

Basic Concepts in Algorithmic Thinking: Sequencing, Selection & Repetition

School Grade: K7/K9

Table of contents

[Introduction](#_heading=h.gjdgxs) **3**

[What is an Algorithm?](#_heading=h.y9fd3ccnzb3t) **3**

[Sequencing](#_heading=h.sx9mm8vv41lj) **4**

[Selection](#_heading=h.7a1rycfnc77v) **6**

[Repetition](#_heading=h.c0s0rsaecwjw) **7**

[Pseudocode](#_heading=h.8i64txlf9n76) **9**

[Solved Problems](#_heading=h.8f8zlhq795e6) **11**

[References](#_heading=h.2et92p0) **11**

[Annexes](#_heading=h.tyjcwt) **11**

# Introduction

Algorithmic thinking is the ability to solve problems by defining a clear and defined sequence of steps that, once faithfully followed, lead to a solution. During our school life, and in particular during the primary and secondary school years, we learn to solve problems through a process of mnemonic learning and imitation of the solutions provided by the teacher. For example, although many students know how to calculate the greatest common denominator of two numbers, few can articulate the list of operations required to reach the result.

This process of defining the steps required to arrive at a solution is extremely important because, in addition to making it possible to automate tasks (e.g. through the creation of a machine capable of carrying out the indicated steps), it leads the individual to a greater awareness of the problem, its solution and consequently to the ability to adapt the found solution to similar or analogous problems: in order to articulate clear and well-defined steps, one must first understand the problem, the conditions from which one starts and those to which one wishes to arrive, and the tools and operations that allow us to modify the system in a programmed and predictable manner.

Although the concepts of 'algorithm' and 'algorithmic thinking' are often immediately linked to the context of computer science, they are indispensable tools in everyday life. One only has to think of cooking and the recipes used to cook any dish, a surgical operation or the construction of a building. The definition of algorithms is one of the processes we use most to pass on knowledge and defined processes.

In this work, the concept of algorithm will be detailed, and some of the most important concepts related to it, such as sequencing, selection, and repetition, will be examined more closely.

# What is an Algorithm?

An algorithm, also often referred to more simply as a procedure, is a set of clear and well-defined steps that can be used to complete a task. An example of an algorithm that can helps us to better understand the concept is provided by the recipe for making coffee using a moka pot. In its simplest version, the recipe can be summarised as follows[[1]](#footnote-0):

* Disassemble the moka pot into its three main components: bottom part, metal filter and top part.
* Pour water into the bottom part of the moka pot
* Place the metal filter in the filter slot, on top of the bottom part of the moka pot
* Fill the metal filter with coffee powder
* Screw the top part of the moka pot on top of its bottom part
* Place the moka pot on a cooker
* Switch on the cooker
* Wait for the coffee to spill out and fill the top of the moka pot
* Switch off the cooker

These instructions can be effectively followed by anyone, even a reasonably sophisticated robot, and always lead to the same result: great coffee.

So all lists of operations are algorithms? Actually, at a formal level, an algorithm is always characterised by four fundamental properties: effectiveness, finiteness of expression, finiteness of computation, determinism.

* 1. **Effectiveness**

An algorithm must be effectively executable by an executor; this means that the executor must be able to understand the description of the algorithm and therefore be able to recognise the language in which the steps making up the procedure are expressed. Well-formed sentences in this chosen language are formally called instructions.

In other words, this property means that if one define an algorithm using a language, for example English, any executor capable of understanding that language will interpret the instructions in the same way.

* 1. **Finiteness of expression**

The procedure described by an algorithm must be articulated in a finite number of instructions. It does not matter how many instructions there are (and one can easily think of procedures with a very large number of instructions) nor how long the procedure is, as long as the number of instructions is not infinite.

* 1. **Finiteness of computation**

Similarly to the finiteness of expression, an algorithm must always be characterised by a finite number of execution steps. In other words, an algorithm must always specify a condition for which the execution ends. Here again, it is not the number of steps that is important, but the notion of finiteness of execution.

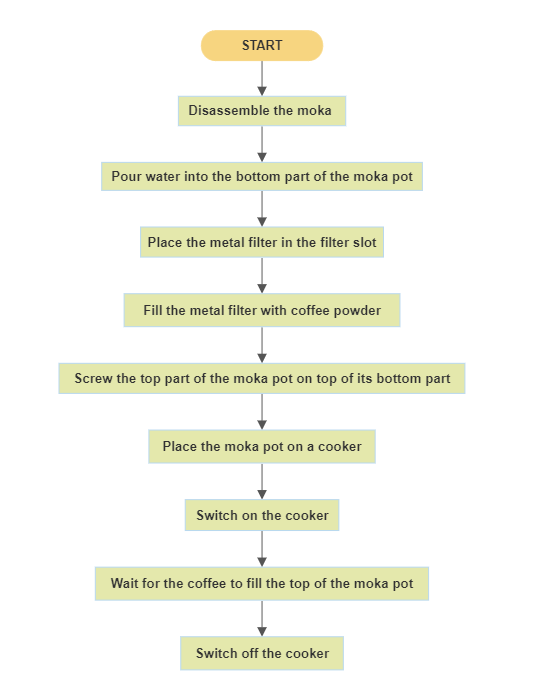
* 1. **Determinism**

An algorithm must be deterministic, which means: applying the same algorithm to the same set of input data will always produce the same output results.

In the light of these four properties, we realise that certain assumptions must be made to keep our coffee recipe an algorithm: a clean, assembled and ready-to-use moka pot must always be present; a cooker must always be available; water and coffee powder must always be available. What if we wanted to make our algorithm more versatile, taking into account these initial conditions and generalising in such a way as to obtain a procedure that is always feasible? In the following chapters we will take a closer look at some of the tools that will allow us to refine our algorithm and, at the same time, better understand this world. In particular, we’ll see three basic structures: Sequencing, Selection, and Repetition. These are in fact the most important, especially thanks to a theorem defined in the 1960s by two researchers: Corrado Böhm and Giuseppe Jacopini [1]. The theorem states that using only Senquencing, Selection and Repetition, any algorithm can be implemented!

# Sequencing

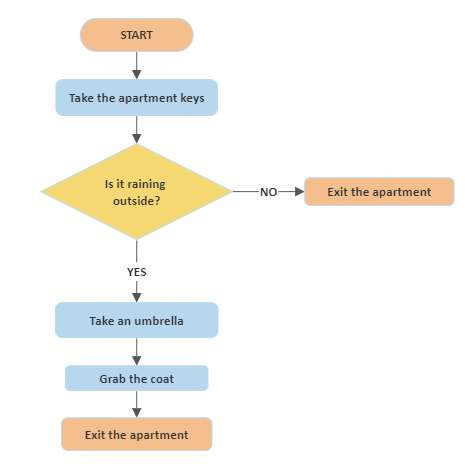
Sequencing means defining an order for the instructions that are part of the algorithm. For example, if we consider the instructions that make up our recipe for making coffee:



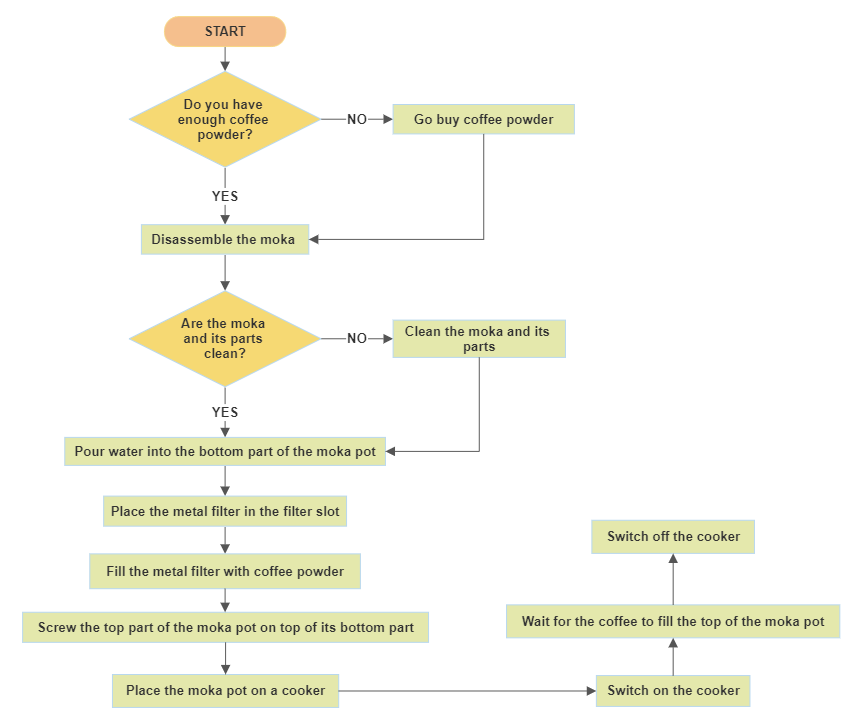
To ensure that the same result is always achieved (ensuring determinism), each step must be performed in the specified order. Following this example, it’s easy to see that changing the order of the steps would lead to different (and unintented!) result: what would happen if you tried to fill the bottom part of the moka pot with water before you dismantled the moka itself?

# Selection

Selection is the construct that allows us to define different 'execution' paths and choose one rather than another depending on the verification of a particular condition (e.g. is the condition true or false?). For example, if we defined an algorithm to collect important objects before leaving the house, we could define two paths depending on whether it is raining outside at the moment or not:



Selection is a very powerful construct that allows us to make an algorithm flexible and adaptable. Taking our algorithm for making coffee as an example, we can start to test conditions to make sure that it is increasingly applicable in everyday reality: what happens if the coffee pot is dirty? What happens if we have run out of coffee powder?



# Repetition

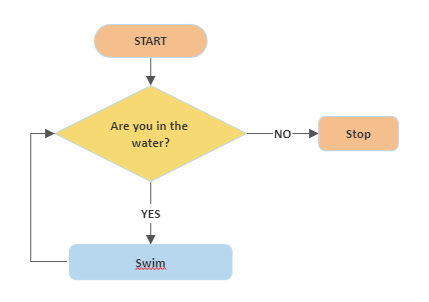
Repetition, often also called iteration or looping, is a construct that allows an instruction (or set of instructions) to be executed more than once without the need to repeat them each time within the context of a sequential instruction block. Commonly, there are three types of repetition: *count-controlled* (also called 'for loop' because of the keywords used in various programming languages to refer to this type of repetition), *while* and *repeat-until*.

* 1. **“Count-controlled” repetition**

Count-controlled repetition is a type of repetition that allows a block of instructions to be repeated a predefined number of times. For example, if we wanted to make explicit the instructions to count down from 10 to 0 we could say "subtract 1 from the current count 10 times in a row". This construct is useful in the context of operations that we already know in advance will be repeated a specific number of times and whose repetition does not depend on any other condition.

* 1. **“While” repetition**

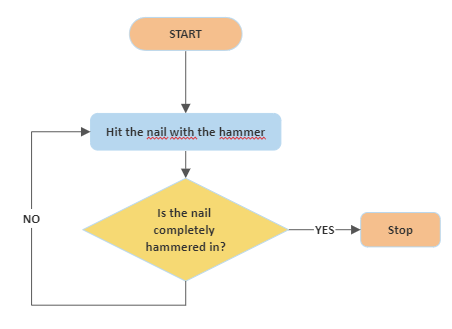
Unlike count control, while repetition allows a block of instructions to be repeated until a condition is met. This construct makes it possible to define the fact that instructions are to be executed several times without knowing in advance how many times they are to be repeated. An example of this type of execution is the following: “while in the water swim”.



It is important to note that when using a while repetition we have no assurance that the instruction block meant to be repeated will be executed at least once. In the given example, if the executor is out of the water from the beginning the instruction “swim” is never carried out.

* 1. **“Repeat until” repetition**

Repeat-until repetition is in fact a very similar construct to while: it too is used when a block of instructions potentially has to be repeated several times and we do not know in advance how many times it will have to be repeated. The only difference is that the stop condition check is carried out after the first iteration of the instruction. In other words, we ensure that the instructions are executed at least once before deciding whether or not to repeat them.



# Pseudocode

Flowcharts are an excellent tool for formalising an algorithm in an understandable and writable language. However, as useful as they are, they have important disadvantages that often make them inconvenient to use: the amount of space required and the non-immediate mapping of typical repetition constructs (e.g. controlled counting). A popular alternative is that provided by pseudocode: a language that resembles programming languages, but abstracts specific keywords from them, remaining generic and closer to the natural language we use every day.

In this chapter, we will briefly introduce a simplified pseudocode that will easily allow us to write our own algorithms using the constructs we introduced. Inside this language, we will use the following rules:

* each line represents an instruction

| this is one instruction  this is an other instruction  this is an other instruction |
| --- |

* blocks of instructions must be represented by the use of indentation: different blocks will have different numbers of empty spaces before the line

| this instruction belongs to **block A**  this instruction also belongs to **block A**  this instruction belongs to **block B**  this instruction also belongs to **block B**  this instruction also belongs to **block B**  this instruction belongs to **block A** |
| --- |

* Selection is identified by the keyword **IF**, followed by a condition to be verified and then a new instruction block to be executed if the condition is true. Following this block, an other instruction block can be used to define instruction to be executed if the condition is false; this block must be introduced by the keyword **ELSE**.

| an instruction  an other instruction  **IF condition**  **this instruction will be executed if condition is true**  **this instruction will also be executed if condition is true**  **ELSE**  **this instruction will be executed if condition is false**  **this instruction will also be executed if condition is false**  this instruction will be executed regardless of the condition specified after IF |
| --- |

* Count-controlled repetition is identified by the keyword **FOR** **X TIMES DO** followed by an instruction block to be repeated X times.

| an instruction  an other instruction  **FOR X TIMES DO**  **this instruction will be repeated X times**  **this instruction will also be repeated X times**  this instruction will not be repeated |
| --- |

* While repetition is identified by the keyword **WHILE <CONDITION> DO** followed by an instruction block to be repeated until <CONDITION> is no longer true. Remeber that if the condition is not initially true the instructions in the following block will not be exectued.

| an instruction  an other instruction  **WHILE CONDITION DO**  **this instruction will be repeated until condition holds true**  **this instruction will also be repeated until condition holds true**  this instruction will not be repeated |
| --- |

* Repeat-until repetition is identified by the keyword **REPEAT** followed by an instruction block. This instruction block must be followed by the keyword **UNTIL <CONDITION>**  that specifies the condition to be checked in order to determine if the instruction block is to be repeated or not..

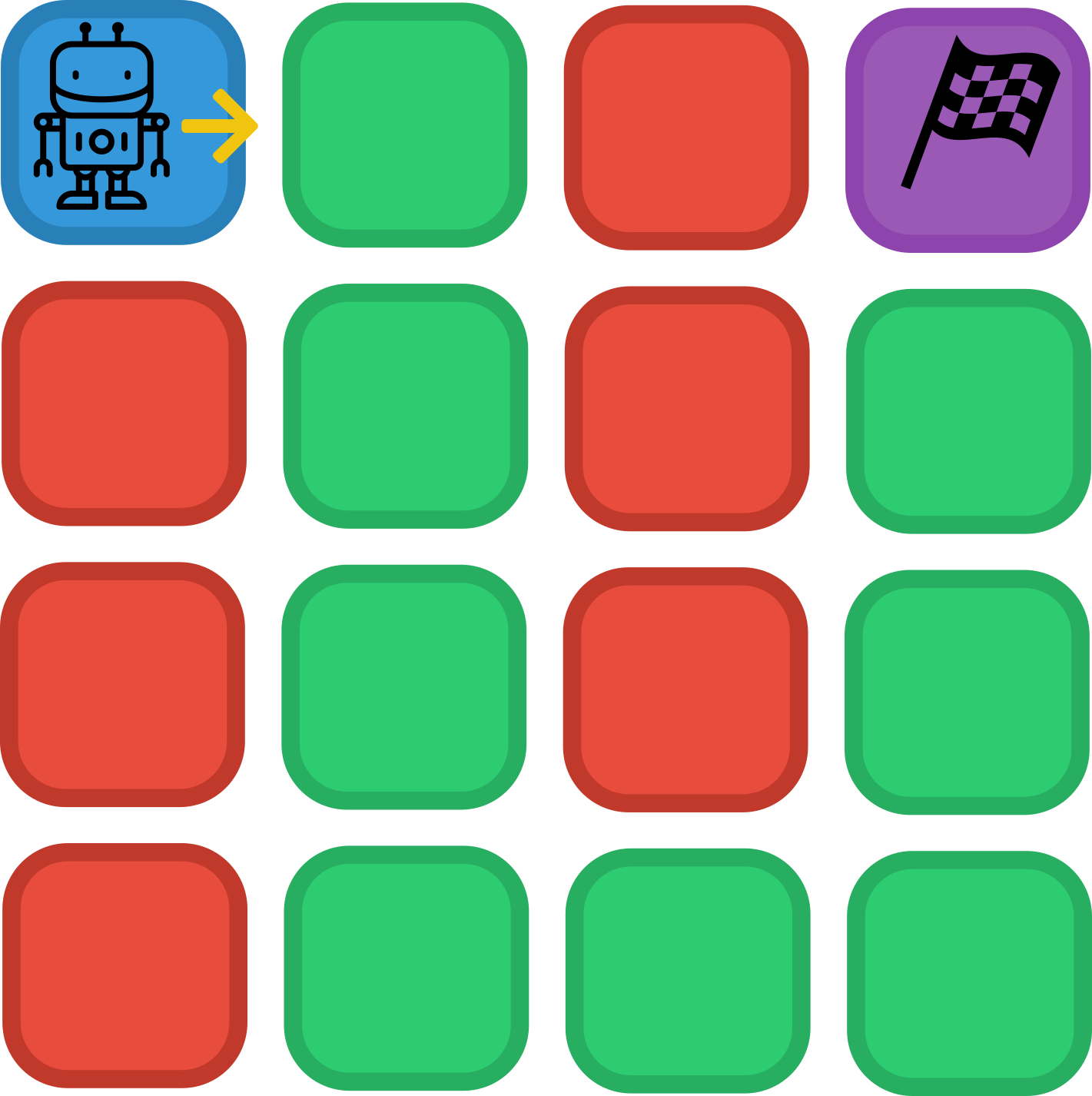
| an instruction  an other instruction  **REPEAT**  **this instruction will be repeated until condition holds true**  **this instruction will also be repeated until condition holds true**  **UNTIL CONDITION**  this instruction will not be repeated |
| --- |

# Solved Problems

1) Bob is a little robot. Being a very simple one, it only carry on 3 operations:

* turn left 90°
* turn right 90°
* walk one tile forward

Referring to the following image, can you help Bob get from the Blue tile to the Purple tile, moving only through the green-coloured tiles? Write two algorithms for this task, one without and one with the use of repetition constructs. Please notice that the yellow arrow indicates the direction Bob starts looking at.



| **Solution with no repetition** | **Solution with the use of repetition constructs** |
| --- | --- |
| walk one tile forward  turn right 90°  walk one tile forward  walk one tile forward  walk one tile forward  turn left 90°  walk one tile forward  walk one tile forward  turn left 90°  walk one tile forward  walk one tile forward  walk one tile forward | walk one tile forward  turn right 90°  FOR 3 TIMES DO  walk one tile forward  turn left 90°  FOR 2 TIMES DO  walk one tile forward  turn left 90°  FOR 3 TIMES DO  walk one tile forward |

# References

[1] Böhm, C., & Jacopini, G. (1966). Flow diagrams, turing machines and languages with only two formation rules. *Commun. ACM, 9*, 366-371.

[2] second reference

1. Moka pot icon created by Dooder and downloaded from Flaticon - “https://www.flaticon.com/free-icons/moka-pot" [↑](#footnote-ref-0)