  In many programming languages array indices start at 0 and we will follow this convention throughout.  Do the examples on the slides as a class exercise, and then ask students to do the questions in **Task 1** of Worksheet 4.  **Two-dimensional arrays**  Introduce the idea of two dimensional arrays that can be created simply by specifying two indices. Note that this can be thought of in terms of a grid or matrix using row and column.  The pseudocode and trace table exercises on the 2-D array slides offer an opportunity to explore assigning data to a 2D array.  Students should then attempt **Worksheet 4 Task 2** to consolidate their understanding of 2D arrays.  Not all students will be able to complete all questions – the last question will challenge the top students!  **Multi-dimensional arrays**  Extend the idea of 2-dimensional array to arrays in 3 or even more dimensions. Don’t spend too much time on this as it is unlikely that students will meet them again on this course!  **Plenary**  Check student understanding using the worksheets.  Recap the main points in this topic.  Give out the homework. | | PowerPoint Guide: Fundamentals Topic 4 Arrays.ppt  Fundamentals Worksheet 4 Arrays  Fundamentals Worksheet 4 Answers  Fundamentals Homework 4 Arrays  Fundamentals Homework 4 Answers  Python and VB coded programs in folders |

|  |  |  |
| --- | --- | --- |
| Topic 5 | Programming Subroutines |  |
| Learning Objectives:   * Be familiar with subroutines, their uses and advantages * Use subroutines that return values to the calling routine * Use arguments/parameters to pass data within programs * Contrast the use of local and global variables | | |
| Content | | Resources |
| **Starter**  Introduce the topic of subroutines by asking, for example, how students would set about writing a long, complex program displaying a menu of several options and then executing code depending on which option was chosen. Have they covered top-down design?  Subroutines include procedures and functions. These can be written by the programmer or form part of the programming language environment.  **Main**  Show the first slide and make sure everyone understands the sequence in which instructions are performed. in some languages the main program is written before the subroutines, in others, after all the subroutines.  Show the examples of built-in functions. Can students think of others?  **Parameters**  Parameters are frequently, though not always, passed to a subroutine. The order in which they are written must be the same in the calling statement as in the subroutine header.  Give out **Worksheet 5** and ask students to complete **Task 1**.  There are more questions on this worksheet than most students will be able to complete in a lesson – question 4 will challenge the most able students.  When students have completed Task 1, go over a solution in class if necessary.  The next two slides show how a function may return a value via a RETURN statement. The local variable name used in the function, e.g. ***product***, is independent of the identifier used in the calling statement.  **Local and global variables**  Explain the concept of **scope** in the context of variables. Subroutines often use local variables, which keeps them independent of the calling program. They can then be re-used in any program, without the programmer having to know what identifiers are used in the subroutine. A further advantage is that the subroutine can be tested independently of the main program, by setting variables and using just a calling statement in a “dummy” main program and observing the result.  If you want to explain the concept of local variables further, here is a more detailed explanation:  Python organises all of the names defined in a program into **namespaces** – collections of all the names that are available to be accessed at a particular point in time during the execution of a Python program. When we start Python, we create two namespaces. The first is the **built-in** namespace and includes all the system-defined names of functions and data types such as int, str, float.  As we use names in a Python program, they are added to the **main** namespace.  When a function is invoked, a new namespace is created corresponding to the function itself. This namespace includes the names that are created inside the function, including any formal parameters, and any names used on the left-hand side of an assignment statement in the body of the function.  These names are referred to **as local variables**.  When the function is completed, the local namespace is destroyed.  When a name is used on the **right-hand** side of a statement,  Python searches through the namespaces in the following order:   1. The current local namespace, if one exists 2. The main namespace 3. The built-in namespace   When you use a name on the **left-hand** side of an assignment statement, Python searches only the current namespace.   1. If the name is not found, a new name is created in the current namespace. 2. If the name is found, then the old reference will be replaced with the object from the right-hand side of the assignment statement.   This means that the same name may exist in many different namespaces, but Python will always use the name governed by the above rules.  **The modular approach**  Discuss the advantages of using subroutines to create a top-down, modular structure. When the students start looking at the Skeleton Program in the preliminary material they will need a thorough understanding of this approach.  The more practice students get on writing pseudocode and turning this into code, the better. Ask them to work on the questions in **Task 2**.  **Plenary**  Depending on the time available, go over the questions or leave some tasks for another time.  Give out **Homework 5**. Python and VB programs are given in the relevant folders. | | PowerPoint Guide: Fundamentals Topic 5 Subroutines.ppt  Fundamentals Worksheet 5 Subroutines  Fundamentals Worksheet 5 Answers  Fundamentals Homework 5 Subroutines  Fundamentals Homework 5 Answers  Python and VB coded programs in folders |

|  |  |  |
| --- | --- | --- |
| Topic 6 | Files and exception handling |  |
| This lesson can be subdivided into two topics, **file handling** and **exception handling routines**, and can be split over two sessions.  Learning Objectives:   * define the terms field, record, file * be able to read from and write to a text file * be able to read from and write to a binary file * understand when and how to use exception handling in a program | | |
| Content | | Resources |
| **Starter**  Ask for examples of data that may be stored in files. Often these are database files or tables, such as stock control, payroll, library, hotel booking or student record applications, for example. What data might files hold in a game program?  Clearly holding all the data in a file or database of some sort is the only way of permanently storing data. Typically, a Python or VB program may provide the “front end” which accesses a database program for storing and retrieving data.  **Main**  Define files, records and fields. It is easiest to talk about these using a tabular format like a database table. Some languages, such as Pascal, have a well-defined record data type consisting of many fields. Others, such as Python, do not implement a record data structure in the same way.  **Text and binary files**  A text file contains only character strings, separated by a comma or other defined separator. A binary file can contain records consisting of many fields of different types such as string, integer, Boolean.  **File open mode**  The programmer must specify whether a file is to be opened for reading, writing or appending, and whether it is a text file or a binary file. The actual filename and pathname must also be specified.  Sometimes the programmer may know how many records are to be read or written, and other times the file is simply read until the end of file is reached. In pseudocode, “While not End of File” is sufficiently explicit. Different programming languages will have different ways of implementing this.  The pseudocode used here for reading from or writing to a file is in the format  WRITE(filename, recordname)  **Handling a text file**  In the example on the slide “Handling a text file” the question is asked: *How many fields are there in each record?*  There is just one field in each record, a single element of the array pupilArray.  In the example on the next slide, two fields separated by a comma are written to a single record. Each record is written on a single line, and the file can be viewed in Notepad. When using pseudocode, you could define the record at the top of the program, e.g.  carRecord = [carModel, carPrice]  and then use pseudocode  write (carFile, carRecord)  to write the record. Implementation will vary according to the language used.  Give out **Worksheet 1**.  This contains an almost complete example of reading from and writing to a text file. Students are asked to insert a few missing lines. This program will be a useful example to help them complete the homework and the final assessment test, so they need to keep it safely in their folders!  **Binary files**  The implementation of binary files varies considerably in different languages. A simple example is given of reading a graphics file and copying it to a new file.  Ask students to complete **Task 2** on the Worksheet..  A simple example is given of reading a graphics file and copying it to a new file.  **Exception handling**  Any program in which the user has to enter data is fundamentally prone to crashing because of user errors in data entry, so every means possible must be used to prevent this from happening. Validation is extremely important, but if that fails to trap all errors, then exception handling routines can be used by the programmer to stop the program from crashing in exceptional circumstances.  Show the examples on the slides of how this can be implemented in different situations.  **Task 3** of the worksheet gives another example of a fairly lengthy program for students to study and answer questions on.  **Plenary**  Go over the answers to the worksheet.  As a general rule the units do not include homework after the final lesson before the assessment, so you may like to give out this homework as soon as they have covered file handling, before the final lesson. It will be a very useful revision exercise before they attempt the Final Assessment, which contains a similar program for students to write. | | PowerPoint Guide: Fundamentals Topic 6 Files and exception handling.ppt  Fundamentals Worksheet 6 Files and exception handling  Fundamentals Worksheet 6 Answers  Fundamentals Homework 6 Files and exception handling  Fundamentals Homework 6 Answers  Python and VB coded programs in folders |

|  |  |
| --- | --- |
| Unit assessment | |
| Learning Outcomes:   * Students will apply their knowledge in answers to a range of questions * Teachers will be able to highlight areas of strength and any gaps in students’ understanding of the material covered | |
| Content | Resources |
| Students should complete the **Fundamentals** **Final** **Assessment Test**.  Answers are available in the **Fundamentals** **Final Assessment Answers** sheet.  The test should be printed and answered on separate sheets of paper.  The test is worth 50 marks, but the final question, worth 15 marks, will probably take at least half an hour for most students to complete. The whole test could take up to 2 hours, as it mainly involves writing short programs. You may wish to omit the final question from the test, or make this an “open book” test. | Fundamentals Final Assessment  Fundamentals Final Assessment Answers  Python and VB coded programs in folders |

Acknowledgements

The authors and publisher would like to thank all contributors for their kind permission to reproduce their photographs or images, screenshots of their websites or other copyright material in the PowerPoint Guides.

Images copyright © Shutterstock

Every effort has been made to trace the copyright holders and we apologise in advance for any unintentional omissions. We would be pleased to insert the appropriate acknowledgement in any subsequent edition of this publication.

This material contains links to relevant websites featured in the teacher’s guide. Every effort has been made to ensure that at the time of distribution, the links remain unbroken, the material remains up-to-date and that links are not inadvertently linked to sites that could be considered offensive. PG Online cannot be held responsible for the content of any website mentioned in this material. It is sometimes possible to find relocated sites by typing the original URL into a browser. Any errors should be reported directly to support@pgonline.co.uk and changes will be made in any subsequent editions of the material.

Artwork



England © 2015 Heather Duncan

Oil on canvas, 100x100cm

www.heatherduncan.com