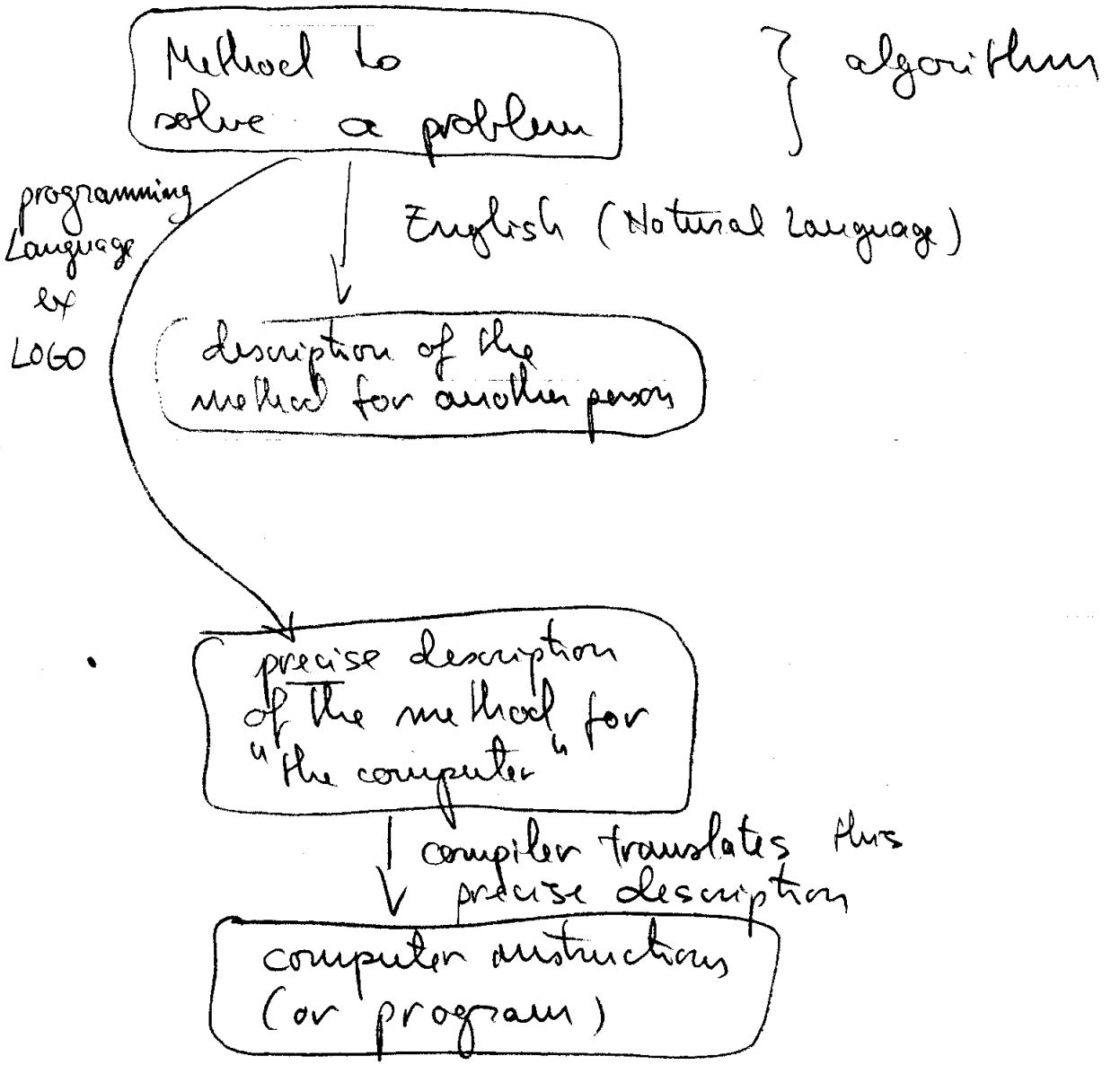
week 10

Programming – algorithmic thinking (Chapter 10)

Goal: to illustrate the concept of programming using a programming language designed for kids (LOGO).

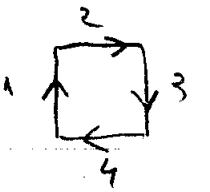


An algorithm (and thus its description) has certain characteristics / parts

- input : data to be transformed
- output : data resulting from transformation
- finiteness : algorithms must stop eventually
- definiteness :
  Algorithms = sequence of steps, of operations on the data
  Each step must be unambiguous so that if algorithm is executed another time by another agent (computer...) the outcome is the same.

Example

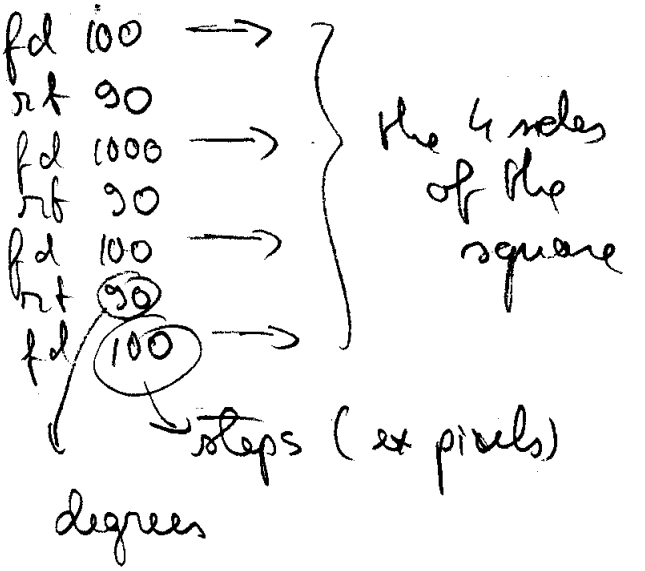
Problem : draw a square
Input : size of the square
Output : the drawing.
Algorithm
- Think of drawing 4 line segments of same length one after another :



→ We can "program" this simple algorithm easily in LOGO.

Drawing a square

Input : square size. Say 100
Output : drawing.



fd → forward
rt → right turn.

Too much writing. Use commands for iteration.

repeat 4 [fd 100 rt 90]

list of commands to be repeated 4 times.

Drawing other more complex figures?

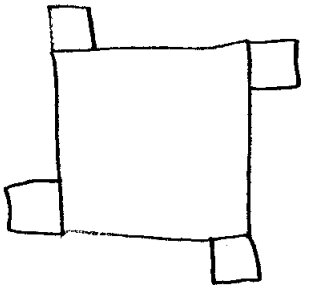


fig A

← ?

We already know to draw a square, so we can think in terms of squares, not lines.
This is called "abstraction" (see Chapter 10, textbook)

We can define new, more complex commands. Ex drawSquare. These are called "procedures" in programming language terminology.

```
to drawSquare
  repeat 4 [fd 100 rt 90]
end
```

(similar to defining a verb)

procedure name! If we want to draw a square now, we just say drawSquare.

How about different sizes of squares?

```
to drawSquareB
  repeat 4 [fd 200 rt 90]
end
```

→ impractical! We shouldn't define a new procedure for a new size.

Use variables

→ The concept of variable is how programming languages treat memory (computer) as storage for data.

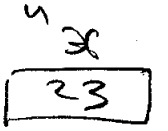
Variables ⇄ names (labels)
⇄ value.

Label in Logo = a word beginning with " (double quote character)

Variables ⇄ retrieve a value ; use command thing
⇄ set a new value ; use command make

For a variable with label "x we write thing "x to mean the value stored by "x.

(Ex) If "x has value 23, then



print thing "x * 10

will print # 230 on screen. We can write expressions using math operators +, −, *, / as in Excel. Print is a command to write on screen the value of a math expression or a label.

Shortcut The command thing "[redacted] can be entered with shortcut :[redacted]. thing "

Example print :x * 10 (outputs 230 if the value of "x is 23)

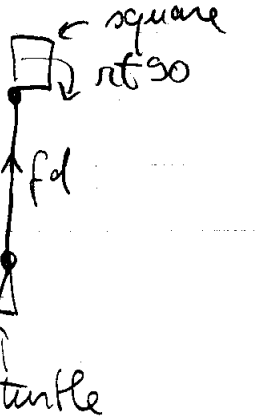
Special variables that define the input to a procedure are called parameters and written immediately after "to [redacted]:

```
to drawSquare :size
  repeat 4 [fd :size rt 90]
end
```

input to the problem of drawing a square

Drawing the shape in fig A:

repeat 4 [fd 400 drawSquare 50 rt 90]



Can we make our drawSquare procedure even more general?

→ drawing a regular n-gon

Input: – length of the edges
– number of vertices } 2 parameters

procedure name (verb) → to drawNgon :size :nvert ← parameters

```
    repeat :nvert [fd :size rt 360/:nvert]
end
```

To tell logo to make the drawing, now that the new procedure has been learned, say:

```
drawNgon 100 7
```

← this tells logo to draw a heptagon with side = to 100 steps.

The values are like objects of verb "drawNgon".
Value 100 is stored in variable "size & value 7 in variable "nvert.

Procedure definitions with parameters are like defining transitive verbs. They need one or more objects (parameters) to complete the meaning of the verb (to perform the action of the procedure).

Other things to try

a) → suppose we want to draw triangle, square, pentagon... ... until we draw 10-gon.

b) → or we want to draw 10 triangles of increasing size, from 50 to 150, in increments of 10.

We can use command "repeat" and query "repcount".

For b):

```
repeat 10 [ drawNgon 50+repcount*10 3]
```

→ use a query called "repcount". Repcount is called a query because it returns us a value. In this way, it is similar to a variable but is NOT a variable. It returns a value about the state of the logo program (ex: position of turtle, orientation, iteration # of a repeat instruction), not a value stored in a box (aka variable).

repcount → returns the iteration # of a repeat command:
- for the first execution of list of commands, it has value 1
- second execution, has value 2, ... etc

try command
repeat 25 [print repcount]
↑
command to write on screen.

Problem a) Draw Δ, square, pentagon ... until 10-gon.

repeat 8 [drawNgon 50 repcount+2]

Problem c) How about drawing like case a) but changing colors?
Try:
repeat 10 [setpencolor repcount ~
drawNgon 50 repcount+2]

Obs – character ~ allows us to continue a command on a second line. Normally logo expects each command to be given on one line (because logo is interpreted language)