# Philosophy of Information Technology

Lecture 5

Mario Verdicchio

Università degli Studi di Bergamo

Academic Year 2025-2026

# Binary encoding of numbers

**Numbers with base 10:**

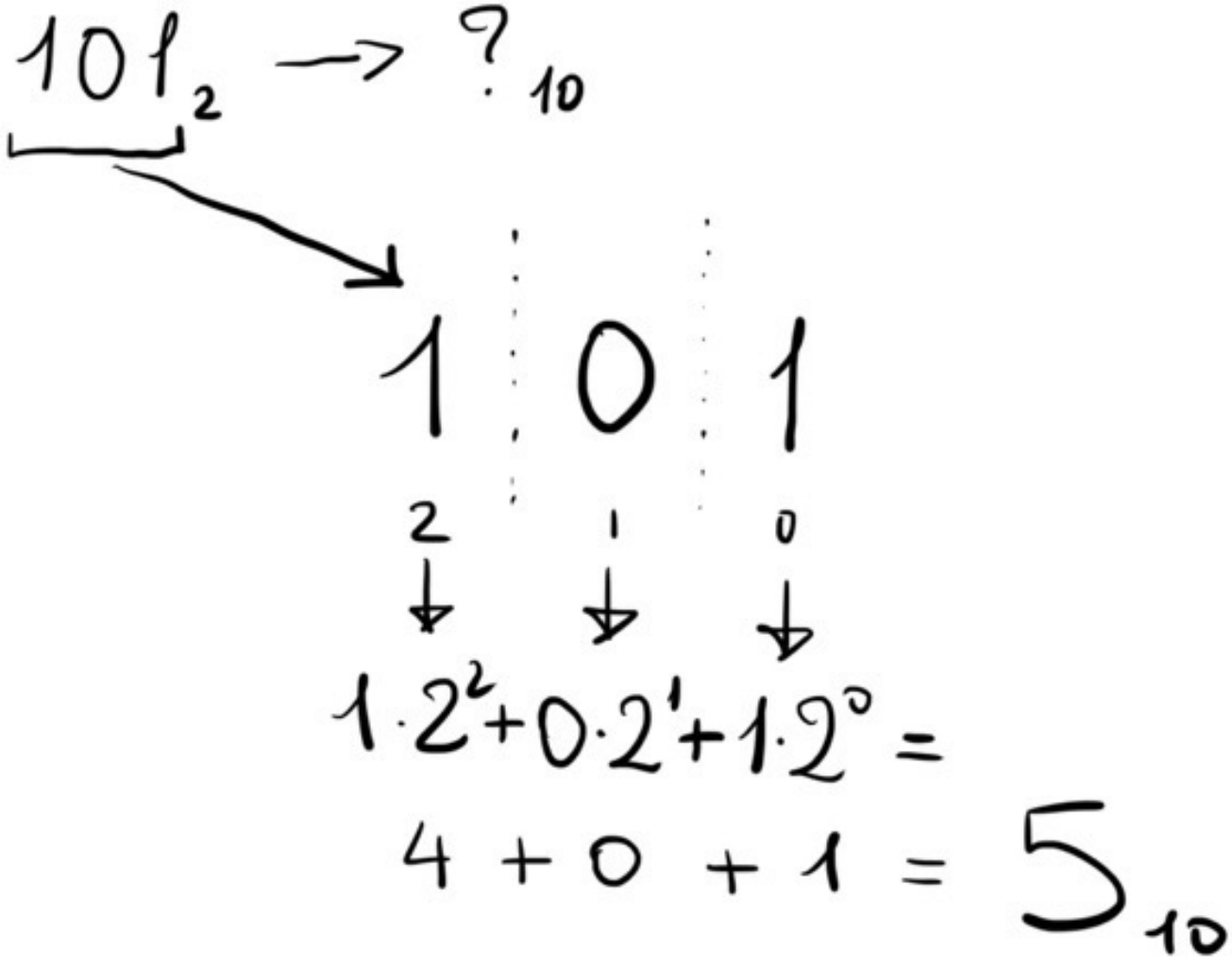
$$215 = 2\times10^2 + 1\times10^1 + 5\times10^0$$

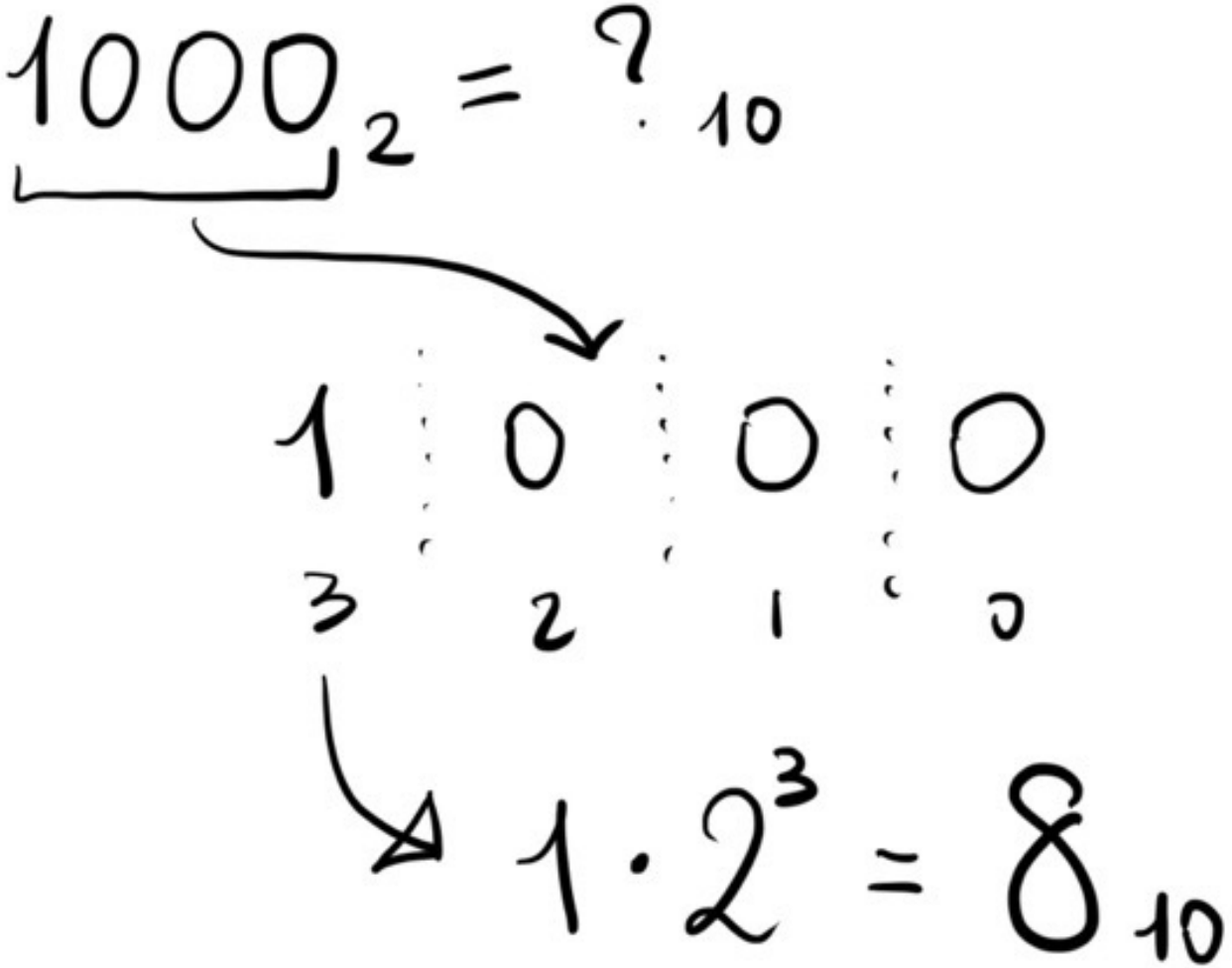
**Numbers with base 2:**

$$110010111 = 1\times2^8 + 1\times2^7 + 0\times2^6 + 0\times2^5 + 1\times2^4 + 0\times2^3 + 1\times2^2 + 1\times2^1 + 1\times2^0$$

## Exercise 1

**Convert the following numbers from base 2 to base 10: 101, 1000, 11011.**

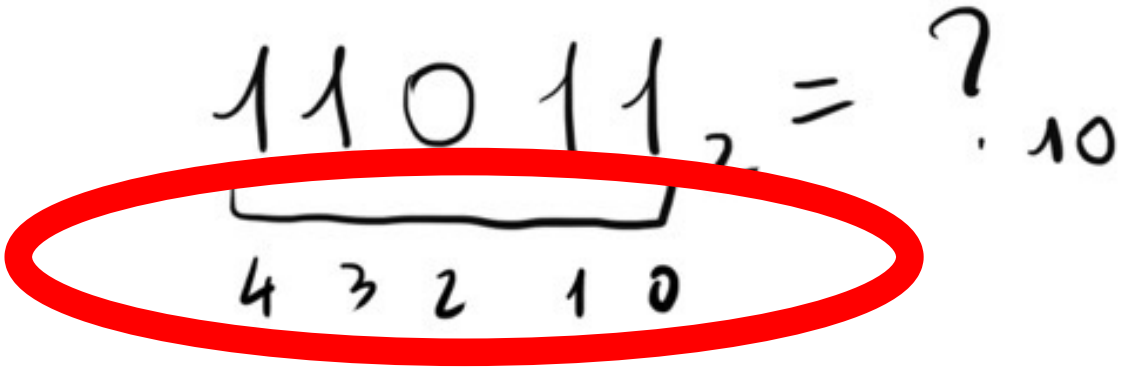

$101_2 \rightarrow ?_{10}$
1 0 1
2 1 0
$1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 =$
$4 + 0 + 1 = 5_{10}$


1000₂ = ?₁₀
1 0 0 0
3 2 1 0
1 · 2³ = 8₁₀

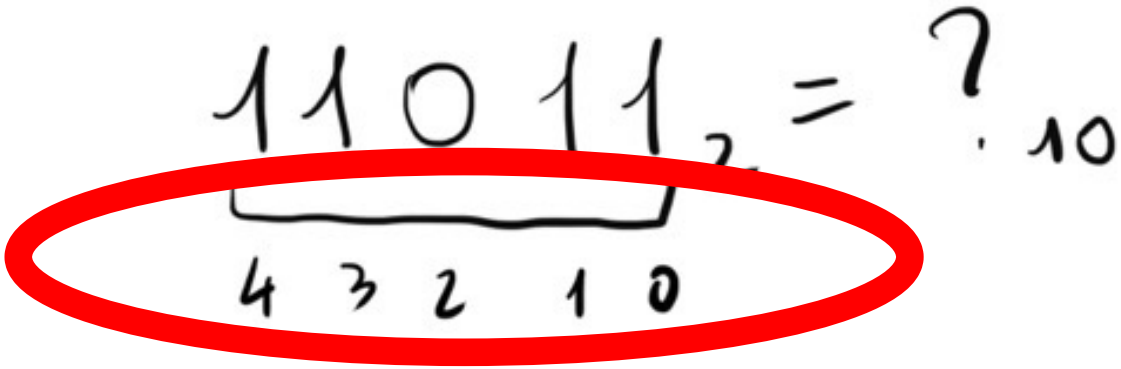
$$\underbrace{11011}_{4\ 3\ 2\ 1\ 0}{}_2 = ?_{10}$$

$$\hookrightarrow 2^4 + 2^3 + 2^1 + 2^0 =$$

$$16 + 8 + 2 + 1 = 27_{10}$$

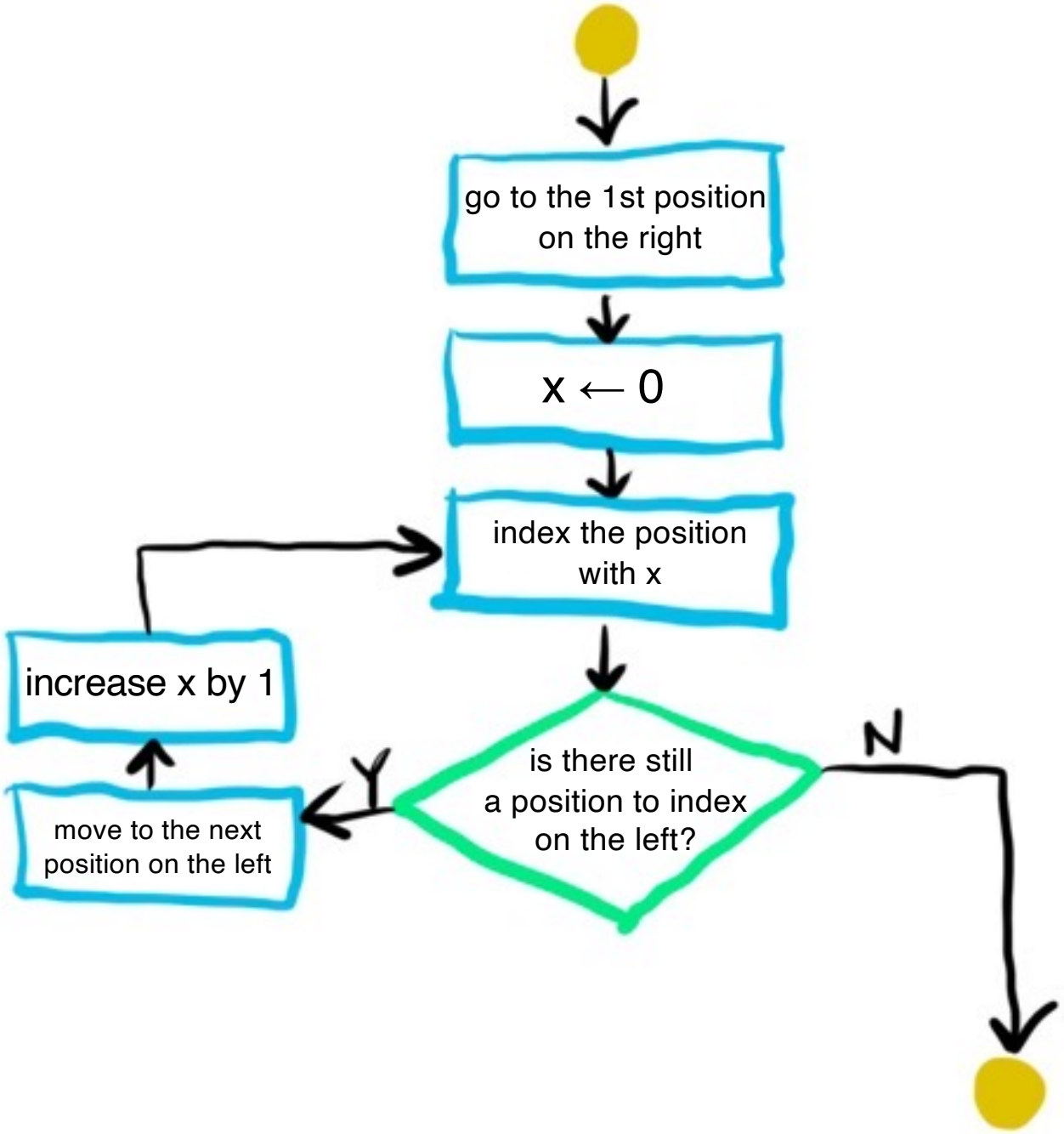


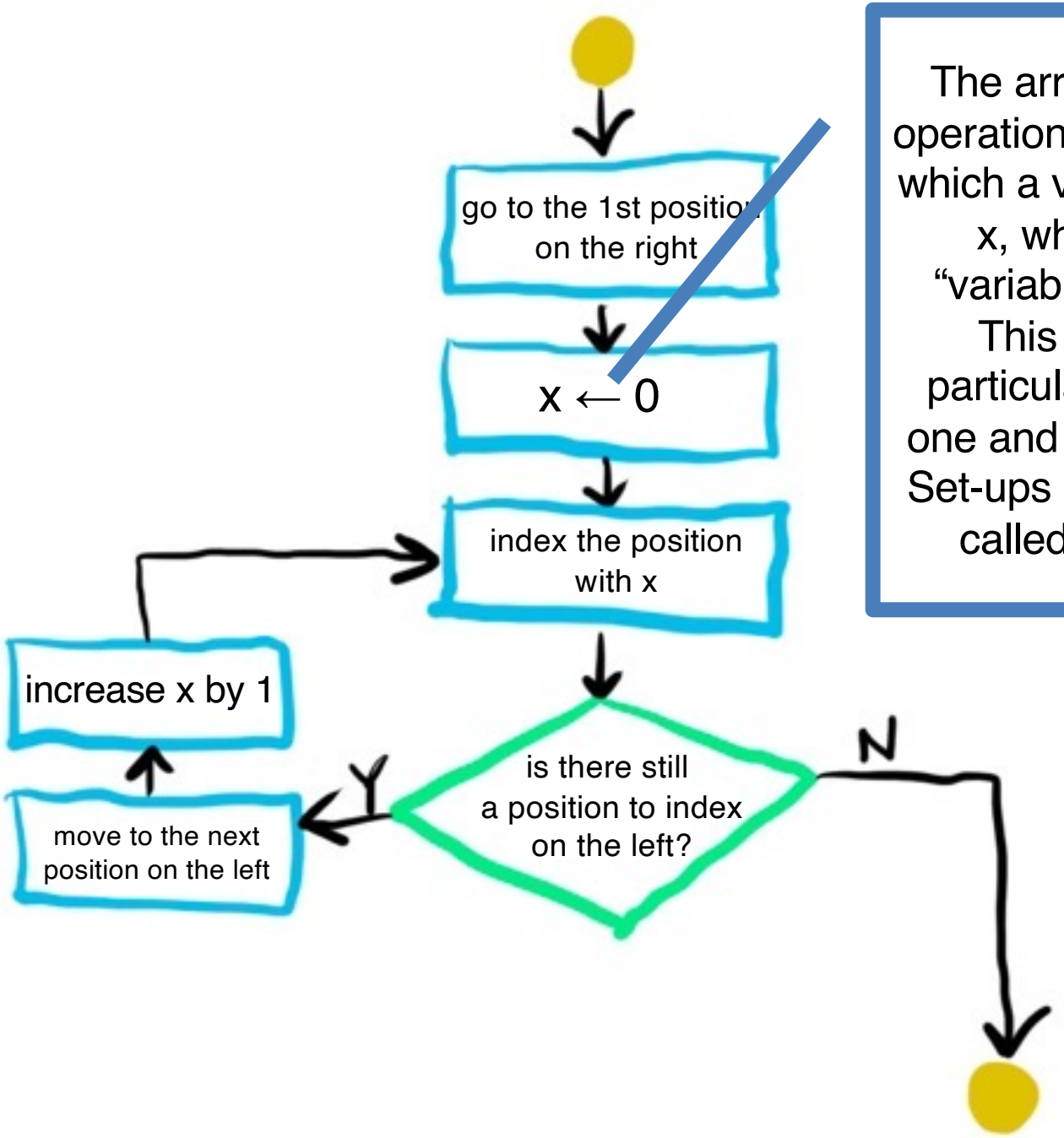

**Assigning a number to the positions of the binary number (also known as “indexing”, that is, establishing an index) may look like a trivial task, but it presents significant differences whether it is done by a human or by a computer.**



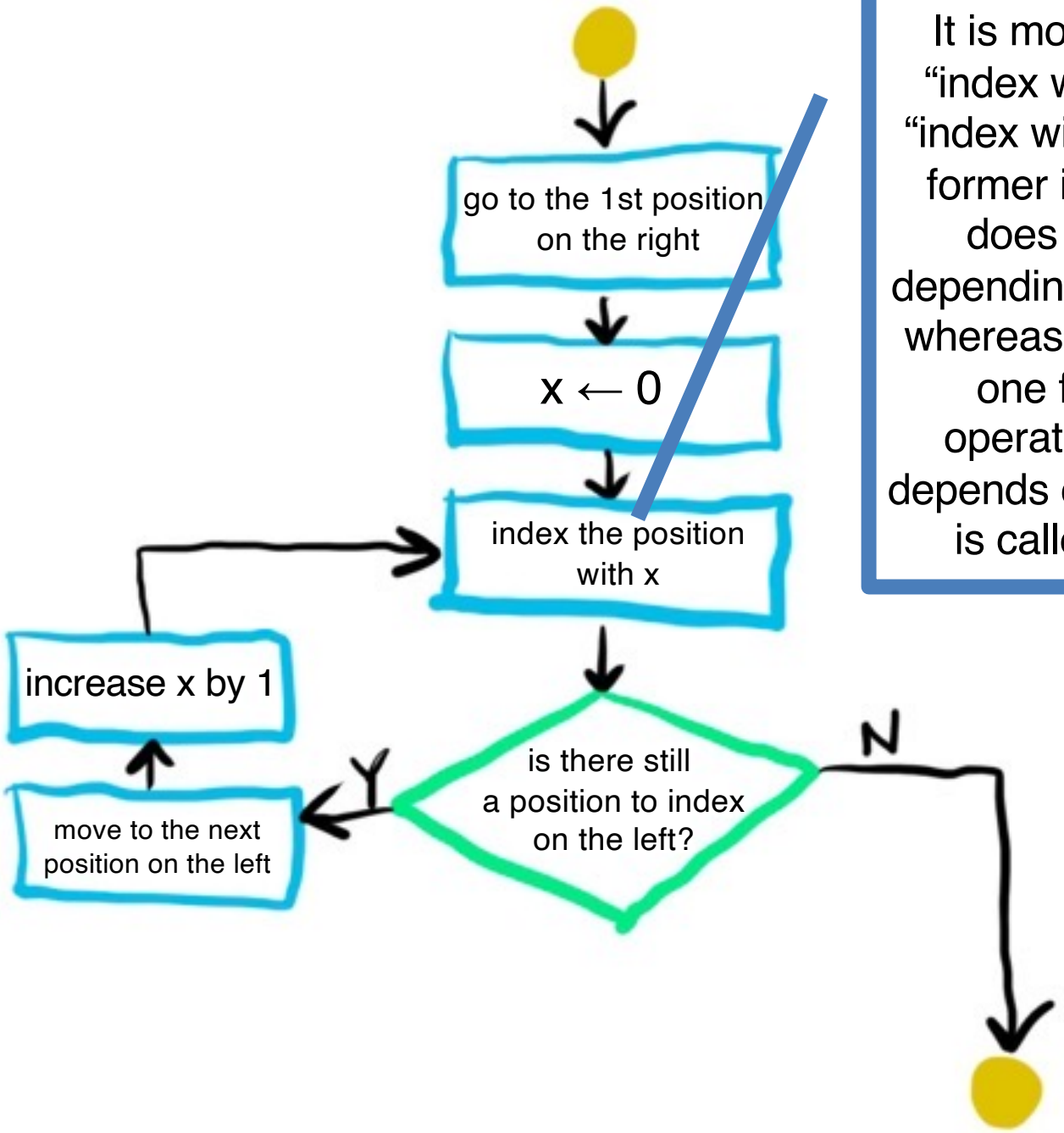

**A human can typically use their eyes to understand how many positions are there at a glance, and then starts writing the indexes from right to left, starting with "0". A computer cannot do that, because they do not have a global perception, but can only treat one data after the other.**

**In the following slide, there is an algorithm that is supposed to guide a computer in the task of indexing the positions.**

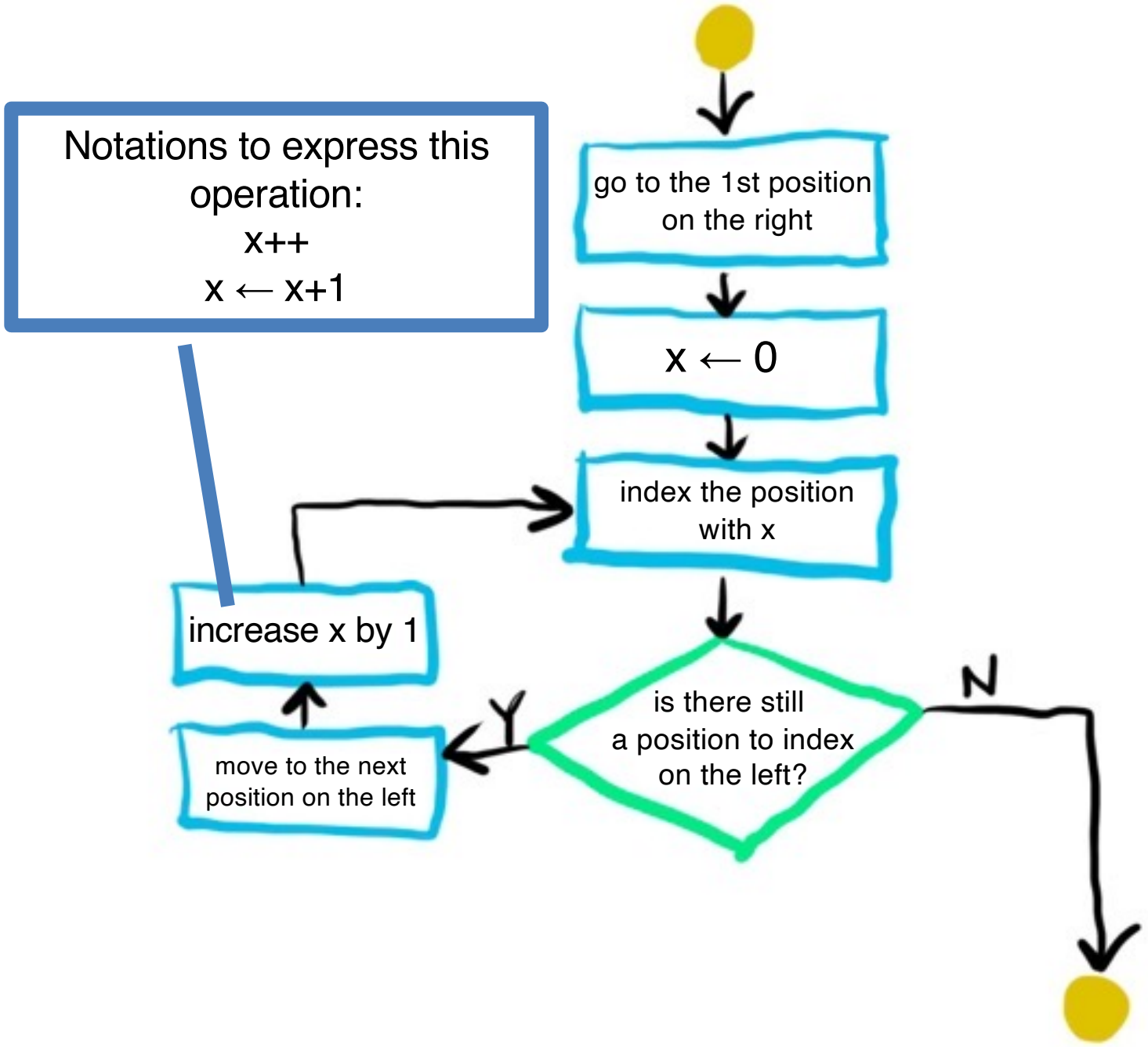

go to the 1st position on the right
x ← 0
index the position with x
is there still a position to index on the left?
Y
N
move to the next position on the left
increase x by 1

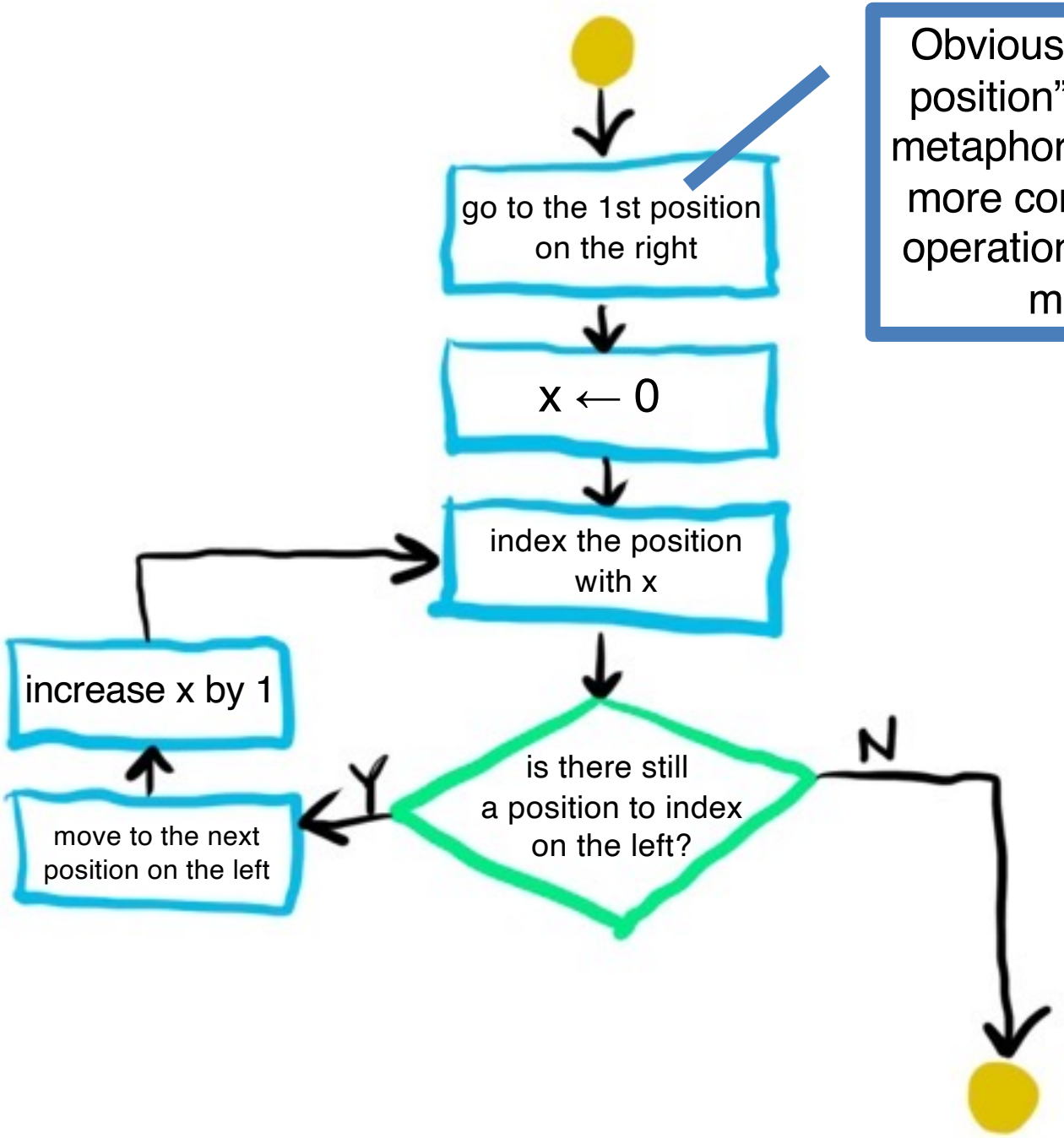



The arrow represents an operation of “assignment”, in which a value is assigned to x, which works as a “variable” or “parameter. This assignment, in particular, is the very first one and works as a set-up. Set-ups in an algorithm are called “initializations”.



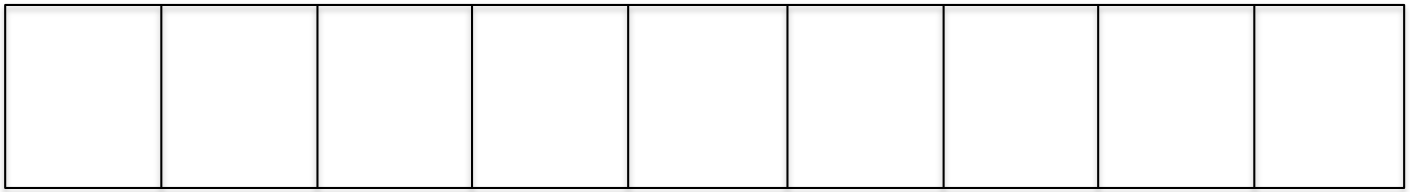

It is more clever to have "index with x" rather than "index with 0", because the former is more flexible: it does different things depending on the value of x, whereas the latter has only one fixed result. An operation whose result depends on a parameter in it is called "parametric".


Notations to express this operation:
x++
x ← x+1
go to the 1st position on the right
x ← 0
index the position with x
is there still a position to index on the left?
Y
N
move to the next position on the left
increase x by 1



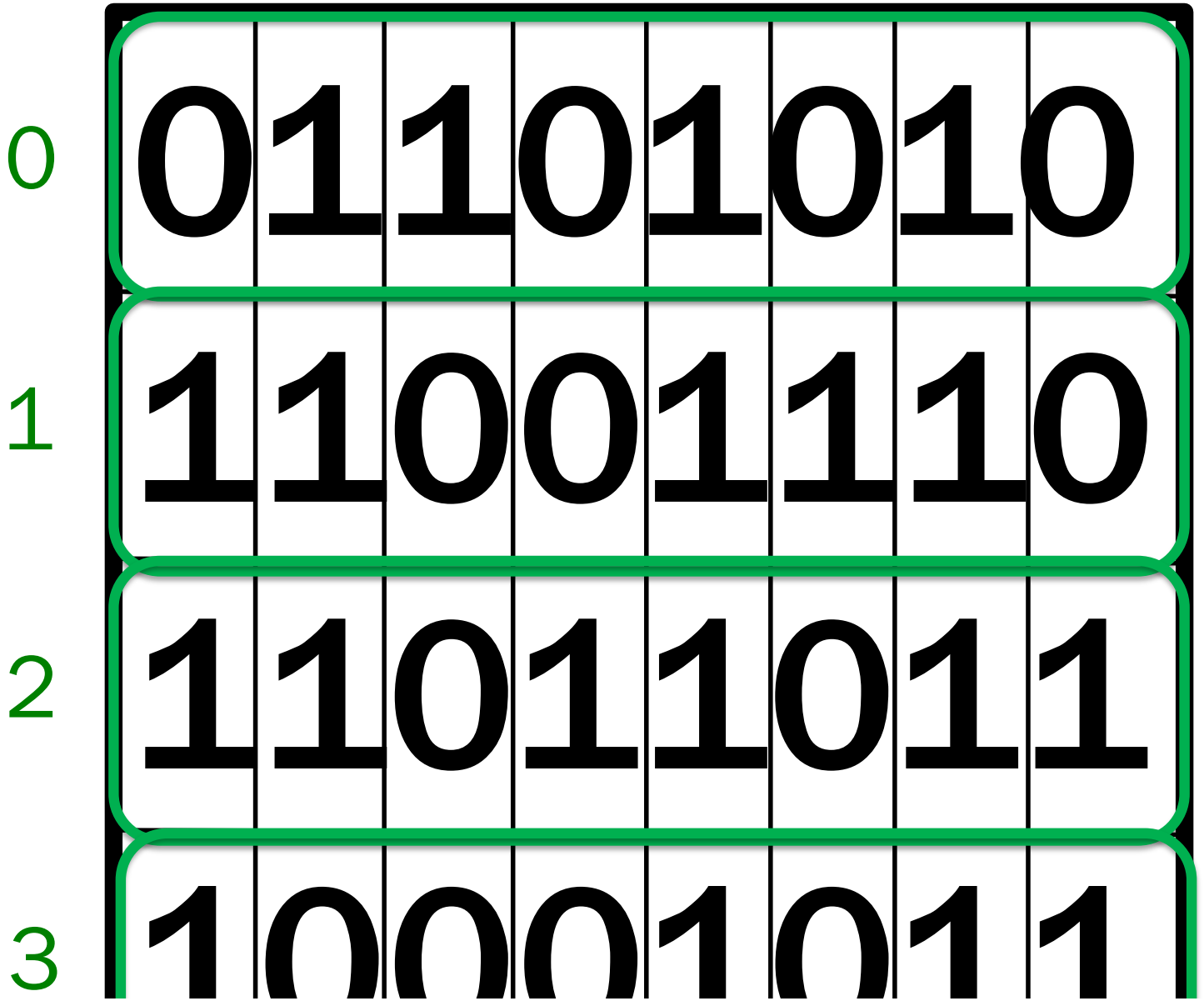

Obviously “going to the 1st position” is a very abstract, metaphorical description of a more concrete data access operation that the computer must execute.

**“First position on the right” implies the existence of a linear structure constituted by a countable (i.e. discrete, amenable to encoding) quantity of positions.**



**These structures, which can be seen as able to contain a multitude of data, are generally referred to in computer science as “structured data” or “data structures”.**

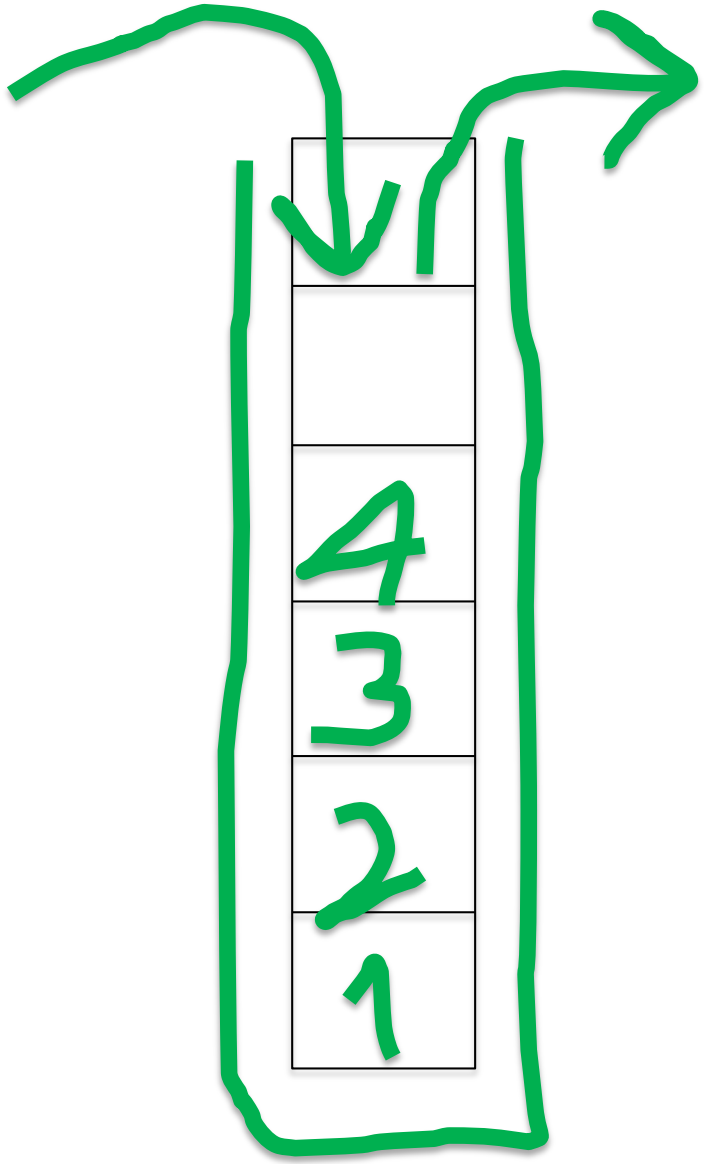
**The memory of a computer is an obvious example of data structure. However, now we are working with creating smaller data structures inside of it.**



# Data structure: definition

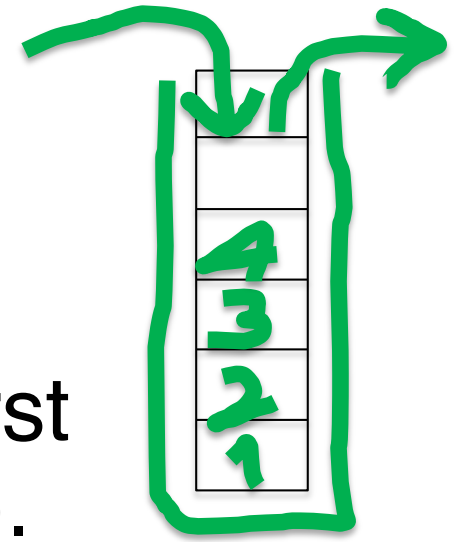
- A data structure is a formal system for organizing elements.
- It is “formal” because, to be useful and compatible with the operations of a computer, it needs to be managed by means of precise rules.
- These rules can be seen as encoding some theory of order (the order that organizes the elements inside the structure)

# Data structure 1: the stack



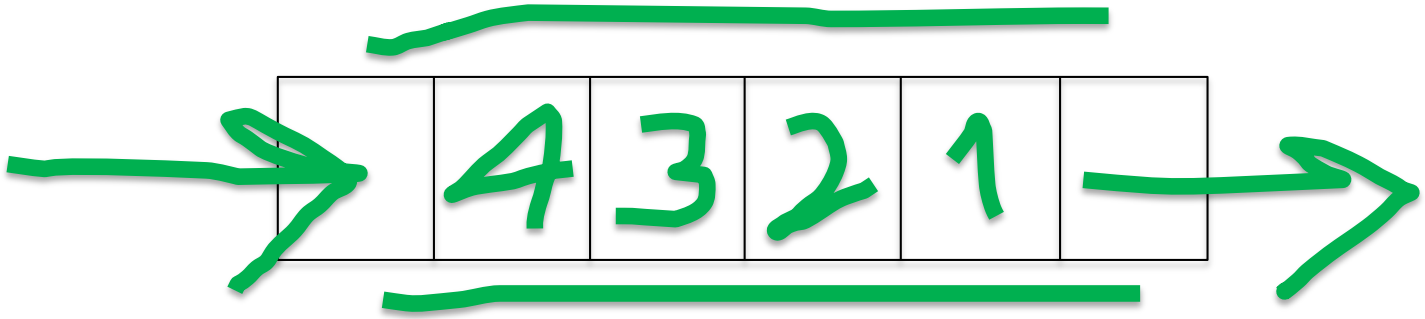
# Data structure 1: the stack

- Access is restricted to the top element.
- Operations:
  - Push (add)
  - Pop (remove)
- Logic behind it:
  the last element inserted is the first removed (LIFO: Last In First Out).




- It encodes a recency-based hierarchy: we get access to what's new.

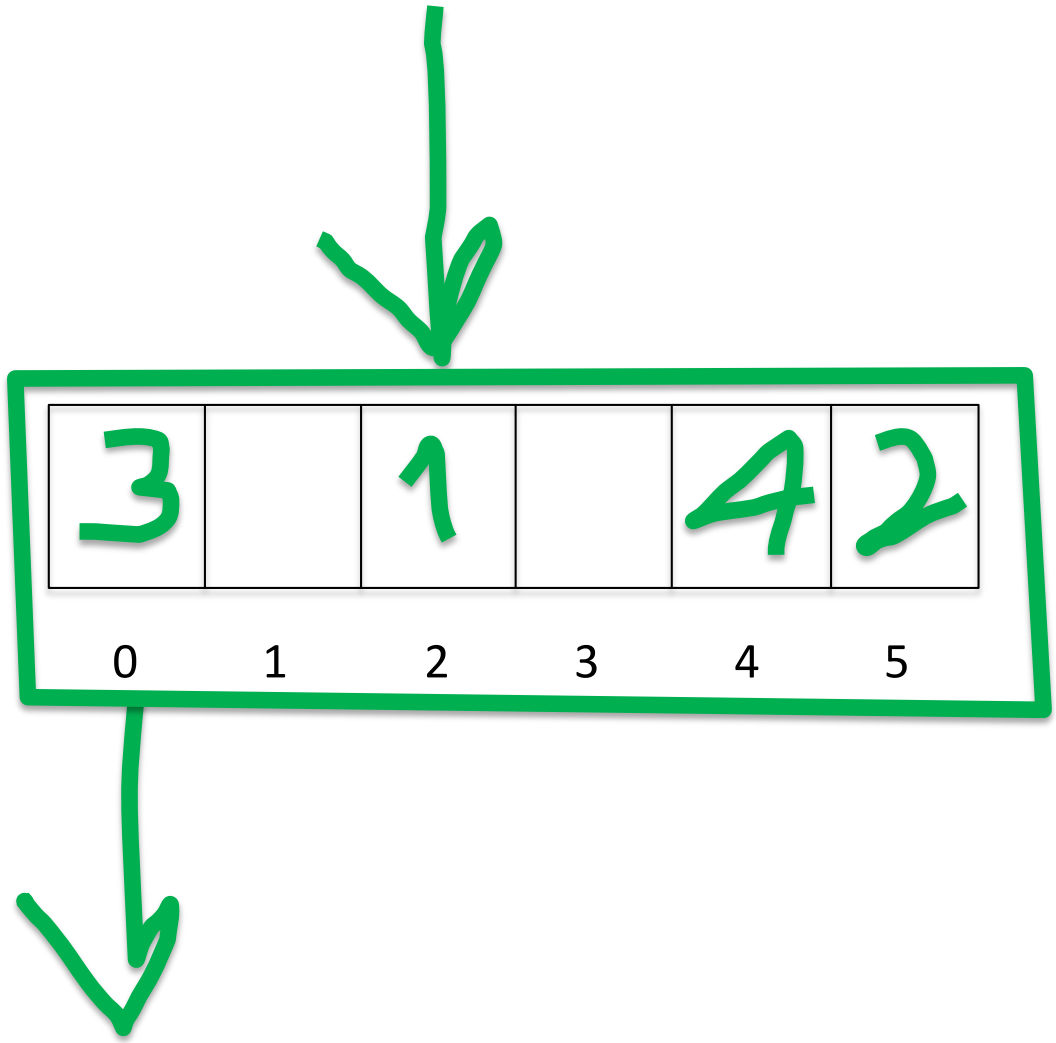
# Data structure 2: the queue



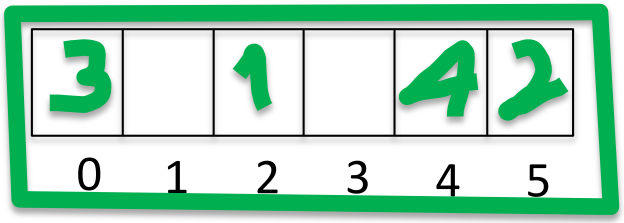
# Data structure 2: the queue


- Elements are processed in order of arrival
- Operations:
  - Enqueue (add at the end of the queue)
  - Dequeue (remove from the front of the queue)
- Logic behind it:
  the first element inserted is the first removed (FIFO: First In First Out).
- It encodes a rule of temporal fairness.

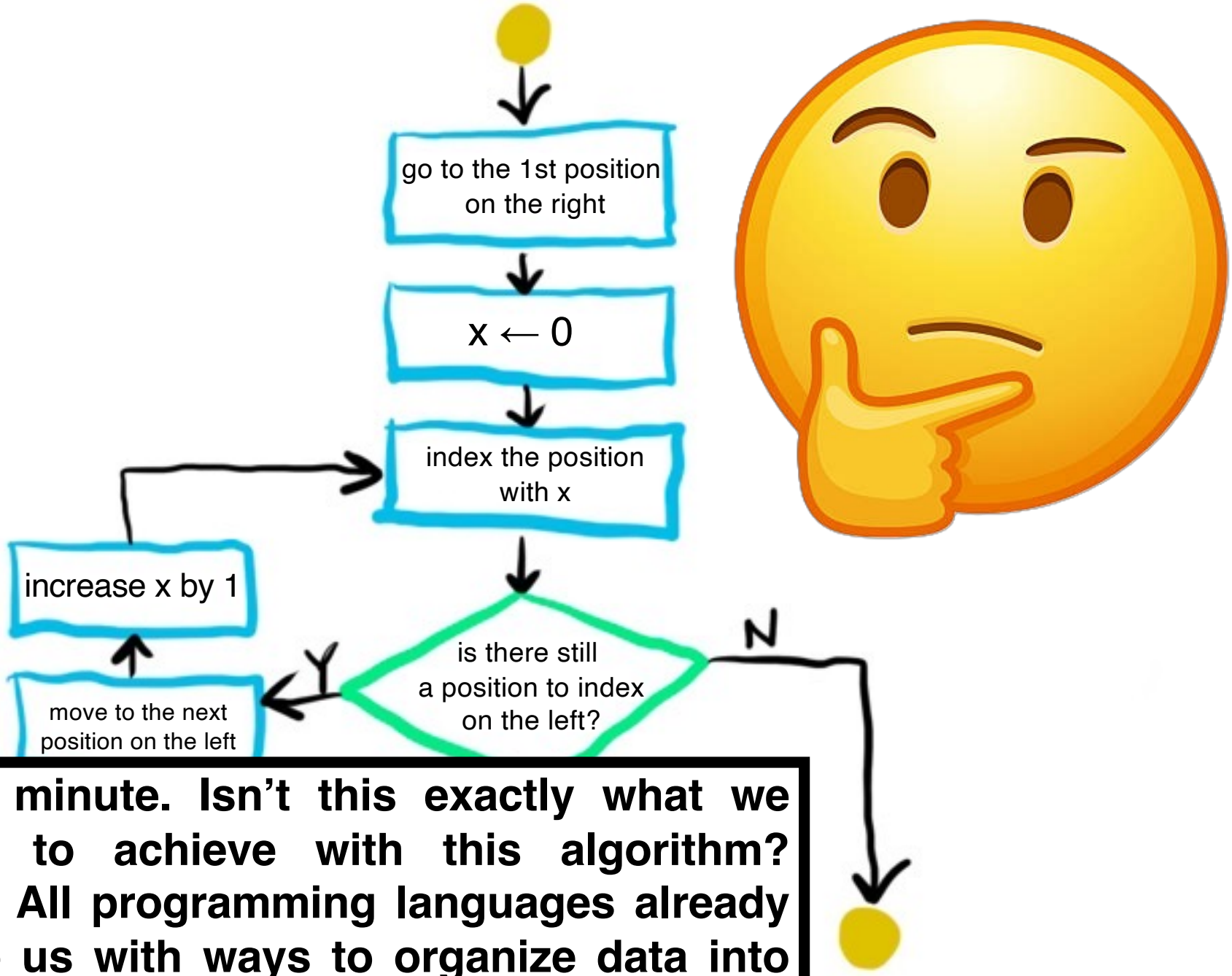
# Data structure 3: the array



# Data structure 3: the array




- Each position has an index.
- Operations:
  - write(i) (write element at position with index i)
  - read(j) (read element at position with index j)
- It represents positional organization.
- Differently from stacks and queues, the size is fixed and determined at the beginning by the largest index.




**Wait a minute. Isn't this exactly what we wanted to achieve with this algorithm? Indeed. All programming languages already provide us with ways to organize data into arrays.**

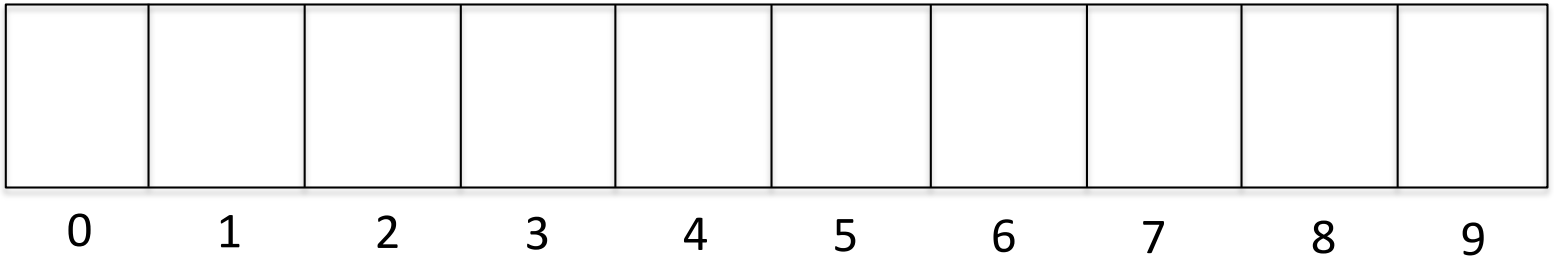
# Working with arrays

Typically, in many programming languages we write

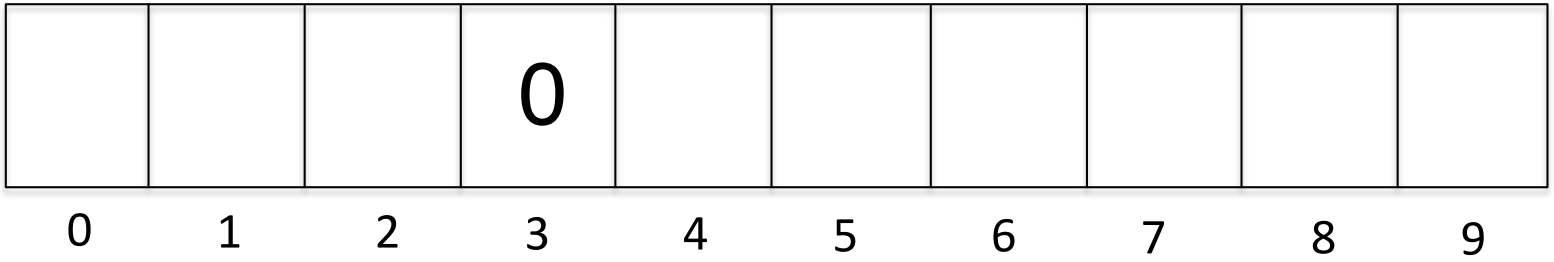
v[10]

to command the computer to reserve a sequence of 10 positions that we collectively refer to as “v”.
A position with index “i” will be accessed by writing “v[i]”

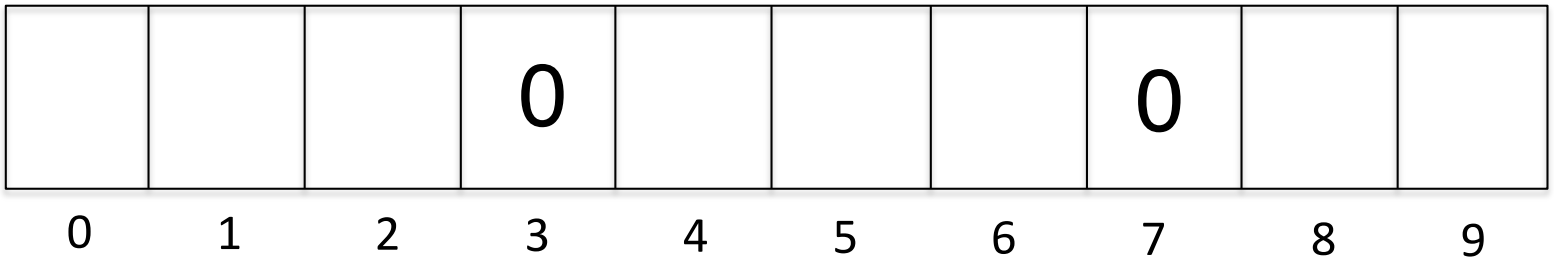
v[10]




v[3] <- 0




v[7] <- v[3]



# Working with arrays

Typically, working with arrays includes repetitive operations.

For instance, if we need to fill the array above with 0s, we have different ways to write it down in a program.

1) the pedestrian way
2) with a for cycle
3) with a while cycle

# 1) The pedestrian way

v[0] <- 0

v[1] <- 0

v[2] <- 0

v[3] <- 0

v[4] <- 0

v[5] <- 0

v[6] <- 0

v[7] <- 0

v[8] <- 0

v[9] <- 0

# 2) With a “for” cycle

- A “for” cycle is a synthetic way to organize an iteration (i.e. a repetitive action)

```
for (<initialization>;<condition>;<step action>)
    <operation>
```

- Example:

```
for (i <- 0; i < 10; i <- i + 1)
    v[i] <- 0
```

# How a “for” cycle is executed

for (<initialization>;<condition>;<step action>)

    <operation>

1) <initialization> is executed (only once)
2) <condition> is checked: if false, the cycle concludes without doing anything else; if true, <operation> is executed
3) after the execution of the operation, <step action> is executed
4) go back to 2)

# 3) With a “while” cycle

- A “while” cycle is another way to organize an iteration

  while (<condition>)

  <operation>

- Example:

```
i <- 0
while (i < 10)
    v[i] <- 0
    i <- i + 1
```

# How a “while” cycle is executed

while (<condition>)

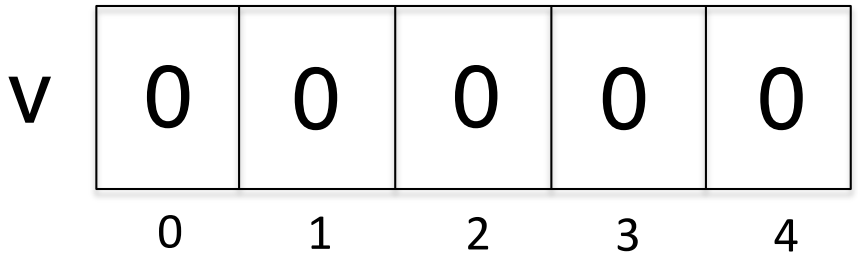
<operation>

1) <condition> is checked: if false, the cycle concludes without doing anything else; if true, <operation> is executed
2) go back to 1)

- Please notice that the “while” cycle has a much simpler structure, so the programmer will have to add operations equivalent to the <initialization> and <step action> from the “for” cycle if needed.

# Exercise 1

- Write the instructions to obtain the following result



# Solution 1.1 (with “for”)

```
v[5]
for (i <- 0; i < 5; i <- i + 1)
        v[i] <- 0
```

# Solution 1.2 (with “while”)

```
v[5]
i <- 0
while (i < 5)
        v[i] <- 0
        i <- i + 1
```

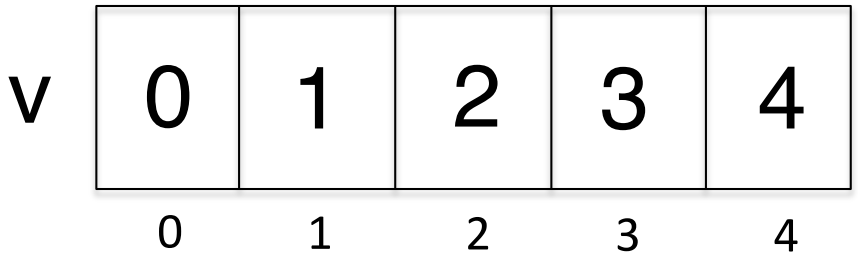
# Solution 1.3 (with “for”)

```
v[5]
for (i <- 4; i >= 0; i <- i - 1)
        v[i] <- 0
```

- Remember that there is never just one algorithmic solution to a problem
- All these solutions obtain the same result
- In particular, 1.1 and 1.2 fill the array in from left to right, whereas 1.3 from right to left

# Exercise 2

- Write the instructions to obtain the following result



# Solutions

- v[5]

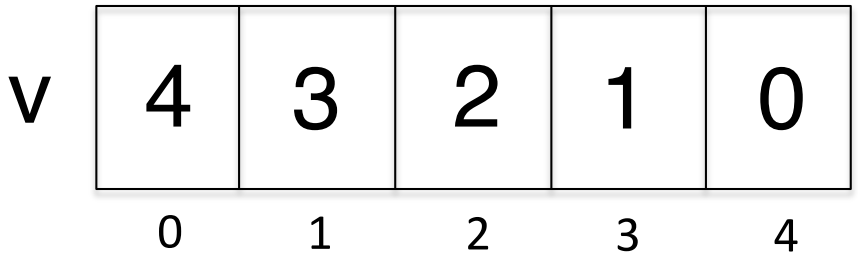
```
for (i <- 0; i < 5; i <- i + 1)
    v[i] <- i
```

- v[5]

```
i <- 0
while (i < 5)
    v[i] <- i
    i <- i + 1
```

# Exercise 3

- Write the instructions to obtain the following result



# Solutions

- v[5]

```
for (i <- 0; i < 5; i <- i + 1)
    v[i] <- 4-i
```

- v[5]

```
for (i <- 4; i >= 0; i <- i - 1)
    v[i] <- 4-i
```