



# TEACHER TRAINING IN COMPUTATIONAL THINKING. TEACHING GUIDE



Tools, reflections, learnings and practical examples resulting from the PECOFIM project

[pecofim.udg.edu](http://pecofim.udg.edu)

Funded by:





“Teacher training in Computational Thinking. Teaching Guide” is a document resulting from the PECOFIM project [2015 ARMIF 00031], funded in the call Ajuts de Recerca per a la Millora en la Formació Inicial de Mestres, granted by the Agència de Gestió d'Ajuts de Recerca (AGAUR).

It has been written by **[Estebanell, M.; López, V.; Peracaula, M.; Simarro, C.; Cornellà, P.; Couso, D.; González, J.; Alsina, A.; Badillo, E.; Heras, R.]**.

Assisted by: **[Freixenet, J.; Muntaner, E.; Sabaté, F.; Serrats, L.; Córdoba, P.; Garcia, N.; Niell, M.; Bertran, A.; Bosch, D.]**

Design and layout: **[Ferrarons, A.]**

Translation and proofreading: **[Weiss, M.]** (english), **[Bosch, D.]** (spanish)

This document is distributed under a Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International license. <http://creativecommons.org/licenses/by-nc-nd/4.0/>

Most of the images used in the document are our own or have been obtained under a Creative Commons license. In the remaining cases, the licenses have been detailed at the foot-image.

Credit as:

**[Estebanell, M.; López, V.; Peracaula, M.; Simarro, C.; Cornellà, P.; Couso, D.; González, J.; Alsina, A.; Badillo, E.; Heras, R.]** (2018). Pensament Computacional en la formació de mestres. Guia didàctica. Girona: Servei de Publicacions UdG.

# TEACHER TRAINING IN COMPUTATIONAL THINKING

## Tools, reflections, learnings and practical examples resulting from the PECOFIM project

Computational thinking, understood as the set of strategies and reasoning required for problem solving and designing systematic solutions, in much the same way as a machine or computer operates, is increasingly present in the educational world. Nowadays, programming activities and educational robots are present both in formal education and in multiple non-formal spaces (campuses, code clubs, museums, etc.), while educational projects and networks promoting this type of activities are emerging everywhere. At the same time, school curricula have begun to include programming and educational robotics as proposals to work on problem solving and help to develop digital and/or mathematical skills. What, though, are the necessary strategies and capabilities that are encompassed in computational thinking (CT)? How should computational thinking be conceived as school content? What can universities, responsible for the initial training of the teachers, do to promote the necessary teaching competence in this field?

This document, resulting from the PECOFIM project, aims to address these different issues.

The PECOFIM project (Computational Thinking in the Initial Teacher Training Process) has been funded by the Research Assistance Program for Improvement in Initial Teacher Training (ARMIF), modality 2, of the Agency for Management of University Grants and Research (AGAUR) -resolution of concession of June 29, 2016, published on June 30, 2016. This project has involved university professors from the University of Girona and the Autonomous University of Barcelona, as well as teachers from different Catalan schools with extensive experience in carrying out activities related to programming and robotics in educational contexts.

This document shares a didactic framework encompassing computational thinking and the role it can play in school as an object of teaching and learning.

Based on practical resources and examples, we hope to contribute to the training of teachers in computational thinking, addressing the what, how and why of computational thinking in school.



El Pensament  
Computacional en la  
Formació inicial de Mestres

Project website: [pecofim.udg.edu](http://pecofim.udg.edu)

# What you will find in this document?

<b>1. What do we mean by Computational Thinking?</b>	<b>6</b>
<b>2. What is the meaning of Computational Thinking in school?</b>	<b>7</b>
2.1. Beyond computational concepts: Computational thinking understood as a set of practices and perspectives	7
2.2. Beyond digital support: Computational thinking understood as a methodology for solving everyday problems with and without a computer	8
2.3. Beyond computing: Computational thinking understood as a cornerstone in the emerging paradigm STE(A)M	9
<b>3. What aspects of Computational Thinking need to be promoted and evaluated?</b>	<b>10</b>
3.1. Characteristic processes of Computational Thinking	13
3.2. Strategies for solving computational problems	18
3.3. Transversal skills in computational contexts	23
<b>4. How can we approach Computational Thinking in teacher training?</b>	<b>28</b>

# 1. What do we mean by Computational Thinking?

In 1980, Seymour Papert introduced the term “computational thinking” in his book *“Mindstorms: Children, computers, and powerful ideas”* (translated into Spanish as “Desafío a la mente”). In 1996, the same Papert, presented the idea in the context of mathematical learning in his article *“An Exploration in the Space of Mathematics Education”*, but it was not until 2006 that Jeannette M. Wing popularized the concept of computational thinking in the field of educational and psychological research, publishing an article in the journal *“Communications of the ACM”* (2006). Specifically, Wing explains:

“Computational thinking as solving problems, designing systems and understanding human behaviour by drawing on the concepts fundamental to computer science.” (Wing, 2006: p 33).

The author suggests that this way of thinking is applicable to the resolution of diverse problems, being a fundamental skill for the whole population and not only for computer scientists and programmers. From this perspective, she emphasizes the need to integrate computational ideas in other disciplines, posing solutions that could also be carried out by humans, and not just by machines.

In the ten years following Wing’s first publication, many authors have focused their attention on the idea of computational thinking, contributing complementary definitions. Berry (2014) and Selby & Woollard (2014) have proposed their own definitions, which in spite of their nuances coincide in understanding computational thinking as the ability to identify problems that can be solved in a way similar to what a programmer would do when giving instructions to a computer using a programming language:

- divide complex problems into smaller size modules,
- sequence long and complex processes in “steps”,
- organize and analyze data recognizing logical patterns,
- start from specific cases to arrive at abstract and generalizable situations,
- use algorithms to automate solutions and
- evaluate the validity of solutions.

## TO LEARN MORE

- Berry, M. (2014). *Computational Thinking in Primary Schools*.  
<http://milesberry.net/2014/03/computational-thinking-in-primary-schools/>
- Selby, C. & Woollard, J. (2014) *Refining and understanding Computational Thinking*.  
<https://eprints.soton.ac.uk/372410/1/372410UnderstdCT.pdf>
- Wing, J. (2006). *Computational Thinking*.  
COMMUNICATIONS OF THE ACM. Vol. 49, No. 3.  
<https://dl.acm.org/citation.cfm?id=1118215>

## 2. What is the meaning of Computational Thinking in school?

Computer technology, closely linked to computational thinking, is not in itself new in school. For decades, in many educational centres the “computer classroom” has been integrated into the school ecosystem. This space, despite having undergone changes associated with technological development, both in terms of devices and software, has often been associated with working with the computer (sometimes as a subject of its own, sometimes integrated in other areas of knowledge).

In our view, the perspective of computational thinking aims to transcend the traditional view of the computer classroom delimited in space and time, understanding school activities related to programming and robotics as an educational strategy for development of a 21st century competency. This should not be an exclusive skill of professionals linked to STEM careers (Science, Technology, Engineering and Mathematics), but of all citizens. In this context, computational thinking in school is presented based on three premises:

- Beyond computational concepts: computational thinking understood as a set of practices and perspectives.
- Beyond digital support: computational thinking understood as a methodology for solving everyday problems with and without a computer.
- Beyond computing: computational thinking understood as a cornerstone in the emerging paradigm STE(A)M.

### 2.1. Beyond computational concepts: Computational thinking understood as a set of practices and perspectives

While computational thinking is often associated as a set of conceptual contents to be taught (the key concepts in learning programming, which are sequences, loops, parallelism, events, conditionals, operators and data), different proposals have also highlighted other dimensions of computational thinking, associated with procedural and epistemological content. On the one hand, we talk about computational practices including those activities related to work in computational contexts and with computational supports. Some examples of these practices are incrementing and iterating processes, testing and debugging of developed programs, reusing and combining programs, and abstracting and modularizing. Finally, we also talk about computational perspectives as the set of attitudes and viewpoints of the discipline itself, which include aspects such as expressing, connecting, questioning and feeling empowerment to create.

#### TO LEARN MORE

- Brennan, K. & Resnick, M. (2012), *New frameworks for studying and assessing the development of computational thinking*. *Proceedings of the American Educational Research Association 2012*, pp. 1-25. <http://scratched.gse.harvard.edu/ct/files/AERA2012.pdf>
- Stephenson, C., & Barr, V. (2011). *Defining Computational Thinking for K-12*. CSTA:- *The Voice of K-12 .Computer Science and Its Educators*, 7(2).

## The dimensions of Computational Thinking according to Brennan and Resnick (2012)

### Concepts

- Sequences
- Loops
- Parallelism
- Events
- Conditionals
- Operators
- Data

### Practices

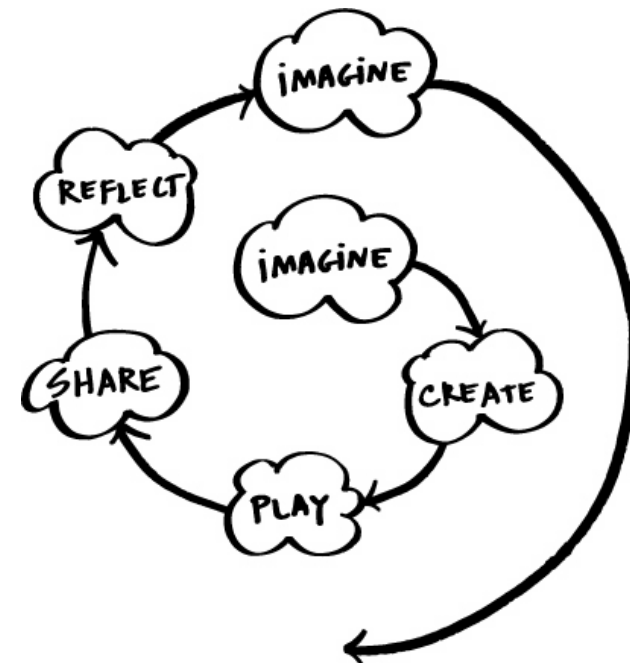
- Decompose problems
- Make incremental changes
- Test and correct
- Reuse
- Persist

### Perspectives

- Express: new means to create.
- Connect: be able to create with others.
- Question: be able to use computation for posing questions and resolving them.

## 2.2. Beyond digital support: Computational thinking understood as a methodology for solving everyday problems with and without a computer

Although computational thinking is associated with the use of digital technologies, we propose to consider it as a way of thinking, doing and communicating that transcends digital support, and which has multiple applications in the resolution of problems of all kinds, particularly everyday ones.



*Spiral of creative thinking according to M. Resnick (2007).*

<http://web.media.mit.edu/~mres/papers/CC2007-handout.pdf>



In a similar direction, Beauchamp (2016) stresses the close relationship between computational thought and the educational approach of “Problem Based Learning”, since all problem solving has different phases: understanding the problem, considering a design, executing it and checking the result.

Additionally, Resnick (2007) defends the close relationship between computational thinking and creative thinking using the term “creative computing” and emphasizes how children learn in the stage of early childhood education, where frequently a learning process takes place with a common denominator: imagine, create, play (experience), share, reflect and re-imagine. According to Resnick, one of the best ways we have to help children and young people is to be sure they have the opportunity to follow their interests, explore their ideas and have tools to make their voices heard. It is not simply a question of teaching them to use technology, but of offering them opportunities to use technology to create things.

#### TO LEARN MORE

- Beauchamp, G. (2016). *ICT and computing in the primary school*. In *Computing and ICT in the Primary School: From pedagogy to practice* (p. 252).
- Griffin et al. (2012). *Assessment and Teaching of 21st Century Skills*  
<http://www.springer.com/us/book/9789400723238>
- Resnick, M. (2007). *All I Really Need to Know (About Creative Thinking) I Learned (By Studying How Children Learn) in Kindergarten*. *Proceedings of the 6th ACM SIGCHI conference on Creativity & cognition*, 1-6

## 2.3. Beyond computing: Computational thinking understood as a cornerstone in the emerging paradigm STE(A)M

The idea of computational thinking has gained increasing interest in recent years in parallel to the emergence of the STEM paradigm (Science, Technology, Engineering and Mathematics). This has arisen as much from the growing demand for professionals with profiles of a scientific, technological, engineering and mathematical nature, as from new educational proposals that call for a greater interrelation between these disciplines throughout schooling. In parallel, we encounter the emergence of makerspaces in education, as well as a commitment to promote the STEAM area, where the arts play a fundamental and transformative role, which allows diversification of tasks and work to be developed, being complementary and helping to express, connect and understand the knowledge being studied.

When considering how computational thinking fits in the STEAM educational environment, we want to emphasize both the opportunities offered by computational thinking as an essential tool in the STEAM field and the opportunities offered by the STEAM field as a context for developing computational thinking. Computational languages and supports offer new opportunities for students to face scientific, engineering and mathematical challenges, such as computational modelling, simulating real-world situations in virtual environments, representing mathematical abstractions, etc. In fact, one of the main documents of reference at present in the STEAM field, the K-12 Next Generation Science Standards (NRC, 2012), includes the use of computational thinking as one of the eight key STEM practices. In addition, developing computational thinking may not only help students to learn more STEAM content, but also to better understand the role of computation in the STEAM professional field and its impact. For example, Weintrop (2016) highlights how computational biology, based on data structures and algorithms, is transforming the very way we think of biology as a science.

At the same time, however, learning in the STEAM field cannot only be enriched by the development of students' computational thinking, but also vice versa. The contexts particular to the STEAM field (construction of scientific explanations and mathematical models, design of engineering solutions, etc.) are especially

suitable for promoting and giving meaning to the use of computational languages, and provide relevant phenomena that are addressed from the computational perspective. In fact, it has often been the STEAM disciplines that have contributed ways of thinking and reasoning from which computational thinking is nourished, such as logical-mathematical thinking or investigative skills.



Example of emerging fields of scientific research based on computational support.

## TO LEARN MORE

- NRC. (2012). A Framework for K-12 Science Education: Practices, Crosscutting Concepts, and Core Ideas. Washington, DC.: National Academy of Sciences.
- Computer Science Teachers Association: Computational Thinking resources. <http://www.csteachers.org/page/CompThinking>
- ISTE's Computational Thinking Toolkit. <https://www.iste.org/explore/articleDetail?articleid=152>
- Weintrop, D., Beheshti, E., Horn, M., Orton, K., Jona, K., Trouille, L., & Wilensky, U. (2016). Defining Computational Thinking for Mathematics and Science Classrooms. Journal of Science Education and Technology, 25(1), 127–147.

### 3. What aspects of Computational Thinking need to be promoted and evaluated?













While there is no single definition of the core elements of computational thinking, in the specialized literature different proposals can be found that detail their fundamental aspects.

Wing, 2006	Stephenson and Barr, 2011	Selby and Woollard, 2013	Selby and Woollard, 2014
<ul style="list-style-type: none"><li>• Recursive thinking and parallel processing.</li><li>• Reformulate a problem that initially seems complicated into a problem that we know how to solve.</li><li>• Simplification, integration, transformation, simulation.</li><li>• Choose a suitable representation, or model the most relevant aspects of a problem in order to make it manageable.</li><li>• Interpret codes as data and data as codes.</li><li>• Use abstraction and decomposition to address large and complex tasks.</li><li>• Judge a design based on its simplicity and elegance.</li></ul>	<ul style="list-style-type: none"><li>• Formulate problems in a way that enables us to use a computer and other tools to help solve them.</li><li>• Logically organize and analyze data.</li><li>• Represent data through abstractions such as models and simulations.</li><li>• Generate automatic solutions through algorithmic thinking (a series of ordered steps).</li><li>• Identify, analyze and implement possible solutions with the aim of achieving the most efficient and effective combination of steps and resources.</li><li>• Generalize and transfer the process of solving one problem to a wide variety of problems.</li></ul>	<ul style="list-style-type: none"><li>• The ability to think in abstractions.</li><li>• The ability to think in terms of decomposition.</li><li>• The ability to think algorithmically.</li><li>• The ability to think in terms of evaluations.</li><li>• The ability to think in generalizations.</li></ul>	<ul style="list-style-type: none"><li>• A mental process.</li><li>• Abstraction.</li><li>• Decomposition.</li><li>• Heuristic reasoning.</li><li>• Logical thinking.</li><li>• Mathematical thinking.</li><li>• Thought focused on engineering.</li><li>• Algorithmic design.</li><li>• Problem solving.</li><li>• Analysis.</li><li>• Evaluation.</li><li>• Generalization.</li><li>• Recursion.</li><li>• System design.</li><li>• Automation.</li><li>• Modelling, simulation and visualization.</li></ul>

# Key aspects of Computational Thinking

Based on these different proposals, we present a list of key aspects to promote and that could be evaluated to help develop CT in school:

- Characteristic procedures of computational languages
- Strategies for solving computational problems
- Transversal skills in computational contexts

Characteristic processes of computational thinking	Strategies for solving computational problems	Transversal skills in computational contexts
 PROCESS COMPUTATIONAL DATA AND VARIABLES	 IDENTIFY AND DELIMIT	 CREATIVITY AND INGENUITY
 REPRESENT COMPUTATIONAL ABSTRACTIONS	 CONSIDER MULTIPLE PATHS	 TEAMWORK
 AUTOMATE WITH COMPUTATIONAL ALGORITHMS	 BREAK DOWN AND SIMPLIFY	 AUTONOMY AND INITIATIVE
 SEQUENCE COMPUTATIONAL STEPS	 TEST, VALIDATE, DEBUG	 INTERACTION AND COMMUNICATION

## TO LEARN MORE

- Wing, J. (2006). Computational Thinking. COMMUNICATIONS OF THE ACM. Vol. 49, No. 3. <https://dl.acm.org/citation.cfm?id=1118215>
- Stephenson, C., & Barr, V. (2011). Defining Computational Thinking for K-12. CSTA:-The Voice of K-12 Computer Science and Its Educators, 7(2).
- Selby, C. & Woollard, J. (2013) Computational Thinking: The Developing Definition. <https://eprints.soton.ac.uk/356481/>
- Selby, C. & Woollard, J. (2014) Refining and understanding Computational Thinking. <https://eprints.soton.ac.uk/372410/1/372410UnderstdCT.pdf>

### 3.1. Characteristic processes of computational thinking

One of the principal aspects of CT is the use of computational languages. While there are a large variety of languages (code languages, block, icons, Mixed-Signal, etc.), some common denominators do exist. One of these denominators is the processing **of data and computational variables**, understanding where they come from, what their meaning is and how they can be used to generate new data, as well as how to recognize patterns to organize this data. The computational challenges faced by students in school often require working with numerical data and its systematic treatment, identification of the variables that allow us to describe and control a computational process, and the relationship between these variables.

That is why we also say that these challenges require developing the ability to **represent computational abstractions** of all kinds: conditions, patterns, geometric relationships, algebraic relationships, etc., which are expressed through code.

In fact, before programming with code, it is advisable for students to first represent their ideas with pencil and paper, as well as through experiential games, and then translate them into computational language.

In addition, one of the most common ways to represent and structure the relationships between variables and the conditions that determine them is through **computational algorithms**. This enables us to automate the operation of a computer program or the behaviour of a robot, using loops, conditionals, operators, variables and parameters, forming an algorithm to solve particular cases of a general problem.

In turn, this depends on developing abilities related to **sequencing computational steps**, knowing how to optimally order the instructions that make up a program by using sequences, repetitions or parallelisms.



PROCESS COMPUTATIONAL  
DATA AND VARIABLES



REPRESENT  
COMPUTATIONAL  
ABSTRACTIONS



AUTOMATE WITH  
COMPUTATIONAL  
ALGORITHMS



SEQUENCE  
COMPUTATIONAL STEPS



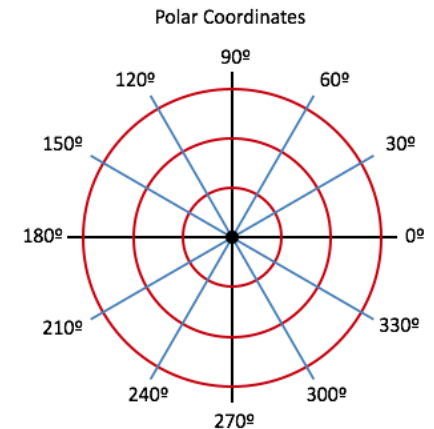
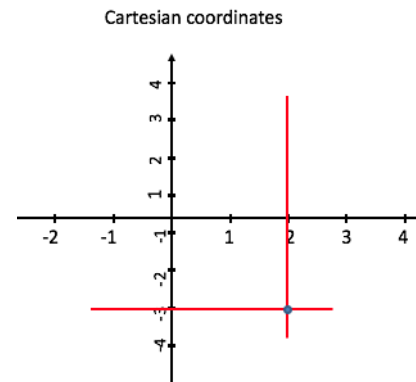
## PROCESS COMPUTATIONAL DATA AND VARIABLES

### PRACTICAL EXAMPLE

Choose the coordinates that help us better define a movement in the plane

When it is proposed to a group of children to design a small video game with Scratch, one of the main challenges encountered is to define the movement of the characters. This language allows us to define this movement from the Cartesian coordinates (X, Y) as well as from the polar coordinates (direction and displacement). Depending on how they want to interact with the characters, the children will have to choose which type of coordinates can be used as data in their code.

For example, if they want to define the movement of a character that hits the ends of the screen, the data that will be most useful to them is the direction expressed in degrees, whereas if they want to define the vertical fall of a character, the data that will be most useful is the Y coordinate.





# REPRESENT COMPUTATIONAL ABSTRACTIONS

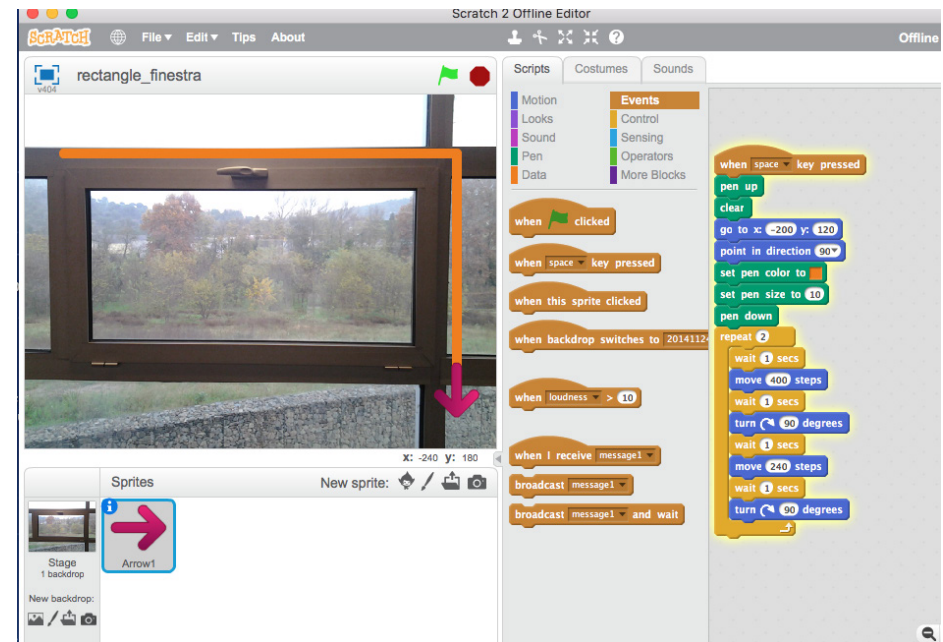
## PRACTICAL EXAMPLE

Reproduce the polygons we see around us

We can propose to the children to identify flat geometric shapes in their physical environment, so that they can photograph them and describe them mathematically with the help of the Scratch programming environment.

To do this, they can position one of the photographs as a backdrop, and program a character so that with one click, they can trace the outline of the shape that appears.

This requires a process of abstraction where data and variables come into play including the length of the sides of the polygon and the angle that the character must rotate on each vertex.







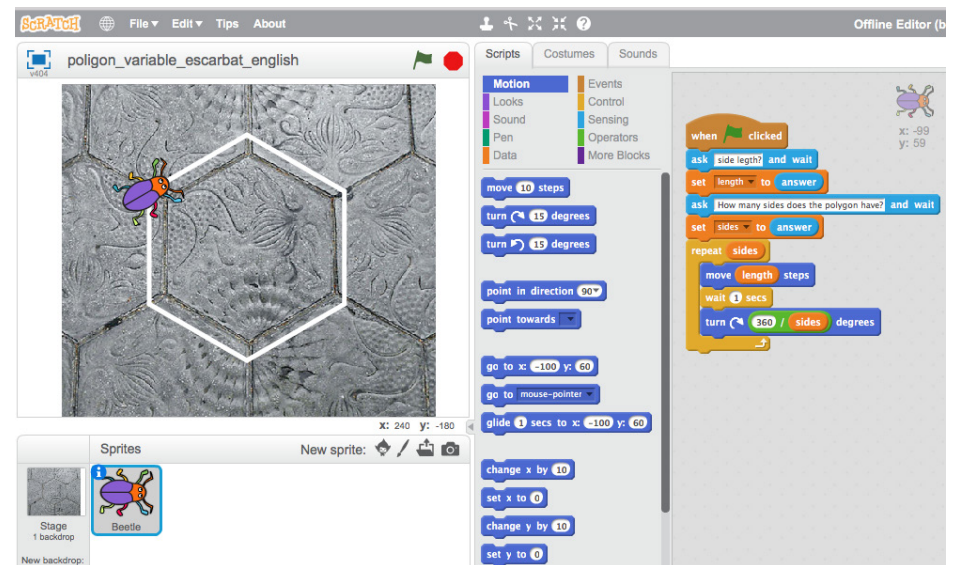
# AUTOMATE WITH COMPUTATIONAL ALGORITHMS

## PRACTICAL EXAMPLE

Regular polygons: how can we generalize their mathematical expression?

Continuing with the previous example, in the event that the shape is a regular polygon, the children can find the relationship between the angle of rotation and the number of sides of the polygon, relating the two parameters with an operator, which in this case will be division.

They will thus be able to construct a general algorithm to draw the outline of any regular polygon through the relationship that states that each angle of rotation is 360 divided by the number of sides of the polygon.







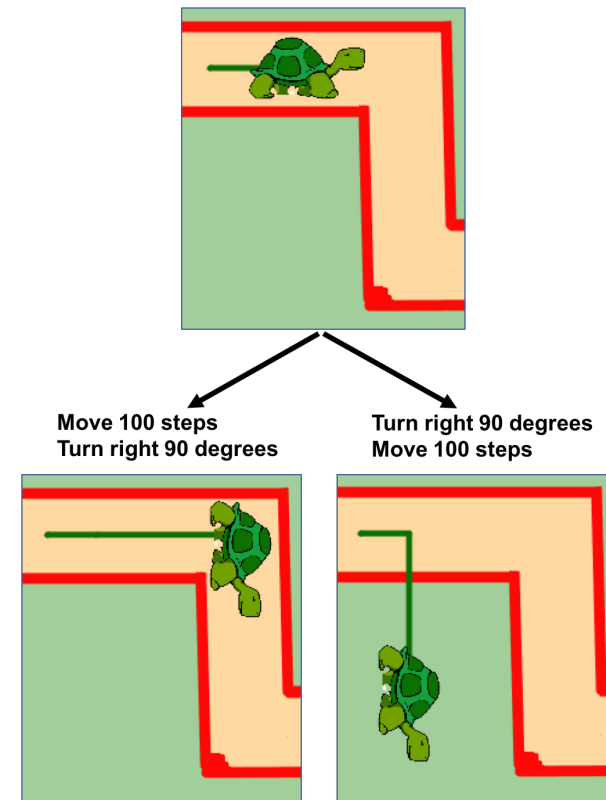
## SEQUENCE COMPUTATIONAL STEPS

### PRACTICAL EXAMPLE

The instructions that are given to a robot do not fulfil the commutative property

If we propose to a group of children to program a robot, or a character that has to follow a specific itinerary, they will have to define a sequence of commands or instructions for this character to follow.

If the two instructions available are “Advance 100 steps” (movement of translation) and “Turn 90 degrees to the right” (rotation movement), the children can verify that the order in which the sequence of instructions is applied does not lead to the same result, and that, therefore, the instructions given to the robot do not fulfil the commutative property.



## 3.2. Strategies for solving computational problems

In parallel to the mastery of key elements of computational languages (their components, structures, rules, etc.), CT has a close relationship with the way in which people face problem solving.

Problem-solving strategies, in fact, have been widely discussed in the educational literature under different umbrellas (problem-solving skills, critical thinking, etc.). However, the skills to develop these strategies are not universal and separate from the contexts in which they are applied, and, therefore, they take on a very particular meaning when it comes to confronting and solving computational problems.

Thus, computational thinking implies the ability to **identify and delimit** a computational problem to be solved, analyzing its possible solutions and optimizing the steps and resources. When this is done, we must also **consider multiple paths**, understanding that there may be several valid paths or solutions.

Problem solving also involves the ability to **break down and simplify**, by modularizing and dividing complex situations into simpler ones. Last, it is also necessary to **test, validate and debug** solutions iteratively, learning from the errors identified, since in computation, debugging involves searching for the error and understanding what is not working.



IDENTIFY AND DELIMIT



CONSIDER MULTIPLE  
PATHS



BREAK DOWN AND  
SIMPLIFY



TEST, VALIDATE, DEBUG



## IDENTIFY AND DELIMIT

### PRACTICAL EXAMPLE

**We ask ourselves what we have and what we need before we start programming a programmable educational robot (Bee-Bot, Cubeto, others)**

When a group of children find themselves faced with the challenge of programming a robot that has to move from one place to another, it is advisable for them to reflect upon the problem they are facing in advance. For example, it is not the same thing to define a movement based on a starting point and a point of arrival, as arriving at a point from a starting point and having made a movement.

It is necessary, therefore, to delimit exactly what the challenge is that they have to address, and identify possible ways to solve it.





## CONSIDER MULTIPLE PATHS

### PRACTICAL EXAMPLE

#### Build a tower with cups (I)

A very common activity designed to reflect on the need for computational language to give clear instructions to a robot is to emulate the programming of a robotic arm that has to build a tower of cups with a given structure.

Divide the children or young people into teams. Some of these can be programmers, charged with writing the program that will allow the robotic arm to build the structure, using a previously agreed-upon set of symbols (indicating to move left, right, up, down, rotate the cup, etc.).

Once they have written the program, another group of children or young people who had not participated in the writing, will execute the instructions and physically construct the tower with real cups. The programmers will have necessarily discussed and reflected on the different ways of building the tower, on what order to position the different cups and the advantages of each option.





## Build a tower with cups (II)

A possible activity to work on debugging could be to provide an example of a code that does not do what it should do, and have the children analyze this code in order to search for the error, thus doing an exercise that involves understanding why a program does not work and correct it to achieve a possible solution.





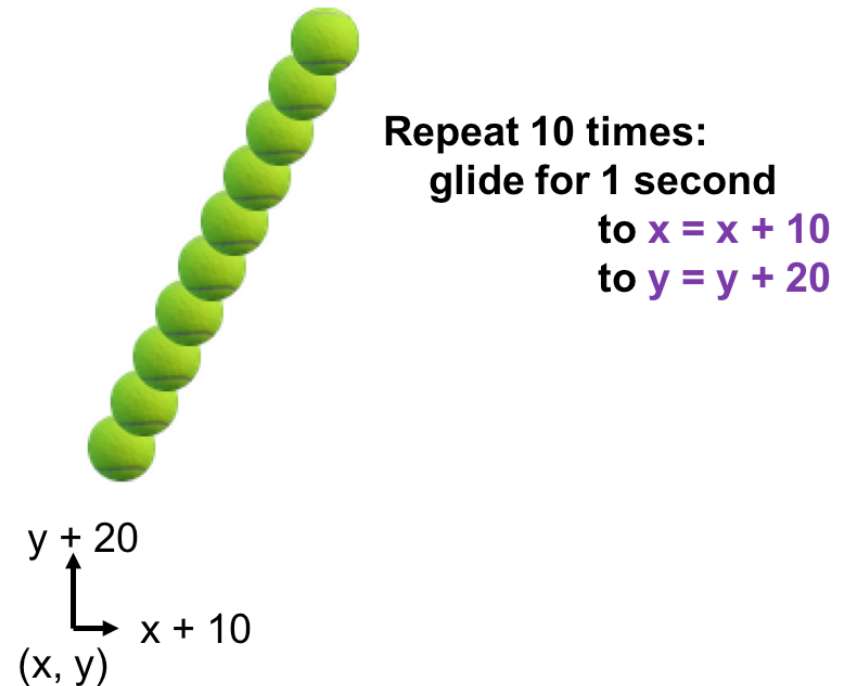
## BREAK DOWN AND SIMPLIFY

### PRACTICAL EXAMPLE

Break down a complex movement into two simpler ones

Children often want to computationally reproduce complex behaviours or phenomena that require previous simplification based on translating a large challenge into several small ones.

Consider, for example, the composition of movements. A complex movement can be composed of several simpler ones, such as a diagonal movement that may be made up of a vertical and a horizontal one.



### 3.3. Transversal skills in computational contexts

In recent years, many proposals have been made to formulate transversal competencies and/or competencies of the 21st century, which encompass classical notions of HOTS (Higher Order Thinking Skills), also called soft skills, and other aspects to take into account the demands of the dynamics—global and profoundly digital of today's society. As with problem solving, the cognitive literature seems to indicate that every competence is developed and expressed in a specific way according to the context, and therefore, the question is not so much which skills are transversal in the abstract or disassociated from school content, but what meaning they take on in computational contexts like those discussed in this document.

Thus, we speak of transversal skills such as **creativity, teamwork, personal autonomy and initiative or interaction and communication** not as transversal competencies to develop “apart” from computational thinking, but within computational thinking. Put another way: it is not necessary to focus on teaching creativity or teamwork in itself, but rather to teach computation by developing creativity and teamwork.



CREATIVITY AND  
INGENUITY



TEAMWORK



AUTONOMY AND INITIATIVE



INTERACTION AND  
COMMUNICATION





## CREATIVITY AND INGENUITY

### PRACTICAL EXAMPLE

Think of ingenious solutions to make the robot follow a line

When a group of children have a robot with wheels and a light sensor, which must be guided using a coloured line drawn on the ground, there are several creative ways to face the challenge.

For example, they can place the robot on the margin of the line (boundary between light and dark) so that the robot moves forward in small sections, turning toward the dark colour when the sensor detects brightness and turning toward the light colour when the sensor detects darkness. To make each turn, only one of the two wheels moves and advances; instead of moving the two wheels in a straight line, each one moves alternately.

To work on creativity it is important to design workshops where different materials are used, let children express their voice, and let them follow their interests. Encourage them to invent poems, stories, games and robots based on what they like, and learn to reuse and mix projects and solutions and freely express their ideas.







## TEAMWORK

### PRACTICAL EXAMPLE

#### Create a collaborative multimedia story

The development of computational programs at a professional level often requires teamwork, and this can be reproduced analogously in the classroom.

For instance, we can propose to a group of children the challenge of creating a story by working collaboratively. Initially, the general storyline is agreed upon among the whole class, but then the children are divided into groups and each one of these groups creates a part of the story, writing a script and programming its digital animation with Scratch.

To view the final result, the students set the computers up in a row and start their programs at the same time, so that characters appear to jump from one screen to another. Coordination between groups must allow both the argumentative coherence of the parts of the story and the coherence of the programs, guaranteeing the synchronization of events based on the agreed timeline from the moment the programs start running.





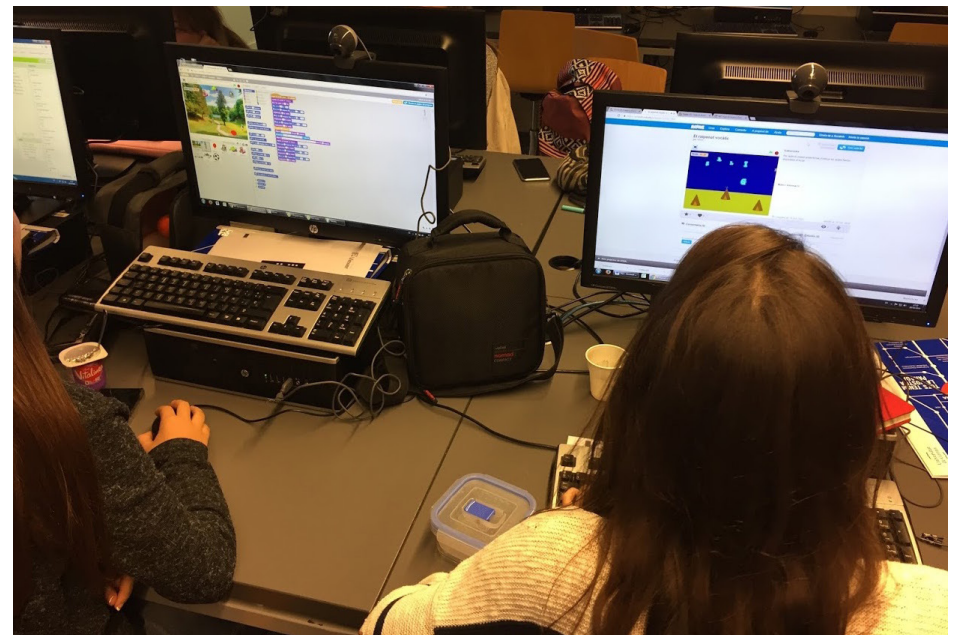
## AUTONOMY AND INITIATIVE

### PRACTICAL EXAMPLES

Decide on the degree of difficulty we want to incorporate when we design a video game

With the final goal being for each work group to create a video game that we can play together, different video game creation tools are presented in the classroom, ranging from the simplest to the most complex: Twine, Scratch, Kodu, Stencyl and Unity. The learning curve of each application is different. This means that if a group chooses to develop their project with a tool that has a certain degree of complexity, they will have to allocate a significant portion of their time to learning, on their own, the operation of the tool.

Experience tells us that approximately a third of the groups decide to develop the video game with a tool that requires a high degree of autonomy to find solutions to the problems that will arise in the course of development of their project.





## INTERACTION AND COMMUNICATION

### PRACTICAL EXAMPLE

Think about how to give the clearest and most understandable instructions to the Child-bot

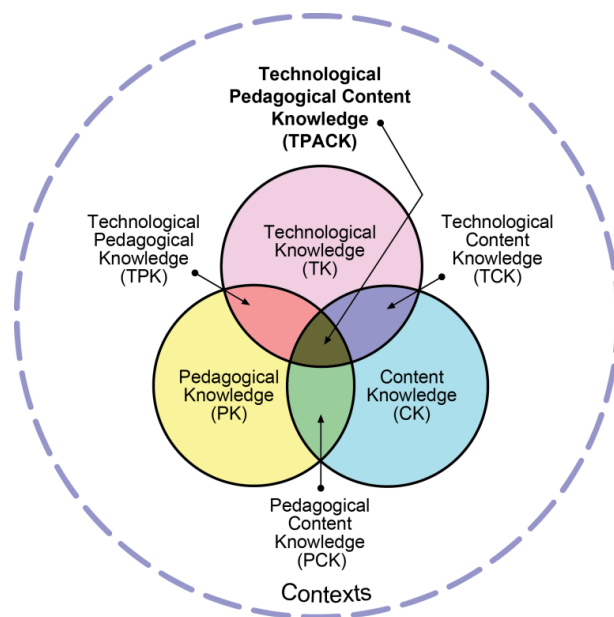
In many workshops and projects that are carried out in the classroom it can be quite valuable, at a given time, for a student to explain a finding she made to her classmates. Similarly, at the end of a workshop she could present the results and process to the class (or to the whole school), thus documenting her work. Still another possibility would be to have the children write down “advice” they would give to a classmate if they had to repeat the workshop.

For example, in an activity called “Child-bot” (*child who runs a program as if he was a robot*), children can write the codes themselves to give the child-robot instructions in what can be called the language of robots (a set of agreed-upon symbols). The children doing the programming must be able to communicate with their classmates-robots, giving them instructions in a clear and understandable way.



## 4. How can we approach Computational Thinking in teacher training?

Due to its being an emerging field, computational thinking still plays an insignificant role in initial teacher training carried out in universities. Some studies have identified this shortcoming in teacher training (Bower & Falkner, 2015). Addressing computational thinking in initial teacher training can be conceived as an integration of what has been called the didactic-technological knowledge of content (Shulman, 2005), usually called TPCK for its acronym in English (also known as the TPACK model, Technological Pedagogical and Content Knowledge). TPCK can be understood as “the ability of teachers to transform their knowledge of content into forms that are didactically powerful and yet adapted to the variety presented by their students in terms of skills and background”.



Model TPCK, or TPACK, Technological Pedagogical and Content Knowledge (Koehler, Mishra and Cain, 2013)

LICENCE: “Reproduced by permission of the publisher, © 2012 by tpack.org”

As with other teacher training processes, this training must be understood at different levels, which are combined and interrelated throughout the learning progression of future teachers during their initial training.

If the ultimate goal of teacher training in computational thinking is for them to be capable of generating learning situations with their future students, it is crucial that the teachers being trained experience firsthand the learning and development of their own computational thinking: a knowledge of existing tools and computational languages, the development of reasoning strategies and problem solving, etc. This is only possible if future teachers themselves face the challenge of solving computational problems as students / learners: program a robot, design a video game or interactive story, develop an app, or also solve computational problems “unplugged”. Therefore, we consider that a necessary first level in teacher training is the **user level**. It is not intended at this level of training that the future teacher consider how to help another person (or their potential students) develop computational thinking, but rather that they raise questions such as how to use a particular computational language to achieve some of the proposed goals (the robot, the video game, the app, etc.).

Only after a person has experienced the fact of facing a computational challenge by designing a small program, developing a code, etc., is it possible to reflect on what they had to do. We understand, therefore, the **reflective user level** as that level where the future teacher is reflecting on what they have done when they had to face a computational challenge: what reasoning did they use? What elements of computational language did they use and why? This is an exercise in metacognition in which the 12 key aspects discussed in the previous section are particularly relevant, since they allow us to reflect on our own actions in a structured way.

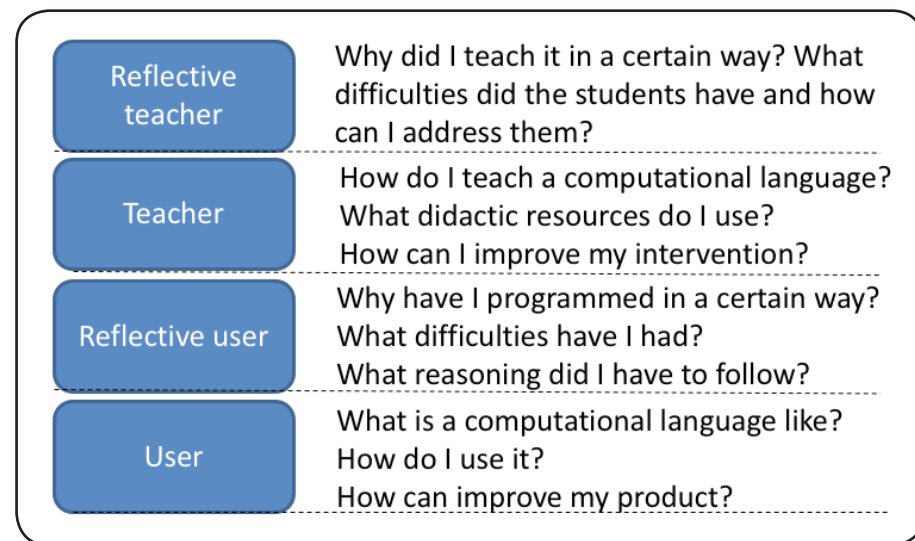
### TO LEARN MORE

- Bower, M. & Falkner, F. (2015) Computational Thinking, the Notional Machine, Pre-service Teachers, and Research Opportunities. Proceedings of the 17th Australasian Computing Education Conference, 27-30.
- Koehler, M. J., Mishra, P., & Cain, W. (2013). What is technological pedagogical content knowledge (TPACK)? Journal of Education, 193(3), 29-37
- Koehler, M. J., Mishra, P., & Cain, W. (2015). ¿Qué son los Saberes Tecnológicos y Pedagógicos del Contenido (TPACK)? Virtualidad, Educación y Ciencia, 10 (6), pp. 9-23. <http://www.punyamishra.com/wp-content/uploads/2016/08/11552-30402-1-SM.pdf>



Only after having reflected as a user does it make sense to situate future teachers before the **challenge of being actual teachers**, that is, deciding what they want to teach, what they hope their students will learn about computational thinking, what resources and strategies to select, etc. In the initial training of teachers there are different moments for this process: in practicums and school stays, in university – school collaboration activities (visits, workshops fairs, etc.), but also having future teachers simulate small classroom situations between peers, the so-called “microteaching”, where some act as teachers and others as students.

Finally, only after having had first-hand experience and having functioned as a teacher is it possible to reflect on the teaching and learning process related to computational thinking, that is, to become a **reflective teacher**. This not only involves asking what and how to teach but why, regarding the difficulties and casuistry of each specific educational context and the strategies to approach them.

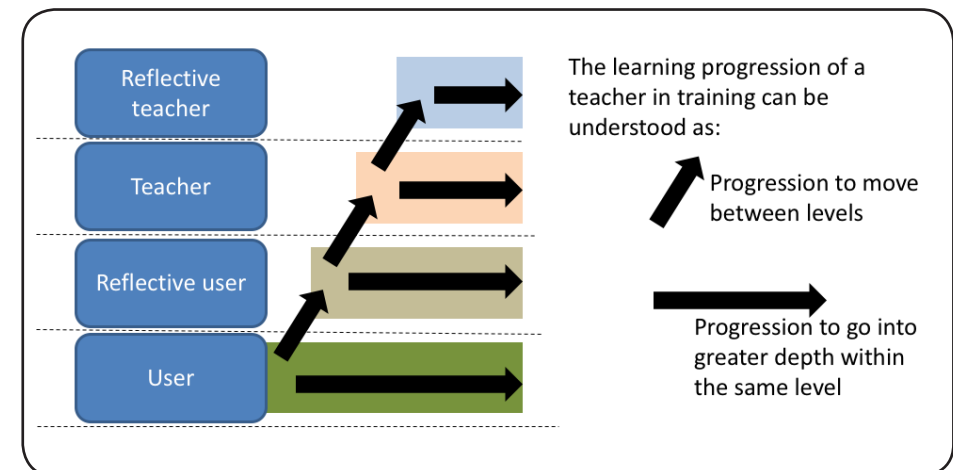


*Levels of progression of future teachers in learning computational thinking.*

The proposal of these four key levels serves to guide what can be the learning progression of future teachers in computational thinking. On the one hand, a learning progression can be understood as the progressive deepening within the same level. For example, at the user level, teacher training can be focused

on learning computational languages and strategies for solving increasingly sophisticated computational problems. In the case of robotics, a possible example would be to introduce the Bee-Bot robots (<https://www.bee-bot.us/>) first, followed by Lego WeDo robots (<https://education.lego.com/en-us>), and then continue with Edison robots (<https://meetedison.com/>), and/or the mbots (<https://www.makeblock.com/steam-kits/mbot>), and/or Lego Mindstorms (<https://education.lego.com/en-us>)

At the same time, a progression can be understood as the transition between levels, so that teachers in training are first users, then reflective users, then teachers and finally become reflective teachers. Thus, for example in a training sequence it would be beneficial to propose that they first create a small video game with Scratch, then reflect on what they learned, followed by a workshop (with other teachers in training or with students). The training sequence will conclude with a reflection on what they learned when they had to teach about this, what difficulties their “students” presented and how they could help to overcome them and/or avoid them.



*Stages in the progression of the expertise of teachers in computational thinking.*

This dual way of understanding the progression of learning in teacher training (becoming increasingly more expert at each level and being able to progress towards the next level) allows us to situate each training activity, and thus devise training itineraries that may be applicable to other teacher training contexts.

## PRACTICAL EXAMPLE:

### Video game design workshop to learn how to program from small challenges

#### USER LEVEL

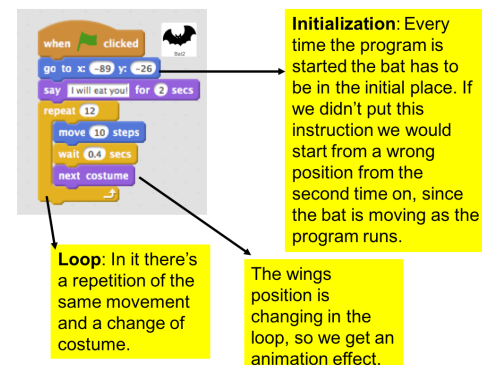
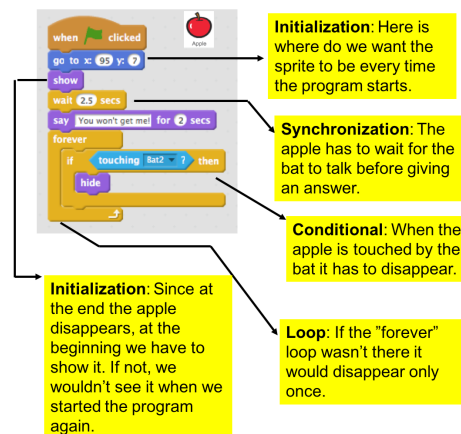
Video game design workshop to learn how to program from small challenges. In order to learn and master the Stencyl programming language, future teachers must create a video game based on the popular video game "Galaga". The training sequence begins with a brief introduction on the basic operation of the tool: the interface, operating rules, pieces of code, etc. Once this programming language has been presented, small challenges are presented to the participants that have to be resolved, with increasing difficulty. Periodically, they join together to solve the challenges that have emerged. At the end of the process participants have acquired a strong enough knowledge of block programming to allow them to both go into greater depth in more sophisticated programming languages in the future, or to begin to ask themselves how this learning they have experienced can in turn be taught by them in the future.

## PRACTICAL EXAMPLE:

### Analyzing a program after having designed it

#### REFLECTIVE USER LEVEL

We can ask a group of teachers in training who have previously participated in an activity where they had to do elaborate a small computer program (for example, with Scratch), to think about why they followed a series of steps, comparing whether all the groups had done it the same way, what they believe they could have done differently, etc. These types of questions lead them to consider not only what they have done, but also the ideas behind their decisions: which concepts, practices and perspectives. In this example, participants are asked to justify why they programmed in a certain way, explaining what would happen if they had not included each of the parts of the programming block.





## PRACTICAL EXAMPLE:

### Microteaching among peers on computational thinking

#### TEACHER LEVEL

After having become familiar with the Scratch programming language (user level) and after having reflected on which aspects of computational thinking are at work when children create a small application using this language (reflective user level), a proposal is made to some teaching degree students to design and implement in front of their classmates a short 20-minute presentation teaching them a particular aspect of Scratch. Thus, the participants in charge of doing this presentation, which we call microteaching, have to think about the learning objectives they are considering, what steps to follow, how to manage the work of the group, what examples they will use, etc. This will enable them, subsequently, to reflect on what worked best and what could be improved in their presentation.



## PRACTICAL EXAMPLE:

### Post-evaluation after having devised Computational Thinking activities in an out of class intervention with children

#### REFLECTIVE TEACHER LEVEL

In a context of an action with real children, a group of students in training can implement proposals they have designed to be developed with children; for example, using a robot or having the children themselves receive orders and act as if they were robots. These students not only have the challenge of carrying out the activity itself, but must also gather evidence of the learning done by the children that participated: conversations among children, critical situations, etc. Once this evidence is collected, the teachers in training meet back at the university to discuss their experience. Through the evidence gathered and a grid of key aspects of computational thinking (see previous section), participants discuss what kind of learning they may have promoted among the children who participated in the workshop, thus reflecting on their practice.

#### TO LEARN MORE

- BBC Bitesize <https://www.bbc.com/education/guides/zp92mp3/revision>
- Bocconi et al., (2016). Developing Computational Thinking in Compulsory Education. Implications for policy and practice. JCR Science for Policy Report. European Commission. [http://publications.jrc.ec.europa.eu/repository/bitstream/JRC104188/jrc104188\\_computhinkreport.pdf](http://publications.jrc.ec.europa.eu/repository/bitstream/JRC104188/jrc104188_computhinkreport.pdf)
- CAS Barefoot <https://barefootcas.org.uk/barefoot-primary-computing-resources/concepts/computational-thinking-further-examples/>
- Computing at School resources: <https://www.computingschool.org.uk/>
- Computer Science Without a Computer: <https://csunplugged.org/en/>, <http://csedweek.org/unplugged/thinkersmith>
- Google Education Computational Thinking microsite <https://edu.google.com/resources/programs/exploring-computational-thinking/>
- Google CT for Educators [https://computationalthinkingcourse.withgoogle.com/course?use\\_last\\_location=true](https://computationalthinkingcourse.withgoogle.com/course?use_last_location=true)
- Kiki Computational Thinking Games <http://games.thinkingmyself.com/>





## Before finishing, some practical advice:

- **You do not have to know everything before you start:** Of course something you learn is that you do not need to know everything about a particular tool. It is necessary to face a challenge or problem, and propose and develop a possible solution. However, it is obvious that when this challenge or problem is worked on in a classroom there will be multiple ideas and paths that students will explore and, in the face of children's doubts, the teacher will have to learn to answer, "explore, investigate", and to raise questions that make them think, and that guide them toward discovering and overcoming the small obstacles they may encounter.
- **Collaborative challenges should be considered:** It is essential that students work in teams and learn to collaborate, communicate, and manage problems and solutions within the framework of teamwork.
- **The spaces where the activities are to be carried out must be taken into account:** It is very important that teachers and future teachers be aware of the spaces where the workshops and activities will be carried out. To ensure the success of a workshop, it is crucial that the spaces be suitable for working in teams of children, that the spaces lend themselves to sharing projects and ideas, and that the equipment be functional.
- **The equipment must be ready before beginning the activity:** The equipment must be ready before beginning the activity. It is important to have tested the equipment and technologies prior to the work session with the boys and girls, so that when the workshop begins the materials are ready for use; this way we will avoid technology-related problems (there is no Wi-Fi, the Wi-Fi doesn't work, there is no Bluetooth, this plugin isn't installed, it doesn't work with this browser, etc.).
- **If in spite of everything something does not work as expected, do not worry; an error, and knowing how to solve it, is part of the challenge!**

