The other art of computer programming. Milestone 2: LOGO.
1960s

Melanie Tarr
Edith Cowan University

Follow this and additional works at: https://ro.ecu.edu.au/ecuworkspost2013
Part of the Computer Sciences Commons, and the Education Commons

10.25958/5e3a746237785
This Book is posted at Research Online.
https://ro.ecu.edu.au/ecuworkspost2013/7443
Learn how Papert programmed a robot called a ‘turtle’

Inventing turtle graphics
Tell which shape is programmed by looking at a few lines of code

Program patterns in LOGO
Use code to make a Koch snowflake and the Hilbert pattern

LOGO
The Other Art of Computer Programming
by Melanie Tarr

1960s
ENGLISH MATHEMATICIAN SEYMOUR PAPERT CREATED AN EMBODIED UNDERSTANDING OF MATHEMATICS THROUGH THE LOGO LANGUAGE. THIS LANGUAGE, UNDERSTOOD BOTH BY HUMANS AND COMPUTERS, DESCRIBES DRAWINGS NOT IN AN ABSOLUTE CARTESIAN GRID, BUT FROM THE PERSPECTIVE OF THE PERSON THAT IS DRAWING. HE CALLED THIS TURTLE GRAPHICS...
# Table of Contents

Turtle graphics .......................................................... 3  
Letters and drawing ..................................................... 4  
Procedures ........................................................................ 5  
Recursion ........................................................................... 6  
States .................................................................................. 7  
Abstraction .......................................................................... 8  
Gestalt ................................................................................. 12  
Recursion ............................................................................. 13  
Koch snowflake ................................................................. 14  
Hilbert pattern ................................................................. 18  

Note: This resource may be useful for the above DT curriculum codes and associated elaborations in the Australian National Curriculum.

- ACTDIK015
- ACTDIK024
- ACTDIP030
- ACTDIK024
The Logo language, created by Seymour Papert, embodied an understanding of mathematics through its use of a simulated turtle. This turtle did not draw in an absolute Cartesian grid, but from the perspective of the person drawing. Papert called this turtle graphics.

The embodied robot took commands from the operator and performed line art with its own form of geometry.

**2 kinds of turtles**

Not everyone could afford a robot turtle so a simulated turtle was used as well. The simulated turtle appeared in the middle of the computer screen.

**The digital turtle**

A turtle rotated through a number of steps to draw a circle. The number of steps was 360.

The turtle line is a turtle trail.

The turtle has a number of steps it moves forward.

The mechanical turtle

Using a small tabletop computer, people gave the turtle commands to make it go forward, back, left, and right, moving over a big piece of paper, drawing pictures as it moved along.

The turtle trails are messy, and the clear-screen command would clean the screen.

**1960s Milestone**

Seymour Papert created an embodied understanding of mathematics through his Logo language. This language, understood both by humans and computers, describes drawings not in an absolute Cartesian grid, but from the perspective of the person who is drawing. He called this turtle graphics.
making letters

WHICH TRAIL IS PRODUCED WHEN THE PREVIOUS CODE IS EXECUTED?

A

B

start

THE ABOVE CODE PRODUCES THE LETTER "N"

turtle code words were called primitives. They could also be shortened, for example,

forward 5
right 90
forward 5

DRAWING AND NOT

sometimes it is important for the turtle not to draw

penup (pu)

so lift the pen up

pen down (pd)

then down

MAPPING

1960s milestone two
Logo procedures are instructions you teach Logo to do. They include all the steps Logo must take to make something happen.

The following example is one that you cannot program in LOGO however it is still a procedure.

The following example are steps that do convert into steps you can program or automate.
recursion and steps to making a shape

visualising things that repeat, iterate or recurse

gestalt - closure

Closure is a psychological principle that says to our brain, “The square is drawn.”

a computer can not see

It doesn't know when to stop so it could keep drawing the square forever
**states of mind**

eg. I’m in a great mood so I will visit the beach

I’m feeling tired so I will have a sleep

When people are in different states of mind, they exercise different judgements.

---

**states in computer science**

In computer science, states work in a similar way.

If a machine is in a certain state (1), it will execute a set of instructions.

If a machine is in another state (2), it will execute a different set of instructions.

---

**so to program a square**

We need to draw four lines and turn at a 90 degree angle each time.

```
TO SQUARE
  REPEAT 4 [FC 20 FT 90]
END
```

---

**a state in logo**

Every machine has a start and an ending state. In this example, it counts to 4 and then stops.

Every machine has a start and an ending state. It stops after repeating the corner procedure 4 times.

An ending condition satisfies the machine and it stops after repeating the corner procedure 4 times.
The essence of computational thinking is abstraction. In computing, we abstract notions beyond the physical dimensions of time and space. Our abstractions are extremely general because they are symbolic.

One of the first examples of abstraction was Alan Turing’s universal machine.
**ABSTRACTING A CHECKERBOARD**

**TO DRAW 8 SQUARES IN A COLUMN YOU COULD CODE**

REPEAT 8 
(SQUARE FD 20)

**TO TURN RIGHT AT THE FIRST COLUMN TO START DRAWING THE SECOND COLUMN**

RT 90 FD 20 RT 90

**THE SQUARE PROCEDURE IS AN ABSTRACTION THAT DRAWS A COLUMN**

90

**MORE ABSTRACTION**

**TO ABSTRACT A COLUMN PROCEDURE**

REPEAT 8 [FD 20 RT 90] 
RT 90 FD 20 RT 90

**TO COLUMNS**

REPEAT 8 [SQUARE FD 20] 
RT 90 FD 40 RT 90
REPEAT 8 [SQUARE FD 20] 
LT 180 END

**TO CHECKERBOARD**

REPEAT 4 [COLUMNS] END

---

**shapes**

This next section shows the programming of a flower drawing. The programming language in this comic is called LOGO and was created for children in the 1960s. Although LOGO can seem very basic, it is also powerful enough to use in Artificial Intelligence Systems. The instruction panels should be read down from picture to egression to algorithm to code. LOGO was written by Marvin Minsky and Seymour Papert. Marvin Minsky is also known as the father of Artificial Intelligence.

**IDEA OR DESCRIPTION**

algorithm/pseudocode

code

**TO CONSTRUCT A CIRCLE**

1. move forward once and left once
2. repeat 360 times

**REPEAT 360 [FD 1 LT 1]**

1. move forward twice and left once
2. repeat 360 times

FD 1

LT 1

---

1960s Milestone two

Melanie Tarr
1. move forward once and left once
2. repeat 90 times

```
REPEAT 90 [FD 1 LT 1]
```

Now go back the other way

Hint: this is just part of a circle

1. move forward once and left once
2. repeat 90 times
3. turn left 90
4. move forward once and left once
5. repeat 90 times

```
LT 90
REPEAT 90 [FD 1 LT 1]
```

TO LEAF
```
REPEAT 2 [REPEAT 90 [FD 1 RT 1] RT 90]
```
End

To construct an arc, hint this is just part of a circle

1. move forward once and left once
2. repeat 90 times

```
REPEAT 90 [FD 1 LT 1]
```

1. run the leaf pattern
2. turn the turtle increasing it by 360/24=15 degrees
3. repeat 24 times

```
TO LEAF
REPEAT 2 [REPEAT 90 [FD 1 RT 1] RT 90]
END
```

```
TO CRYSANthemUM
REPEAT 24 [LEAF RT 15]
END
```

1. turn 180 degrees
2. forward 200 steps

```
TO STEM
RT 180
FD 200
END
```

```
TO LEAVES
REPEAT 2 [LEAF LT 90]
END
```

1. turn 180 degrees
2. forward 200 steps

```
TO STEM
RT 180
FD 200
END
```

```
REPEAT 6 [FD 100 RIGHT 360/6]
```

```
REPEAT 360 [FD 1 LT 1]
```

```
RT 30
REPEAT 3 [FD 122 RT 120]
LT 120
REPEAT 3 [FD 122 RT 120]
```

Circle the shape produced by the code.
gestalt - parts of a whole

The unified whole is greater than the sum of its parts.
Recursion in computer science is when a procedure calls itself as part of the solution through recursion we discover the emergence of pattern.

To draw a picture, we would like to draw a picture of a pattern.

What steps repeat?

1. Draw a line forward a number of steps
2. Turning 90 degrees to start the next line
3. Add 5 steps to the number of steps

We also need to increase the line length by 5 steps each time we draw it, so we need to add another command in that adds 5 to the step number. This would be step 3 so our instructions to Logo now reads:

1. Drawing a line forward a number of steps
2. Turning 90 degrees to start the next line
3. Add 5 steps to the number of steps.

Logo remembers the number of steps it takes forward from the variable N. Variables are covered in chapter 10. The person operating the program inputs the first number of steps, then Logo generates the rest by adding 5 each repeat to the number of steps. The command:

```
1. FD :N
2. RT 90
3. :N +5
```

To convert instructions to code:

```
procedure name :<arguments>
statements
.
.
repeat procedure
end
```

The format for this recursive procedure is as follows:

```
TO MAZE :N
FD :N RT 90
MAZE :N +5
END
```
Koch Snowflake

1. Start with a line.
2. Divide the line into 3 equal parts.
3. Draw an equilateral triangle using the middle segment as a base.
4. Erase the base of the equilateral triangle.
5. Repeat steps 2 - 4 until you have finished the flake.

Milestone: 1960s

Melanie Tarr
```turtle
[repeat 3 [rt 120 side :size :level]]
  ( side 100 1 )
  [if :level = 0 [fd :size stop]] >>>

[s side :size / 3 :level - 1 lt 60] >>>
  ( side 33.3333333333333333 0 )
  [if :level = 0 [fd :size stop]] >>>
  side stops

[s side :size /3 :level - 1 lt 60] >>>
  side stops
```
[if :level = 0 [fd :size stop]] >>> side stops
(side 100 1)

[if :level = 0 [fd :size stop]] >>> side stops
(side 33.33333333333 0)

[if :level = 0 [fd :size stop]] >>>
(side 100 1)
[if :level = 0 [fd :size stop]] >>>
(side 33.33333333333 0)
[if :level = 0 [fd :size stop]] >>>
(side 33.33333333333 0)
[if :level = 0 [fd :size stop]] >>>
(side 33.33333333333 0)
[if :level = 0 [fd :size stop]] >>>
(side 33.33333333333 0)
[if :level = 0 [fd :size stop]] >>>
(side 33.33333333333 0)
[if :level = 0 [fd :size stop]] >>>
(side 33.33333333333 0)
[if :level = 0 [fd :size stop]] >>>
(side 33.33333333333 0)
[if :level = 0 [fd :size stop]] >>>
(side 33.33333333333 0)
[if :level = 0 [fd :size stop]] >>>
(side 33.33333333333 0)
[if :level = 0 [fd :size stop]] >>>
(side 33.33333333333 0)
[if :level = 0 [fd :size stop]] >>>
(side 33.33333333333 0)
[if :level = 0 [fd :size stop]] >>>
(side 33.33333333333 0)
[if :level = 0 [fd :size stop]] >>>
(side 33.33333333333 0)
[if :level = 0 [fd :size stop]] >>>
(side 33.33333333333 0)
[if :level = 0 [fd :size stop]] >>>
(side 33.33333333333 0)
[if :level = 0 [fd :size stop]] >>>
(side 33.33333333333 0)
[if :level = 0 [fd :size stop]] >>>
(side 33.33333333333 0)
[if :level = 0 [fd :size stop]] >>>
(side 33.33333333333 0)
[if :level = 0 [fd :size stop]] >>>
(side 33.33333333333 0)
[if :level = 0 [fd :size stop]] >>>
(side 33.33333333333 0)
[if :level = 0 [fd :size stop]] >>>
(side 33.33333333333 0)
[if :level = 0 [fd :size stop]] >>>
(side 33.33333333333 0)
[if :level = 0 [fd :size stop]] >>>
(side 33.33333333333 0)
[if :level = 0 [fd :size stop]] >>>
(side :size / 3 :level - 1 lt 60] >>>
(side 33.333333 0)
[if :level = 0 [fd :size stop]] >>>
side stops

[sid :side / 3 :level - 1 rt 120] >>>
(side 33.333333 0)
[if :level = 0 [fd :size stop]] >>>
side stops

[sid :side / 3 :level - 1 lt 60] >>>
(side 33.333333 0)

[sid :size / 3 :level - 1] >>>
(side 33.333333 0)
[if :level = 0 [fd :size stop]] >>>
side stops
side stops
snowflake stops

1960s Milestone tZo

Melanie Tarr
hilbert pattern

hilbert 20 2

[h :size :level 1]
  ( h 20 2 1 )
  [if :lev = 0 [stop]]
  [lt :par * 90]

rt :par * 90

fd :size

[h :size :lev - 1 :par]
  ( h 20 0 1 )
  [if :lev = 0 [stop]]
  h stops
  [fd :size]

[h :size :lev - 1 :par]
  ( h 20 0 -1 )
  [if :lev = 0 [stop]]
  h stops
  [rt :par * 90]

[h :size :lev - 1 :par]
  ( h 20 0 -1 )
  [if :lev = 0 [stop]]
  h stops
  [rt :par * 90]

[h :size :lev - 1 :par]
  ( h 20 0 1 )
  [if :lev = 0 [stop]]
  h stops
  [rt :par * 90]

[h :size :level - 1 0 - :par]
  ( h 20 1 -1 )
  [if :lev = 0 [stop]]
  [lt :par * 90]
hilbert pattern

hilbert 20 2
hilbert pattern

hilbert 20 2

[rt :par * 90]
[h :size :lev - 1 :par]
[h 20 0 1]
[if :lev = 0 [stop]]
h stops
[fd :size]

[h :size :lev - 1 :par]
[h 20 0 1]
[if :lev = 0 [stop]]
h stops
[rt :par *90 ]

[h :size :lev - 1 :par]
[h 20 0 0 -1]
[if :lev = 0 [stop]]
h stops
[lt :par *90 ]

[h :size :lev - 1 :par]
[h 20 0 -1]
[if :lev = 0 [stop]]
h stops
[fd :size]

[h :size :lev - 1 :par]
[h 20 0 0 -1]
[if :lev = 0 [stop]]
h stops
[rt :par *90 ]

[h :size :lev - 1 :par]
[h 20 0 0 -1]
[if :lev = 0 [stop]]
h stops
[lt :par *90 ]

[h :size :lev - 1 :par]
[h 20 0 0 -1]
[if :lev = 0 [stop]]
h stops
[rd :par *90 ]

[h :size :lev - 1 :par]
[h 20 0 0 -1]
[if :lev = 0 [stop]]
h stops
[rd :par *90 ]

[h :size :lev - 1 :par]
[h 20 0 0 -1]
[if :lev = 0 [stop]]
h stops
[lt :par *90 ]

[h :size :lev - 1 :par]
[h 20 0 0 -1]
[if :lev = 0 [stop]]
h stops
[rd :par *90 ]

[h :size :lev - 1 :par]
[h 20 0 0 -1]
[if :lev = 0 [stop]]
h stops
[rd :par *90 ]

[h :size :lev - 1 :par]
[h 20 0 0 -1]
[if :lev = 0 [stop]]
h stops
[lt :par *90 ]

[h :size :lev - 1 :par]
[h 20 0 0 -1]
[if :lev = 0 [stop]]
h stops
[rd :par *90 ]

[h :size :lev - 1 :par]
[h 20 0 0 -1]
[if :lev = 0 [stop]]
h stops
[rd :par *90 ]

[h :size :lev - 1 :par]
[h 20 0 0 -1]
[if :lev = 0 [stop]]
h stops
[lt :par *90 ]

[h :size :lev - 1 :par]
[h 20 0 0 -1]
[if :lev = 0 [stop]]
h stops
[rd :par *90 ]

[h :size :lev - 1 :par]
[h 20 0 0 -1]
[if :lev = 0 [stop]]
h stops
[lt :par *90 ]

[1960s Milestone two]

Melanie Tarr
hilbert pattern

hilbert 20 2

[h :size :lev - 1 0 - :par
  ( h 20 0 -1
  [if :lev = 0 [stop]
   h stops
  [fd :size]
  [rt :par * 90]
[h :size :lev - 1 :par
  ( h 20 0 -1
  [if :lev = 0 [stop]
   h stops
  [fd :size]

[lt :par * 90]
  h stops
hilbert stops