endorsed for BTEC

REVISE BTEC NATIONAL Computing REVISION GUIDE

Includes FREE online edition

Pearson

endorsed for BTEC

REVISE BTEC NATIONAL Computing

REVISION GUIDE

Series Consultant: Harry Smith

Authors: Steve Farrell, Mark Fishpool, Christine Gate and Richard McGill

While the publishers have made every attempt to ensure that advice on the qualification and its assessment is accurate, the official specification and associated assessment guidance materials are the only authoritative source of information and should always be referred to for definitive guidance.

This qualification is reviewed on a regular basis and may be updated in the future. Any such updates that affect the content of this Revision Guide will be outlined at **www.pearsonfe.co.uk/BTECchanges**. The eBook version of this Revision Guide will also be updated to reflect the latest guidance as soon as possible.

A note from the publisher

In order to ensure that this resource offers high-quality support for the associated Pearson qualification, it has been through a review process by the awarding body. This process confirms that this resource fully covers the teaching and learning content of the specification or part of a specification at which it is aimed. It also confirms that it demonstrates an appropriate balance between the development of subject skills, knowledge and understanding, in addition to preparation for assessment.

Endorsement does not cover any guidance on assessment activities or processes (e.g. practice questions or advice on how to answer assessment questions), included in the resource nor does it prescribe any particular approach to the teaching or delivery of a related course.

Pearson examiners have not contributed to any sections in this resource relevant to examination papers for which they had prior responsibility.

Examiners will not use endorsed resources as a source of material for any assessment set by Pearson.

Endorsement of a resource does not mean that the resource is required to achieve this Pearson qualification, nor does it mean that it is the only suitable material available to support the qualification, and any resource lists produced by the awarding body shall include this and other appropriate resources.

Pearson

Introduction

Which units should you revise?

This Revision Guide has been designed to support you in preparing for the externally assessed units of your course. Remember that you won't necessarily be studying all the units included here – it will depend on the qualification you are taking.

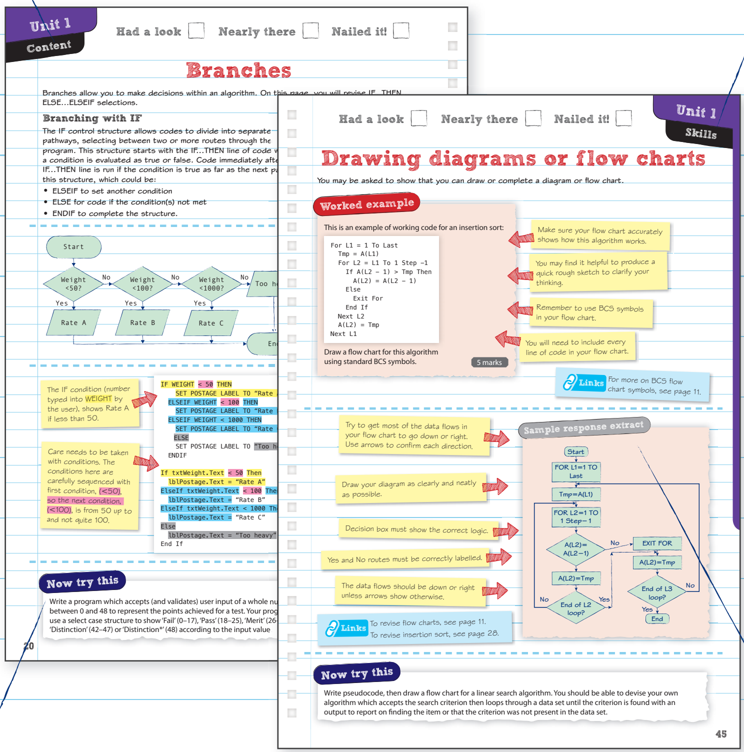
BTEC National Qualification	Externally assessed units
For both: Extended Certificate Foundation Diploma	1 Principles of Computer Science 2 Fundamentals of Computer Systems
Extended Diploma	1 Principles of Computer Science 2 Fundamentals of Computer Systems 3 Planning and Management of Computing Projects 4 Software Design and Development Project

Your Revision Guide

Each unit in this Revision Guide contains two types of pages, shown below.

Content **pages** help you revise the essential content you need to know for each unit.

Skills **pages** help you prepare for your exam or assessed task. Skills pages have a coloured edge and are shaded in the table of contents.



Unit 1 Content

Had a look □ Nearly there □ Nailed it! □

Branches

Branches allow you to make decisions within an algorithm. On this page, you will revise IF...THEN ELSE...ELSEIF selections.

Branching with IF

The IF control structure allows codes to divide into separate pathways, selecting between two or more routes through the program. This structure starts with the IF...THEN line of code w a condition is evaluated as true or false. Code immediately afte IF...THEN line is run if the condition is true as far as the next pa this structure, which could be:

- ELSEIF to set another condition
- ELSE for code if the condition(s) not met
- ENDIF to complete the structure.

The IF condition (number typed into WEIGHT by the user), shows Rate A if less than 50.

Care needs to be taken with conditions. The conditions here are carefully sequenced with first condition, (<50), so the next condition, (<100), is from 50 up to and not quite 100.

```
IF WEIGHT < 50 THEN
   SET POSTAGE LABEL TO "Rate
 ELSEIF WEIGHT < 100 THEN
   SET POSTAGE LABEL TO "Rate
 ELSEIF WEIGHT < 1000 THEN
   SET POSTAGE LABEL TO "Rate
  ELSE
   SET POSTAGE LABEL TO "Too h
 ENDIF

If txtWeight.Text < 50 Then
 lblPostage.Text = "Rate A"
ElseIf txtWeight.Text < 100 The
 lblPostage.Text = "Rate B"
ElseIf txtWeight.Text < 1000 Th
 lblPostage.Text = "Rate C"
Else
 lblPostage.Text = "Too heavy"
End If
```

Now try this

Write a program which accepts (and validates) user input of a whole nu between 0 and 48 to represent the points achieved for a test. Your prog use a select case structure to show 'Fail' (0–17), 'Pass' (18–25), 'Merit' (26- 'Distinction' (42–47) or 'Distinction*' (48) according to the input value

20

Had a look □ Nearly there □ Nailed it! □

Unit 1 Skills

Drawing diagrams or flow charts

You may be asked to show that you can draw or complete a diagram or flow chart.

Worked example

This is an example of working code for an insertion sort:

```
For L1 = 1 To Last
  Tmp = A(L1)
  For L2 = L1 To 1 Step -1
    If A(L2 - 1) > Tmp Then
      A(L2) = A(L2 - 1)
    Else
      Exit For
    End If
  Next L2
  A(L2) = Tmp
Next L1
```

Draw a flow chart for this algorithm using standard BCS symbols. 5 marks

Make sure your flow chart accurately shows how this algorithm works.

You may find it helpful to produce a quick rough sketch to clarify your thinking.

Remember to use BCS symbols in your flow chart.

You will need to include every line of code in your flow chart.

Links For more on BCS flow chart symbols, see page 11.

Try to get most of the data flows in your flow chart to go down or right. Use arrows to confirm each direction.

Draw your diagram as clearly and neatly as possible.

Decision box must show the correct logic.

Yes and No routes must be correctly labelled.

The data flows should be down or right unless arrows show otherwise.

Links To revise flow charts, see page 11. To revise insertion sort, see page 28.

Sample response extract

Now try this

Write pseudocode, then draw a flow chart for a linear search algorithm. You should be able to devise your own algorithm which accepts the search criterion then loops through a data set until the criterion is found with an output to report on finding the item or that the criterion was not present in the data set.

45

Use the **Now try this** activities on every page to help you test your knowledge and practise the relevant skills.

Look out for the **sample response extracts** to revision questions or tasks on the skills pages. Post-its will explain their strengths and weaknesses.

Contents

Workbook also available for externally assessed units
ISBN 9781292150192

Unit 1: Principles of Computer Science

Unit 2: Fundamentals of Computer Systems

Unit 3: Planning and Managing of Computing Projects

Unit 4: Software Design and Development Project

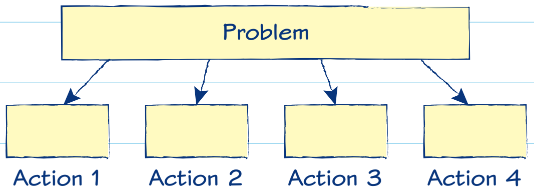
A small bit of small print
Pearson publishes Sample Assessment Material and the Specification on its website. This is the official content and this book should be used in conjunction with it. The questions in *Now try this* have been written to help you test your knowledge and skills. Remember: the real assessment may not look like this.

Identifying problems and processes

Computational thinking enables you to analyse a problem, break it down into smaller parts, recognise patterns within the problem and finally identify a strategy to solve it. Over the next few pages, you will revise the first stage of computational thinking – decomposition.

Decomposition – step 1

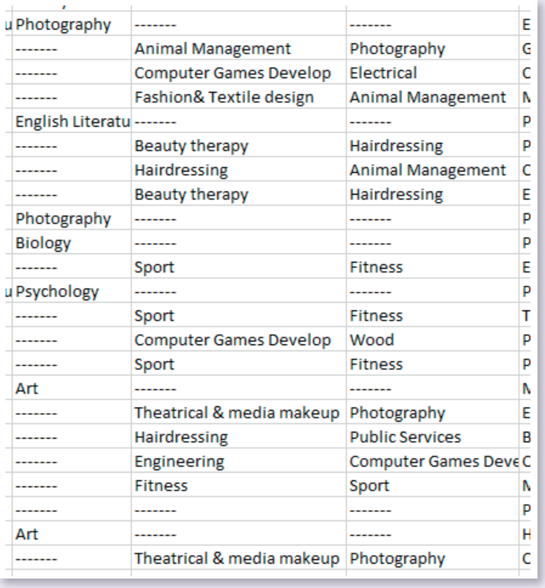
Every computer program is made up of a number of processes or actions. Before an app can be written, the problem to be solved and the processes (actions) inside the app need to be identified. The first step in decomposition involves identifying and describing the problem and the processes required to solve it.



Case study: App to collate spreadsheets

An app is being developed to bring together several spreadsheets from different members of a team into one workbook which can then be used for a mail merge.

The mail merge is to be from a worksheet in the workbook using data which is selected and copied from member worksheets.



u	Photography	-------	-------	E
	-------	Animal Management	Photography	G
	-------	Computer Games Develop	Electrical	C
	-------	Fashion& Textile design	Animal Management	M
	English Literatu	-------	-------	P
	-------	Beauty therapy	Hairdressing	P
	-------	Hairdressing	Animal Management	C
	-------	Beauty therapy	Hairdressing	E
	Photography	-------	-------	P
	Biology	-------	-------	P
	-------	Sport	Fitness	E
u	Psychology	-------	-------	P
	-------	Sport	Fitness	T
	-------	Computer Games Develop	Wood	P
	-------	Sport	Fitness	P
	Art	-------	-------	M
	-------	Theatrical & media makeup	Photography	E
	-------	Hairdressing	Public Services	B
	-------	Engineering	Computer Games Deve	C
	-------	Fitness	Sport	M
	-------	-------	-------	P
	Art	-------	-------	H
	-------	Theatrical & media makeup	Photography	C

Only data which are names of courses are to be copied to the worksheets for mail merges. The cells with dashes in them are to be ignored.

The problem must be clearly described in language that will be familiar to the user. Later, the solution will be checked against the problem to ensure all needs have been met.

1. Copy and paste member worksheets into designated worksheets in the master workbook.
2. Run macro code to loop into each of the designated worksheets. Loop down each row of data and loop along the columns in each row.
3. Select every data item that is not dashes and copy.
4. Move to appropriate cell in merge worksheet, then paste special as value.
5. Complete loops when blank cells are found.
6. Save workbook.

Once the problem has been described, the processes needed to program the solution can be identified. These become the framework of the solution, with the detailed steps needed to implement each process added at the next stage.

Now try this

You are designing an app to allow builders to price jobs.

(a) Write down a list of outputs needed for this app.
(b) Write down a list of inputs needed for this app.
(c) Choose one input and describe the actions needed to test it for validity.

Think about what data is actually needed and don't include more than that.

Breaking down problems and processes

Once the problem and processes have been identified, the next stage of decomposition is to break down the problem and processes into a sequence of steps.

Case study App to support a competition

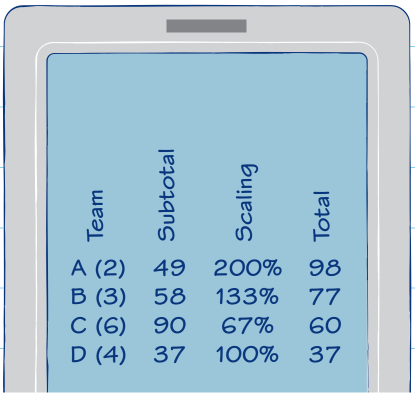
An app is being developed to support a charity treasure hunt. There will be four teams of up to six people. Each team will be given photographs of various places within the treasure hunt boundary and clues on sheets of paper. When calculating the scores, the following rules must be applied:

- Photo questions are each worth 1 point.
- Other questions are each worth 2 points.
- Deduct 1 point for a wrong answer.
- Blank answers are each worth 0 points.
- The points total is scaled according to how many are in the team: each member is worth 25%, and the total is then inverted. So a team of four has a scaling of 100% and a team of two has 200%.

Last year, the results were calculated in a spreadsheet. This year the organisers plan to use a smartphone app.

Steps in calculating points using app

1. Enter the team name with number of members in the team.
2. Enter number of correct photo questions.
3. Enter number of correct other questions.
4. Enter number of wrong questions.
5. Calculate points for correct other questions by multiplying number correct by 2.
6. Calculate subtotal by adding adjusted other questions to correct photo questions, then subtracting number of wrong questions.
7. Calculate team scaling.
8. Multiply subtotal by team scaling.
9. Repeat steps 1 to 8 for all teams.
10. Sort by totals to show winning team.



Team	Subtotal	Scaling	Total
A (2)	49	200%	98
B (3)	58	133%	77
C (6)	90	67%	60
D (4)	37	100%	37

Now try this

A local independent corner store would like to check its stock using a mobile phone app.

(a) What do you think would be needed to make this practical?

(b) What steps are needed to make the app work using the phone's camera to scan barcodes on stock items?

Think specifically about how barcodes can be used.

Communicating problems and processes

On this page, you will revise the final stage of decomposition – how to describe and communicate the key features of problems and processes to clients and other programmers.

Communicating algorithms

Describing problems and processes as a set of structured steps – an algorithm – will enable clients and programmers to understand how a proposed solution will work. At this stage, mistakes in the understanding of the problem or design flaws may become apparent before the project moves to the coding phase.

Links Revise algorithm design on page 8.

Case study — Restaurant tablet

A restaurant is planning to use a tablet for each table so diners can browse the menu using video footage of the ingredients and dishes, and read reviews from other diners before choosing their meal. The tablet will display the final bill, at which point diners can log into their PayPal account or make a payment using card or cash to the waiter. Clients may then enter feedback about their meal.

Using pseudocode

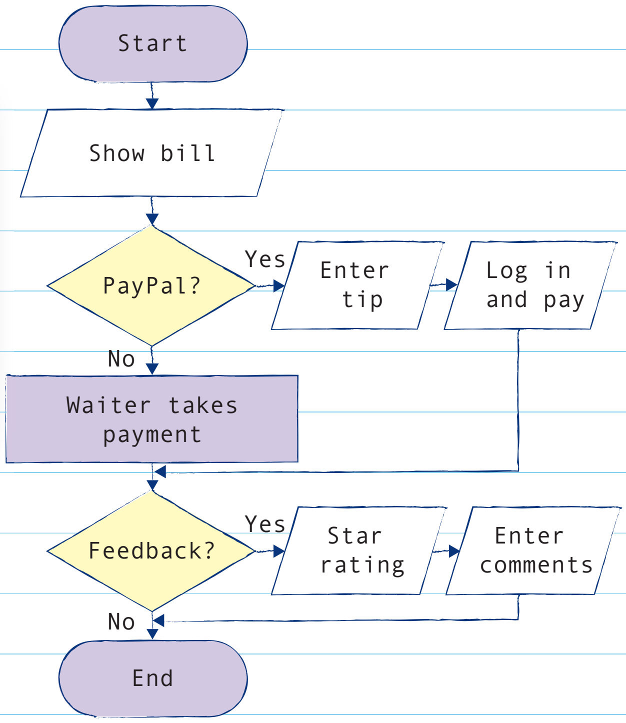
Pseudocode can be used to explain to clients and other programmers how code will work. This pseudocode shows the process by which a customer pays and is given an option to rate their experience at the restaurant.

```
If PayPal
  Enter tip amount
  Log in to PayPal
If not PayPal
  Waiter takes payment
User selects whether they want to leave feedback
If yes
  Enter star rating
  Write comments
Show thank you screen on the tablet
```

Links Revise how to produce, apply and interpret pseudocode on pages 9–10.

Using a flowchart

You could also use a flowchart to show the algorithms required to demonstrate the processes.



Now try this

An independent jeweller is setting up a website as a retail outlet. It would like to offer a loyalty discount for customers who purchased within the last year as well as giving a reduction of 10% on the sale for more than one item and a choice of postage rates of next day or economy. The jeweller is VAT registered, so VAT needs to be added to the sale after any discounts and postage.

Describe the processes that would be needed to calculate a quote for a customer visiting the website.

What actions are needed to produce the quote? Each action will be a process.

Pattern recognition

Once the problem and processes have been described, the next step involves pattern recognition where you look for repeating features within problems and between problems. This will enable you to create code that can be reused in other apps.

Common elements and individual differences

- Identifying common elements or features in problems needing coded solutions or within systems requiring maintenance can result in producing program code that can be re-used in other apps.
- Identifying any differences and individual elements within problems that can utilise common code need interpreting so library code can be adapted by using appropriate parameters or branching within subroutines.

Code libraries

Code libraries are used by many organisations to improve the effectiveness of programming teams by keeping copies of program segments that are easy to find and to re-use.

Documentation is an essential element of a code library as this is needed to clearly identify code segments with how they can be used as reliable building blocks for new apps.

Debugging time can be reduced by using library code as these program segments will have already been extensively tested and signed off as fit for purpose.

Parameters

Parameters are vital for much re-usable code to control the values that can be passed into a subroutine and so make the workings of this code reliable and predictable.

This code uses 0, 99, 2, 5 as fixed numbers, making it very inflexible as the code would need editing to be used elsewhere.

```
Sub ColumnSort(ByRef SA (,))
Dim Z, FirstRow, Passes, Item As Integer
Dim Temp As String
FirstRow = 0
Passes = FirstRow
While Passes <= 99
  Item = FirstRow
  While Item <= 99
    If SA(2, Item) > SA(2, Item + 1) Then
      For Z = 0 To 5
        Temp = SA(Z, Item)
        SA(Z, Item) = SA(Z, Item + 1)
        SA(Z, Item + 1) = Temp
      Next Z
    End If
    Item = Item + 1
  End While
  Passes = Passes + 1
End While
End Sub
```

An example of code that is hard to re-use.

This code uses parameters NoCols, FirstRow, LastRow, Col instead of fixed numbers, making the code much more re-usable.

```
Sub ColumnSort(ByRef SA(,), NoCols, FirstRow,
LastRow, Col)
Dim Z, Passes, Item As Integer
Dim Temp As String
Passes = FirstRow
While Passes <= LastRow - 1
  Item = FirstRow
  While Item <= LastRow - 1
    If SA(Col, Item) > SA(Col, Item + 1) Then
      For Z = 0 To NoCols
        Temp = SA(Z, Item)
        SA(Z, Item) = SA(Z, Item + 1)
        SA(Z, Item + 1) = Temp
      Next Z
    End If
    Item = Item + 1
  End While
  Passes = Passes + 1
End While
End Sub
```

An example of the same code tweaked to make it re-usable.

Now try this

Write down all the benefits and disadvantages of reusing code within an organisation. Do the positives outweigh the negatives?

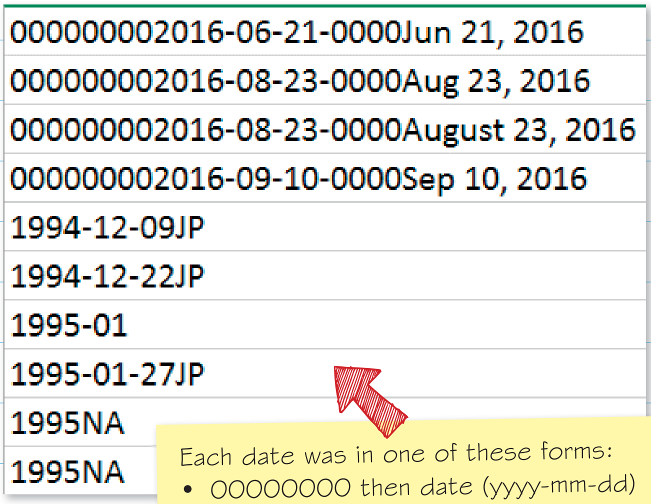
Use a single line for each item so it's easy to see whether the positives outweigh the negatives.

Describing patterns and making predictions

Once you have identified repeating features, you will need to describe the patterns. You can then make predictions based on these patterns which will enable you to design program algorithms.

Case study: Cleaning dates data

Research was carried out to find computer games titles that had been on sale during the last three decades. Over 10 000 items of data were downloaded from the internet. Some dates were not recognised as such by the spreadsheet. The number of these entries made manual editing a poor option due to the time it would take and the errors that might be introduced into the data set by so much repetitive work.



000000002016-06-21-0000Jun 21, 2016
000000002016-08-23-0000Aug 23, 2016
000000002016-08-23-0000August 23, 2016
000000002016-09-10-0000Sep 10, 2016
1994-12-09JP
1994-12-22JP
1995-01
1995-01-27JP
1995NA
1995NA

Each date was in one of these forms:

- 00000000 then date (yyyy-mm-dd) then -0000 then the date as text
- date (yyyy-mm-dd) then two letters
- date (yyyy-mm)
- date (yyyy) then two letters.

Cleaning the dates data

Search and replace could be used to clean some of the data (for example, removing 00000000). Code could edit the data into a consistent yyyy-mm-dd format, which could then be used as dates in the spreadsheet.

Code was written to loop down the data, copying each item into a variable that was then edited into the yyyy-mm-dd form, according to the type of form it started with.

Pseudocode 1

```
Select top cell of the dates column
Start loop
Copy the cell into a variable, CellContent
If left 8 characters of CellContent = 00000000
  CellContent = mid(CellContent, 9,10)
If length of CellContent =12
CellContent = left(CellContent, 10)
```

```
If length of CellContent =7
  CellContent = CellContent + "-01"
If length of CellContent =6
  CellContent = left(CellContent,4) + "-01-01"
Set active cell to CellContent
Move down a cell
Loop if active cell not empty
```

Pseudocode 2

Loops

Loops are used to repeat code that uses patterns in the data for processing, such as deleting parts of data items that are not wanted.

Links: For more on loops, see page 19.

Now try this

1 Create a spreadsheet to generate the first 20 numbers in the Fibonacci sequence.

2 Produce an algorithm that calculates the *n*th term of the Fibonacci sequence.

Each term in the Fibonacci sequence is the sum of the previous two terms: 1, 1, 2, 3, 5, 8, 13.

Pattern generalisation and abstraction

After pattern recognition, the next step is to generalise and abstract these patterns to identify all the information necessary to solve a problem. To help you do this, you need to revise variables, constants, key and repeated processes, inputs and outputs.

Representing a problem as code

Identifying information that is necessary to solve an identified problem is an essential part of the programming life cycle. Parts of a problem or system can be represented in code as variables, constants, key processes, repeated processes, inputs and outputs.

	Definition
Variables	Values in a problem or system that may change Usually input by the user or may result from calculation
Constants	Values in a problem or system that remain fixed while the code runs
Key processes	Processes that are essential to understanding of a problem or how a system works
Repeated processes	Processes that occur multiple times within a problem
Inputs	Values read or entered into the system
Outputs	Information presented to the user

Case study — Workout app

You recently saw a television programme which suggested that every opportunity to exercise should be taken as 'every little helps!'

As you spend a lot of time using a computer, you think that an app to help encourage a work out whilst using a computer might be useful.

The mouse could be moved to the corners of the screen and clicked, exercising the lower arm, wrists and fingers. These actions could be repeated with the other side of the body.

Even the toes and feet could be exercised by placing the keyboard on the floor and alternatively tapping the space bar and numeric keypad to ensure the foot has some movement.

Key and repeated processes

```
Start button mouse click
  Initialise TapCount, StartTime to 0
  Initialise Target, to random between 1–4
  Set image(Target) to active
Target image mouse click
  IF Target matches image
    Increment TapCount
    Play success sound
  ELSE
    Play fail sound
  Call Update statistics
Key press event
  IF space or number pressed
    Increment TapCount
    Play success sound
  ELSE
    Play fail sound
  Call Update statistics
Update statistics subroutine
OUTPUT TapCount
  FOR Countdown = 5 TO 1 STEP –1
    Display Countdown
```

Mouse inputs are to click onto one of 4 target images placed at the four corners of the form. Clicking on the correct target image will increment (add 1) to the variable, TapCount.

A start button can start the workout by clearing the variables, TapCount, StartTime, to zero.

The screen will show the target images and workout statistics. Target images can have three variants for active, inactive and correct click. Workout statistics can show time taken, number of correct taps/clicks, number of incorrect taps/clicks, average speed and accuracy percentage.

Keyboard inputs are to tap the spacebar or one of the number keys. Tapping any of these will increment (add 1) to the variable, TapCount.

Speakers can make a sound each time a correct click or key press is made or a different sound if a wrong click or key press is made.

Now try this

Write down two key processes and two repeated processes for a website shopping cart.

Think about the actions that take place in the website shopping cart.

Representing the new system

The last element of pattern generalisation and abstraction is to represent the new system using variables, constants, key processes, repeated processes, inputs and outputs. Filtering and ignoring any information not needed to solve the problem will enable you to focus on the actual problem.

Links For more information on variables, constants, key processes, repeated processes, inputs and outputs, see page 6.

Filtering information

Before writing a program, think carefully about what is actually needed to help solve the problems you are asked to code.

In a database, for example, it is very easy to add fields to a table so there is a place for every possible aspect of the data subject. A better approach is to look at the information that is required from the system, which can then be matched to the data needed to populate the reports and screens outputting from the system.

Case study Health club members list

A system is being written to handle the members list for a health club. It will hold all the information needed for the club's day-to-day operations.

The system should be quick and easy to use as well as using validation techniques to reduce errors typed into the system.

Members could be issued cards which allow scanning into the system by barcode, swiping or NFC (contactless near-field communication).

Reports can be used to extract data from the system onto paper and data can be exported for use in mail merges.

Printed outputs

Reports from the health club members database could include:

- schedule showing the activities booked for that day, week or month
- members list summary
- member details with all the information about an individual member
- members activity log detailing the activities undertaken over a period of time
- members payments due statement with what is currently owed to the club
- booking receipt to confirm an activity has been reserved.

Tables

The database could include the following tables:

- members
- activities
- bookings
- payments.

Forms

These forms will allow easy navigation of the database to make it more user friendly:

- main menu to click buttons navigating to other parts of the system
- members to add, edit or delete a member
- bookings to add, edit or delete a booking
- payments to record a payment
- reports to choose a report for printing.

Now try this

Create a data dictionary identifying the fields needed for the tables in a database to keep track of a health club members list.

The tables are listed in bullets on this page. What fields would each table need?

Algorithm design

The final stage in computational thinking is to design the algorithm using a step-by-step strategy to solve the problem. This will enable you to clearly understand how the program will work.

Designing an algorithm

1. Define the overall purpose of the program.
2. Divide into the processes needed.
3. Plan the steps needed for each process.
4. Check the algorithm against the original need to confirm it will be fit for purpose.

Algorithm design

There are often many algorithms in a program which can be at different levels of detail. Designing a program can start with an overall algorithm to summarise how the system works, with other algorithms providing detail needed to design smaller sections of code.

Links For more information on standard algorithms, see pages 26–29.

Case study

Stock control system

The owner of a second-hand furniture shop is considering writing a stock control program because she quite enjoys coding and wants to have control over how the app looks and behaves.

The app is to run on a PC in the shop as a restricted version so customers can find out if there is anything in the back stock room that interests them.

New item pseudocode

This algorithm enters a new item of stock, checks all the data present and then saves to a stock data file.

```
If any field has not been completed
  Display message
  Show * next to every field not completed
  Place cursor in first field not completed
When all fields completed
  Generate stock number
  Copy fields to Stock file
  Save Stock file
Clear fields on the New item form
```

Stock search pseudocode

This algorithm searches for an item in a stock data file and shows the results on-screen.

```
Click on search button
  If search textbox is empty and furniture checked
    Loop around stock array
      Display all furniture
  If search not empty and furniture checked
    Loop around stock array
      Display all furniture matching textbox
  If search empty and other checked
    Loop around stock array
      Display all non-furniture items
  If search not empty and other checked
    Loop around stock array
      Display other items matching textbox
Place cursor into search textbox
```

Sale pseudocode

This algorithm allows the user to record details of the sale of an item and then save to the data file.

```
Select item sold
Select customer
If customer not known
  Open New customer form
  Enter new customer
  Click on Confirm button
  Copy fields to Customer array
  Save Customer file
  Close form
Show customer information
Click confirm sale
Save Stock, Customer and Sales files
Print receipt
Clear fields on the Sales form
```

Now try this

A car electronic cruise control keeps the vehicle at a constant speed by using the accelerator, gear change and brake, until the driver cancels cruise control by using the brake.

Write pseudocode to design a cruise control system that also links into the sat-nav to prevent the vehicle from exceeding speed limits.

What inputs and outputs are needed for the control device?

Structured English (pseudocode)

There are two main methods you can use to plan program algorithms – pseudocode (structured English) and flow charts. On this page, you will revise commonly used pseudocode terms and how to apply them. Pseudocode can be converted to a programming language to implement:

```
REPEAT UNTIL the end of file
  READ into Sectors(Y)
  Increment Y
Set LastSector to Y – 1
Close the data file
```

→

```
Do
  Input(1, Sectors(Y))
  Y = Y + 1
Loop Until EOF(1)
Lastsector = Y – 1
FileClose(1)
```

Representing operations

- **BEGIN…END** can be used for any code which you want to keep separate or simply to show where your algorithm starts and finishes.
- **INPUT/OUTPUT** are for any part of the algorithm that allows data in or out such as typing into a textbox or displaying a result.
- **PRINT** is used when a hard copy is produced.
- **READ/WRITE** are for when data are read into the algorithm from a file or written out to a file.

Representing decisions

- **IF…THEN…ELSE…ELSEIF (ELIF)** are used for branches in the algorithm.
- Simple branches use **IF…THEN** to define a test condition and action for condition is met which are usually indented or you could use BEGIN…END for them.
- **ELSE** is used when actions are required when an IF…THEN condition is not met.
- **ELSEIF (or ELIF in some programming languages)** is for actions to be carried out if the previous IF…THEN condition is not met and a further test needs to be made.
- **WHEN** is used to represent select case structures with several branches possible based upon the contents of a variable.

Representing repetition

Each of these are written as a single pseudocode line to define the loop followed by the repeated code indented in the code.

- **FOR** is the unconditional FOR…NEXT loop with the pseudocode line showing how many times the loop iterates.
- **REPEAT UNTIL** is a conditional loop with the pseudocode line defining what ends the loop.
- **WHILE/WHILE NOT** are conditional loops with code defining what allows the iteration.

Dos and don'ts

👍 Use program command words to identify branch and loop structures.

👍 Use indents to show what's included in a structure.

👍 Summarise sections of code.

👎 Don't write actual code which is ready to run.

👎 Don't produce pseudocode in your program editor.

👎 Don't include too much detail about how code will do an action such as swapping items.

Now try this

Produce pseudocode for spreadsheet code to copy rows in a worksheet to one of three other worksheets based upon contents of first cell in each row. A fourth worksheet is used for copies of rows where first cell does not match. This needs to be able to handle any number of rows, starting in a cell named 'FirstCell'.

Read the requirement carefully then consider how you would explain the algorithm in simple words.

Interpreting pseudocode

Pseudocode is used to plan program algorithms. It enables the programmer to visualise how a program will work and to see improvements to the logical structures and processes after reading it. On this page, you will revise how to interpret and develop pseudocode.

Case study

Zilch

Zilch is a game played with six dice where the players each take turns throwing the dice to earn points. A target score is set.

The game is won by the player who goes over the target score after the same number of turns as the other players.

When a player has a turn they keep on throwing until either they 'stick' to keep their points or throw a non-scoring combination of dice – 'Zilch' – when their points for the turn are zero.

Zilch dice points rules

There are six dice with several possible points schemes in use. We shall use the scoring below:

1, 2, 3, 4, 5, 6	3000 pts
Three pairs	1500 pts
Three the same	dice number × 100 pts
Dice showing 5	50 pts
Dice showing 1	100 pts

Interpreting and developing code

The process calculating the outcomes of each stage of the game will produce points earned and dice left for the next throw. This will be large and complex, so needs to be broken down into sub-processes, making them easier to focus on and write.

How would you do this if playing with real dice? The first process is to find out if the highest score is thrown, then next highest and so on. Each of these algorithms will be a section of pseudocode.

Preparation for identifying the highest score can take place inside this loop by counting how many of each number has been thrown in the Totals() array. This loop can also show the dice number on the screen.

The structure of this pseudocode can be evaluated against the requirement to identify a throw of 1, 2, 3, 4, 5, 6 using dry runs.

The code here is reasonably effective. Less effective code could use another FOR loop to count how many of each number was thrown. Less effective code might test for a number being thrown more than once, rather than 0.

The highest score is 1, 2, 3, 4, 5, 6 with the method shown here using an array, Totals(), to find out if each number has been used once. Before the check, each item in Totals() is set to 0 using a FOR loop.

A FOR loop throws the dice using the variable, Throw, to hold the number for each dice.

```
Set Score to 0
FOR X = 1 TO 6
  Set Totals(X) to 0
FOR X = 1 TO 6
  Set Throw to random number between 1–6
  Set Dice(X) to Throw
  Increment Totals(Throw)
  Show Dice(X) on the form with its number
Set Winner to True
FOR X = 1 TO 6
  IF Totals(X) = 0 THEN set Winner to False
IF Winner
  Add 3000 to Score
  Show Score on the form
  END subroutine
```

The variable, Winner, is set to true then a FOR loop iterates through the Totals() array, changing Winner to false if any of the numbers were not thrown.

Now try this

Produce a description of how this pseudocode calculates a score in Zilch:

```
FOR X = 1 TO 6
  IF Dice(X) = 1
    Add 100 to Score
    Decrement DiceLeft
  IF Dice(X) = 5
    Add 50 to Score
    Decrement DiceLeft
Show Score on the form
```

The sequence is important. Start with the first line of the pseudocode and interpret the meaning. Remember, indents show how much code is inside a structure such as a FOR loop.

Flow charts

Flow charts provide a pictorial complement to pseudocode, helping you to plan algorithms. British Computer Society (BCS) symbols are commonly used in flow charts.

Flow chart shape	Description
▭	**Process** used for anything in code that cannot be represented by any of the other symbols.
◇	**Decision** shows where there is a choice of two paths with the condition that needs to be met written inside the symbol.
▱	**Input/output** shows every place where data or events come into or leave the algorithm.
──▶	**Connectors** are used to reduce the need to draw lines across the flow chart.
⬭	**Start/end** symbols show the entry and exit points in the algorithm.

Alarm flow chart

This flow chart illustrates how an alarm works on a mobile phone.

The flow chart has to begin and finish with start/end symbols. The rest of it needs to show the routes that are possible in the code.

The alarm is set with an input from the user.

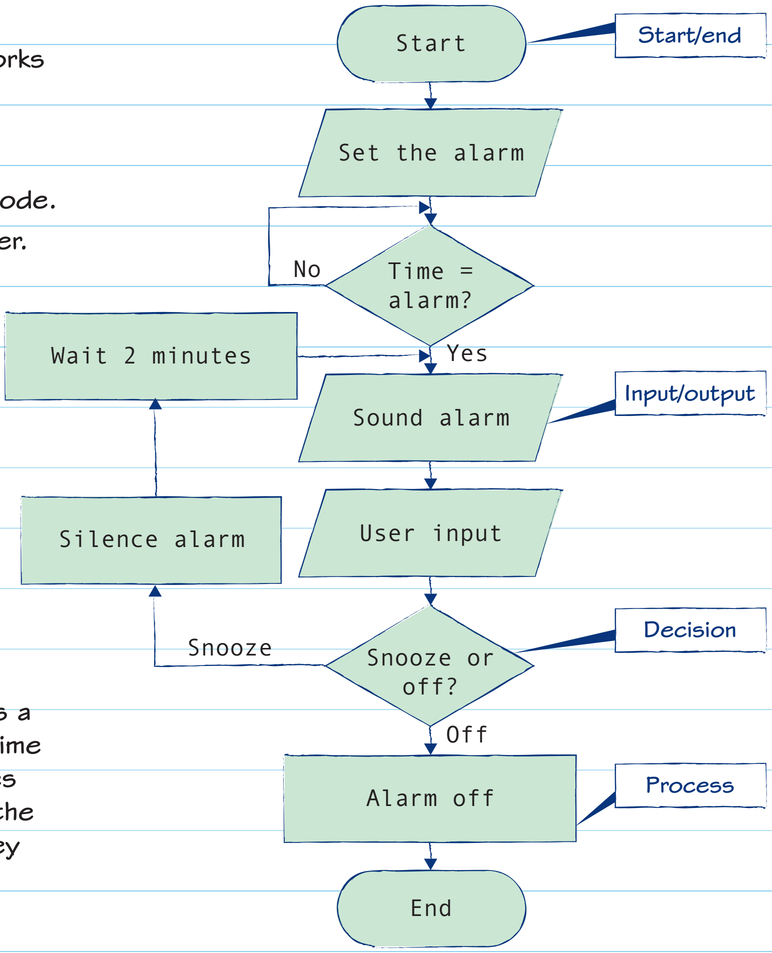
A decision is used to show how the system regularly checks the actual time to the alarm. When they match, the yes branch is taken so the alarm is sounded.

The user can now input into the system to either turn the alarm off or snooze.

If snooze, the app enters a process to wait a short time before the alarm sounds again.

If the user turns the alarm off, a process disables the alarm and the app ends.

Waiting is shown in this flow chart both as a process (wait 2 minutes) and as a loop (time = alarm?). These are both valid techniques for showing the delay with the author of the flow chart able to choose the method they prefer to show this delay.



Now try this

An independent jeweller is setting up a website as a retail outlet. It would like to offer a loyalty discount for customers who purchased within the last year as well as giving a reduction of 10% on the sale for more than one item and a choice of postage rates of next day or economy. The jeweller is VAT registered, so VAT needs to be added to the sale after any discounts and postage.

Produce a flow chart to illustrate this algorithm.

There should be a symbol in the flow chart for each line of your pseudocode.

Make sure you use the correct symbols.

Handling data within a program

Programming paradigms can be used to build computer code to handle data within a program. On this page, you will revise common data-handling techniques and structures provided within programming languages to process data.

Reading a data file

```
FileOpen(1, EmployersFile, OpenMode.Input)
Y = 0
Do
    Y = Y + 1
    For X = 0 To 9
        Input(1, Employers(X, Y))
        TempEmployers(X, Y) = UCase(TempEmployers(X, Y))
    Next X
Loop Until (EOF(1))
FileClose(1)
LastEmployer = Y
```

Code to read a data file usually needs an indefinite loop to repeat reading each line from the disk until it reaches the end of file.

A variable, EmployersFile, has been set to name the data file before the code here opens it for input. A variable, Y, is set to zero before the Do loop to keep track of each line read in from the EmployersFile and which item in the array, Employers(), is used to store the input data.

A nested loop uses the variable, X, to keep track of the data for each employee and which item in the array is used to store the input data. This loop uses a UCase() function to force each data item to upper case.

After EmployersFile is closed variable, Y, is used to set variable, LastEmployer, so the program knows the subscript number of the last employee record in the array, Employers().

Writing a data file

```
FileOpen(2, NameOfFile, OpenMode.Output)
LineOfPrint = "Allocations(6-200)"
PrintLine(2, LineOfPrint)
LineOfPrint = " ,"
Xvalue = 6
For X = 0 To Xvalue
    LineOfPrint = LineOfPrint & X & ","
Next X
PrintLine(2, LineOfPrint)
For Y = 0 To LastAllocation
    LineOfPrint = Y & ","
    For X = 0 To Xvalue
        LineOfPrint = LineOfPrint & Allocations(X, Y) & ","
    Next
    PrintLine(2, LineOfPrint)
Next Y
FileClose(2)
```

Code to write a data file can use a definite loop to repeat writing each line to the disk as the code knows how many records are in the array.

A variable, NameOfFile, is set to name the data file before the code here opens it for output.

This code uses a variable, LineOfPrint, which is built up into each line to be written to the disk with a comma between each data item.

Two FOR…NEXT loops are used to write to disk.

The first loop produces column headings for when the data file is opened into Excel with the name of the array, Allocations(6-200), in the first column and numbers from the loop variable, X, for which array item is in the other columns.

The second loop writes the data to disk. After this loop, the data file is closed.

Now try this

Produce a program to read in a data file, make changes to the data and write it back to the disk. The data file can be created using Excel and saved as CSV (comma separated variables). Use a calculation in the spreadsheet to produce a reference number in the first column. The program you write can add 'REF' to these numbers before writing them back to disk. You can open the new data file in Excel to confirm the reference numbers have been set by your code.

Include a comma between each item when writing to disk.

Constants and variables

On this page, you will revise the data types you can use to define constants and variables.

Constants and variables

Constants and variables are very similar, with both naming a place in memory where data can be held. A constant does not change when code runs, the contents of a variable is usually changed by calculations or user input.

Arrays

An array is a variable which can contain many different values, each of these identified by the subscript (number) inside brackets at the end of the array name. A lot of code uses arrays to hold data records for the program.

Links For more on arrays, see page 23.

Text variables and constants

These can be combined (concatenated), searched or part of the string can be selected and used, such as the first three characters.

- **Alphanumeric strings** are used to hold combinations of letters from the alphabet and numbers, such as AB3076.
- A **character** is a single letter or number such as A, B, 6. A string is one or more characters. In a program, these variables can be used for any combination of words, spaces or numbers such as an address or a name.
- **Strings** can hold alphanumeric characters as well as other characters including escape codes such as CrLf (carriage return/line feed).

Numeric variables and constants

- **Floating point (real)** variables are used to contain numbers which may have a fractional part. These variables can hold a range of values which depends upon the type used. A single has a range of -3.4028235E+38 to 3.4028235E+38, using 4 bytes of memory. A double is ±1.79769313486231570E+308 and uses 8 bytes of memory.
- **Integer** variables and constants are used to contain whole numbers. These variables can hold a range of values, which depends upon the type of integer used. A short integer has a range of –32768 to 32767, using 2 bytes of memory. A long integer has a range of –2147483648 to 2147483647 and uses 4 bytes of memory.

Date/time

Declaring a variable as a date can save the programmer a lot of effort as there are functions available to calculate dates, such as DateAdd, and these variables can show the date in whatever format is required for the app.

A date variable can also be used for time. The actual content of date variable is a number with the whole part the date (number of days since 1 January 1900) and the fractional part the time, e.g. 6 am is .25, midday is .5 and so on.

Boolean

A Boolean variable has only two possible values – true or false.

It is a good data type for use in a conditional statement, such as IF Found THEN, where Found is the Boolean variable.

Boolean variables can also be used to represent **objects**, such as option buttons in code.

Now try this

Write code which uses six different data types to hold information entered by the user. Process each of the entries using a method appropriate to the data type, for example, Boolean, to make a decision, using appropriate output to show the results of your processing.

Decide upon the data types with what you'll do with them before you start coding.

Managing variables

You need to revise the difference between local and global variables and when to use them, as well as the use of naming conventions to give meaningful names to objects in your code.

Managing variables

Managing variables helps to get the best performance from an app in terms of reliability, although there can be a very small reduction in speed due to the creations and releasings of **local variables**.

The minor speed hit is more than compensated for by the extra reliability due to much more control over where variables exists in the code.

Good program design is very clear on where a variable is used or changed and so if another part of the code tries to use such a variable an error is generated to alert the programmer.

Parameters

A parameter is an argument in brackets after the name of a subroutine or function code passing a value into this code.

A parameter is a local variable to the subroutine or function unless it is defined by reference, in which case it is the same variable as was used by the code calling the subroutine or function.

The default for a parameter is by value, which means that what's inside a variable is passed into the subroutine or function and so does not affect this variable elsewhere.

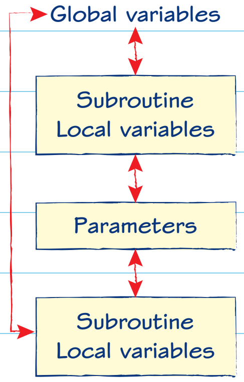
Global and local variables

The scope of a variable defines which parts of the code can see or use it.

A global variable exists everywhere in the code and only ceases when the app closes.

A local variable exists inside a subroutine or function subprogram whilst that code is running then ceases when the subprogram ends.

If a subroutine calls another subroutine, any local variables in the calling subprogram would not be seen by the called subroutine unless passed in as a parameter.



Global can be used anywhere; local and parameters are private to the subroutine.

Naming conventions

The programmer has a lot of choice of the names of variables used in code, although there are a number of reserved words, such as open, which cannot be used as variables because they are part of the programming language.

A good variable name helps to document the code because it describes what the variable contains. Capitalisation can be used to help see words used in a variable name, e.g. CarColour.

Bad names are anything meaningless or which mislead about the use of the variable.

Variable names – dos and don'ts

👍 OKtoGo
👍 VATdue
👍 NameOfFile

👎 oktogo (poor capitalisation)
👎 Var1 (meaningless)
👎 Axx (meaningless)

Now try this

Create a poster showing how the scope of variables affects where they can be used in a program.

You can use circles to show each scope.

Arithmetic operations

Programming paradigms can be used to implement arithmetic operations, which include mathematical functions such as + and *, relational operators such as = and <, Boolean operators such as NOT, as well as date and time.

Mathematical operators

Mathematical operators are plus (+), minus (-), divide (/ or DIV) and multiply (*). Remember that the computer will always use BIDMAS (brackets, indices, divide, multiply, add, subtract) for the order in which a calculation is worked out.

The following calculation needs to set Pay by working out the HourRate + Supplement before multiplying by Hours, but gives a wrong result:

Pay = HourRate + Supplement * Hours

This is because brackets should be used to calculate the addition before multiplying:

Pay = (HourRate + Supplement) * Hours

Relational operators

Relational operators are frequently used in code, especially for conditions which control a branch into a choice of coding routes.

In these examples:

Pay has been set to 3.9

Cost has been set to 4

Equals	Pay = Cost	False
Less than	Pay < Cost	True
More than	Pay > Cost	False
Not equal to	Pay <> Cost	True
Less than or equal to	Pay <= Cost	True
More than or equal to	Pay >= Cost	False

Modulo

When a number is divided into another, the remainder (rem) is called the modulo or modulus, (MOD), which is often useful in calculations carried out by code needing the number remaining after division.

Modulo operator examples

10 % 3 returns 1 in Python code.

7 mod 4 returns 3 in VB.NET code.

= MOD(4, 3) returns 1 in an Excel cell.

23 rem 4 returns 3 in Prolog code.

Boolean operators

Boolean operators can be complex calculations but always end with a result of True or False.

In these examples:

Car has been set to True

Diesel has been set to False

Opposite (NOT)	NOT Diesel	True
All of them (AND)	Car AND Diesel	False
Any of them (OR)	Car OR Diesel	True

Date/time operators

Usually a date in program code is held internally as a whole number (the day count from 1/1/1900) and time as the fractional part of a number, e.g. .75 is 6pm, so 6pm on 17 October 2017 is held as 43025.75, so simple arithmetic can often be used. Excel® has a known bug which calculates 1900 as a leap year.

Other programming languages make it very difficult for the programmer to reach the underlying numbers. It is much easier and practical to use the date and time functions provided.

Now try this

Create an Excel spreadsheet to show expressions illustrating mathematical, relational, Boolean and date/time operators. Copy and paste another version of each of your examples so a printed copy shows both the calculation workings and the result.

Use a single quote (') at the start of each of the copied examples so the workings print.

Arithmetic functions

Arithmetic functions enable you to code arithmetical operations – random, range, round, truncation – in a program. The Excel spreadsheet is used to demonstrate these arithmetic functions on this page.

Arithmetic functions

random()	Generates a random number.
range()	Creates an array of elements using the range of values in the brackets.
round()	Rounds a number up or down to the nearest whole number.
truncation()	Rounds a number down to the number of decimal places in the brackets.

Using the range() function

One argument will create a range of integer numbers from 0 to one before the argument, e.g. range(5) creates 0, 1, 2, 3, 4.

Two arguments create a range of integer numbers from first argument to one before the last, e.g. range(2,6) creates 2, 3, 4, 5.

Three arguments create a range of integer numbers with the last argument defining how much each item increments, e.g. range(1,12,3) creates 1, 4, 7, 10.

In code, **10 in range(1,4)** will return false.

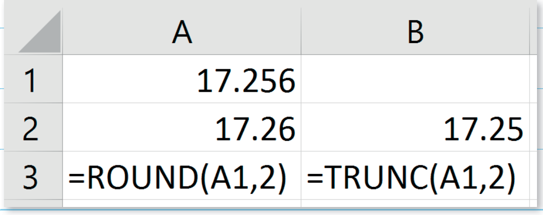
ROUND() and TRUNCATE() functions

These are both used to specify the number of decimal places showing for a number.

The round() function will adjust a number to fit with the least significant digit rounded up or down.

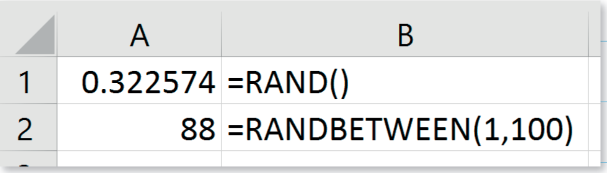
The truncate() function simply removes any digits that do not fit.

Round and Trunc



	A	B
1	17.256	
2	17.26	17.25
3	=ROUND(A1,2)	=TRUNC(A1,2)

RAND and RANDBETWEEN



	A	B
1	0.322574	=RAND()
2	88	=RANDBETWEEN(1,100)

Using the random() function

The random() function will usually generate a random number larger than 0 and less than 1.

Some programming languages accept an argument inside the brackets to define the largest random number that can be produced.

Excel offers the RANDBETWEEN() function which accepts two arguments to define the scope of random numbers that are generated.

Now try this

Create an Excel spreadsheet to generate test data where the A column contains random numbers between 3 and 50, the B column a random date up to a year before today (assuming 365 days in the year) and the C column a random letter between A and Z.

Use the NOW() function as part of your calculation for the date.

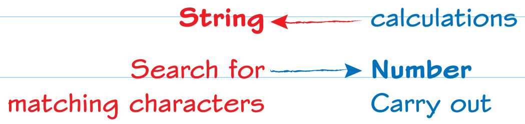
Use the CHAR() function as part of your calculation for the letter.

String handling and general functions

String handling and other built-in general functions convert between different types of number and strings and perform general operations such as dealing with data files and printing.

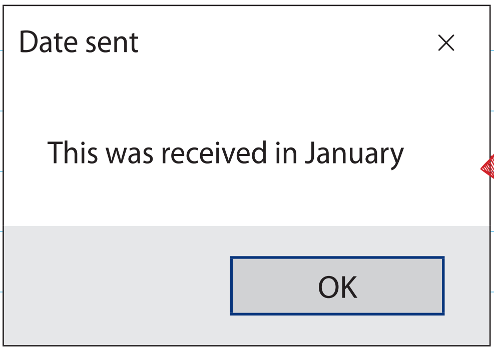
String conversions

Converting to numeric allows code to use numbers stored as strings in calculations. CInt() converts to integer, CDbl() to double data type for floating point (float) numbers. CStr() can convert a number into string if there is need for searching or extracting part of the number.



Manipulating strings

- **Concatenation** is joining together two or more strings. The '+' character is used for concatenantion by C, Java, Python, VB.NET among others whilst the '&' character is unique to Visual Basic. With numbers, '+' performs addition, but it concatenates strings.
- **Length** is how many characters are in a string. Many languages have a len() function to return this number.
- **Position** is where a character or group of characters are in a string. VB.NET uses the IndexOf method, Python the find method.


Date sent

This was received in January

Making it work

The VB.NET code below uses concatenation to join text with month name extracted from date variable, DoR, after converting into a string variable, strDoR. DoR is formatted to "09 January 2017" (long date) before conversion, so number of characters in the month can be calculated as length of date string minus 8 (2 digits day, 4 year, 2 spaces around month).

Month position is calculated from 1 more than index of first space plus 2 (index starts at 0).

When this code runs, MsgText will contain "This was received in January"

```
DoR = "9/1/2017"
strDoR = CStr(Format(DoR, "long date"))
MonthLen = Len(strDoR) – 8
DoRpos = strDoR.IndexOf(" ") + 2
DoRmonth = Mid(strDoR, DoRpos, MonthLen)
MsgText = "This was received in "
MsgText = MsgText & DoRmonth
```

Input and open functions

Input lets users enter into a variable. This Python code shows a prompt of 'How many?', storing the response in Quantity:

```
Quantity = input("How many?")
```

Open connects code to a data file with an argument defining type of access, e.g. read, write. This Python example shows a file, Data.csv, being opened to read the data:

```
DataFile = open("Data.csv","r" )
```

Range and print

Range is a function to return a range object. This Excel code example shows a calculation, =Rand(), being entered into a range of cells:

```
Range("A2:D12").Formula = "=Rand()"
```

Print sends text to screen or other output such as a data file. This Python code writes the contents of a variable, DataVar to a file, data.txt, opened in write mode as DataFile:

```
DataFile = open("data.txt","w")
print(DataVar, file=DataFile)
```

Now try this

Create code to extract and use part of a date converted to string in a short sentence as described in the Making it work box above.

Use some form of console or message box output to check what is in the variables as you develop this code.

Validating data

Programming paradigms can be used to build effective validation techniques into code, improving the validity of inputs with post-check actions aiding further accuracy.

Validation check techniques

Checking data as it is entered for validity is a basic technique used in many ways to check the **data type** and **range** with any **constraints** before the code attempts to process the entry.

This screening helps to reduce errors and gives the user the opportunity to correct mistyping at the time of entering the data.

Data types and boolean

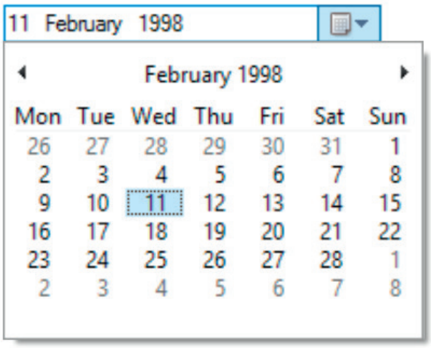
- Checking for the correct data type is basic validation preventing a lot of data entry errors, e.g. by rejecting text when a number is needed.
- Boolean logic can be applied to a data entry where an input could use a choice of validation rules, e.g. a vehicle registration could be in the form of XX99XXX or X999XXX.

Range

Many validation checks ensure inputs are within a range of values, such as age to make sure someone is not too young.

An age can be entered into a textbox with simple validation to ensure a number has been entered within an acceptable range, such as between 18 and 25.

A date of birth is much better data as this would still be useful for years after the data entry as an up-to-date age can be calculated from the current date obtained from the computer clock.



11 February 1998

February 1998

Mon	Tue	Wed	Thu	Fri	Sat	Sun
26	27	28	29	30	31	1
2	3	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	1
2	3	4	5	6	7	8

Constraints

Validation using constraints is essential for data entry such as a reference number with a clear structure. A reference number should be fixed with a set number and combination of letters and digits, e.g. STR00234, which are very straightforward to check. In this example the first three characters would be letters, the next five characters constrained to numbers and the overall number of characters must be eight.

Post-check actions

An app should include post-check actions which provide feedback to the user on why a validation check has failed and what the user needs to do to correct their entry:

- **Enforcement action** usually clears the bad data from the screen.
- **Advisory action** usually keeps the data but also sends a warning message.
- **Verification action** asks the user to confirm their data is correct.

Now try this

What validation can be applied to a data entry requiring a UK postcode?

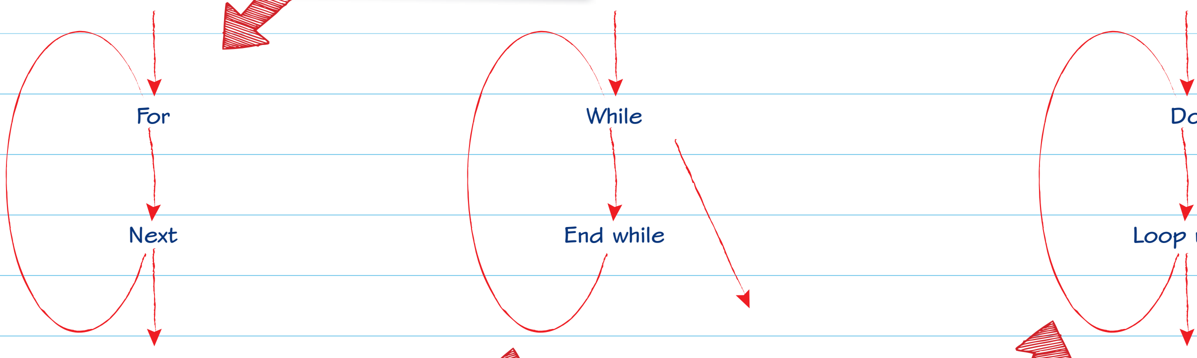
What positions do the letters and numbers occupy in a postcode? Are there any further techniques available?

Loops

Control structures include loops, also known as iterations, which repeat code as many times as needed. This page revises how to improve the effectiveness of code iterations by appropriate use of REPEAT, FOR, WHILE structures and any mechanisms needed to break out of them.

Unconditional loops

The classic unconditional loop is FOR…NEXT where a loop variable is used to keep count of the number of iterations with the NEXT line in this structure used to determine when the loop is complete.



Pre-conditional loops

Conditional loops will continue until an event occurs or a condition is met.

The condition can be at the start, e.g. WHILE, which is known as a pre-conditioned loop, so the code inside the loop will not be run at all if the condition is not met when this structure is executed.

Post-conditional loops

If the condition is at the end, e.g. REPEAT UNTIL or LOOP UNTIL, it is a post-conditional loop so code inside the loop will run at least once, even if the condition is not met as the test is after the body of the loop.

These structures offer the programmer more control over how the loop will work.

Using loops

FOR…NEXT	✓ looping through arrays ✓ generating test data
WHILE REPEAT UNTIL	✓ reading in data files ✓ checking for user attempts, e.g. passwords

Breaking out of a loop

When running a loop, the programming environment creates a structure which needs to end properly or there might be problems if the code runs for a long time.

Programming languages include commands such as BREAK, EXIT FOR or EXIT DO to finish a loop early.

Most code should not need to exit, as a conditional loop should respond to such situations. An unconditional loop requiring an early exit should probably be conditional.

Now try this

Produce a guide on appropriate uses for each loop type. Include examples of how the condition test for a conditional loop can best be used at the start or at the end of the loop.

Think of a situation where iteration code should not be run if a condition is not met.

Branches

Branches allow you to make decisions within an algorithm. On this page, you will revise IF…THEN…ELSE…ELSEIF selections.

Branching with IF

The IF control structure allows codes to divide into separate pathways, selecting between two or more routes through the program. This structure starts with the IF…THEN line of code where a condition is evaluated as true or false. Code immediately after the IF…THEN line is run if the condition is true as far as the next part of this structure, which could be:

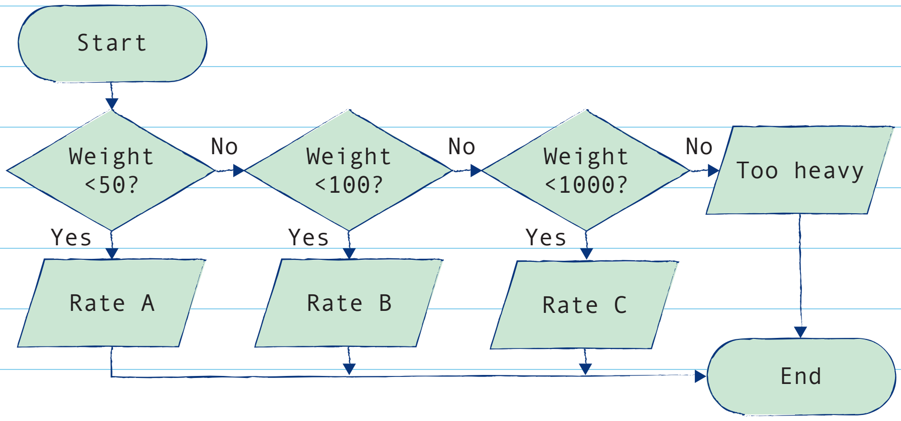
- ELSEIF to set another condition
- ELSE for code if the condition(s) not met
- ENDIF to complete the structure.

Case study: Postage rates

An app could be written to allocate a postage rate according to the weight of a shipment:

Weight	Rate
Below 50 g	A
50 g or more and below 100 g	B
100 g or more and below 1000 g	C
1000 g or more	Too heavy

The app will allow the user to type a weight into a text box, txtWeight, then show the appropriate rate on-screen.



The IF condition (number typed into WEIGHT by the user), shows Rate A if less than 50.

Care needs to be taken with conditions. The conditions here are carefully sequenced with first condition, (<50), so the next condition, (<100), is from 50 up to and not quite 100.

```
IF WEIGHT < 50 THEN
  SET POSTAGE LABEL TO “Rate A”
 ELSEIF WEIGHT < 100 THEN
  SET POSTAGE LABEL TO “Rate B”
 ELSEIF WEIGHT < 1000 THEN
  SET POSTAGE LABEL TO “Rate C”
  ELSE
  SET POSTAGE LABEL TO “Too heavy”
 ENDIF

If txtWeight.Text < 50 Then
 lblPostage.Text = “Rate A”
ElseIf txtWeight.Text < 100 Then
 lblPostage.Text = “Rate B”
ElseIf txtWeight.Text < 1000 Then
 lblPostage.Text = “Rate C”
Else
 lblPostage.Text = “Too heavy”
End If
```

ELSEIF statements respond to other weights with ELSE line running code not met by any other condition showing “Too heavy”.

Now try this

Write a program which accepts (and validates) user input of a whole number between 0 and 48 to represent the points achieved for a test. Your program will use a select case structure to show ‘Fail’ (0–17), ‘Pass’ (18–25), ‘Merit’ (26–41), ‘Distinction’ (42–47) or ‘Distinction*’ (48) according to the input value.

Be very careful to code for the grade boundaries and use test data to ensure they are met.