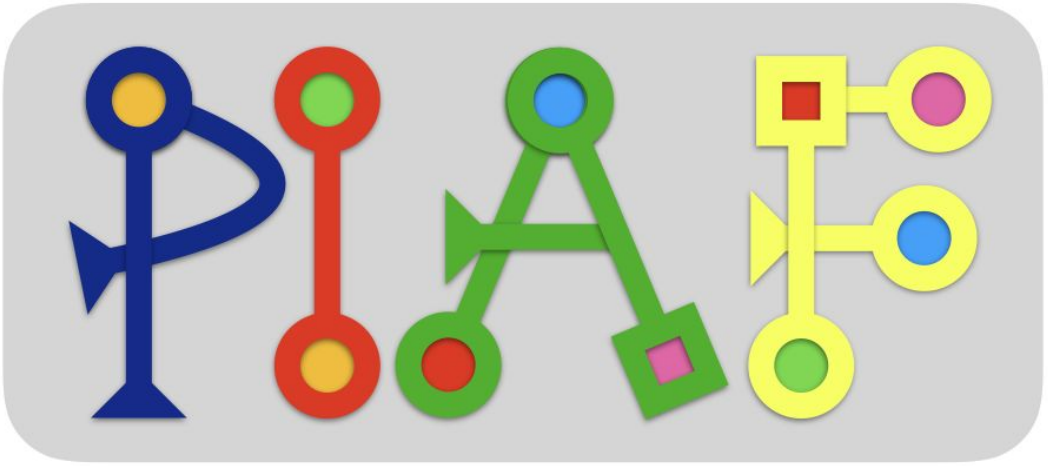

# *PIAF : développer la Pensée Informatique et Algorithmique dans l'enseignement Fondamental*

## Referential of competencies

## Appendix 1: Description and examples

Erasmus+

LIÈGE université

UNIVERSITÉ DE LORRAINE

UNIVERSITÉ DU LUXEMBOURG

UNIVERSITÄT DES SAARLANDES

# Table of contents


**Table of contents** **1**

**Competency 1: Abstracting away / generalizing** **2**

C 1.1 Name objects and (sequences of) actions 2

C 1.2 Differentiate (i) object and action, and (ii) atomic actions and non-atomic actions 2

C 1.3 Identify the input parameters of a sequence of actions 3

C 1.4 Describe the outcome of a sequence of actions 4

C 1.5 Predict the outcome of a sequence of actions 5

C 1.6 Using objects whose value can change 5

C 1.7 Recognize existing objects and (sequences of) actions that can be used to reach a similar goal 6

**Competency 2: Compose/decompose a sequence of actions** **7**

C 2.1 Order a sequence of actions to reach a goal 7

C 2.2 Complete a sequence of actions to reach a simple goal 7

C 2.3 Create a sequence of actions to reach a simple goal 8

C 2.4 Create a sequence of actions to reach a complex goal 8

C 2.5 Combine sequences of actions to reach a goal 9

C 2.6 Decompose goals into simpler subgoals 9

Competency 3: Control a sequence of actions **10**

C 3.1 Repeat a sequence of actions a given number of times 10

C 3.2 Repeat a sequence of actions until a goal has been reached 10

C 3.3 Create a sequence of actions which relies on simple conditions 11

C 3.4 Create a sequence of actions which relies on complex conditions 11

Competency 4: Evaluate objects or sequences of actions **12**

C 4.1 Compare two objects according to a given criterion 12

C 4.2 Compare two sequences of actions according to a given criterion 12

C 4.3 Improve a sequence of actions according to a given criterion 13

**Competency 5: Translate between representations** **14**

C 5.1 Represent objects or sequences of actions in various formal representations 14

C 5.2 Translate objects or sequences of actions from an informal representation to another representation (formal or not) and vice versa 14

C 5.3 Translate objects or sequences of actions between formal representations 15

**Competency 6: Build a sequence of actions iteratively** **16**

C 6.1 Test a sequence of actions with respect to a given goal 16

C 6.2 Check and correct a sequence of actions with respect to a given goal 17

C 6.3 Extend or modify a sequence of actions to reach a new goal 18

# Competency 1: Abstracting away / generalizing

## C 1.1 Name objects and (sequences of) actions

**Definition**
Being able to give names to objects, actions and sequences of actions.

**Background**
It is important to build up the habit to give names to existing objects / actions so that these can be referred to and that solutions can be expressed easily and straightforwardly.

Note that some values (objects) may be known and will not change (e.g. a family name, a street number), in that case they are called constants. Some values are not known and / or may change depending on the context (e.g. temperature), and are thus called variables. This concept of variable is discussed in competency C 1.6 below.

**Examples**

1. Given a story with several similar (e.g. all sportsmen) but distinguishable characters, being able to give them names in order to uniquely identify them and tell a story about them.

2. One learner does an action, a second one sees it and has to simply tell it to a third one, who doesn't see the first learner and has to reproduce the action.

3. Two learners agree on four words corresponding to four simple (sequences of) actions. A third learner comes in. The first learner says one of the four words, the second learner executes the corresponding commands. From this, the third learner should be able to learn the language and then perform the requested commands.

4. Describe the steps necessary to get dressed, and then just call this sequence of actions “getting dressed”.

## C 1.2 Differentiate (i) object and action, and (ii) atomic actions and non-atomic actions

**Definition**
Being able to tell whether something (picture, sentence, sentence) refers to an object, an action, or else a sequence of actions.

**Background**
Objects and actions can be seen as metaphors of the concepts of *expressions* and *instructions* used in computer science. Typically, expressions have a value while instructions do not (they change the state of the environment [memory, screen, etc.]).

This competency refers to the ability to tell parts of an algorithm (e.g. recipe, movement, etc.) which have values (typically represented by nouns) from those which do not (typically represented by verbs).

Furthermore, some actions can be decomposed into sub-actions, in which case these are called non-atomic actions. Actions which cannot be further decomposed are called atomic actions.

**Examples**

1. Given words/groups of words, being able to say if it is an object (a mountain, a tree, an image, the number 5) or an action (jump, put on something, sing, think).

2. Given cards with words, being able to classify them into two groups (objects vs. actions), and then try to see which actions can apply to which objects.

3. Imagine (i) what actions one can perform on an object or (ii) on what objects an action can be performed.

4. Classify these actions into atomic and non atomic: get ready for school, put a sock on, dance, order animals by size, turn 90 degrees left, multiply a number by two, give the first letter of the alphabet,...

## C 1.3 Identify the input parameters of a sequence of actions

**Definition**
Being able to tell what is required (objects or sequences of actions) in order to perform a sequence of actions.

**Background**
Sequences of actions can be seen as functions (in a mathematical sense). Its input parameters are often simply called input.
When learning algorithmics, it is good practice to start with clearly and precisely list the input and the outcome (result) of an algorithm prior to writing the algorithm itself.

**Examples**

1. If I want to say who is the tallest out of three people, I need to know their size.

2. For getting dressed I need to know the pieces of clothing I have.

3. To prepare a recipe, I need to know the ingredients at hand.

4. To cut someone's hair, I need scissors and someone with hair.

# C 1.4 Describe the outcome of a sequence of actions

**Definition**
After executing a sequence of actions (informal, algorithm, program), being able to tell what happened. The learner should describe the final situation (which objects have been modified / created / suppressed, or what solution to a problem has been found).

Note that the construction of the sequence is not part of this competency, but of competency C 2.3.

**Background**
This competency is strongly linked to competency C1.3 above. Both competencies are needed to define effective sequences of actions.

**Examples**

1. Given a program (i.e. in Scratch) the learner can say that the cat has moved from top to bottom three times, or the turtle is in the car or the box contains 3 pens.

2. Given a recipe (without the title), the learner is able to tell what dish has been prepared.

3. Here is a map (grid 3x3), with start position (1,1), and end position (3,3):

| | | |
|---|---|---|
| | | End |
| | | |
| Start | | |

Here is an algorithm:

Move Up
Move Up
Move Right
Move Right

Imagine a robot is put on position (1,2), the learner should be able to execute this algorithm and tell if it allows the robot to reach the end position.

## C 1.5 Predict the outcome of a sequence of actions

**Definition**
Being able to tell, from a sequence of actions, what will happen if it is executed. In contrast to competency 1.4, this competency is about providing a *prediction* without actually executing the sequence of actions.

**Background**
This competency is important to prevent learners from overusing a "trial and error" approach (that is, changing the program as long as it fails without trying to understand/predict why).

**Examples**

1. Taking the grid from the last example illustrating competency C 1.4, being able to predict whether the robot will reach the end position, but with the constraint that the program/algorithm is not actually run.

2. Give children a starting position on a grid (e.g., 1,1 on the grid above) and a program (such as Move Up Move Right Move Down Move Left) and ask them what figure will the turtle draw if I run this program from this initial position.

3. Give the learners a criterion to compare persons (for instance, their size), and ask them what would be the outcome of comparing persons pairwise such that, at each step, the person with the greater value is reused in the next comparison.

## C 1.6 Using objects whose value can change

**Definition**
Being able to handle values which may vary depending on the context, and refer to them using named abstractions (also known as identifiers).

**Background**
This competency introduces the concept of computer variable, which corresponds to the association of a name with a memory slot. Introducing the name is called “*variable declaration*” and associating this name with a value is called “*variable affectation*”.

**Examples**

1. Give children cards with numbers. The teacher will spell out an addition piece by piece. The children will have to compute the result of the addition on the fly (without knowing its members in advance) and choose / update the result card.

2. Create a program in Scratch where every five clicks the cat meows, or where the score of the player is the number of times the cat has been clicked.

## C 1.7 Recognize existing objects and (sequences of) actions that can be used to reach a similar goal

**Definition**
Given a new problem (that is, a problem whose solution is not yet known), being able to realise that it is similar to an already known problem, and that an already known solution to this problem can be partly or fully reused to solve the new problem.

**Background**
Computational thinking also includes being able to build on already existing work rather than inventing everything from scratch.

Note that modifying a solution to adapt it to a new problem is discussed in competency C 2.5.

**Examples**

1. You have a list of recipes: tomato soup, strawberry cake, vegetable curry. If you need to make an onion soup, which of these recipes will you use?

2. The learners are given the algorithm to compute the biggest value out of three values. Then, the teacher asks which part can be reused to compute the smallest value out of three values.

# Competency 2: Compose/decompose a sequence of actions

## C 2.1 Order a sequence of actions to reach a goal

**Definition**
Given an unordered list of actions and a goal, being able to combine these actions in a valid order to build a sequence that achieves the goal.

**Background**
Before being able to create an entire sequence of actions (see C 2.3), an easier activity towards this competency would be to *order* already existing parts of a sequence of actions. So, the learner does not need to identify all the parts needed to achieve the goal, but only their correct order.

**Examples**

1. Let the learners imagine they have to bake a cake, and give them unordered parts of the recipe, they should be able to correctly order these parts (e.g. 1. Take a bowl, 2. measure the quantity of flour, etc).

2. Give the kids the different steps to get dressed and ask them to order them correctly (several ones are acceptable).

## C 2.2 Complete a sequence of actions to reach a simple goal

**Definition**
Given a limited sequence of actions and a simple goal (not relying on concepts such as conditions and loops), being able to complete the sequence of actions (by adding missing actions) to reach the goal.

**Background**
In algorithmics, learning often starts with getting inspiration from existing solutions (resp. problems). In this context, it is important to be able to evaluate how far is a solution from an existing solution, and to complete it if needed.

**Examples**

1. A drawing is almost finished and the learner has to add the missing action(s).

2. Considering the goal to bring a sprite to some place on the screen. An existing program does part of the way, the learner has to complete the missing actions (possibly copying and pasting already existing ones) towards bringing the sprite to the target location.

## C 2.3 Create a sequence of actions to reach a simple goal

**Definition**
Given a simple goal (not relying on concepts such as conditions and loops), being able to identify all the necessary actions and put them in the right order.

**Background**
This ability can be seen as an extension of the ability to order (see C 2.1) or complete (see C 2.2) an existing sequence of actions.

**Examples**

1. The learner is put in the middle of an empty room and given a set of possible actions. Ask them to tell the list of actions necessary to open a door: e.g. turn towards the door, walk to the door, grab the handle, push it downwards, pull the door.

2. Draw a square: Go forward, turn right, go forward, turn right, …

## C 2.4 Create a sequence of actions to reach a complex goal

**Definition**
Being able to identify and combine actions to reach a more complex goal (goal requiring conditions, loops, etc.).

**Background**
This competency is an extension of competency C 2.3 above.

**Examples**
1. Find your way through a maze without touching the walls.

2. Describe the necessary actions to add two integers.

3. Given a map with initial and final positions (possibly several checkpoints to visit), and additional rules (obstacles,...), describe the actions to reach the objective while satisfying the rules.

## C 2.5 Combine sequences of actions to reach a goal

**Definition**
Given already known sequences of actions achieving known goals, being able (i) to tell, in a given context (new goal to achieve), relevant from irrelevant sequences, and (ii) to connect them together in a valid order (that is, which allows to reach the new goal).

**Background**
Here the focus is on the *combination* of sequences. There is another layer of abstraction. It is not only about combining actions or objects but sequences (generally referred to by a name). This competency extends C 1.7 that only required to identify the reusable parts.

**Examples**

1. Take the sequences of actions for preparing individual dishes, and combine them (in the right order) to provide a full menu from the aperitif to desert.

2. Given two sequences of actions respectively drawing a square and a triangle, combine them to draw a simple house.

3. Make a complex game (with several levels) composed of smaller games that already exist.

4. Given the function max(a,b) that computes the maximum of a and b, identify that max(max(a,b),c) computes the maximum of a, b and c.

## C 2.6 Decompose goals into simpler subgoals

**Definition**
Given a complex goal, being able to split it into several subgoals, which can be achieved more easily. The subgoals can then be tackled by different people.

**Background**
When designing large software, learners often face a “blank sheet” wondering “where do I start?”. It is important to be able to tackle complex issues by first splitting them into smaller (and easier to solve) issues.
Here it is no longer about adapting a solution to a problem, but rather about adapting a complex problem to solutions by decomposing it.

**Examples**

1. What are the main steps to get the school bag ready for tomorrow? (e.g., find school bag, check the topics on tomorrow’s schedule, get material for each course into the bag)
2. Order a list of dishes for a group of people (e.g., get each one’s order, group them, send the compiled list to the kitchen)
3. Given a complex geometric form, calculate its surface by decomposing it into several simpler forms.

# Competency 3: Control a sequence of actions

## C 3.1 Repeat a sequence of actions a given number of times

**Definition**
Being able to appropriately use repetitions of (sequences of) actions *a given number of times* in order to reach a goal.

**Background**
A first famous family of repetitions in CS are the ones that are executed a given (known in advance) number of times (**for** loop in programming languages).

**Examples**

1. Ask the learners to write an algorithm that adds 3 egg yolks into the bowl. If the children don't see the advantage of a loop, then ask them to add 25 egg yolks.

2. Given a robot that can only turn 45° left, create a sequence of actions that allows him to follow a given path (every time the robot has to turn, the program will need to repeat the Turn Left action as many times as needed to turn the desired angle).

## C 3.2 Repeat a sequence of actions until a goal has been reached

**Definition**
Being able to control repetitions of (sequences of) actions *until a condition is met*, in order to reach a goal.

**Background**
Repetitions that are executed until a condition is met (the number of repetitions is not known in advance) are also a famous family in CS (**while** loop in programming languages).

**Examples**

1. Mix the ingredients until you get a smooth dough.

2. Move forward until you reach the wall (in this case we assume that there is a sensor to detect close obstacles).

## C 3.3 Create a sequence of actions which relies on simple conditions

**Definition**
Being able to use simple conditions (a simple condition can be expressed using a single criterion) to allow (or disallow) some (sequences of) actions.

**Background**
Controlling the execution of some (sequence of) actions by means of conditions is central in computational thinking, see also competency C 4.1 below.

**Examples**

1. **If** *the dough is thick* **then** add 20 cl of water and mix.

2. **If** *the robot faces the wall* **then** it moves backwards, **else** it stops.

3. Ask one person his/her age, and tell him (**depending on the answer**) “you’re an adult now” **or** “wait a bit before driving a car”.

## C 3.4 Create a sequence of actions which relies on complex conditions

**Definition**
Being able to combine conditions by means of logical operators (and, or, not) to allow (or disallow) some (sequences of) actions.

**Background**
When designing algorithms, it is very useful to combine several conditions to describe the solution to a problem.

While developing computational thinking,  it is important to realize that every condition is an object as any other (in CS they are called booleans). Boolean objects can be constant (as the expression “2>1” that is always true) or variable (“the temperature is above 10°C today” is not always true).

**Examples**

1. *If x is strictly less than 0* ***and*** *strictly greater than 24,* then it is not a valid hour.

2. If *the dough is thick* ***and*** *its temperature is below 10°C* ***and*** *we are on a Saturday* ***or*** *a Sunday* then go to the 24/7 shop **and** buy milk.

3. If *the robot faces the wall* ***and*** *there is a wall on its left too* ***or*** *the temperature is above 10°C* then it stop else it turns 45° twice**.**

# Competency 4: Evaluate objects or sequences of actions

## C 4.1 Compare two objects according to a given criterion

**Definition**
Given two (or more) objects, and a comparison criterion, being able to compare these objects with respect to the criterion.

**Background**
In CS, it is common to reason about objects with an order relationship, like numbers (order on reals or integers) or words (lexicographic order). Being able to apply this notion of order on various objects (age or size of people) is part of computational thinking.

**Examples**

1. The teacher chooses the criterion (size, weight, width,...) and the children have to order the objects depending on the order.

2. Make cards with objects on them (people, animals, things you find in a kitchen) and ask the kids to group them into several categories with respect to a criterion they chose (size, number of legs, things they eat, color of skin, material), and then ask them to change the criterion and the grouping. Notice that here, there is not always an order (e.g. the colors).

3. Compare weights using a beam balance. This one is a good tool since it allows to compare two weights (like a processor that applies binary operations). It prevents the learners to claim that they can compare all the values at once, as novices tend to do.

## C 4.2 Compare two sequences of actions according to a given criterion

**Definition**
Being able to compare sequences of actions with respect to several criteria: readability, number of lines, execution time, or other criteria.

**Background**
When a solution (sequence of actions) is found, it is tempting not to question its quality. Computational thinking is also about knowing that a problem may have 0, one or more solutions, and about being critical about solutions (which requires to be able to compare them).

**Examples**

1. There are several routes to travel from Nancy to Liège. Find the fastest, the most ecological, the cheapest,…

2. You have two robots, one that is fast moving but slow turning and the other one is fast turning but slow moving. Sometimes a longer way is more efficient than a shorter one, depending on the robot. Give several mazes to the students with different features (e.g., amount of turning moves required), and let them decide which robot is best suited for which maze.

3. I know how to cycle, drive and walk. Now, I am given a journey to make, which means of transport fits best (based on parking, distance, traffic...)?

## C 4.3 Improve a sequence of actions according to a given criterion

**Definition**

Being able to think about the possibility to improve a sequence of actions with respect to a criterion and modify it accordingly.

**Background**

As a follow-up of competency C 4.2, it is important, once a solution (sequence of actions) is known to be sub-optimal with respect to a given criterion, to be able to search for ways to improve it.

**Examples**

1. Give the learners  an algorithm for the robot to go from point A to point B, and ask for a path with less turns/jumps/steps.

2. Provide a recipe with repetitive actions (add one egg, adding one egg, add one egg), and ask the learners to make the code more readable by, e.g., adding a loop for adding the eggs.

# Competency 5: Translate between representations

## C 5.1 Represent objects or sequences of actions in various formal representations

**Definition**
Being able to use precisely defined representation systems to write down information (objects or sequences of actions).

**Background**
Computational thinking entails being able to use a precisely defined language (called formal language) which can be efficiently and unambiguously processed by another human or a machine. This competency 5.1 therefore refers to the ability to translate a word, a picture or a number (or a combination of these) into a precisely defined code/language (alphabet, symbols…). This translation is generally called *encodin*g in CS.

**Examples**

1. Translate the arabic number 2 into another formal representation (e.g. roman numerals, sign language, etc.)

2. Associate a number (using e.g. ASCII table) to a character of the keyboard.

3. Send a secret message to someone by ciphering and deciphering it using a code (using a correspondence table).

## C 5.2 Translate objects or sequences of actions from an informal representation to another representation (formal or not) and vice versa

**Definition**
Being able to describe objects or sequences of actions in different representation systems, no matter how precisely defined (formal *vs* informal representations) these are. This competency implies being able to switch from an informal representation to either another informal one, or to a formal one, and vice-versa.

**Background**
An informal representation refers for instance to expressions in everyday language (intuitive, not precisely defined). A formal one refers to a precise language with precise and unambiguous rules (which can be a graphical representation, a pictogram, a flowchart, an instruction in some programming language, etc).

**Examples**

1. Associate ambiguous pictures (that can have several interpretations) with descriptions of activities (informal => informal).

2. Tell a story to the learners, and they draw one picture for every scene (informal => informal)

3. Starting from a sequence of actions (movements of a person or a robot) the learner has verbalised in everyday language, use different pictograms with a precise signification (related to an action) to describe this sequence of actions (informal => formal)

4. Describe a sequence of actions represented in pictograms in your own words (everyday language) (formal => informal)

## C 5.3 Translate objects or sequences of actions between formal representations

**Definition**

Being able to use another formal language/code to represent a formally described object or sequence of actions.

**Background**

This competency is an extension of competency C 5.2 above, with the extra constraint that the representation systems for both pieces of information are based on precisely defined rules (formal representations).

**Examples**

1. Use a flowchart to represent the actions that have been described in pseudocode.

2. Have two different languages for a robot: on the one hand “go north, south, east, west” and, on the other hand, ”forward, turn left, turn right” and ask the children to translate a program from one of these two languages to the other.

# Competency 6: Build a sequence of actions iteratively

## C 6.1 Test a sequence of actions with respect to a given goal

**Definition**
Being able to check whether a given sequence of actions produces the expected outcome or not and explain why.

**Background**
It is possible to think about a problem in terms of input/outcome pairs (outcome for a given input data). These pairs are often called tests. In order to make sure that a given sequence of actions is valid (i.e. that it does what it has been designed for), one has to either prove it mathematically (using a formal representation of the sequence) or to check it with respect to all possible input/outcome pairs. The latter is generally not feasible (too many such pairs exist), and choosing the right (representative) pairs is not easy.
This competency therefore aims to develop in the learner the habit of verifying and iteratively amending a sequence of actions, based on tests.

**Examples**

1. Take the noun “Toto”, count how many letters it contains, and check it is 4.

2. Here is a map (grid 3x3), here is the entrance (position (1,1)) and here is the exit (position (3,3)):

| | | |
|---|---|---|
| | | Out |
| | | |
| In | | |

Here is an algorithm:

Move Up
Move Up
Move Right
Move Right

Check that when the robot starts at the entrance, and the algorithm is run, it leads it to the exit.

# C 6.2 Check and correct a sequence of actions with respect to a given goal

**Definition**
From a given sequence of actions which fails to reach a given goal, being able to modify it so that it does.

**Background**
Tests are not only useful to check whether a sequence of actions is valid. When these are well chosen, they also make it easier to find out why a given sequence is not valid, and thus to correct it.

**Examples**

1. Transform 120 grams into kilograms.
   The suggested sequence: Take the number of grams, move left two digits and put a dot → the outcome is 1.2kg. The student needs to identify the problem in this code and fix it.

2. Let us assume there is a map (grid 3x3) with (1,1) being the entrance and (3,3) being the exit:

| | | |
|---|---|---|
| | | Out |
| | | |
| In 😊 | | |

Does the following algorithm makes it possible to go from the entrance to the exit? If not, fix it.

Move  right
Move left
Move up
Move right

## C 6.3 Extend or modify a sequence of actions to reach a new goal

**Definition**
From a given sequence of actions whose outcome is known (the sequence is complete with respect to a previous goal), being able to reuse and update it to reach another (new) goal.

**Background**
In CS, to design complex programs, it is good practice to start with a simpler goal. We first write the corresponding program, simple but functional, and then we extend it step by step with new functionalities while ensuring that the modifications will not break what has been done so far. This method is called an iterative approach.

Note that this competency differs from C 2.2 where we started from an incomplete sequence of actions (that was thus not fulfilling its goal). Here we progressively extend a functioning sequence of actions, getting closer and closer to a final objective.

**Examples**

1. Having programmed a drawing (e.g. a square in Scratch), use it in another program to program a more advanced drawing (e.g., a flower (consisting of several squares slightly rotated from one another)).

2. The learners start from a program (given or that they have developed themselves) that implements a maze game where the player moves a character from the entrance to the exit without touching the walls. They then extend this program to add a ghost that moves on the screen, going through the walls, and that the character has to avoid or else the game is lost.