

## UCL/CAS Training for Master Teachers and Teachers

### Algorithms and Programming Module 1

*"Would you tell me, please, which way I ought to go from here?"*

*"That depends a good deal on where you want to get to."*

*"I don't much care where –"*

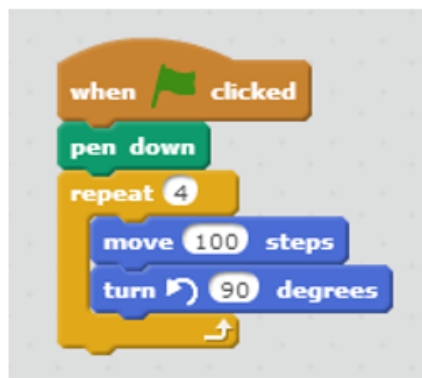
*"Then it doesn't matter which way you go."*

*- Lewis Carroll, Alice's Adventures in Wonderland & Through the Looking-Glass*

UPL

```
repeat 4 [fd2 lt]
```

Scratch



Python

```
from turtle import *  
#  
for paces in range(4):  
    fd(100)  
    lt(90)
```

# WORKBOOK 2

## PATTERNS & SYMMETRY

### ACTION GEOMETRY

## UNPLUGGED PROGRAMMING

## CONTROL STRUCTURE: REPETITION

- ❖ 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**

**Programming control Structures 2: Repetition..... 3**  
 Action Geometry ..... 3

**The Repeat Control Structure in Unplugged Programming ..... 4**

**Mission 1 Tracking and Stacking the Code ..... 7**  
 What’s in our Toolbox? ..... 7  
 Explanation: Symmetry in the Code and in the Drawing ..... 7

**Mission 2 Read and Crack the repeat Code ..... 11**

**Mission 3 Crack the Code by Drawing ..... 12**

**Mission 4: Match your Drawings ..... 16**

**Mission 5 Writing Programs to produce Drawings ..... 18**

**Mission 6 Writing Programs to fit the Drawings ..... 19**

**Mission 7 Patterns, Symmetry, Algorithms..... 20**  
 What’s the Answer? And can you say How you did it ..... 21  
 Explanation: Working it out, algorithmic thinking ..... 21  
 Explanation: There’s More than One Way to Do It..... 21

**Mission 8 Generalising Results..... 22**  
 Explanation \*\* The Process of Generalising ..... 22  
 Explanation \*\*\* A Different sort of Sequence..... 22  
 Explanation\*\*\*\* Generalising in Simple Algebra ..... 22

**PROGRAMMING CONTROL STRUCTURES 2: REPETITION**

<i>Table 1 (extract) from Workbook 1</i>		<i>Return Program ?</i>
9	<code>* fd1 lt fd1 rt    fd1 lt fd1 rt.....program(3)</code>	<b>yes</b>
10	<code>  * fd1 lt fd1 lt fd1    fd1 lt fd1 lt fd1.....                           program(4)</code>	<b>yes</b>
11	<code>* fd2 lt lt        fd2 lt lt.....program(1)</code>	<b>yes</b>
12	<code>  * fd1 lt    fd1 lt        fd1 lt    fd1 lt.....   program(2)</code>	<b>yes</b>

In the examples of Table 1 (above), you may have noticed the occurrence of repeat patterns in the code. We have separated the code with space to help pick out the repeat patterns

In the code below, we have written the repeated patterns in program(1) and program(2) from Table 1 underneath each other (stacked them), to pick out the repeated pattern in sequence as follows in **program(1a)** .

```
fd2 lt lt
fd2 lt lt.....program(1a)
```

Another example of a building block, with a more obvious and extended symmetric repeated pattern in the code and in the figure, is the square.

**ACTION GEOMETRY**

The symmetry of the square is reflected in the four equal sides and the four equal internal angles of the figure. An extended similar definition applies to regular simple polygons, e.g. pentagon has five equal sides with five equal internal angles as we shall see later in the Course.(Workbooks 9 and 10) But, as we see in Table 2, the symmetry of the square is reflected in the code that we use to draw it. The symmetry can be picked out in the ‘action geometry’ method we use to draw the square.

Table 2. The Square		
<pre>fd2 lt  fd2 lt fd2 lt  fd2</pre>	<pre>fd2 lt  fd2 lt fd2 lt  fd2 lt</pre>	
Incomplete repeat pattern	<code>program (2)</code> completes repeat pattern	A square with side of 2 paces

Both programs in Table 2 draw the same square but the RETURN `program (2)` has completed the symmetry of the four ‘`fd2 lt`’ components, the four ‘forward and left turn’ symmetric code components of the square in the ‘action geometry’.

In our unplugged programming, it helps to write the program on separate lines (stack them) to pick out a repeating pattern in sequence. So `program (2)` we may stack as:

```
fd2 lt
fd2 lt
fd2 lt
fd2 lt.....program (2a)
```

which picks out the pattern repetition in sequence, and shows the symmetry of the program and of the components of the drawing (for a similar example see the unit square in Figure 3: ‘draw a side and turn the corner’) that it represents.

- ❖ An interpretation of the code as *a physical characteristic*, for example, *a kind of motion*, or *drawing a shape*, helps to ‘**crack the code**’.

**THE REPEAT CONTROL STRUCTURE IN UNPLUGGED PROGRAMMING**

In our UPL toolbox, we have a **repeat** control structure, (as in most programming languages), which enables us to take advantage of repeat patterns of code in sequence, by replacing the repetitive code with just **one repeat** instruction from our toolbox. So far, if we take `program (1)`, we have rewritten it in the form `program (1a)`, which makes clear what code is repeated, and how many times.

```
fd2 lt lt fd2 lt lt.....program (1)
```

```
fd2 lt lt
fd2 lt lt.....program(1a)
```

We take the repeated pattern of code and put it into our UPL toolbox **repeat** control structure as follows in the form **program (1b)** .

```
repeat 2[fd2 lt lt].....program(1b)
```

In UPL, (Unplugged Programming Language), when ‘Walking the Talk’ we might say:

```
"Repeat twice (pause), forward2--left-turn--left-turn,
(pause)"
```

In UPL, when “Chalking the talk”, the repeat instruction is written with the repeat pattern enclosed in square brackets [ ], and is obeyed 2 times for **program(1b)** or 4 times for **program(2b)** , see below, (or as many times as the number after the **repeat** instruction) in sequence so that **program(1)** , **(1a)** and **(1b)** are all exactly equivalent programs.

Similarly, in the case of the square,

```
fd2 lt fd2 lt fd2 lt fd2 lt.....program(2)
```

we may rewrite as

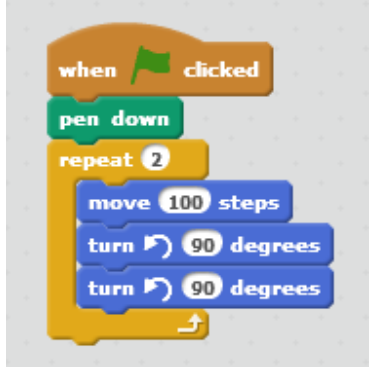
```
fd2 lt
fd2 lt
fd2 lt
fd2 lt.....program(2a)
```

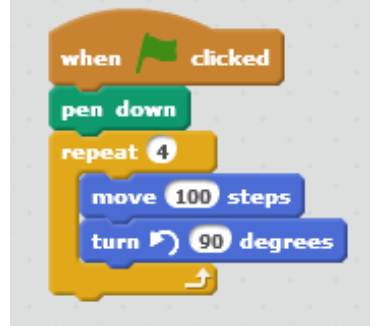
and finally **program(2)** becomes:

```
repeat 4[fd2 lt].....program(2b)
```

and **program(2)** , **(2a)** and **(2b)** are exactly equivalent programs.

- ❖ We can use the **repeat** structure in our ‘Walk the talk’, if we make it clear what is to be repeated (included in the repeat square brackets --- perhaps by intonation) and how many times. In the examples below, we show the relationship between the repeat structure in all three languages. The process of algorithmic thinking, and coding is essentially the same in each languages. Any program we have written so far, can easily be transcribed to Scratch, Python, Coffeescript or Logo. Being able to manage the necessary familiarity with the Scratch and/or Python programming environments is the key to when to make the transition to a screen language.
- ❖ The match of UPL to Scratch and Python showing the equivalent repeat structure in the corresponding programs are shown in Figures 1 and 2. The full transition to Scratch 2.0 can be undertaken at any time with the help of Workbook 6.

<pre>repeat 2[fd2 lt 1t]</pre>		<pre>from turtle import * # for i in range(2):     fd(100)     lt(90)     lt(90)</pre>
<p><i>UPL program(1b)</i></p>	<p><i>Scratch program(1b)</i></p>	<p><i>Python program (1b)</i></p>
<p align="center"><b>Figure 1. Repeat Programming Structure</b></p>		

<pre>repeat 4[fd2 lt]</pre>		<pre>from turtle import * # for i in range(4):     fd(100)     lt(90) </pre>
<p><i>UPL program(2b)</i></p>	<p><i>Scratch program (2b)</i></p>	<p><i>Python program (2b)</i></p>
<p align="center"><b>Figure 2. Repeat Programming Structure</b></p>		

- ❖ In the mapping of a program in UPL to Scratch and Python a pace in UPL is equivalent to 50 pixels on the screen in Scratch or Python so `fd2` in UPL maps to `fd(100)` in Python and the ‘move 100 steps’ block in Scratch.

**MISSION 1 TRACKING AND STACKING THE CODE**

**WHAT'S IN OUR TOOLBOX?**

1. **forward1: fd1**  
 our pet/robot goes forward 1 pace, drawing a trail as it goes. On squared paper a pace is the length of the side of a square. We can make it go forward any number of paces e.g.  
**forward3: fd3** our pet/robot goes forward 3 paces.
2. **left turn: lt**  
 our pet/robot turns left through 90 degrees -- just that, no movement forward
3. **right turn: rt**  
 our pet/robot turns right through 90 degrees -- just that, no movement forward
4. **repeat 2[code]**  
 our pet/robot executes the code in the square brackets [ ] the number of times (2) stated.

**EXPLANATION: SYMMETRY IN THE CODE AND IN THE DRAWING**

A repeating pattern in the code may give rise to symmetry in the code, which may reflect symmetry in the way the drawing was produced by the code.

<pre>fd1 lt fd1 lt fd1 lt fd1 lt  repeat 4[fd1 lt]</pre>	
<p>Code for unit square</p>	<p>Building block: the unit square (Return program)          Representation of the four symmetric repeats of [fd1 lt]</p>
<p><i>Figure 3. Symmetry in the code and symmetry in the 'action geometry' of drawing</i></p>	

Look for repeated patterns of code in sequence for each program. (not all the programs have repeat patterns) If you find a repeated pattern in sequence, stack the programs -- see program (1a) and 2(a) above. For this to be possible, the repeat patterns -- the longer the better (3 or more instructions) -- must be next to each other in the code and in sequence.

1. In **program (3)** and **program (4)** below, stack the programs and label them as **program (3a)** and **program (4a)**. Complete the drawing for each program and crack the code. Are they RETURN programs?

`fd1 lt fd1 rt      fd1 lt fd1 rt.....program(3)`

`fd1 lt fd1 lt fd1      fd1 lt fd1 lt fd1.....program(4)`

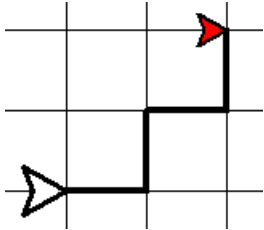
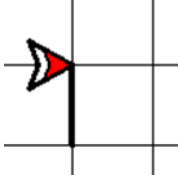
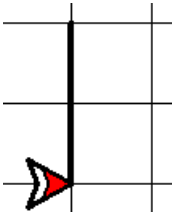
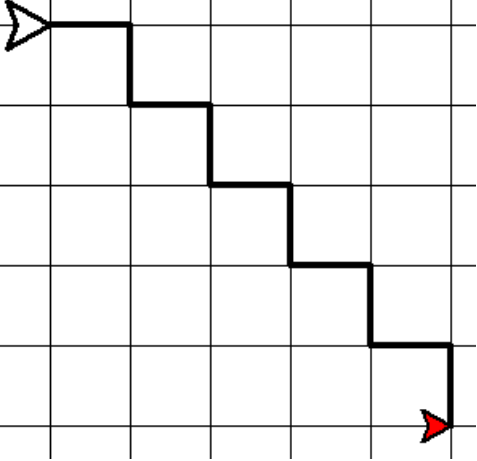
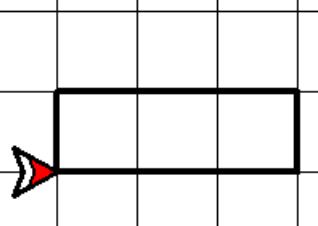
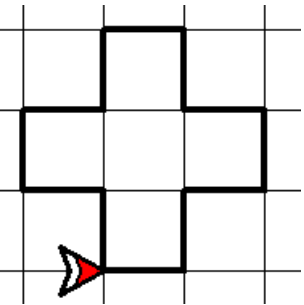
2. Write the code as a repeat statement in **program (3a)** and **program (4a)** from question 1, and label your programs **program (3b)** and **program (4b)** respectively.
3. Match the stacked code in Table 4 with the drawings in Table 5 and put the results in Table 3
4. Write the `repeat` statement for each program in Table 4

Table 3						
<b>Table 4</b>	<b>1</b>	<b>2</b>	<b>3</b>	<b>4</b>	<b>5</b>	<b>6</b>
<b>Table 5</b>						



<i>Table 4. Repeat statements from the stacked code</i>	
<i>Stacking the Code</i>	<i>Repeat Statement</i>
1	<pre>fd1 lt fd1 rt fd1 lt fd1 rt</pre>
2	<pre>rt fd1 rt rt fd1 rt</pre>
3	<pre>lt fd2 lt lt fd2 lt</pre>
4	<pre>fd1 rt fd1 lt fd1 rt fd1 lt fd1 rt fd1 lt fd1 rt fd1 lt fd1 rt fd1 lt</pre>
5	<pre>fd1 lt fd1 rt fd1 lt fd1 lt fd1 rt fd1 lt fd1 lt fd1 rt fd1 lt fd1 lt fd1 rt fd1 lt</pre>
6	<pre>fd3 lt fd1 lt fd3 lt fd1 lt</pre>

*Table 5. Line drawings with repeat patterns*

		
A	B	C
		
D	E	F

**MISSION 2 READ AND CRACK THE REPEAT CODE**

1. 'Walk the talk' and so crack the code in the repeat instructions in Table 6.

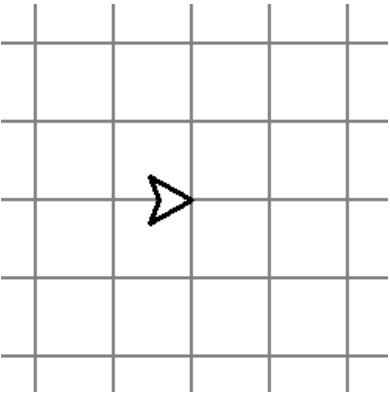
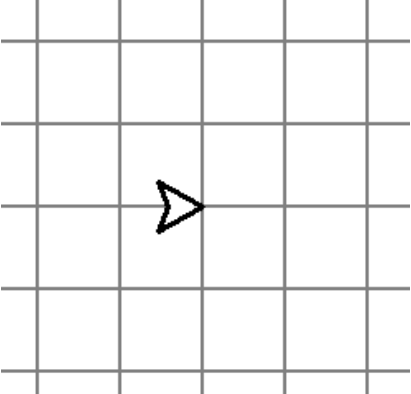
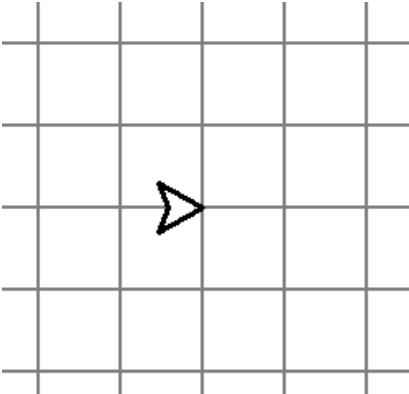
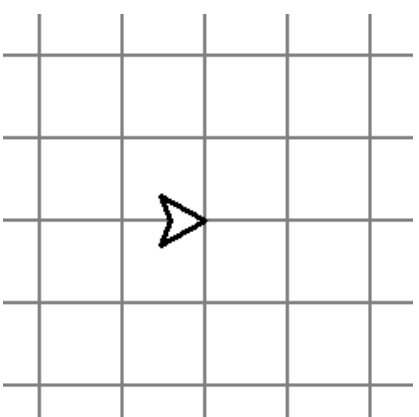
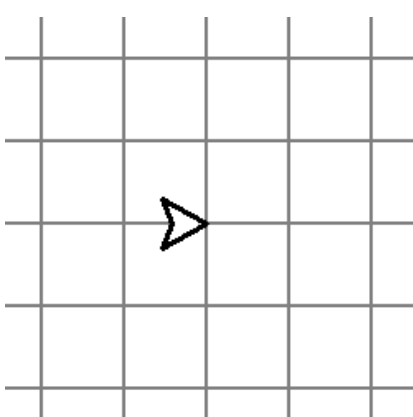
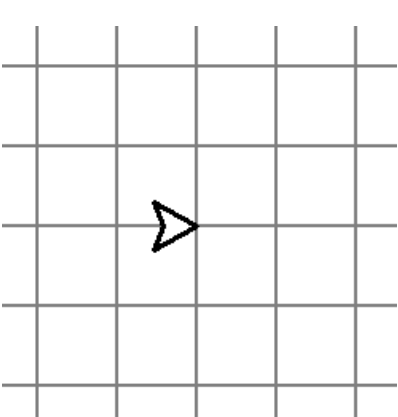
<i>Table 6. Read and Crack the repeat UPL programs</i>		<i>RETURN Program ?</i>
1	<code>repeat 4[fd2 lt]</code>	
2	<code>** repeat 2[fd1 lt fd1 lt fd1 lt fd1 lt lt]</code>	
3	<code>repeat 4[lt]</code>	
4	<code>repeat 2[fd3 lt fd1 lt]</code>	
5	<code>repeat 2[fd2 lt lt]</code>	
6	<code>** repeat 4[fd1 lt fd1 lt fd1 lt fd1 lt lt]</code>	
7	<code>** repeat 2[fd1 lt fd1 lt fd1 lt fd1 lt rt]</code>	
8	<code>** repeat 3[ fd1 lt fd1 lt fd1 lt fd1 lt rt]</code>	
9	<code>** repeat 4[fd1 lt lt fd1 rt]</code>	
10	<code>** repeat 2[fd1 lt fd1 rt fd1 rt fd1 lt fd1 lt fd1 lt]</code>	
11	<code>** repeat 2[fd1 rt fd1 rt fd1 rt fd1 lt]</code>	
12	<code>** repeat 2[fd1 lt fd1 lt fd1 lt fd1 rt]</code>	
13	<code>** repeat 2[fd1 lt fd1 lt fd1 lt fd1 lt fd1]</code>	
14	<code>** repeat 3[fd1 lt fd1 lt fd1 lt fd1 lt lt]</code>	
15	<code>**** repeat 4[repeat 4[fd2 lt] lt]</code>	
16	<code>**** repeat 2[lt repeat 4[fd1 lt]</code>	
17	<code>**** repeat 4[repeat 2[fd2 lt lt] lt]</code>	
18	<code>*** repeat 2[fd1 lt fd2 lt]</code>	

\*difficulty grading

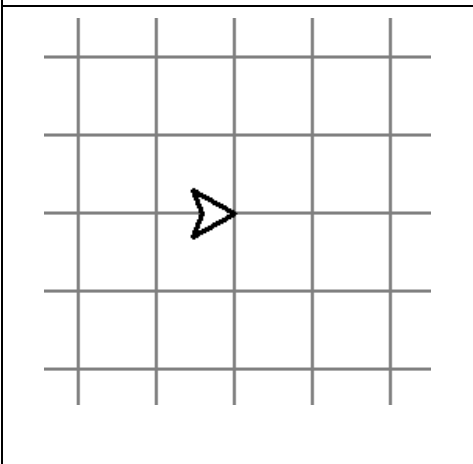
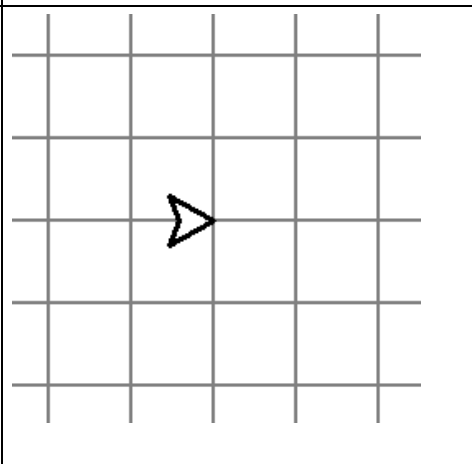
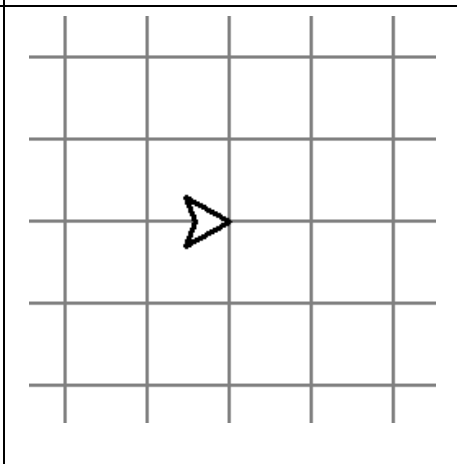
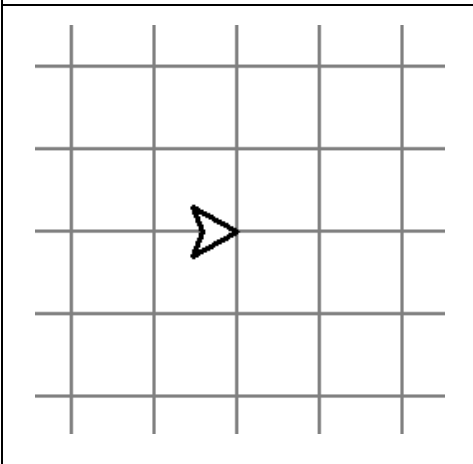
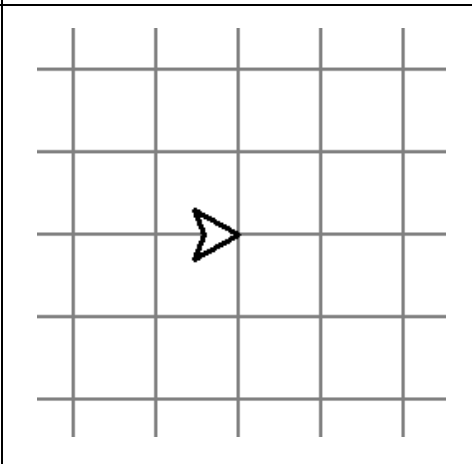
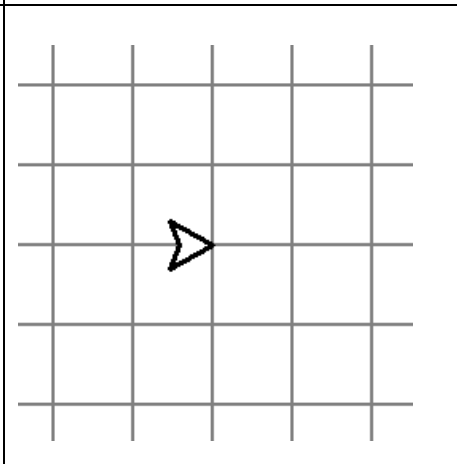
### MISSION 3 CRACK THE CODE BY DRAWING

1. Draw the figures for the programs in Table 7, and so *Crack the Code*
2. Make sure you record for each drawing:
  - A starting point and direction
  - A path
  - A finishing point and direction
3. State for each program if it is a RETURN program.

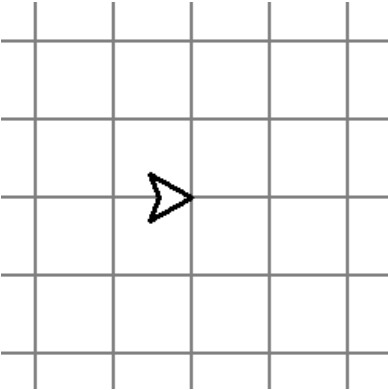
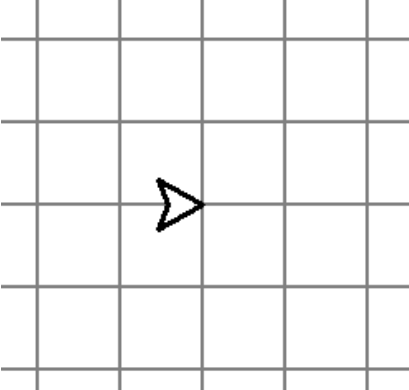
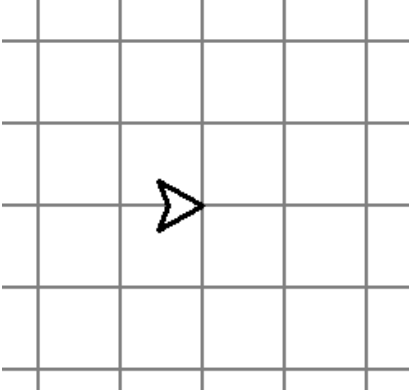
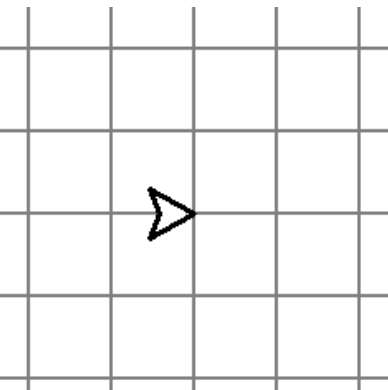
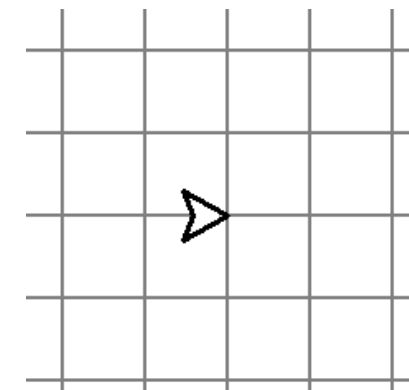
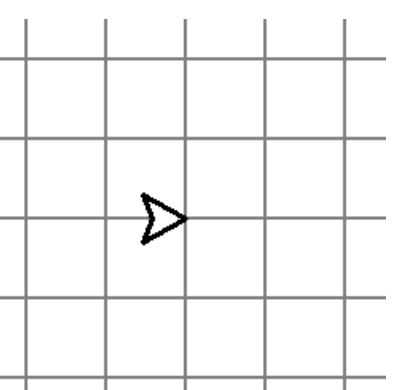
*Table 7: Read, Track and Crack the Code*

<pre>repeat 4[fd2 lt]</pre>	<pre>** repeat 2[fd1 lt fd1 lt fd1 lt fd1 lt lt]</pre>	<pre>repeat 4[lt]</pre>
		
<p>1</p>	<p>2</p>	<p>3</p>
<pre>repeat 2[fd3 lt fd1 lt]</pre>	<pre>repeat 2[lt fd2 lt]</pre>	<pre>** repeat 4[fd1 lt fd1 lt fd1 lt fd1 lt lt]</pre>
		
<p>4</p>	<p>5</p>	<p>6</p>

*Table 7: (cont'd) Read, Track and Crack the Code*

<p style="text-align: center;">**</p> <pre>repeat 2[fd1 lt fd1 lt fd1 lt fd1 lt rt]</pre>	<p style="text-align: center;">**</p> <pre>repeat 3[fd1 lt fd1 lt fd1 lt fd1 lt rt]</pre>	<p style="text-align: center;">**</p> <pre>repeat 4[fd1 lt lt fd1 rt]</pre>
		
7	8	9
<p style="text-align: center;">**</p> <pre>repeat 2[fd1 lt fd1 rt fd1 rt fd1 lt fd1 lt fd1 lt]</pre>	<p style="text-align: center;">**</p> <pre>repeat 2[fd1 rt fd1 rt fd1 rt fd1 lt]</pre>	<p style="text-align: center;">**</p> <pre>repeat 2[fd1 lt fd1 lt fd1 lt fd1 rt]</pre>
		
10	11	12

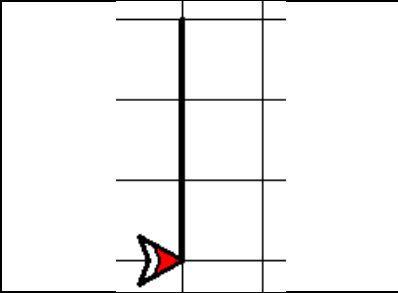
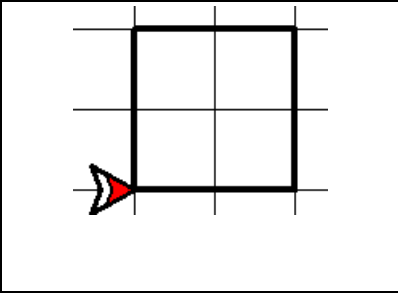
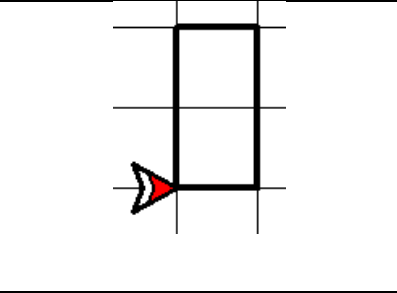
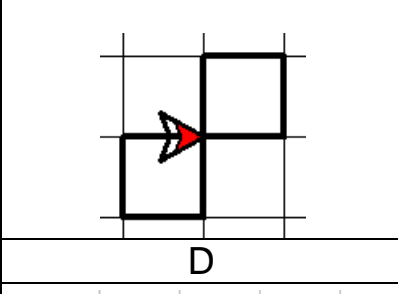
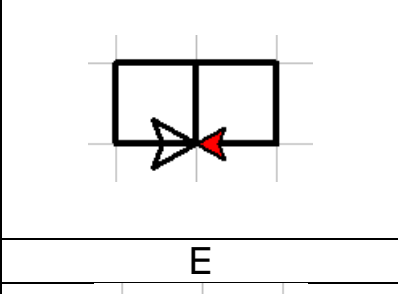
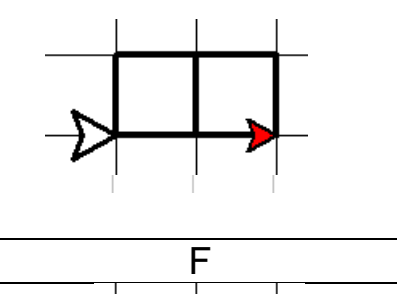
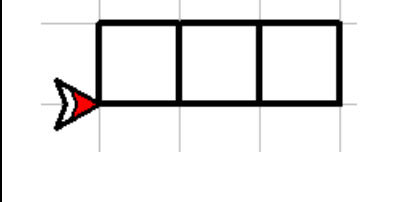
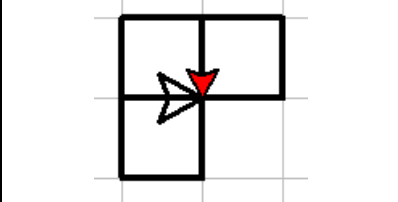
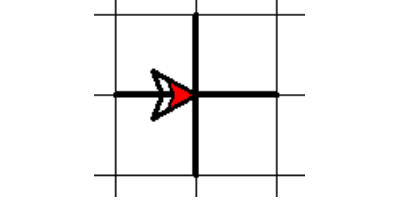
*Table7(cont'd): Read, Track and Crack the Code*

<p><b>**</b>                      repeat 2[fd1 lt fd1 lt                      fd1 lt fd1 lt fd1]</p>	<p><b>**</b>                      repeat 3[fd1 lt fd1 lt                      fd1 lt fd1 lt lt]</p>	<p><b>***</b>                      repeat 2[fd1 lt fd2                      lt]</p>
		
<p><b>13</b></p>	<p><b>14</b></p>	<p><b>15</b></p>
<p><b>repeat 2[fd2 lt lt]</b></p>	<p><b>****</b>                      repeat 4[repeat 2[fd2                      lt lt] lt]</p>	<p><b>****</b>                      repeat 4[repeat 2[fd2                      lt lt] lt]</p>
		
<p><b>16</b></p>	<p><b>17</b></p>	<p><b>18</b></p>

**MISSION 4: MATCH YOUR DRAWINGS**

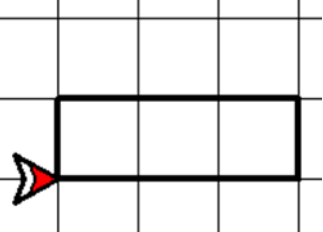
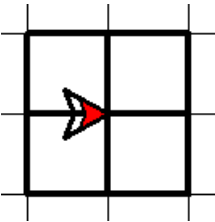
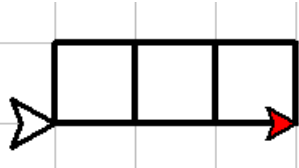
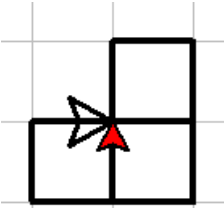
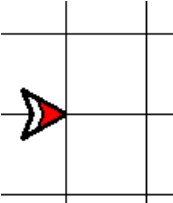
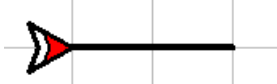
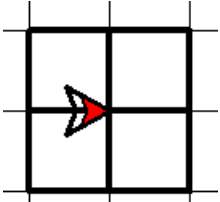
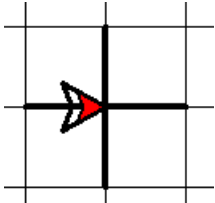
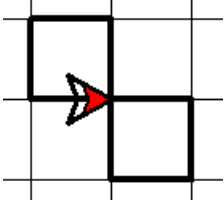
- Compare and match your drawings for Table 7 with the line drawings in Table 9 and enter the results in Table 8. For example, the repeat program in Table 7(2) produces the drawing in Table 9(C).

<i>Table 8.</i>																		
Table 7	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
Table 9		E																

<i>Table 9. Drawings for repeat programs</i>		
		
A	B	C
		
D	E	F
		
G	H	I

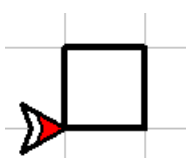
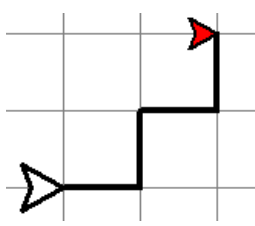
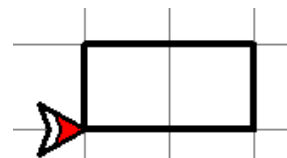
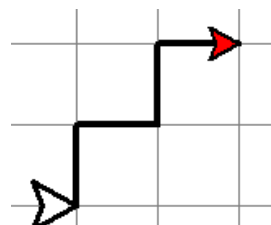
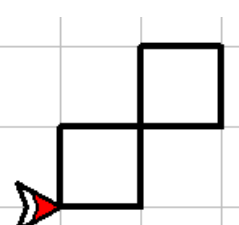
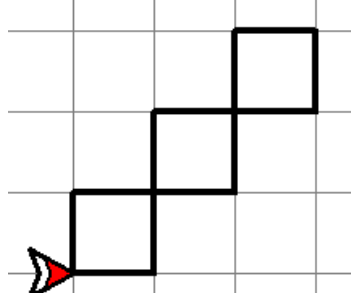
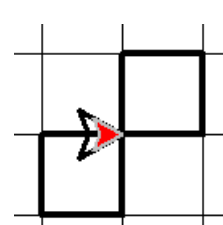
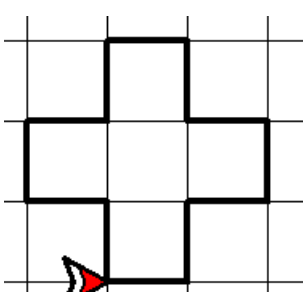
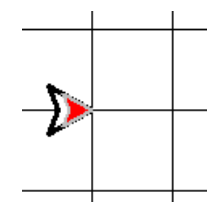


*Table 9 (cont'd) Drawings for repeat programs*

		
J	K	L
		
M	N	O
		
P	Q	R

MISSION 5 WRITING PROGRAMS TO PRODUCE DRAWINGS

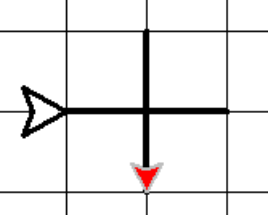
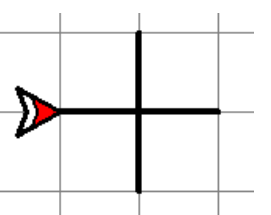
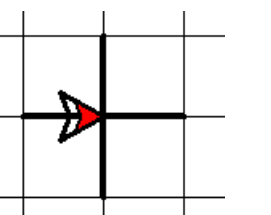
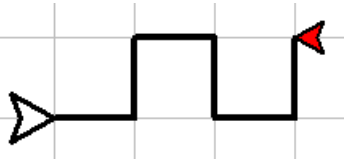
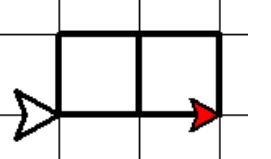
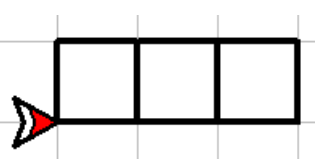
**Table 10. Drawings which can involve repetitive patterns**

		
1	2	3
		
4	5	6
**	**	***
		
7	8	9

- Write programs to produce the drawings in Table 10. Take note of the repetition in the drawing and in your code. Build a program that uses as much repeated pattern as you can. Stack the repeat patterns (the longer the better) and rewrite your programs with repeat statements. Some drawings can be done in more than one way. (Drawings graded as 2\* or more will require some thinking).
- \* Some of the programs, which produce the drawings in Tables 9 and 10, are not RETURN programs. Add instructions to the end of these programs to bring the red arrowhead back to the starting point of the large white arrowhead and pointing in the same starting direction to make the programs into RETURN programs.



MISSION 6 WRITING PROGRAMS TO FIT THE DRAWINGS

<i>Table 11. Plus sign, row of squares: different programs, symmetry</i>		
		
1	2	3
		
4	5	6

1. \*\*\*Write a program in UPL for the drawing of a plus sign in Table 11, starting with 11.1  
 Whichever point you start from, your pet/robot starting direction is to the right → and your program takes it from there.
2. \*\*\*Make your program into a RETURN program.
3. \*\*\*Take your starting point in 11.2 and draw the plus sign by turning left at the centre each time you approach it. Write a program to draw the plus sign using the symmetry of the code to make a repeat instruction. And make your program into a RETURN program.
4. \*\*\* Try a similar approach to program drawing 11.3
5. \*\*\*\* In 11.4 write the code to draw the figure. Then put the code in a repeat loop to repeat the code twice. Crack the Code for the repeat instruction.
6. \*\*\*\* In 11.5, Write a program for the drawing. Symmetry? Is there a repeat pattern? Could you program to add a third square in the row? And then turn it into a Return program as in 11.6

## MISSION 7 PATTERNS, SYMMETRY, ALGORITHMS

- ❖ We have seen how symmetry in drawings is reflected in repeat patterns in the code which generates them. And how often this is related to RETURN programs. In this next mission, we examine patterns in general and the idea of an algorithm which produces the pattern. Later in the Course we shall be trying to recognise and identify patterns and symmetry in our coding to discover and explore properties of regular figures in our 'action geometry' and in pictures that we produce from our programs.

Identify the pattern in the following examples:

1) Replace the underlines with letters to make English words:

a) \_ \_ \_ M \_ T \_ Y

b) P \_ \_ T \_ R \_ \_

These are patterns that might be found in a crossword. Search this page to find words that fit the crossword clues a) and b).

2) What are the next 2 numbers in the following sequences?

a) 1 2 3 4 5 6 ...

b) 2 4 6 8 10 12 ...

c) 1 3 5 7 9 11 ...

d) \*0 2 4 6 8 10 ...

e) 4 8 12 16 20 24 ...

f) 5 9 13 17 21 25 ...

g) 1 4 9 16 25 36 ...

h) \*3 5 7 9 11 13 ...

i) \*\*3 6 12 24 48 96 ...

j) \*\*2 4 8 16 32 ...

k) \*\*1 3 7 15 31 ...

WHAT'S THE ANSWER? AND CAN YOU SAY HOW YOU DID IT

EXPLANATION: WORKING IT OUT, ALGORITHMIC THINKING

- ❖ Can you say how you worked out the answer in each case? That's a start to --- **algorithmic thinking**.

What is the secret 'code' for, or 'way of working out', the underlying pattern in each sequence? Saying how you worked it out is sometimes more difficult than the actual working it out, but making this explicit is the process of devising an algorithm that may serve as a basis for a program to perform the task.

So in a) we might say:

Start with 1, and that is the first number. The next number you get by adding 1 to the previous number and so on ... This is part of the set of integers or counting numbers that we use every day.

In b) we might say:

Start with the counting numbers 1, 2, 3, 4, 5 ... multiply each by 2 to get the sequence in b)

$$\begin{aligned} 1 \times 2 &\rightarrow 2 \\ 2 \times 2 &\rightarrow 4 \\ 3 \times 2 &\rightarrow 6 \\ 4 \times 2 &\rightarrow 8 \\ &\dots \end{aligned}$$

This looks like the 2 times table! What we are trying to do is recognise how we get from 1, 2, 3, 4,... to the sequence b) 2, 4, 6, 8, ... which of course is the *operation* of the 2 times table.

In effect, we are taking the numbers 1, 2, 3, 4, ... and doing the same thing to each one to produce another sequence. This *doing the same thing to each number* is the **algorithm** that we are working out which produces this new sequence b).

- 3) In question 2), the second number in each sequence is: b) 4, c) 3, d) 2, e) 4, f) 5, g) 4, h) 5, i) 6, j) 4, k) 3. How do you 'get' these second numbers from 2, (the second number in 1, 2, 3, 4, ...)?  
 We will find a different algorithm for each sequence.

- 4) What is the 4<sup>th</sup> number in each sequence? How do you get these numbers from 4?  
 5) In b) you do the same thing to 2 as you do to 4. You multiply them both by 2 to get the numbers in b). So what is the 7<sup>th</sup> number in each sequence a) to k). And what did you do to the 7 for each answer.

EXPLANATION: THERE'S MORE THAN ONE WAY TO DO IT

In trying to solve a problem in computing, we will often come up with more than one way of tackling a problem --- we call it **algorithmic thinking**. Deciding which algorithm/program is better or more effective, and how well it does the job, we call **evaluation**.

- 6) \* In question 1, there is more than one solution that satisfies the crossword question as it is asked. Can you find another solution?

## MISSION 8 GENERALISING RESULTS

### EXPLANATION \*\* THE PROCESS OF GENERALISING

1) \*\* **generalising**: In a) The 5<sup>th</sup> number is 5, the 7<sup>th</sup> number is 7, we can say what any number in the sequence is, because we know *how to work out* the numbers that make up the pattern in the sequence. We start with 1 as the first number in the sequence and then add 1 to get the next number, and add 1 to that to get the next number and so on...

In b) The 5<sup>th</sup> number is 10 --- how does 10 come from 5?

The 7<sup>th</sup> number is 14 --- how does 14 come from 7?

Is the *way* that we get 10 from 5 the same as the *way we* get 14 from 7?

If so, then **this way** is possibly the algorithm. We may need to test it with another number to see if it works for more items in the sequence.

What are the 5<sup>th</sup> and the 7<sup>th</sup> numbers in sequences c) to h)? How did you do it? Describe the algorithm for each sequence.

### EXPLANATION \*\*\* A DIFFERENT SORT OF SEQUENCE

2) In the next examples, what are the next two numbers in the sequence? Is there a pattern? Can you describe how you work out the next numbers in the sequence? If you can say how you do it then you have an algorithm for the sequence. The algorithms are perhaps not as straightforward to work out as our previous examples.

i. \*\*\* 0 2 1 3 2 4 3 5 ...

ii. \*\*\* 1 1 2 3 5 8 13 ...

The sequence in ii) is called the **Fibonacci** sequence. We shall be making use of the Fibonacci sequence and some of its applications in the real world, when we are looking at pictures and regular shapes in 'action geometry' later in the Course.

### EXPLANATION\*\*\*\* GENERALISING IN SIMPLE ALGEBRA

3) \*\*\*\* Another way to describe the algorithm is in a form from algebra where we would say:

In a) the *r*<sup>th</sup> number is *r*,

In b) the *r*<sup>th</sup> number is: double *r*, or 2 times *r*, or 2 multiplied by *r*, or we may say the *r*<sup>th</sup> number is  $2 \times r$  or  $2r$ . (These are just different ways of saying or writing the same thing).

Express in terms of *r* as in b) the *r*<sup>th</sup> number in sequences c) to k)?

4) \*\*\*\* Write the first 5 numbers in the following sequences where the algorithm to work out the *r*<sup>th</sup> number is described in an algebraic form as:

a.  $r+1$  ( $1+1=2, 2+1=3, \dots$ ) => 2, 3, 4, 5, 6...

b.  $3+r$

c.  $2 \times r - 1$

d.  $2 \times r + 3$

e.  $r \times r - 1$

\*is the difficulty rating.