COMPUTING AT SCHOOL
EDUCATE · ENGAGE · ENCOURAGE
In collaboration with BCS, The Chartered Institute for IT

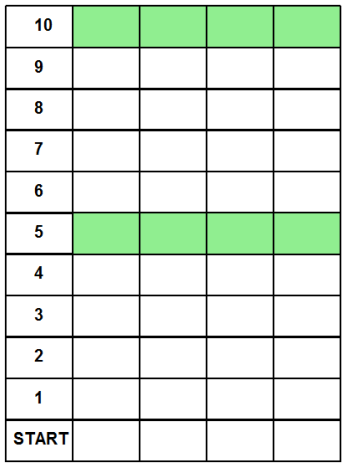
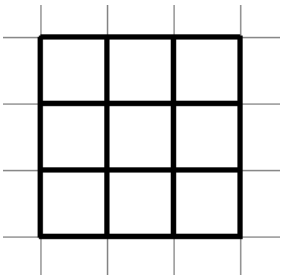
UCL ENGINEERING
Change the world

UCL/CAS Training for Master Teachers and Teachers

Algorithms and Programming Module 1

'—so long as I get SOMEWHERE,' Alice added as an explanation.
'Oh, you're sure to do that,' said the Cat, 'if you only walk long enough."
--- Lewis Carroll, Alice's Adventures in Wonderland &Through the Looking-Glass

Table 1

	
AQADO BOARD	A SIMPLIFIED MODEL

WORKBOOK 4
UNPLUGGED PROGRAMMING
COMPUTATIONAL THINKING
PROJECT AQADO

- Addressed to Teachers
  Activities are graded: easy to hard – 0 to 5*. You should attempt every activity marked without a star or marked with one *.

CONTENTS

SOLVING A PROBLEM, COMPUTATIONAL THINKING, A TOOLBOX FOR PROGRAMMING

- Let's see how programming and computational thinking come together by undertaking a project.

A. In AQA GCSE 2015, as part of their controlled assessment, students were asked to write a program for the computer to host a simple board game, AQADO. Let's take on a part of that program, namely, to build a program to draw the AQADO board. Our approach will probably generalise (G) to drawing boards for a number of popular games.

B. First we use a simple but comprehensive model of Computational Thinking: **ADAGE**

   i) **A**lgorithmic Thinking (AT)

   ii) **D**ecomposition (D)

   iii) **A**bstraction (A)

   iv) **G**eneralisation (G)

   v) **E**valuation (E)

Choosing a model of Computational thinking in itself is an example of Decomposition (breaking down into components) and Abstraction (selecting a tool and hiding a lot of detail).

A MODEL OF COMPUTATIONAL THINKING APPLIED TO PROGRAMMING PROJECTS

- When tackling a problem we are usually being asked how to solve it by thinking of a systematic approach with a sequence of steps. (Algorithmic Thinking)
  Approaches that often prove effective are:

1. Break the problem into separate parts (tasks); use a simple model(s) of the task. (Decomposition D and Abstraction A)
2. Find a *human* solution (algorithm) for the task on the way to, or as part of, a computer programmed one. (Algorithmic Thinking AT)
3. Select an appropriate toolbox, and create tools to add to our toolbox that help to perform the task (Algorithmic Thinking AT, Abstraction A)
4. See how well the solution fits; review and compare other solutions (Evaluation E)
5. Seek to generalise solutions for wider application (Generalisation G)

This is a *static* model of the process of computational thinking applied to programming. In practice, it is more likely to be used *dynamically* like a program, where the elements of the processes described above are revisited and repeated.

THE AQADO BOARD AND OUR SIMPLIFIED MODEL BOARD

- We tackle the board first (D), choose a simplified model of it: general enough to capture the essence of the structure; yet simple enough for us to set aside detail (D and A), and to investigate different approaches (E). We first attempt to solve the problem of drawing our model board: a 3x3 array of squares as in Figure 1. Our model for AQADO is also a model for all sorts of board games and pastimes from noughts and crosses (3x3), draughts, chess and reversi (8x8), sudoku (9x9), snakes and ladders (10x10), crosswords (13x13)… (Generalisation G).

In our simple model, we have postponed the rectangular shape, colouring, numbering and writing on the board to be completed as separate tasks (D). Even with our simple model, there are many different ways to draw it. We are looking to build a program(s)

1. that will draw the simple model, and, in which we choose effective building blocks to include as ***user-functions*** to make the job easier (AT and A).
2. that will take advantage of the physical symmetry of the drawing to include ***repetition*** in the code in order to make the code relatively easy to write and read (AT and E)
3. that will be in a form we can readily adapt to generalise for wider application, in particular, the Aqado board (E and G).

TWO MORE INSTRUCTIONS TO HELP US MOVE WITHOUT DRAWING

We add two more instructions (***system-functions***) for our pet/robot from the pool of operations, common to Scratch, Python, Coffeescript and Logo, which are essential tools when we want to move the pet/robot without drawing. This is particularly useful when we are trying to implement a *human* algorithm in our problem solving approach, where a human can move to any position on the paper without drawing.

These instructions are:
`penup or up` for short. Raises the pen. Instructions to move the pet/robot after this instruction do not draw until the **`pendown`** instruction is met.

`pendown or down or pd` Lowers the pen. Instructions to move the pet/robot after this instruction draw until the **`penup`** instruction is met.

OUR TOOLBOX: INSTRUCTIONS AND PROGRAMMING CONTROL STRUCTURES

PROGRAM INSTRUCTIONS

1) **`forward1,2.3,… or fd1,2,3,…`**
2) **`left turn or lt`**
3) **`right turn or rt,`**

4) **`penup`** or **`up`** for short.
5) **`pendown`** or **`down`** for short.

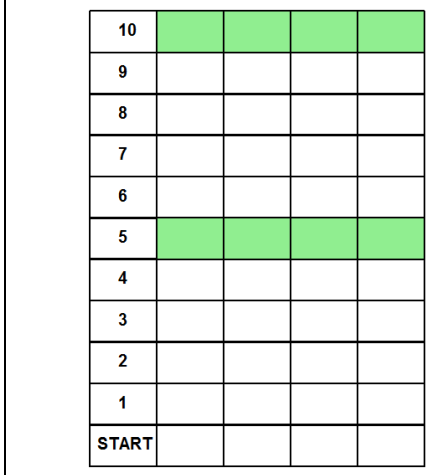
USER-FUNCTION INSTRUCTIONS

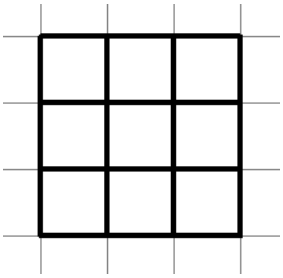
6) **`about turn`** or **`at`**
7) **`square1,2,3`**… or **`sq1,2,3`**…
8) **`fetch1,2,3`**… or **`fe1,2,3`**…

PROGRAMMING CONTROL STRUCTURES

- ***sequence*** (in the code)
- ***repetition*** (looking for symmetry in the drawing to use repetition in the code AT)
- ***functions*** (choosing and defining useful building blocks and defining ***user-functions*** AT and A)

Table 1: AQADO board and our simplified model.





AQADO BOARD	A SIMPLIFIED MODEL

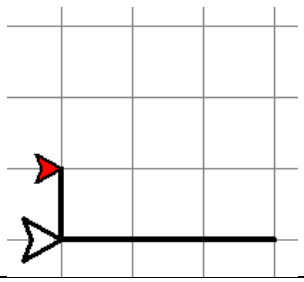
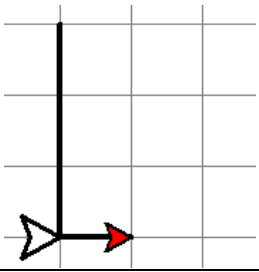
MISSION 1: PROGRAMMING WITH USER-FUNCTION FE3 AS A BUILDING BLOCK

We have decided to tackle initially a simpler problem: to draw the simplified model, a 3x3 array of unit squares.

1) One way you might start: draw the 3x3 grid on paper and write the program that copies your drawing movements using the Toolbox above. Look out for repetition in the structure and in your process of drawing. You may be able to enclose code in a repeat loop if code is next to a repeat of itself. Or if code is repeated at separated intervals the code could be defined as a ***user-function*** and the function name used wherever the repeated code appears.
2) Another way is to imagine you are the pet/robot and experiment with ways your pet/robot might traverse the framework in the least number of instructions from its base starting point. Look for symmetry in the code and use the **`repeat`** instruction, which reflects symmetry in the drawing and shortens the code. Where code of a building block occurs more than once in a drawing give the code a name as a ***user-function***.
3) In this mission, use the basic instructions **`fd, lt and rt,`** the repeat control structure, **`repeat,`** and the ***user-function***, **`fe3,`** from our toolbox.

   Experimenting with the code for the drawings in Table 2, or otherwise, write a program to draw the simple model of 3x3 squares in Table 1, making use of symmetry and repetition wherever you can.
4) ** Adapt your program to draw a 4x4 model of unit squares.

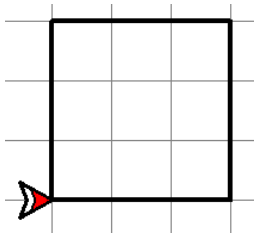
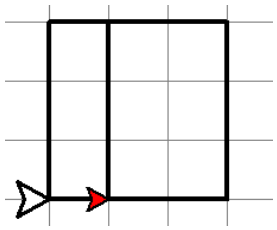
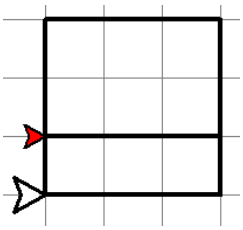
Table 2: Using `fe3` horizontally and vertically as a building block

	
`fe3 lt fd1 rt` ***What's next?***	**`lt fe3 rt fd1`** ***What's next?***

MISSION 2: PROGRAMMING WITH USER-FUNCTION SQ3 AND FE3 AS BUILDING BLOCKS

1) In this mission, use the basic instructions **`fd, lt and rt`**, the repeat control structure, **`repeat,`** and the ***user-functions***, **`sq3`** and **`fe3`** in our toolbox.
   By experimenting with the code for the drawings in Table 3 or otherwise, write a program, using the tools above, to draw the simple model of 3x3 squares in Table 1, making use of symmetry, and therefore repetition in the code, wherever you can.
2) ** Adapt your program to draw a 4x4 model of squares**.

Table 3: Using user-functions **`sq3`** ***and*** **`fe3`**

		
`sq3`	**`sq3 fd1 lt fe3 rt`** ***what's next?***	

MISSION 3: PROGRAMMING WITH USER-FUNCTION RE3 AS A BUILDING BLOCK

In this task we define another ***user-function***: **`rectangle`** or **`re`** -- with arguments 1, 2, 3 … (paces) -- which, in an extended form, will be useful later to complete the Aqado board.

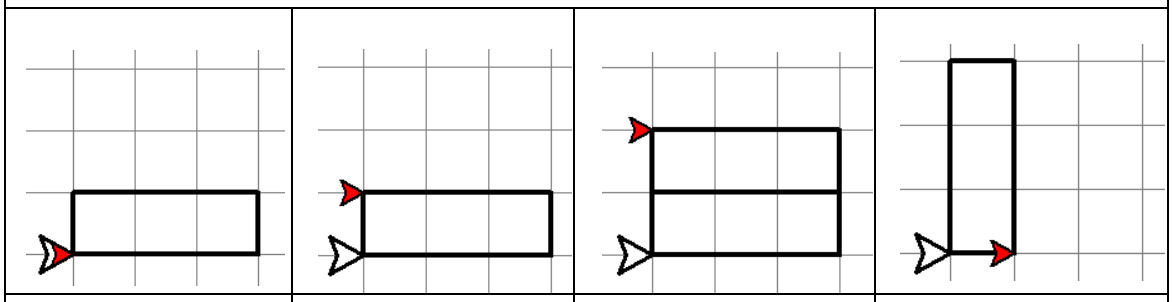
Formally we can define **`re`** as a function with a parameter p, which stands for arguments 1, 2, 3 :

```
rep -> repeat 2[fdp lt fd1 lt]
```

But in this task, all we need to know is that the ***user-function*** we have named **`re3`**, a RETURN program, draws a rectangle as depicted in Table 4A with the code:

```
repeat 2[fd3 lt fd1 lt]
```

Table 4

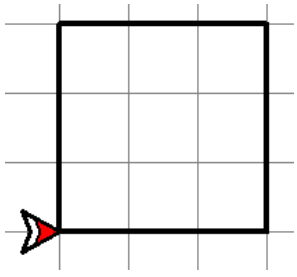
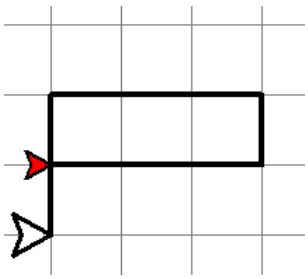
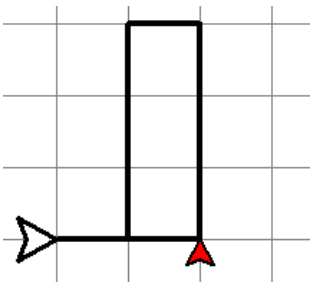


1) *Use the basic instructions **`fd, lt and rt`**, the repeat control structure **`repeat`** and the ***user-function*** **`re3`** added to our toolbox as a building block..
   By experimenting with the code for drawings in Table 4, A – D, or otherwise, write a program to draw the simple 3x3 model in Table 1.
2) **Extend your program to build a 4x4 array of squares

MISSION 4: PROGRAMMING WITH USER-FUNCTION SQ3 AND RE3 AS BUILDING BLOCKS

1) In this mission, use the basic instructions **`fd, lt and rt`**, the repeat control structure **`repeat`** and the ***user-functions sq3 and*** **`re3`** as building blocks from our toolbox. By experimenting with the code for drawings in Table 5, A – C, or otherwise, write a program to draw the simple 3x3 model in Table 1.
2) **Extend your program to build a 4x4 array of squares

Table 5

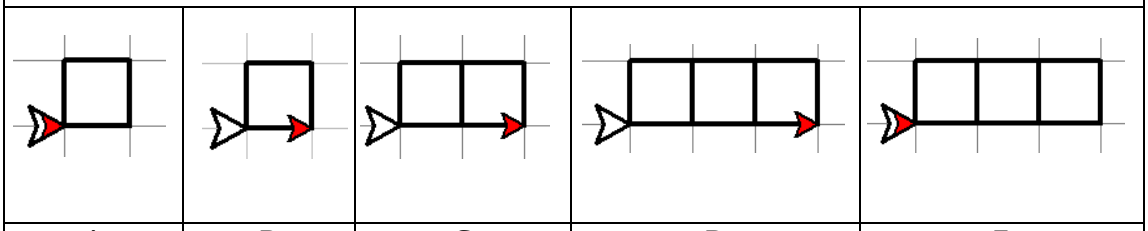
A	B	C
		

MISSION 5: PROGRAMMING WITH USER-FUNCTION SQ1 AS A BUILDING BLOCK

1) **In this mission, use the basic instructions **`fd, lt and rt`**, the repeat control structure, **`repeat,`** and the ***user-function***, **`sq1`** from our toolbox. Experimenting with the code for the drawings A – D, in Table 5, write a program to draw the RETURN program for the row of squares in E in Table 5.
2) ***Making use of symmetry and repetition in the code wherever you can, define a ***user-function*** for a bigger building block: **`rowofsq`** or **`rs`** for short,--- a row of squares -- where **`rs3`** for short represents the RETURN program for the drawing E in Table 6.

5) **Use ***user-function*** **`rs3`** defined as the RETURN program which draws the structure E in Table 6 to build a program to draw the simple model in Table 1.
6) **Extend your program to build a 4x4 array of squares

Table 6: Sticking unit squares together in a row



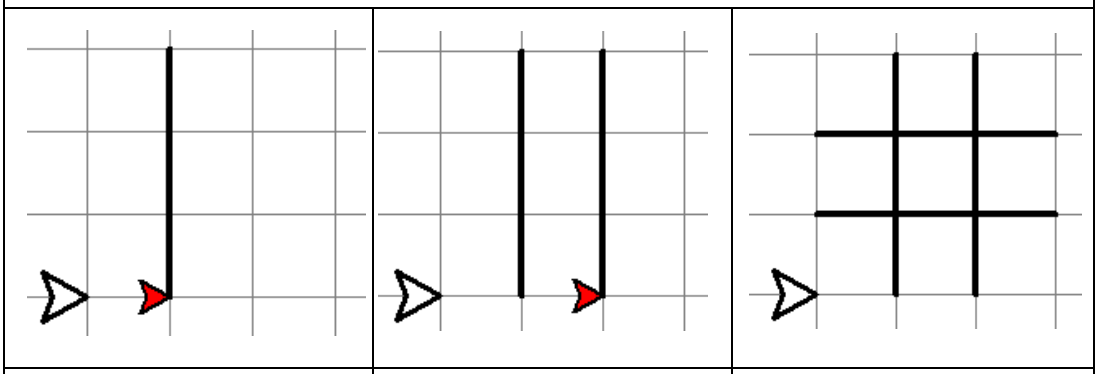
A	B	C	D	E

MISSION 6: FINDING AN ALGORITHM **TO DRAW THE** NOUGHTS AND CROSSES **GRID**

1) **Drawing the grid for noughts and crosses differs from the 3x3 simple model for Aquado in that if our starting point is off the grid see C in Table 6, we have to move on occasion without drawing. This means we have to use the **`penup or up`** and **`pendown or down`** instructions in our toolbox. Apart from that the algorithm we devised in Mission 1 can be used.

2) ** Alternatively, start the pet/robot at a starting point on the noughts and crosses grid, and draw just the grid without using the instructions **`up`** or **`down`**

Table 7: noughts and crosses grid



A | B | C

MISSION 7: CRACKING THE CODE – SEQUENCE, REPETITION AND FUNCTIONS

Read, Track and Crack the Code for the programs in Table 8:

	Table 8: Crack the Code: – functions and repeats	*Return Program ?*
1	`sq3 fd2 lt re3 fd2 rt fd1 at re3`	no
2	`repeat 3[re3 lt fd1 rt] fd1 rt re3`	
3	`** lt fd1 rt re3 rt fd1 lt repeat 3[fd1 lt re3 rt]`	
4	`** sq3 lt fd1 rt re3 rt fd1 lt fd2 lt re3`	
5	`** repeat 3[re3 lt fd1 rt] repeat 2[fd1 rt fe3]`	
6	`** repeat 3[fd1 lt re3] repeat 3[fd1 lt fe3]`	
7	`**** repeat 2[repeat 3[fd1 lt re3]]`	
8	`**** columnsp or cop -> repeat p[fd1 lt rep rt] repeat 2[co3 lt]`	
9	`**** prongp or prp -> repeat p[lt fd1 rt fep] fe3 pr3 rt pr3`	
10	`**** stackp or stp -> repeat p[rep lt fd1 rt] st3 rt st3`	
11	`**** repeat 2[co4]`	
12	`**** st4 rt st4`	
13	`**** co4 repeat 4[fd1 lt fe4]`	
14	`**** rowofsqp or rsp -> repeat p[sq1 fd1] at fdp at repeat 3[rs3 lt fd1 rt]`	
15	`**** repeat 4[rs4 lt fd1 rt]`	

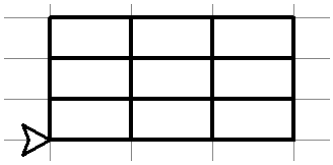
MISSION 8: PROGRAMMING TO DRAW THE AQADO BOARD

1) *** Choose one of the algorithms in missions 1-6, to draw the simple model (see Table 9 (A)), with the individual cells or boxes as rectangles 2 paces long, 1 pace high as defined in (**`re2`**).
   As an example, ***Crack the Code*** in Table 10.

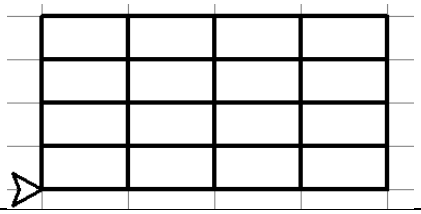
2) **** Adapt (generalise) the program you have chosen so that it draws a 4x4 grid of rectangles (see Table 9 (B))
   As an example, ***Crack the Code*** in Table 11.

3) ***** And generalise it further to draw the Aqado grid of 11 rows and 5 columns of rectangles **`(re2)`**
   As an example, ***Crack the Code*** in Table 12.

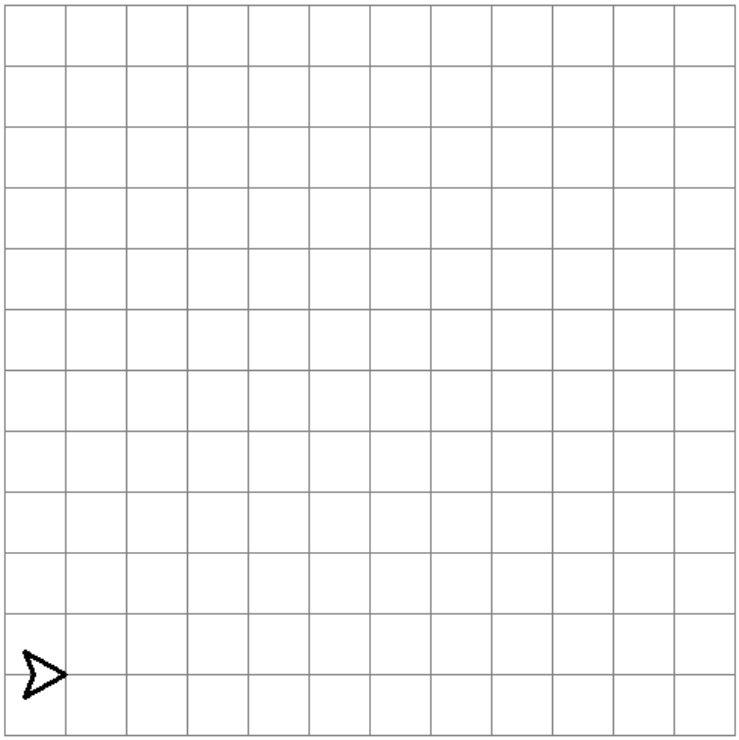
Table 9 : Introducing rectangular cells



(A) 3x3 array of rectangles **`(re2)`**

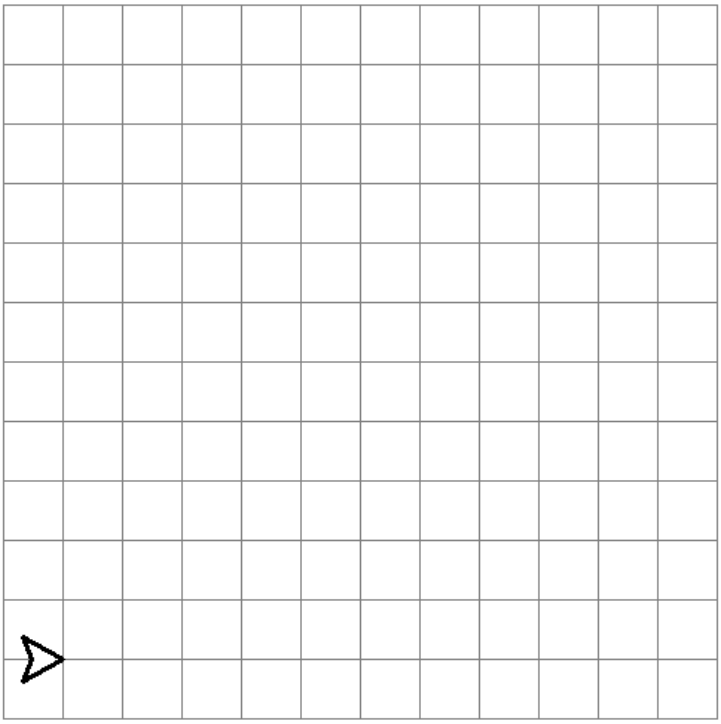


(B) 4x4 array of rectangles **`(re2)`**

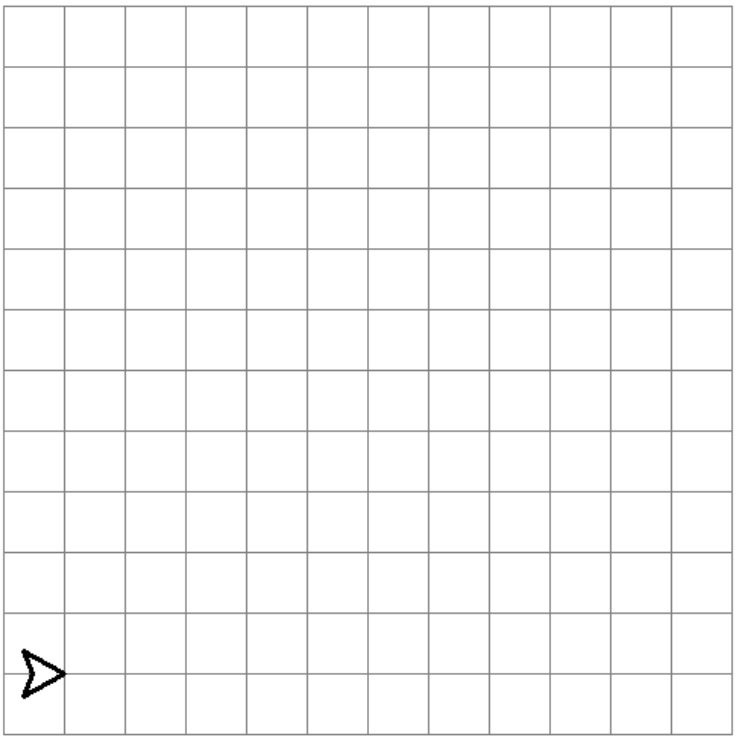
Table 10 Crack the Code:

`**** repeat 3[re6 lt fd1 rt] repeat 2[fd2 rt fe3 lt]`

c

Table 11 Crack the Code:



```
**** repeat 4[re8 lt fd1 rt] repeat 3[fd2 rt fe4 lt]
```

Table 12 Crack the Code:

`***** repeat 11[re10 lt fd1 rt] repeat 4[fd2 rt fe11 lt]`

MISSION 9: FUNCTIONS WITH TWO PARAMETERS

**** If your original Algorithm involves the sq3 user-function, you might want to define an extension of that as a user function **`outlinep,q`** or **`oup,q`** where p and q can take argument values 1, 2, 3 …; and p paces stands for the number of columns and q paces stands for the number of rows contained in the **`outline`**. Formally, we write

```
oup,q  -> repeat 2[fdp lt fdq lt]
```

See Table 10 for examples of outline grids **`ou4,2`** and **`ou5,3.`**

4) Crack the code

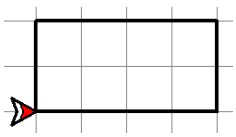
`ou3,3`

5) *** What is the **`outline`** or **`ou`** instruction that draws the outline enclosing the Aqado-like grid of unit squares with 5 columns and 11 rows?

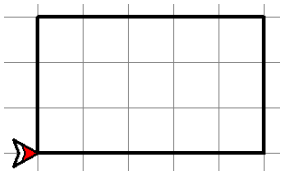
6) **** Adapt your program so that the outline grid is large enough to contain rectangles (**`re2`**) of the specified Aqado board instead of unit squares, see Table 8.

7) ****Complete the grid for the Aqado board

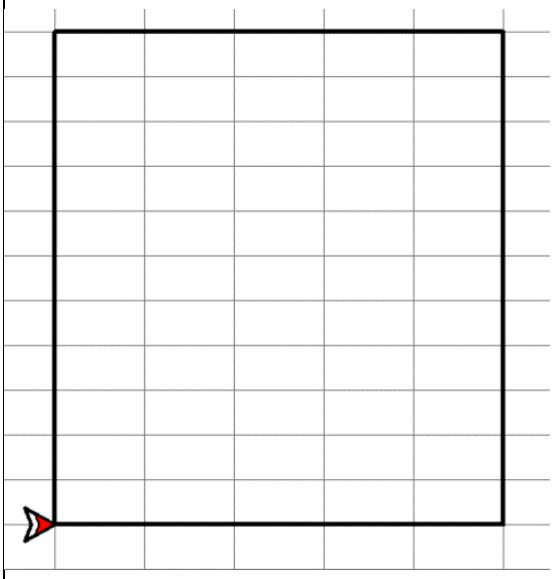
Table 10: outline grids



`ou4,2`



`ou5,3`



Aqado outline