

# Computational thinking

## 1. Introduction

Computational thinking is *not* about thinking like a computer (computers don't think). In fact, computational thinking does not even need a computer for it to take place. Instead, it is a way to understand a problem and then to logically work out a good solution.



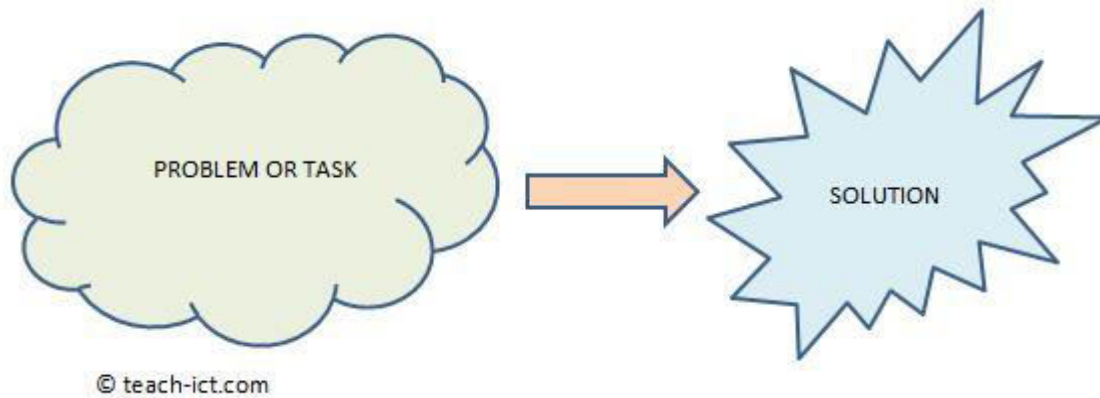
The four elements of computational thinking are:-

- Decomposition
- Pattern Recognition
- Abstraction
- Algorithmic thinking

This section will describe what these mean.

# Computational thinking

## 2. Decomposition (1)



Simple tasks often have simple solutions. For example, if the task is to open a door then the solution is to simply walk up to the door and open it.

But some problems are a bit more complicated - for example, a weekly trip to buy groceries. You can break down large problems like this into several smaller, simpler tasks. This is "**decomposition**".



The weekly shop task breaks down into five smaller tasks. These are:-

- Budget - work out how much money you are going to spend
- Itemise - make a list of items and make sure that you can buy them with your budget
- Identify shop - decide which shops you are going to visit.
- Travel - work out how to get to the shops
- Shopping - in your chosen shops, collect the items, bag them up and pay for them. Then return home, unpack them, and put them in their correct place.

This approach can be used on *any* task. The skill is to be able to break up a problem into more manageable tasks and this takes practice.

## Computational thinking

### 3. Decomposition continued.

Let's decompose a different problem:

*Task: Create a new game (not necessarily a computer game).*



Your challenge is to create a new game.

This is an open-ended problem. It is not well-defined. The way to approach it is to break it down (decompose) it into a set of questions that need answering.

1. Type - what type of game is it going to be? e.g. card game, board game, computer game.
2. Theme - what style of game is it going to be e.g. quiz, strategy, simulation, pure chance etc
3. Audience - who is the game aimed at? e.g. small children, teens, adults, multi-player etc.
4. Shape - does it have challenges? does it have rewards?
5. End - how does the game end? how do you win or lose?
6. Name - what is going to be the name of the game?

Until you have decided what type of game it is going to be, you cannot move forward with the task. You *have* to break it down into more detail.

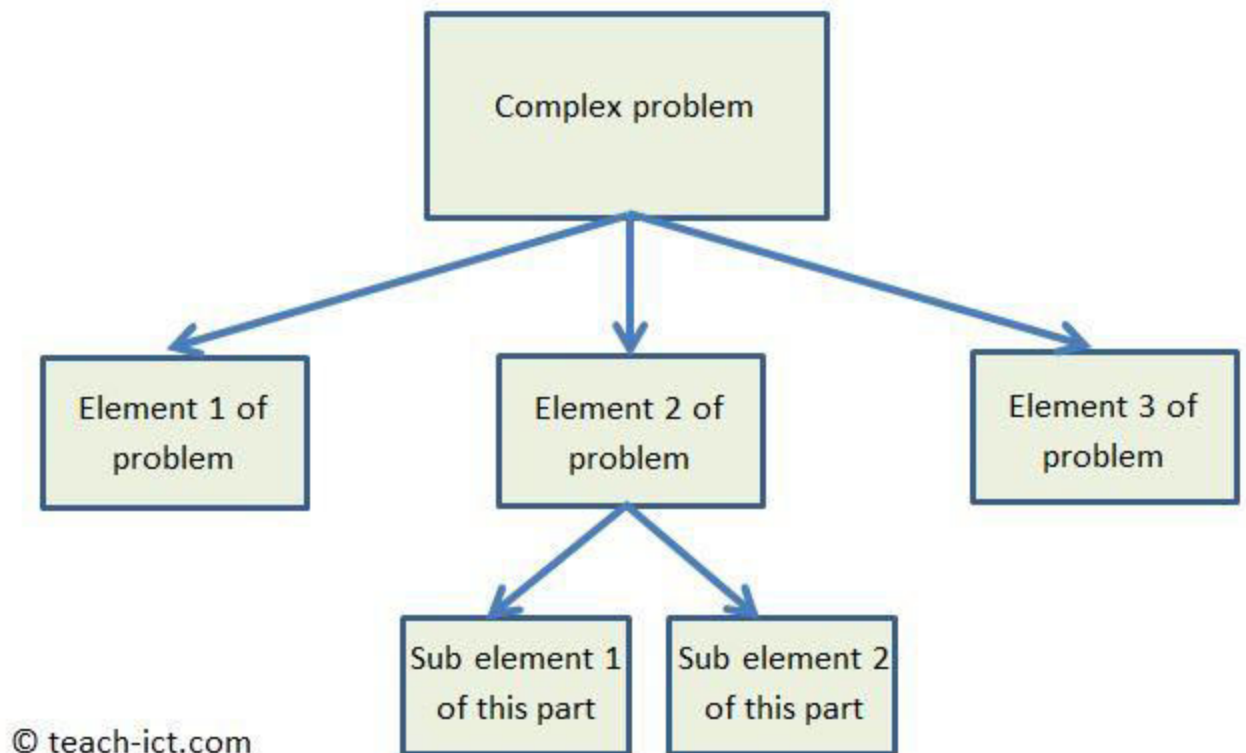
This is what decomposition is about: defining the problem in more detail and making it easier to solve.

## Computational thinking

## 4. Structure Diagrams - Top-Down design

It can be easier to see how a problem is decomposed by making a diagram. 'Top-Down' diagrams are a popular way to do this.

Top-down diagrams are a type of tree diagram. Put the whole problem in a box at the top. Then put boxes beneath it, describing how that problem can be broken down into smaller elements.



Each sub-problem can then split into smaller sub elements if needed.

The diagram is complete when all of the problems are broken down in enough detail to find an easy solution for them.

## Computational thinking

### 5. Pattern recognition

Many problems are solved by doing some of the steps in similar or repetitive ways. This similarity of steps is called a '**pattern**'.

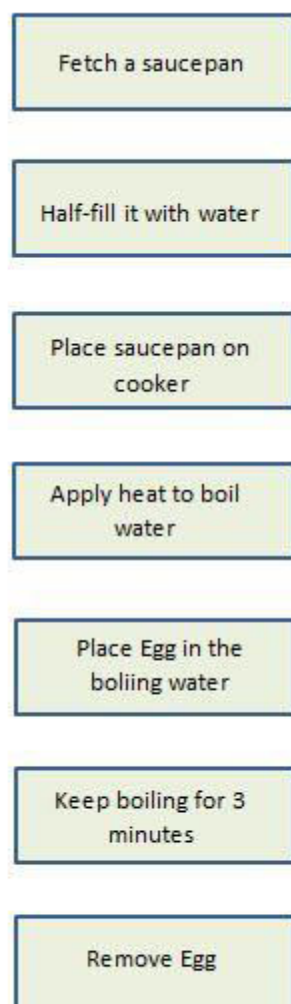
A **pattern** is where there is some repetitive parts to the overall problem. This means more than one part of a problem can be dealt with by simply altering the information going into the pattern.

In computer programming patterns include loops, subroutines, code libraries and creating classes.

Spotting a pattern when solving a problem is one of the skills for computational thinking.

We can use a simple example such as how to boil an egg.

The steps for doing this could be :-



© teach-ict.com

These steps will allow you to boil an egg successfully. Well done - you are on your way to becoming a cook!

Now for our next task we could cook some pasta.

The steps for cooking the pasta are shown next to those for boiling an egg:

BOIL: EGG	COOK: PASTA
Fetch a saucepan	Fetch a saucepan
Half-fill it with water	Half-fill it with water
Place saucepan on cooker	Place saucepan on cooker
Apply heat to boil water	Apply heat to boil water
Place <b>Egg</b> in the boiling water	Place <b>Pasta</b> in the boiling water
Keep boiling for <b>3 minutes</b>	Keep boiling for <b>10 minutes</b>
Remove <b>Egg</b>	Remove <b>Pasta</b>

© teach-ict.com

Do you notice how similar they are? The only differences are shown in red. In the egg task, it involves the egg and a time of 3 minutes and the other involves pasta and a time of 10 minutes. All of the other parts of the problem are a 'pattern'.

We can give this pattern a name - 'BOIL' and it looks like this

**PATTERN: BOIL**

Fetch a saucepan

Half-fill it with water

Place saucepan on  
cooker

Apply heat to boil  
water

Place **ITEM** in the  
boiling water

Keep boiling for **TIME**

Remove **ITEM**

© teach-ict.com

The thing to be boiled has been given a general name of 'ITEM' and the boiling time is given the name 'TIME'.

Now this pattern can be used to boil an egg, pasta or in fact any other food by using the pattern with the relevant extra information. Like this:

BOIL Egg 3 minutes  
BOIL Pasta 10 minutes  
BOIL Carrots 15 minutes  
BOIL Potatoes 20 minutes  
.....

The first bit of information (Egg, Pasta ...) is swapped for ITEM in the pattern and the second bit of information (3 minutes, 10 minutes ...) is swapped for TIME in the pattern.



Being able to create a pattern for solving a more general problem is a very powerful way of saving time and effort for the next similar problem that comes along.

# Computational thinking

## 6. Abstraction

Another powerful aspect of computational thinking is called 'abstraction'.

**Abstraction** means to examine a problem and identify its most essential details and to disregard any non-essential details.

To understand the concept of abstraction have a look at the following two images:



The top image shows a map of London, including the roads, tube stations, parks, river etc. This is brilliant if you are wanting to walk or drive around



London. However, if you want to use the Underground, it isn't much use as it has far too much detail and much of that detail is not needed.

The second diagram shows a map of the London Underground. All of the unnecessary detail has been removed so that you can see just the information that you actually need.

Removing all of the unnecessary detail is what is meant by 'abstraction'

# Computational thinking

## 7. Abstraction continued

Let's look at another example of abstraction

### **Problem**

Imagine that you are working for the Highways Authority and have been asked to produce a new road sign for a 'T' junction (a road junction where the current road ends and the next road is at a right angle).

The sign must follow these rules:

1. It must take no more than 1 second for a driver to look at it and understand its meaning.
2. It must not contain any words.
3. It must use the standard shape and colours for all UK road signs.

### **Determining a solution**

Here is the first solution that our designer came up with\*:



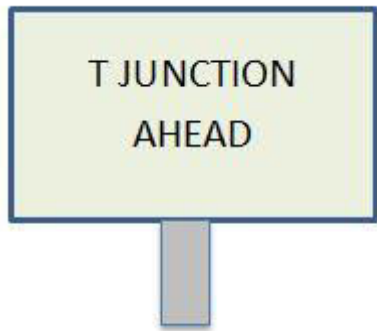
A nice try but this sign isn't going to be any use because 'point 1' said that:  
*'any sign must take no more than 1 second for a driver to look at it and understand its meaning'*

If you were driving on a road, it would take you longer than 1 second to take out the clutter in the image and understand what the sign was trying to tell you.

Also, point 3 said that the sign:  
*'It must use the standard shape and colours for all UK road signs'*.

So our artist's sign clearly isn't going to do the job. Let's have another go.


Here is our next attempt - is it any better?



This one is a bit better because all of the unnecessary detail has been removed (abstraction). However, point 2 in our list of rules said that the sign: *'must not contain any words'*

So we need to go back to the drawing board.

First of all, let's represent a road as a simple thick black line like

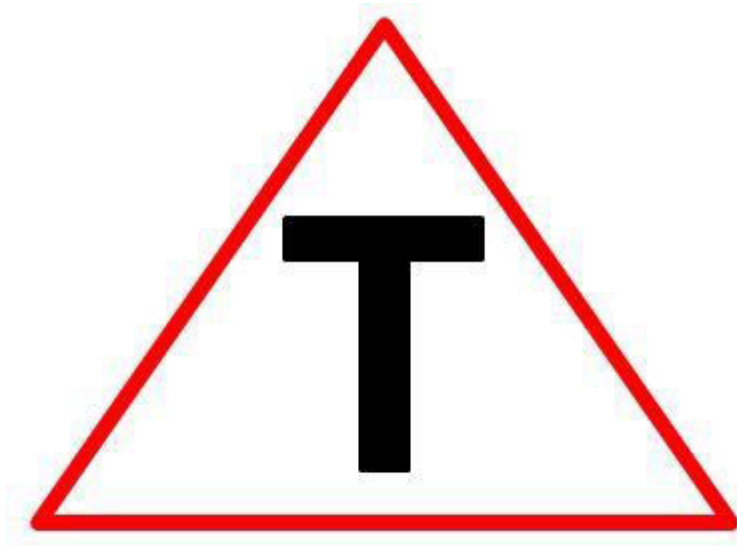
this . This is an 'abstraction' of a road (roads have a substantial width to them, so a *thin* black line would not be right for a sign plus it would be hard to see from a distance).

The second essential feature of a T junction is that it has another road at right angles to it. So a good abstraction of this is:-



As this is for a road sign, we want the idea of going *forward* along the road, so it is better to have the shape be upright rather than on its side.

Here is our final attempt:



Now this one is much better:

It uses a standard UK highway sign shape (the red triangle)

It does not contain any words

And the most important, all of the clutter and unnecessary detail has been removed (abstraction).

### *Conclusion*

From the example above, using abstraction to remove any non-essential features guides us towards a solution.

The same method of abstraction can be used for many other problems including those in computer programming.

\*image courtesy of Wikimedia Commons

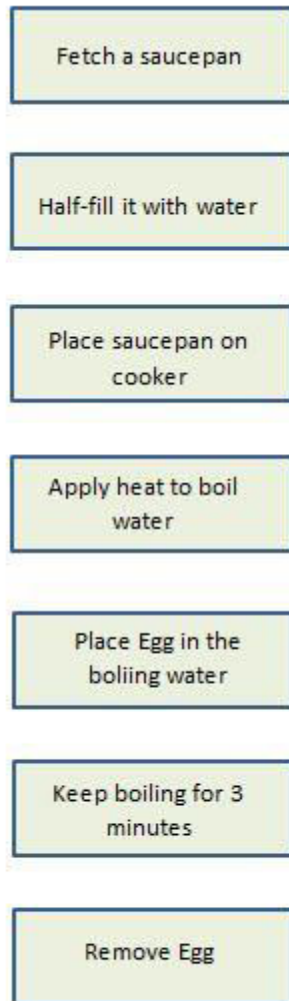
## **Computational thinking**

### **8. Algorithmic thinking**

We have discussed decomposition, pattern recognition and abstraction as methods to make it easier to find a solution to a problem. The final technique lays out the solution in a step-by-step manner. The name for this is 'algorithm'.

**Algorithm** - is a list of step-by-step instructions that, when followed, will solve a problem.

Let's go back to the earlier example of boiling an egg :-



© teach-ict.com

The diagram lays out the steps (algorithm) needed to boil an egg.

Being able to lay out an algorithm when writing a computer program is vital. This is because a computer needs to be told what to do, step by step.

There are two popular methods to set out an algorithm

- Pseudocode - text statements written for each step of the algorithm.
- Flow chart - this uses a set of standard shapes and arrows to make a diagram of the algorithm

Algorithms often include a decision to be made at some point in the task. In order to handle this, a *conditional statement* is used such as an IF statement. The pseudo code below describes this

IF (condition is true) then carry out this step

For example, let's set out an algorithm in pseudocode for making a cup of tea but also considers whether milk needs to be added or not

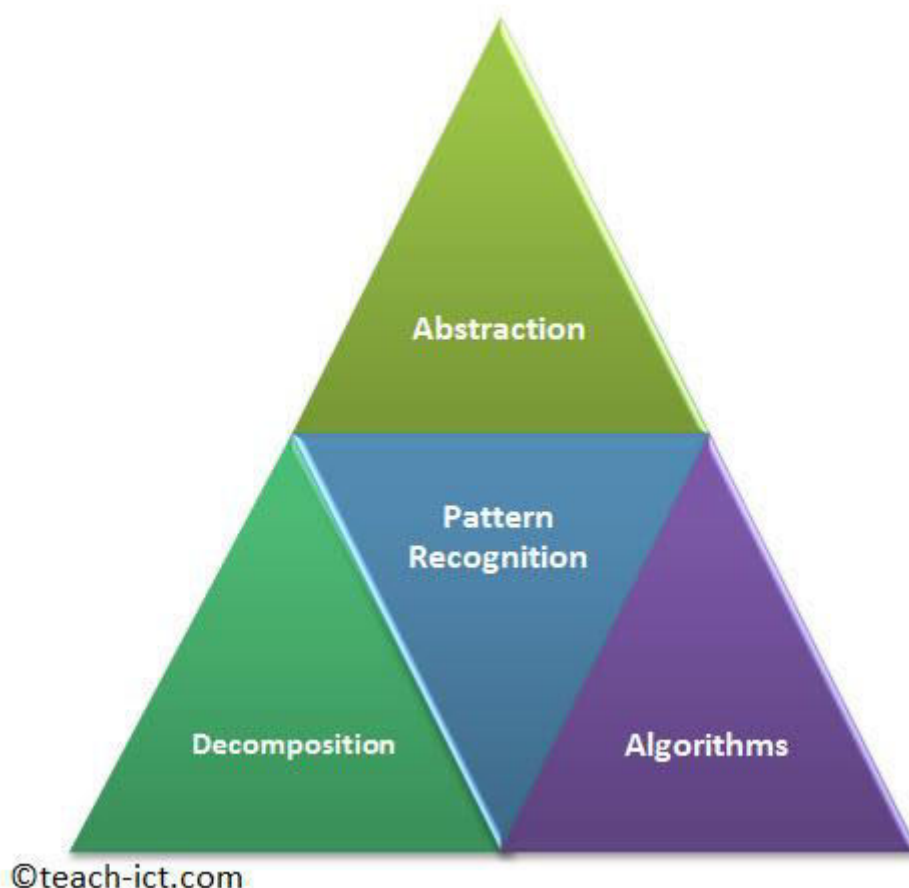
- Put tea bag in the cup
- Pour enough boiling water into the cup to reasonably fill it
- IF milk is needed THEN add milk
- Remove tea bag
- End

Working out the algorithm for solving a problem requires thinking in a clear and logical way.

## Computational thinking

### 9. Summary

The four main methods used for computational thinking are shown in the diagram below:



**Decomposition** - breaking down a problem into simpler parts. A top-down tree diagram can show this graphically.

**Pattern recognition** - being aware of any repetition in solving the task, this pattern can simplify the steps needed to solve the problem and furthermore, if the pattern can be made general, it can be used to solve other problems as well.

**Abstraction** - recognising the most essential elements of a problem and disregard irrelevant details.

**Algorithm** - working out the steps needed to solve the problem. An algorithm can be set out as pseudocode or shown graphically in a flow diagram.

## Writing pseudocode

### 1. Algorithm

This section takes a closer look at pseudocode - one of the techniques used to describe an algorithm. As a reminder, the definition of an algorithm is shown below.

**Algorithm** - is a list of step-by-step instructions that, when followed, will solve a problem.

Algorithms have three kinds of data flow. These are

- Sequence: One command follows another
- Selection: A choice is made between two or more data paths
- Iteration: The same group of command is repeated a certain number of times

The two main techniques for producing an algorithm are:

- Pseudocode
- Flowcharts

We will cover various parts of how an algorithm is put together. This includes Operators and Variables, Conditionals, and Loops.

These concepts are also used in actual programming languages. This section is intended as an overview, so we will discuss each of the concepts in more detail in their own section. You can find those more detailed sections here:



- Operators and Variables
- Conditionals (Selection)
- Loops (Iteration)

# Writing pseudocode

## 2. Pseudocode

Pseudocode is a text-based way of setting out an algorithm. Here is a very simple example:

- START
- Fetch a tea cup
- Boil some water
- Place a tea bag into the cup
- Pour on boiling water
- Stir teabag
- Remove teabag
- END

Pseudocode is not a formal computer language and so it has no particular rules governing how it should be written. You may see many different versions in books, on the web and even in your exam papers.

What is important is that pseudocode should be easy to read, unambiguous and error free. Especially because programmers often use it to form the basis of their programming code.



# Writing pseudocode

### 3. Being precise

The pseudocode on the previous page for boiling water and making tea is not very clear:

- START
- Fetch a tea cup
- Boil some water
- Place a tea bag into the cup
- Pour on boiling water
- Stir tea bag
- Remove tea bag
- END

How much water needs to be boiled? How much should be poured onto the tea bag? How long should the tea bag be stirred for? What do we do with the tea bag when it is removed?

The pseudocode can be written more precisely in order to answer these questions:

- START
- Fetch a tea cup
- Add 300 ml of water to the kettle
- Boil the water until the kettle switches itself off
- Place a new tea bag into the bottom of the cup
- Pour 200 ml of boiling water from the kettle, into the cup
- Leave the tea bag to stew for 10 seconds
- Use a metal spoon to stir the tea bag for 3 seconds
- Using the spoon, remove the tea bag from the cup
- Dispose of the tea bag
- END

Note the 'START' and 'END' statements. These confirm the points where the algorithm begins and where it has been fully laid out.

You could also number each step in between, to make it easier to talk about individual lines of the algorithm, but it is not essential.

If the task of the computer programmer was to write code for controlling a tea-making robot, this extra precision is essential.

## Writing pseudocode

## 4. Keywords

You can divide up the words in a segment of pseudocode into *action words* that do something, and the words that specify what is to be done.

There is a convention that action words, like "print" or "input", are put in upper case in order to make them stand out. Everything else is left in lower case. This makes it much clearer to see the structure of the pseudocode.

For example

```
START
DISPLAY "What is your first name?"
INPUT firstName
DISPLAY "What is your surname?"
INPUT surname
FullName = COMBINE firstName, surname
PRINT FullName
```

The keywords are in capitals - these are the actions the algorithm carries out.

When it comes time for a programmer to change pseudocode into actual source code, they will have to decide how to actually perform the task. How this is done will vary from one programming language to another.

Pseudocode is more about working out the general flow of an algorithm, to act as a guide for the programmer. So vague keywords are fine.

## Writing pseudocode

## 5. Operators and variables

### *Operators*

One of the most common things that programs are used for is doing some kind of mathematics. Even for tasks where humans can intuitively keep track of the numbers involved, computers need to use maths to do the work.

So it's important that algorithms are able to talk about maths. Since pseudocode is meant as a rough guide, you could spell out the task exactly for simple maths "ADD 5 onions to soup", or you can just indicate that maths will need to be done when the program is converted into source code. For example "CALCULATE the value of X"

## *Variables and assignment*

Values that a program has to keep track of are called "variables". The "X" in the previous paragraph, for example, is a variable. As well as calculating values of variables, you can just tell the program what its value is. This is called "assignment":

START

SET N to 5

ADD N Onions to Soup

ADD N Carrots to Soup

END

Common "assignment" keywords used in pseudocode like this are SET, INIT (short for "initialise"), or just using an = sign or arrow

# Writing pseudocode

## 6. Conditionals (Selection)

The very simplest algorithms will robotically perform the same tasks every single time. But ideally you want algorithms to be able to do different things depending on the current conditions. This is done using "conditionals".

The most common format for conditional statements is using the keywords "IF, THEN, ELSE".

For example:

```
START
IF EndOfSchool
    THEN Prepare_to_go_home()
ELSE
    Prepare_for_Next_Lesson()
END IF
END
```

You can combine more than one condition together in an algorithm by using "ELSE IF":

For example

```
START
IF EndOfSchool
```

```

    THEN Prepare_to_go_Home()
ELSE IF Lunchtime
    THEN Visit_Cafeteria()
ELSE
    Prepare_for_Next_Lesson()
END IF
END

```

Notice that the tasks performed within the IF statement are indented. This is common practice when writing pseudocode, as it makes it much easier to read. It's not required, just as very little else is *strictly* required when writing pseudocode, but it's a very good idea.

# Writing pseudocode

## 7. Loops (Iteration)

Sometimes you want an algorithm to specify that a task has to be carried out more than once. You could write out each repetition, like this:

```

START
SET Sticks IN Basket TO 0
IF 0 Sticks IN Basket
    THEN ADD 1 Sticks TO Basket
IF 1 Sticks IN Basket
    THEN ADD 1 Sticks TO Basket
IF 2 Sticks IN Basket
    THEN ADD 1 Sticks TO Basket
ELSE
    PRINT "There are now 3 sticks in the basket"
END IF
END

```

You can see that this would start to get very long and repetitive if you had to carry out the same task more than a handful of times. Thankfully, there's a shortcut: loops.

There are several types of loop: FOR loops, WHILE loops, and REPEAT loops. We will go into more detail on the differences between them later, but they all do the same kind of thing - they carry out a task more than once.

For example:

```

START
Sticks = 0
WHILE Sticks <= 10
    ADD 1 to Sticks
END WHILE
PRINT "There are ten sticks"

```

END

This loop will keep adding sticks until there are ten sticks, and then end. The algorithm will then move on to the next step.

Note that much like conditional statements, it's good practice to indicate where a loop will end within an algorithm. This makes it easier to pick out the loop from the steps before and after it.

## Writing pseudocode

### 8. Exam Reference Language

The pseudocode we have described so far is still perfectly fine to use.

However, for this syllabus (OCR J277) an Exam Reference Language has been introduced. This has some advantages over informal pseudocode, namely

- Used to pose exam questions formally
- Has a more formal structure than pseudocode
- More precise
- More consistent

As an update to the J276 syllabus, a few extra items have been added

- Random number command, `random(...)` to help with learning programming techniques
- File handling commands, to simplify and standardise file statements these include
  - `open()`
  - `close()`
  - `readLine()`
  - `writeLine()`
  - `endOfFile()`
  - `newFile()`
- Iteration, added the DO WHILE loop
  - `do ----- until ....`
- Iteration, added the STEP command for FOR loops
- Casting, Added FLOAT as a data type
- String handling, the + symbol is used to denote concatenation

# Flow charts

## 1. Algorithm

This section describes flowcharts - one of the techniques used to describe an algorithm. As a reminder, the definition of an algorithm is shown below.

**Algorithm** - is a list of step-by-step instructions that, when followed, will solve a problem.

The two main techniques for producing an algorithm are:-

- Pseudocode
- Flowcharts

For an amusing introduction to flowcharts (it's not essential though), have a look at the two minute clip from the 'Big Bang Theory' where Sheldon creates an algorithm on 'How to Make a Friend' and lays it out as a flowchart.

You will need YouTube access and a set of headphones.

# Flow charts

## 2. Setting out an algorithm

An algorithm breaks down a task into a series of steps.

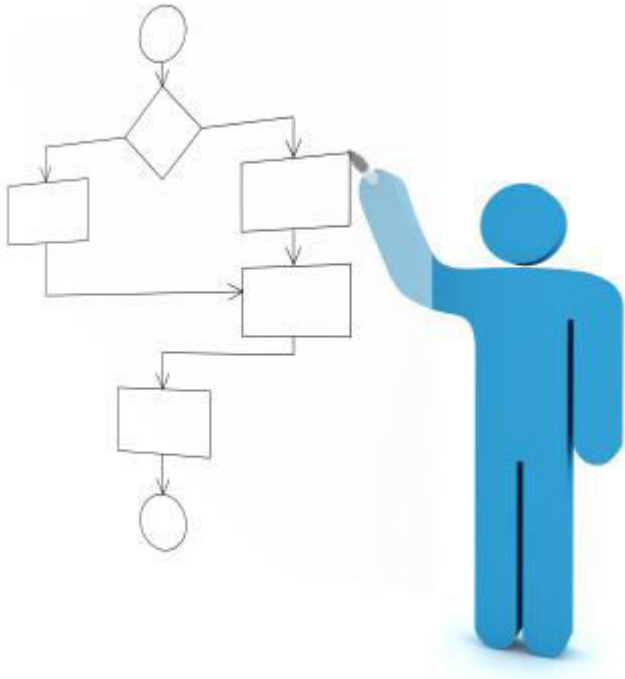
For example, let's take making a sandwich. What steps do you need to follow?

- Fetch two slices of bread
- Fetch knife
- Spread butter on each slice of bread
- Place one slice of bread on a plate with the buttered side facing up
- Add 1 slice of ham on top of the buttered bread
- If required add mustard
- Put second slice of bread on top of ham (with buttered side facing towards ham)
- Use knife to cut bread diagonally in half



There are lots of stages to this simple task. Some of them need to be done in a certain order. For example, you need to get the knife before you can butter the bread. You need to add the ham before you can cut the sandwich. There are also choices to be made such as the mustard.

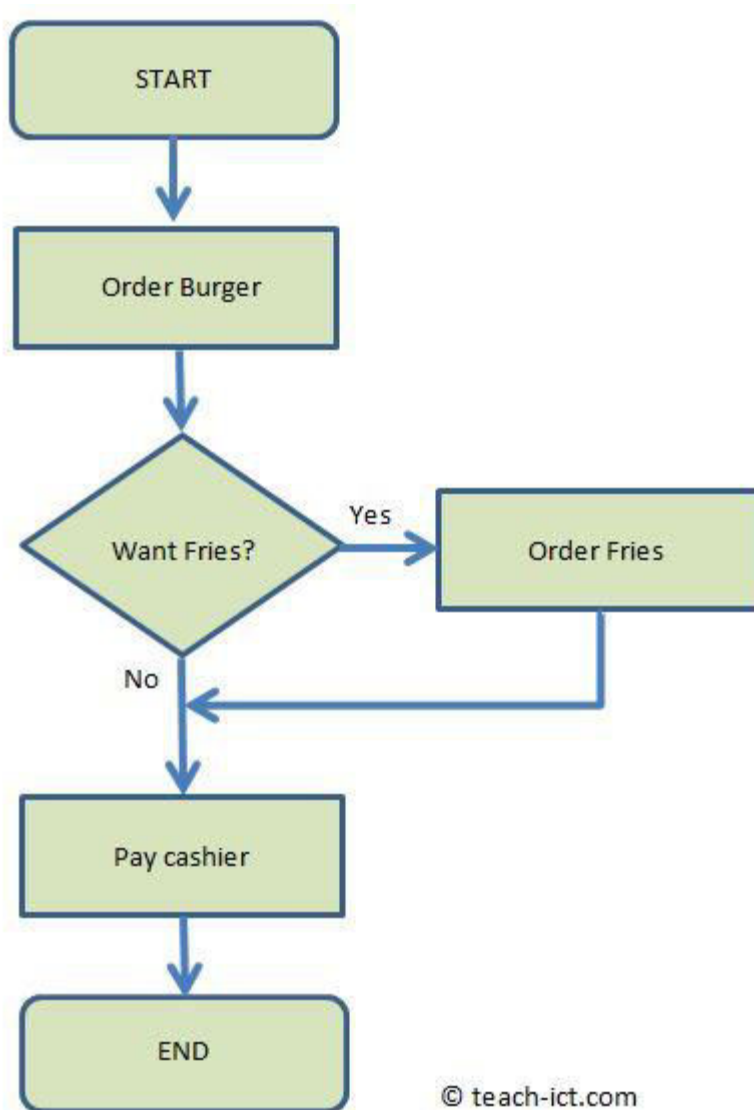
Flow charts are useful for laying out the process as a visual diagram.



## Flow charts

### 3. What is a flowchart

A flowchart is a diagram that shows the breakdown of a task or system into all of the necessary steps.



Each step is represented by a symbol. Connecting arrows show the step-by-step progression through the task.

Have a look at the diagram above. It shows the process of ordering a burger.

There is a clear start, a series of steps, a clear direction of flow and a clear end or finish point.

This is a very simple flowchart. For some tasks or systems, the flowchart can be very complex and detailed.


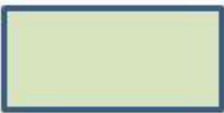
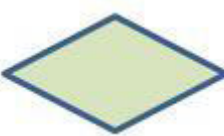



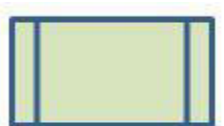
## Flow charts


### 4. Flowchart shapes

There is a recognised standard (ANSI) that sets out the shapes that can be used in a flowchart and what they mean.

Using standard shapes means that anyone can view a flowchart and at least understand how the algorithm flows.

Here are some of the more common shapes

Flowchart shapes	
Its meaning	Shape
Terminator - the start or end of the algorithm	
Process or Action - something is being carried out	
Decision - there are always two outputs, yes no for example	
Data - an input or an output	
Document - a document is being used	
Delay - need to wait a certain time before carrying on	
Pre-defined process - this points to an algorithm that has already been defined somewhere else.	

Flowchart shapes	
Its meaning	Shape
Flow - the arrow indicates the direction	

## Flow charts

### 5. Flowchart worked example

**Problem:** You have been shopping and your basket now contains several items, each with a different price. Work out how many items are in your basket, and how much the basket will cost. Also, work out the average price of the items in the basket. Display all of these results.

**Task:** break down the problem (decompose) into a set of instructions. Then display this algorithm as a flow chart.

**Solution: Steps needed**

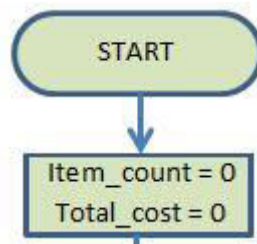
1. Enter the cost of each item (no need to identify the item itself)
2. Keep a count of how many items were entered
3. Work out the total cost as the sum of all the prices
4. Work out average cost by dividing the total cost by the number of items
5. Display the total cost, average cost and the number of items

Let's build up the flow chart step by step. Starting with the start terminator



We need to keep track of how many items we have bought and the total cost of them, so we need to set a starting value for each of them.

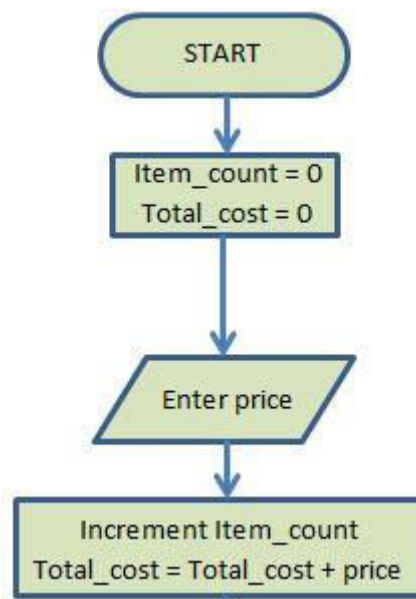
Since we haven't bought anything yet, both the "item counter" and "total cost" variables are set to zero.



Next we allow the person to enter the price of a single item. This is an input, so the input box is used.

Since we now have one more item than we did before, the item counter is incremented by 1.

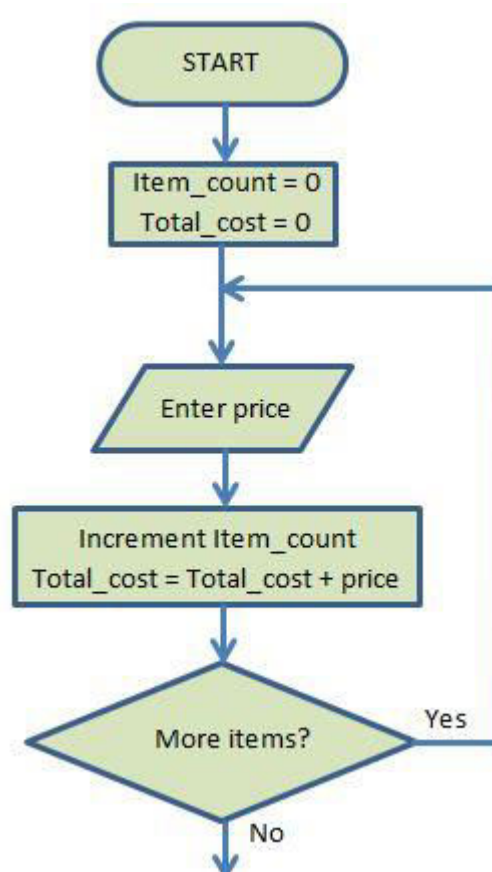
The "total\_cost" variable is updated by adding the newest item price to it, allowing us to keep track of how much money we are spending.



Next, we need to check if the person still has more items to enter, so a question is asked using the diamond decision box.

If the answer is Yes, the algorithm loops back and prepares to add the next item.

Once all items have been entered, then the algorithm takes the No branch and continues on.



A reminder of the original problem:

**Problem:** You have been shopping and your basket now contains several items, each with a different price. Work out how many items are in your basket, and how much the basket will cost. Also, work out the average price of the items in the basket. Display all of these results.

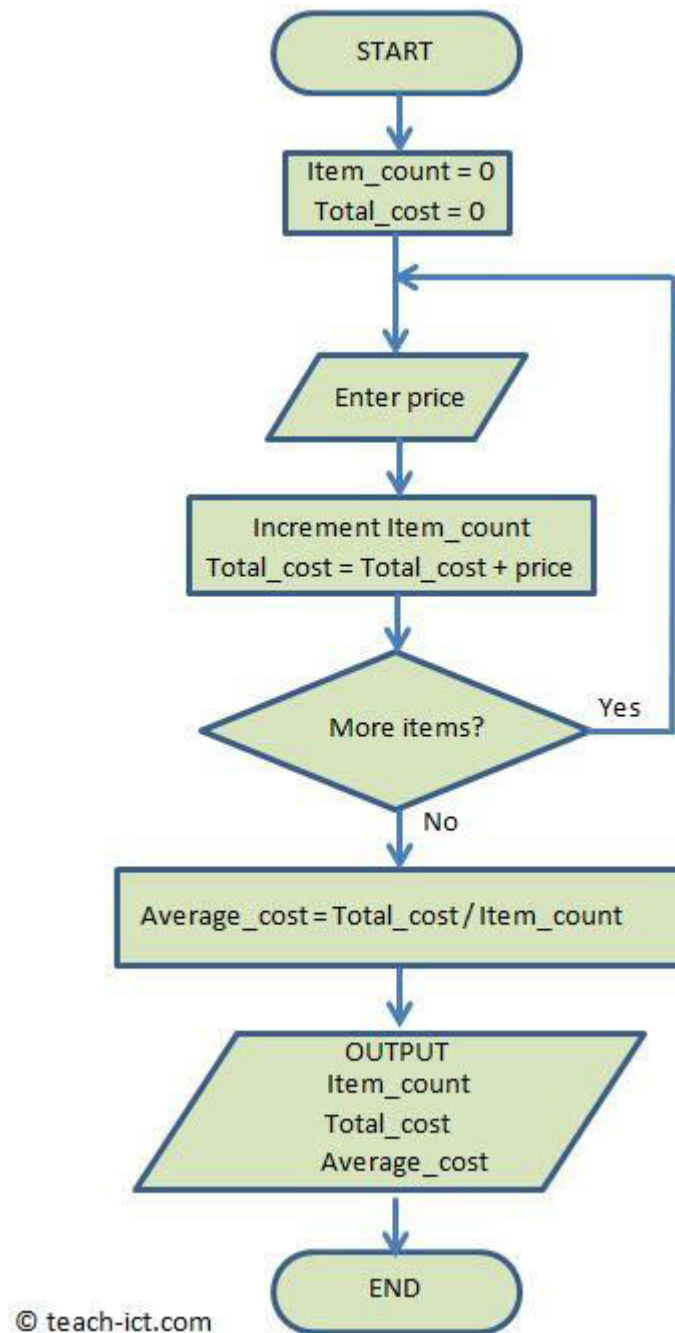
Now that all items have been entered:

- item\_count tells us how many items we have bought.
- Total\_cost tells us the total cost of all of these items

The only things left to do are to calculate the average price and to display the results.

The first is an action, and so it uses the square-shaped process box. Displaying the results is an output, so it uses the parallelogram-shaped data box.

The finished flowchart ends with a terminator.



## Flow charts

### 6. Flowchart Pros and Cons

As flowcharts are just one way of describing an algorithm, it is useful to consider why you might want to use them over, for example, writing the algorithm in pseudocode.

#### *Advantages*

- Excellent tool to be able to see the flow of the algorithm.



- It clearly shows which decisions have to be made along with inputs and outputs.
- Just by looking at a flowchart you get a sense of how simple or complicated the algorithm is.

### *Disadvantages*

- Drawing the shapes tidily and lined up can be time consuming. Although there are specialist applications designed just for flowcharts.
- Very awkward to change a flowchart once completed.
- Not so useful for very complicated algorithms.