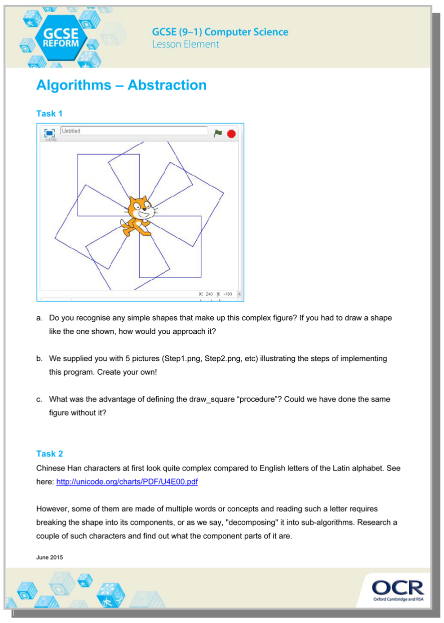
Algorithms – Abstraction

Instructions and answers for teachers

***These instructions should accompany the OCR resource 'Algorithms - Abstraction' activity** which supports OCR GCSE (9 –1) Computer Science*

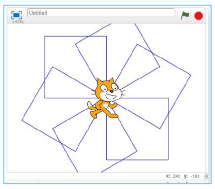


GCSE REFORM

GCSE (9–1) Computer Science
Lesson Element

Algorithms – Abstraction

Task 1



a. Do you recognise any simple shapes that make up this complex figure? If you had to draw a shape like the one shown, how would you approach it?

b. We supplied you with 5 pictures (Step1.png, Step2.png, etc) illustrating the steps of implementing this program. Create your own!

c. What was the advantage of defining the draw_square "procedure"? Could we have done the same figure without it?

Task 2

Chinese Han characters at first look quite complex compared to English letters of the Latin alphabet. See here: http://unicode.org/charts/PDF/U4E00.pdf

However, some of them are made of multiple words or concepts and reading such a letter requires breaking the shape into its components, or as we say, "decomposing" it into sub-algorithms. Research a couple of such characters and find out what the component parts of it are.

June 2015

OCR
Oxford Cambridge and RSA

The Activity:

This resource comprises of 7 tasks.

These simple activities help the students understand the concept of abstraction and how algorithms are used to solve problems.

This activity offers an opportunity for English skills development.

This activity offers an opportunity for maths skills development.

Associated materials:

'Algorithms – Abstraction' Lesson Element learner activity sheet.

Introduction

Abstraction is finding patterns and packaging similar looking code into self-contained chunks of code. Packaging allows faster program development as it allows programmers to re-use previously created code, so we don't have to repeat it. This "folder of code" exists on one place in the program and other parts of the program "link" to it, just like web pages link to each other. Imagine, instead of sending somebody a link to something you liked on the internet, you had to copy and paste it inside your message! It would be much less efficient.

Apart from the extra time and memory used by this operation, if the page got updated, your recipient will have outdated information, while when sending a link, if they choose to follow it they will get the real up-to-date version of the page. It makes for easier maintenance - change the code in one place and the links to this code from elsewhere in your program will not need to be changed. Additionally, when programming as a team, other programmers don't need to know how your part works, they can just link to it - they become users of your program, its workings are hidden from them by what we call "the layer of abstraction".

Processing data and implementing formulas requires a particular sequence of steps. It is not always the case and depends on the task. In Cooking, the order in which you add ingredients matters. In Maths, addition doesn't care about which order you add numbers in, while division does. Similarly, in programming, command-line interfaces and batch processing require predetermined order for user input, while with graphical user interfaces, buttons can be clicked in different order, so, no set algorithm for user input is needed.

Task 1

This activity will require a widely accessible programming package Scratch 2.0, either in its off-line (application https://scratch.mit.edu/scratch2download/) version or its online version (https://scratch.mit.edu/). It is important that procedures (custom blocks) are supported.

Task 1: Drawing polygons in Scratch (will also work in any language with “pen” capabilities, eg Python Turtle). A square is an example of a polygon with 4 sides and 4 angles. As the angles of a complete polygon need to add to 360 degrees (otherwise, there will be gaps in its perimeter), each of the 4 angles is 360/4=90 degrees, as the pupils will know.

Learners will develop an algorithm to draw a square in Scratch 2.0, first line by line (see file Scratch1.png), then as a procedure (see file Scratch1.png), then as a generic polygon. As you can see, at this stage, calling this procedure doesn't tell us how the shape is drawn; we just use it by supplying the parameters to draw the polygon as we need it.

Task 2

Decomposition is the process of breaking a complex problem down in to smaller more manageable portions. Decomposition makes great use of abstraction by identifying common patterns in these portions and reusing the code written for one part on other parts.

In this task, learners will look at Chinese letters which are made up of simpler parts and research the process through which the meaning gets “composed”.

Task 3 – Algorithmic Thinking

A well-known problem involves getting some creatures across the river in the same boat (such as described here: http://en.wikipedia.org/wiki/Fox,_goose_and_bag_of_beans_puzzle). Discuss what happens if the farmer uses the wrong algorithm to carry them across?

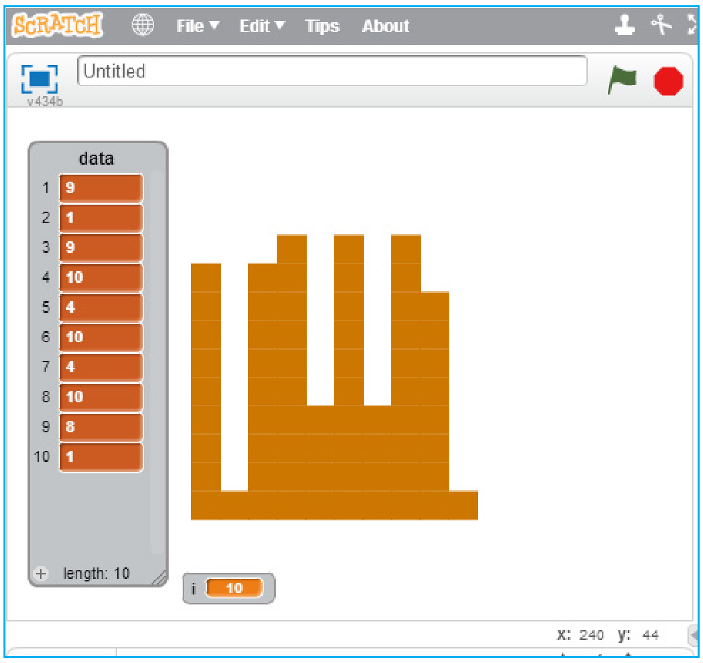
Can you make this problem more complex by introducing another creature that is dangerous to any of those present in the puzzle already.

Task 4 – Standard searching algorithms

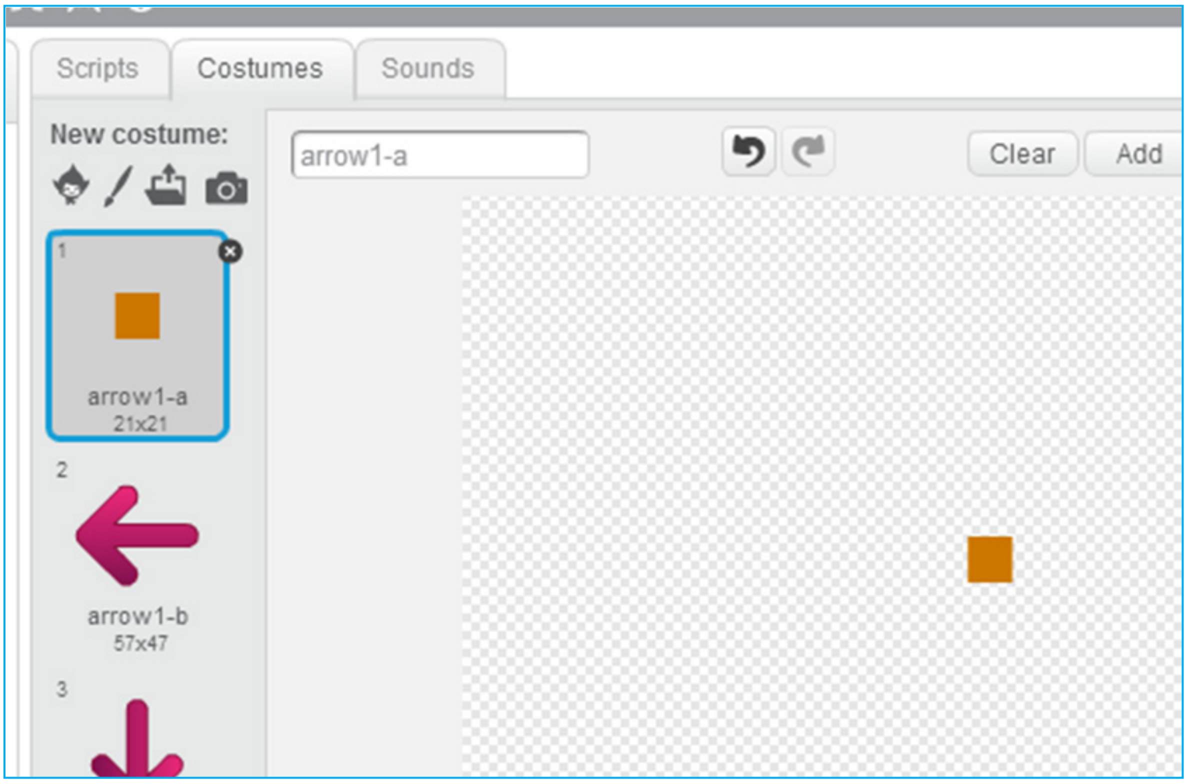
Issues:

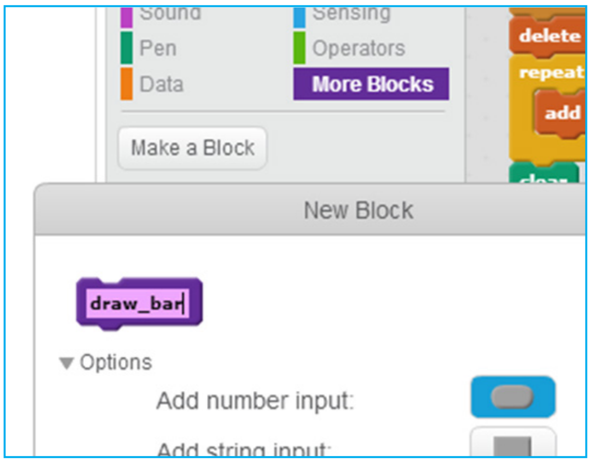
The learners need to be aware that both searching methods are used and have their place. Regardless of the algorithm used, the learners need to be aware (it's a common mistake) that the algorithm must allow for a condition where the search is over but the item is not found. The choice of the searching algorithms depends on the data that needs to be searched. The data that is unsorted will have to be searched in a linear, no-stone-unturned-until-found, door-to-door [picture of travelling salesperson/policeman or another suitable analogy for someone visiting every single house on a street]. This is because we don't know enough about the data to use any shortcuts. We can say that unsorted data is a poorer quality than sorted as the linear search is inefficient and processor intensive. However, storing data in unsorted state works better for weaker processors that might be overwhelmed by the process of simultaneously recording new data and sorting it.

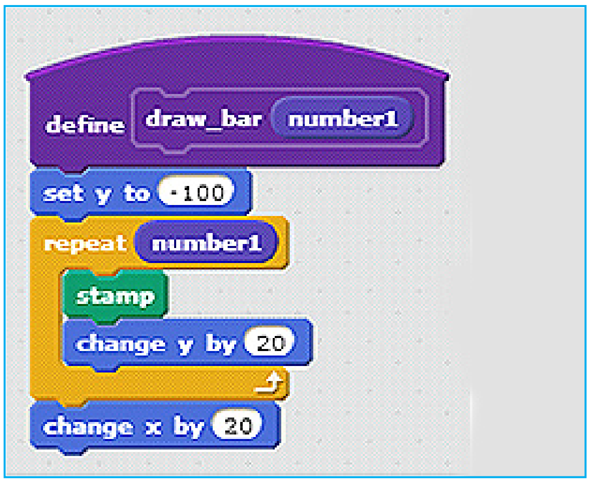
Binary search is much more efficient as we need to look at the fewer items before we find the one we need. How is that achieved?



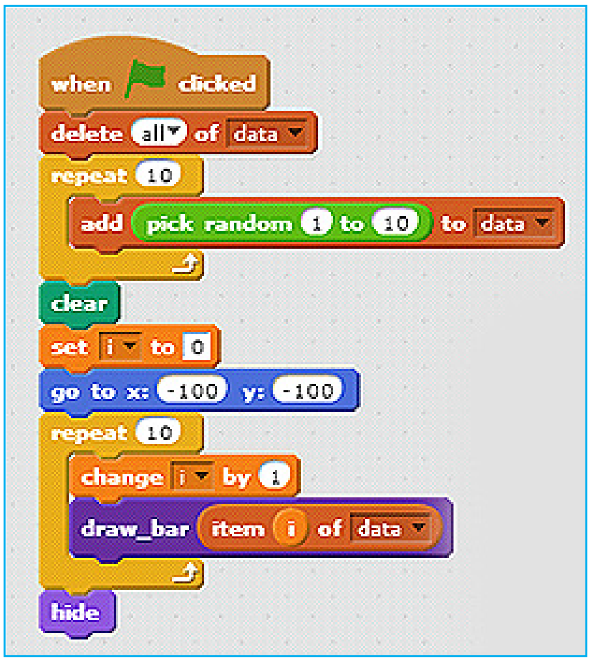
	data
1	9
2	1
3	9
4	10
5	4
6	10
7	4
8	10
9	8
10	1







```
define draw_bar (number1)
set y to -100
repeat (number1)
    stamp
    change y by 20
change x by 20
```



```
when [flag] clicked
delete all of data
repeat 10
    add pick random 1 to 10 to data
clear
set i to 0
go to x: -100 y: -100
repeat 10
    change i by 1
    draw_bar item i of data
hide
```

Version 1

When booking into this hotel, you make it to your floor and this sign is the first thing you see. If you are looking for room 318, you will not have to search the entire floor, just the left side. To simplify the search for your room, the hotel conveniently numbered their rooms in sequence and placed the entrance to the floor in the middle, so that you can halve your search effort and time. You know by now that "binary" could be translated "one of the two options" - which is what the hotel gives you here - left or right. Imagine that if you do go left and about half way through that side of the floor, the corridor turns and branches out into further 2 halves with numbers "319-321" and "322-327". Which way will you turn?

Binary search

A hotel has 120 rooms located on 3 floors and 4 lifts/staircases. Design a floor plan (or signs for a given floor plan) for each of lifts/staircases. Assuming a person starts at lift 2. How many signs will they see before they find their room?

Given pseudocode for a binary search, write it as (a) structured English; (b) code in your preferred high level language

Linear search

Given pseudocode for a linear search, write it as (a) structured English; (b) code in your preferred high level language

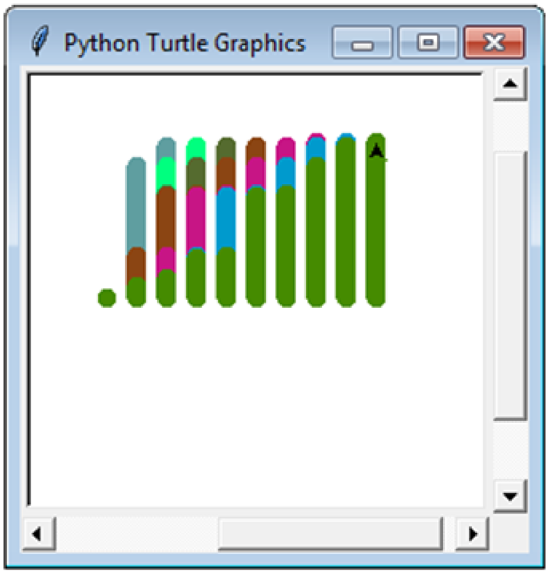
Task 6

Sorting and searching could end up being quite dry unless there is a visual element to them. There are many visualisations of the searching and sorting algorithms available on the internet, such as the one at http://www.sorting-algorithms.com. You will find that learners understand the algorithms better if they also can plot the results using a language with graphics abilities, eg Scratch/BYOB/Snap/AppInventor/Python/Basic, etc. The tasks would involve generating a random list of numbers and then applying an algorithm to it. Provided is an example of visualising a list in Scratch 2.0, using abstraction via a procedure. This procedure uses the “stamp” feature (which imprints the copy of the sprite’s costume onto the Stage. The sprite’s costume should be a smallish square (although, most symmetric shapes should work, as well).

Learners should be given an opportunity to implement just the plotting of a list via the provided example and then they can design.

For the extension activity, here is an example of using Python Turtle to do a visualisation of the Insertion Sort.

```
>>>
Insertion sort
[66, 0, 76, 21, 52, 51, 6, 78, 10, 20]
Time taken: 7.737010955810547 seconds
[0, 6, 10, 20, 21, 51, 52, 66, 76, 78]
```



```
import turtle; from random import sample, choice; import time
COLOURS = ["firebrick", "dark goldenrod", "sienna", "chartreuse4",
           "DeepSkyBlue3", "medium violet red", "red4", "saddle brown",
           "dark olive green", "spring green", "cadet blue", "deep pink"]

screen = turtle.Screen()
t = turtle.Turtle(); t.speed(0); t.width(10)

def draw_col(x, y):
    t.penup()
    t.goto(x, 0)
    t.pendown()
    t.setheading(90)
    t.forward(y)

def draw_sort(s):
    for i in s:
        draw_col(s.index(i)*15, i)

def insertion_sort(s=sample(range(100), 10)):
    startTime = time.time()
    print("Insertion sort")
    print(s)
    for i in range(1, len(s)):
        val = s[i]
        j = i - 1
        while (j >= 0) and (s[j] > val):
            s[j+1] = s[j]
            j = j - 1
        s[j+1] = val
        t.pencolor(COLOURS.pop())
        draw_sort(s)
    endTime = time.time() - startTime
    print ("Time taken: " + str(endTime) + " seconds")
    return s


def quick_sort(s=sample(range(100), 10)):
    startTime = time.time()
    print(s)
    less = []
    equal = []
    greater = []

    if len(s) > 1:
        pivot = s[0]
        for x in s:
            if x < pivot:
                less.append(x)
            if x == pivot:
                equal.append(x)
            if x > pivot:
                greater.append(x)
        print(less+equal+greater)
        return quick_sort(less)+equal+quick_sort(greater)
    else:
        endTime = time.time() - startTime
        print (endTime)
        return s
```

```
if __name__ == "__main__":
    print(insertion_sort())
    #print("Quick sort")
    #s = quick_sort()
    #print(s)
    screen.exitonclick()
```

Task 7

Pseudocode, flow charts and structured English are used to describe the code before it is created. They are “language-agnostic” – meaning pseudocode doesn’t contain any references to any specific languages and in fact, the choice of the language used to do a program happens after the pseudocode is complete. The knowledge of pseudocode is needed for both exams and controlled assessment task submissions.

Learners should be fluent in converting problems between the 3 techniques (and, of course to an actual code of a language like Python or JavaScript). This exercise asks pupils to create a dictionary – a “Rosetta Stone”.