

# LibreLogo - Handbook

## Turtle Geometry

Andreas R. Formiconi

Versione 0.1

January 2018



Figure 1.1: Jan Fabre, "Searching for Utopia", 2003

This work is released under the  
Creative Commons Attribuzione 2.5 Italia license.

In order to read a copy of the license visit  
<http://creativecommons.org/licenses/by/2.5/it/> or write to Creative Commons,  
PO Box 1866, Mountain View, CA 94042, USA.



# Contents

1.1	Aknowldegments . . . . .	4
<b>I</b>	<b>Basics of Logo and LibreLogo</b>	<b>5</b>
1.2	Preface . . . . .	7
1.3	Acknowledgments . . . . .	10
<b>II</b>	<b>Basics of Logo and LibreLogo</b>	<b>11</b>
1.4	Preface . . . . .	13
<b>2</b>	<b>LibreLogo</b>	<b>15</b>
2.1	How to manage graphics in Writer . . . . .	21
<b>3</b>	<b>La genesi</b>	<b>23</b>
3.1	La paura della matematica . . . . .	23
3.2	<i>Mathophobia: The Fear for Learning</i> . . . . .	24
<b>4</b>	<b>Il LOGO</b>	<b>31</b>

## 1.1 Acknowledgments

First of all I thank all the students who worked beyond my expectations, even by sending works that they were not supposed to do. Their explosion of creativity was extremely useful. Some of these contributors deserve specific mentions. First the remarkable "Creativity Exercises" of Marta Veloce, Primary Education student, that inspired chapter ???. Then Alberto Averono, teacher of a secondary school, who in a training course about coding further developed Marta's exploration. Another Primary Education student, Eleonora Aiazzi, sent a moving text, "When a professor treats you as a child", who was extremely encouraging about the teaching method I was experimenting. A well experienced primary school teacher, Antonella Colombo, gave me back a wonderful documentation of Papert's syntonik learning she tried to trigger in her classroom. Her work opens the trip in chapter ??, from the draw of a circle to the Halley's orbit. Thanks to Piero Salonia, who corrected the drafts of the first Italian version of this manual. Finally, thanks to the beautiful world of Linux and its sharp tools - scp, ssh, rsync, grep, find, nmap, latex, bibtex and so on - which gives true superpowers, unknown in the Graphical User Interface World - the Internet of true freedom.

**Part I**

**Basics of Logo and  
LibreLogo**



## 1.2 Preface

This is an English version derived from the “Piccolo Manuale di LibreLogo” that I wrote for the students of the Primary Education curriculum at the university of Florence. The manual was intended to aid the students in developing their coding activities within the “Laboratorio di Tecnologie Didattiche” (Laboratory of Educational Technologies). The programming language is LibreLogo, which is an implementation of Seymour Papert’s Logo language within the Writer word processor of the LibreOffice suite. LibreLogo is a plugin available by default in Writer since version 4.0 of LibreOffice. It has been written in Python by László Németh. The specific documentation can be found at <http://librelogo.org>. From there a description of all the LibreLogo commands can be downloaded in 33 different languages <sup>1</sup>. Apart from this, some extended documentations can be found only in Hungarian, as far as I know as of March 2018: there are a pretty technical handbook written by László Németh itself [5] and a more classroom oriented one by Lakó Viktória [7]. Some of the examples presented in this work have been inspired by those in Lakó Viktória’s book, but the perspective is different here. First of all I have put a strong emphasis on pedagogical aspects, beyond the mere technical facts, following the line of thought of Seymour Papert, as reported, for instance, in *Mindstorms* [6]. Secondly, during the couple of years since the writing of the first version of the “Piccolo Manuale di LibreLogo”, a number of reflections, exercises and hands-on practices got back from students and other sources, have been added. The overall result is a rather broad discussion of possible uses of LibreLogo, both in a vertical dimension, from primary school examples to tertiary education level exercises, as well as in a transversal dimension, through a variety of disciplines. Finally, this is not a true translation of the “Piccolo Manuale di LibreLogo” but an English and somewhat more concise rewriting.

---

<sup>1</sup>The commands dictionaries can be downloaded from [https://help.libreoffice.org/Writer/LibreLogo\\_Toolbar](https://help.libreoffice.org/Writer/LibreLogo_Toolbar)





# Contents

### 1.3 Acknowledgments

First of all I thank all the students who worked beyond my expectations, even by sending works that they were not supposed to do. Their explosion of creativity was extremely useful. Some of these contributors deserve specific mentions. First the remarkable "Creativity Exercises" of Marta Veloce, Primary Education student, that inspired chapter ???. Then Alberto Averono, teacher of a secondary school, who in a training course about coding further developed Marta's exploration. Another Primary Education student, Eleonora Aiazzi, send a moving text, "When a professor treats you as a child", who was extremely encouraging about the teaching method I was experimenting. A well experienced primary school teacher, Antonella Colombo, gave me back a wonderful documentation of Papert's syntonik learning she tried to trigger in her classroom. Her work opens the trip in chapter ??, from the draw of a circle to the Halley's orbit. Thanks to Piero Salonia, who corrected the proofs of the first Italian version of this manual. Finally, thanks to the beautiful world of Linux and its sharp tools - scp, ssh, rsync, grep, find, nmap, latex, bibtex and so on - which gives true superpowers, unknown in the Graphical User Interface World - the Internet of true freedom.

**Part II**

**Basics of Logo and  
LibreLogo**



## 1.4 Preface

This is an English version derived from the "Piccolo Manuale di LibreLogo" that I wrote for the students of the Primary Education curriculum at the university of Florence. The manual was intended to aid the students in developing their coding activities within the "Laboratorio di Tecnologie Didattiche" (Laboratory of Educational Technologies). The programming language is LibreLogo, which is an implementation of Seymour Papert's Logo language within the Writer word processor of the LibreOffice suite. LibreLogo is a plugin available by default in Writer since version 4.0 of LibreOffice. It has been written in Python by László Németh. The specific documentation can be found at <http://librelogo.org>. From there a description of all the LibreLogo commands can be downloaded in 33 different languages<sup>2</sup>. Apart from this, some extended documentations can be found only in Hungarian, as far as I know as of March 2018: there are a pretty technical handbook written by László Németh itself [5] and a more classroom oriented one by Lakó Viktória [7]. Some of the examples presented in this work have been inspired by those in Lakó Viktória's book, but the perspective is different here. First of all I have put a strong emphasis on pedagogical aspects, beyond the mere technical facts, following the line of thought of Seymour Papert, as reported, for instance, in *Mindstorms* [6]. Secondly, during the couple of years since the writing of the first version of the "Piccolo Manuale di LibreLogo", a number of reflections, exercises and hands-on practices got back from students and other sources, have been added. The overall result is a rather broad discussion of possible uses of LibreLogo, both in a vertical dimension, from primary school examples to tertiary education level exercises, and in a transverse dimension, through a variety of disciplines. Finally, this is not a true translation of the "Piccolo Manuale di LibreLogo" but an English rewriting, characterized by a somewhat more concise exposition of the same facts.

---

<sup>2</sup>The commands dictionaries can be downloaded from [https://help.libreoffice.org/Writer/LibreLogo\\_toolbar](https://help.libreoffice.org/Writer/LibreLogo_toolbar) :



## Chapter 2

# LibreLogo

LibreLogo is the implementation of the famous Logo language within the word-processor Writer. Writer is the word processing application of the LibreOffice, analogously to word, which is part of MS Office suite. Logo was created by Seymour Papert in the seventies to improve the learning of math. Seymour Papert, born in South Africa in 1928, first studied math in Johannesburg and successively in Cambridge. Between 1958 and 1964 he got his PhD at the University of Geneva with Jean Piaget: interesting collaboration between a mathematician and a pedagogist. Since 1964 he was a researcher of the MIT Artificial Intelligence laboratory where, in 1967 he got the position of codirector with Marvin Minsky, a relevant scientist in the field of artificial intelligence. The laboratory is the same where Richard Stallman, father of free software, would have worked a few years later. Free software is a beautiful reality but, ironically, it is incredibly widespread and, at the same time, so few people know something about. And it is right in the school context that it should be much more diffused, because of its relevant ethic and educational features. Probably, the best known free software is the Linux operating system, but there are many more: LibreOffice, analogous of MS Office, Gimp for manipulating digital bitmap images, Inkscape for vector images, Audacity for audio recorded files, OBS for streaming and screencasting, shotcut for video cutting and many others. Seymour Papert is famous for having invented Logo, a programming language for drawing by giving commands to a "Turtle". However, in the first version, conceived in the Seventies, the Turtle was a robot which was able to draw by means of a pencil.

When computers arrived in the homes in the 80s, Logo became a software and as such was described by Seymour Papert in *Mindstorms*. In order to understand the pedagogical value of Papert's thought, let's read this passage, taken from *Mindstorms* (pp. 7-8) [6]:

I take from Jean Piaget a model of children as builders of their own intellectual structures. Children seem to be innately gifted learners, acquiring long before they go to school a vast quantity of knowledge by a process I call "Piagetian learning", or "learning without being taught". For example, children learn to speak, learn the intuitive geometry needed to get around in space, and learn enough of logic and rethorics to get around parents - all this without being taught. We must ask why some learning takes place so early and sponta-

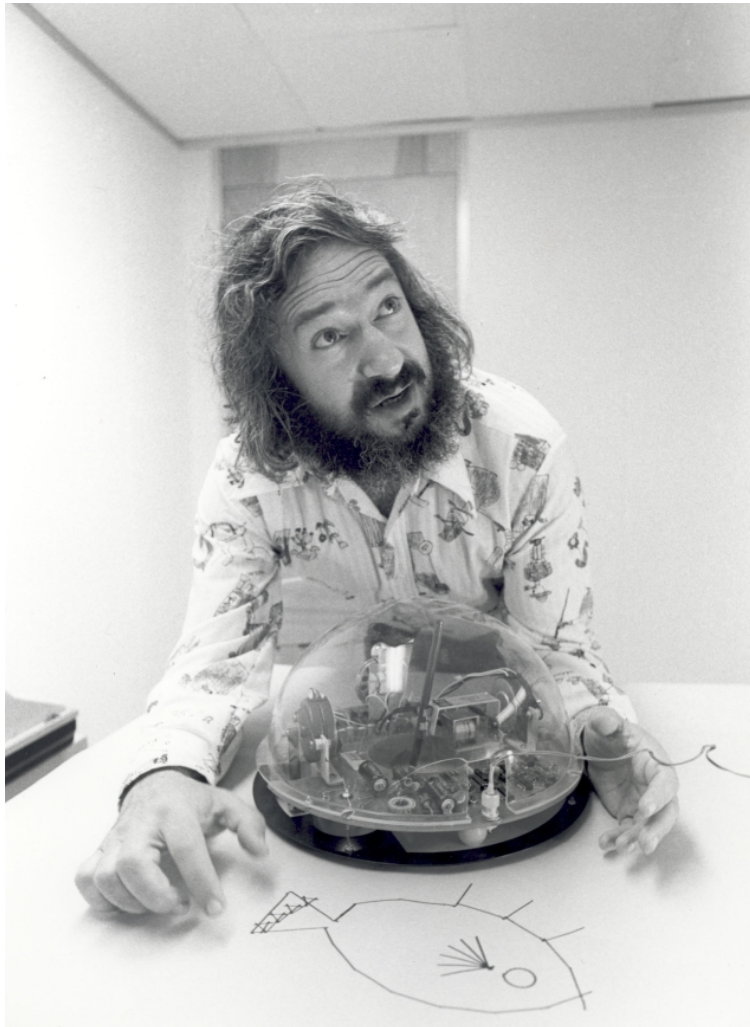


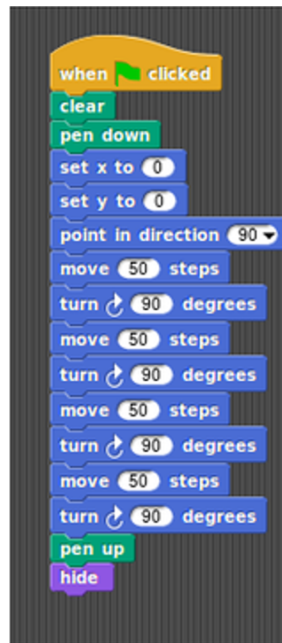
Figure 2.1: Seymour Papert shows one of the first versions of Logo, when it was kind of a robot for drawing.



neously while some is delayed many years or does not happen at all without deliberately posed formal instruction.

If we really look at the "Child as a Builder" we are on our way to an answer. All builders need materials to build with. Where I am at variance with Piaget is in the role I attribute to the surrounding cultures as a source of these materials. In some cases the culture supplies them in abundance, thus facilitating constructive Piagetian learning. For example, the fact that so many important things (knives and forks, mothers and fathers, shoes and socks) come in pairs is a "material" for the construction of an intuitive sense of number. But in many cases where Piaget would explain the slower development of a particular concept by its greater complexity or formality, I see the critical factor as the relative poverty of the culture in those materials that would make the concept simple and concrete.

In the 90's Logo circulated as a program installable from a floppy disk. Once launched, it produced a black screen on which instructions could be written in sequence, one after the other. The instructions represented the movements given to the turtle on the screen. Then, with a special command, you could "execute" the sequence of commands, and so the turtle would move leaving a mark on the screen. Logo had a great resonance as an experimental method for teaching math and a wide variety of versions have been derived, reaching to generalizations such as the current Scratch. However, it has not been widely distributed in schools and maybe it has been more successful among kids than among teachers. Probably it was too early. Using Logo means writing code, an activity that is not part of the preparation of most teachers, including those who teach science. Today it is perhaps different, we talk a lot about *coding*, even if perhaps not always with full knowledge of the facts. The situation has evolved so much that *coding* can mean so many different things. Moreover, from the 1980s to the present, the variety of programming languages has grown enormously. Nowadays, among the logo derivations, Scratch is the most renowned educational programming language. Mitchel Resnick, a former student of Papert, is the project leader of Scratch and now is still following it at the MIT Media Laboratory. Scratch goes far beyond the production of graphics and allows you to create animations and video games, thus also allowing one to experiment with rather sophisticated programming techniques. Another innovative aspect is that it is structured as a web service and this has allowed the creation of a large community of living dissemination and exchange of programs. From the operational point of view, Scratch differs from Logo in that it is a visual language. The commands are in fact made up of coloured blocks that can be interlocked. The program comes out from the execution of these sequences of commands connected together, as in a puzzle. It's an attractive system that's a bit like Lego, where the instructions you give are stuck together like bricks. The joints ensure that instructions are combined only in legitimate ways, protecting against the typical and frequent spelling and syntactical errors that anyone who is writing software in the conventional text mode would encounter. Many of these languages have emerged, in addition to Scratch. The most famous are Snap!, Alice, Blockly, Android App Inventor, just to name a few. The following figure shows the difference between a text code and a visual code. The code is used to draw a square. Left the LibreLogo version and right the Snap! In



```

CLEARSCREEN
PENDOWN
HOME
FORWARD 50
RIGHT 90
FORWARD 50
RIGHT 90
FORWARD 50
RIGHT 90
FORWARD 50
RIGHT 90
PENUP
HIDETURTLE

```

Scratch this simple code would be identical. I used Snap! since I have a certain preference for this language. Snap! represents an enhancement of Scratch, which makes it more similar to a generic language, while maintaining the visual form. Among these features there is the possibility of saving the code in a standard format (XML) which is readable and editable by any text editor. For those who are used to working with softwares, this is a very important element. The code is not "optimal", in any sense. It is only intended to compare instructions in two different environments.

A special feature of Scratch is that it has created a large software sharing community. This happened thanks to the fact that it was conceived as a web service, which allows the writing of programs and the possibility of running them but also the realization of a social environment for sharing and remixing the programs. Visual languages have also drawbacks. They are (apparently) easy, fun and colorful, their effectiveness would seem guaranteed but the scientific evidence is not so clear. There are in fact various studies that show that visual languages do not facilitate so much the learning of "real" languages [2]. It seems that they are advantageous to understand the simplest constructs of programming but studies where the deep understanding about what a given algorithm does do not show substantial differences between visual and textual languages [4]. The research of Colleen Lewis is of particular interest. She compared the results obtained with Logo and Scratch in a class of children between 10 and 12 years [1]. The results showed that the learning of some specific coding constructs was facilitated by Scratch but, on the other side, the kids showed a higher level of self-esteem when introduced to programming with Logo.

And even if in the initial phases kids prefer visual tools, later on, once they try conventional textual programming, they may perceive the limits of visual coding:

- as limiting their creativity because less powerful
- because they feel visual coding less real: "if you have to do something real, nobody will ever ask you to encode it with visual educational software" [3]

It is on the basis of such considerations that we decided to focus on the Logo language, as an introductory tool to programming. Pretty a number of Logo versions are available nowadays. We here focus on a version that is available by default in Writer, the word processing application included in the LibreOffice office suite<sup>1</sup>, analogous of the widespread MS Office suite. The latter is a "proprietary product", i.e. the company that produces it sells it but without distributing the source code in clear, according to the conventional industrial model, with which the intellectual property is jealously kept secret. LibreOffice is a free software, and as such is ideal for use in any educational context. Firstly, because it carries an ethical message. In fact, free software is defined by four types of freedom<sup>2</sup>:

- The freedom to run the program as you wish, for any purpose (freedom 0).
- The freedom to study how the program works, and change it so it does your computing as you wish (freedom 1). Access to the source code is a precondition for this.
- The freedom to redistribute copies so you can help your neighbour (freedom 2).
- The freedom to distribute copies of your modified versions to others (freedom 3). By doing this you can give the whole community a chance to benefit from your changes. Access to the source code is a precondition for this.

It should be noted - on this point many are confused - that *open source* software is different from (*free software*) since the ethical aspect is missing: open source software assumes that the source code is available in clear, but does not mention the four freedoms mentioned above and, in particular, the two specifications that characterize the ethical value of *free software*: "so as to help others" in the third freedom and "so that the whole community benefits" in the fourth freedom. Free software is developed by communities that may join together in non-profit societies. *Open source* is developed by private economic actors who adhere to the shared development paradigm because it fits well into their marketing strategies: there are companies that develop *open source* projects alongside traditional proprietary products because they find it convenient for their marketing strategies. LibreOffice's functionality can be enriched by means

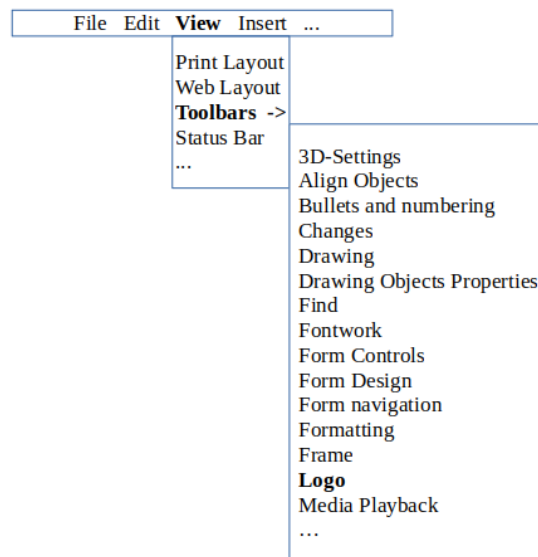
---

<sup>1</sup>There is another similar project called OpenOffice. Many ask what are the differences with LibreOffice. A small history of the evolution of these two software, which have a common origin, can be found here (July 2016): <http://www.navigaweb.net/2014/04/differenze-tra-openoffice-e-libreoffice.html>. At present, LibreOffice is convenient because it incorporates more features and is updated more frequently.

<sup>2</sup>Free software definition according to the free Software Foundation: <https://www.gnu.org/philosophy/free-sw.en.html>

of many *plugins*. LibreLogo is one of them and, from version 4.0 on, the LibreLogo plugin is included by *default* <sup>3</sup> in the program. But what does it mean to use Logo within a *word processor* like Writer, considered that this is a normal word processor while Logo is a kind of a drawing language? Simple: With the LibreLogo toolbar you can produce images that are integrated into the document as if they were imported. It's a brilliant idea, due to Németh László, who reproduced the features of Logo within LibreOffice. In reality, it has further increased them, taking advantage of the Python language, with which he wrote the plugin. Using LibreLogo is very simple: you open a document in Writer, you write some code in Logo language, as you would write any other text, and then you run it by pressing the appropriate button in the LibreLogo *toolbar*; if the code is correct, the turtle executes the code drawing a figure in the text, right in the middle of the page.

This design can then be managed like any other LibreOffice graphics. The first time you launch LibreOffice the Logo toolbar is not active. Therefore you need to activate it, with the appropriate menu command: **View** → **Toolbars** → **Logo**:













Once this is done, you must close the program and relaunch it to see, among the other toolbars also the LibreLogo one:



where the icons have the following meanings:

<sup>3</sup>Di *default* which means that this is the normal behavior. Those who use Linux (for Windows or Mac this problem does not exist) should take note of the following. Until the release of LibreOffice 4 excluded, install the LibreOffice extension from <http://extensions.libreoffice.org/extension-center/librelogo>. Instead, from version 4 on, install the Office-Library package directly, with the command `sudo apt-get install library-Library`. Then you need to restart LibreOffice, if it is already open. Successively, activate the toolbar in View-¿Toolbars-¿Logo. Close and relaunch

	FORWARD 10	Forward by 10 points (we will see the meaning of point successively)
	BACK 10	Back by 10 points
	LEFT 15	Left by -15 degree
	RIGHT 15	Right by 15 degree
		Executes the program. Starting with version 4.3, in a newly opened document it executes a standard sampled program.
		Stops the running program
	HOME	Brings the Turtle in its initial condition: at the center with nose up.
	CLEARSCREEN	Cancels all the graphics - leaving the text.
		Allows to write a command and run it at once.
		Adjust the whole text making it uppercase. At the same time, it translates the commands in the language of the document. Currently, the dictionary file for the Italian language is set up but the commands have to be written in English. I will fix this.


## 2.1 How to manage graphics in Writer

The interaction between LibreLogo and Writer is particular for graphics. It may seem cumbersome at first but you actually have to get used to it and learn two or three rules. The probably unique feature of LibreLogo is that by running <sup>4</sup> a script you get a graphic object in the same place where you have written the code, that's it in ODT document page. These objects are of "vectorial" type, that is, they are composed by a set of geometric objects.


<sup>4</sup>In jargon, by "running a program" we intended to execute all its instructions. Today, with modern languages, programs are often called *script*. In general, a program is a complete software and maybe also very complex. A *script* tends to be a smaller, more specific fragment of code but these categories may overlap widely.


They are different from *raster* or *bitmap*, that consist of a matrix of pixel<sup>5</sup>. The graphic objects produced by LibreLogo are completely similar to those produced with the handwriting tools available in Writer, accessible through the special *toolbar*, under the menu item **View** → **Toolbars** → **Drawing**:



As such, drawings made with LibreLogo can be moved, copied, or saved like any other graphic object. One useful thing to understand is that such objects are often actually a composition of distinct objects. We will do many of them in this manual. To use them as a single object, use the grouping function, as follows: first, you delimit the region that includes the objects to group, by selecting *pointer*  in the drawing bar and then by outlining the desired rectangular box with the mouse and holding down the left button. Please note that the mouse cursor must be in the shape of an arrow and not the typical you have when inserting text, in the shape of a capital I, because this is where you insert text and not graphics. The fact that the graphic (and not textual) cursor is active is also understood by the fact that, at the same time, another toolbar is activated for controlling the graphics:



When you select the region containing the graphic objects, icons are activated in this bar, including the icon for the grouping function: . Pressing this will group all graphic objects in the selected region into a single graphic object that can be copied elsewhere or saved.

Another useful trick is to properly "anchor" the graphics to the document, where we have to use them. The key to determine the anchorage in the usual graphic bar is this: . By clicking on the arrow on the right of the anchor, you can select four anchor types: 1) "on page", 2) "in paragraph", 3) "in character" and 4) "as character". In the first case the graphics are associated to the page and do not move from it, in the second to a paragraph, in the third to a character and in the fourth case it behaves as if it were a character. What is the most appropriate anchorage is something that you learn from experience. Most of the graphics in this manual have been anchored "to the paragraph", except for small images that are in line with the text, as in the previous one, these are anchored "as a character".

These concern the management of graphics in Writer in general. Using LibreLogo, the only difference is that the graphics are produced through the instructions we put in the code. LibreLogo places the graphics in the middle of the first page of the document, even if the code text extends on the following pages. It may happen that the graphics overlap the text of the code itself. At first glance the result may be confusing and one can believe something wrong is going on. None of this. The graphics are produced to be used somewhere else. It is simply a matter of selecting it, as we have just described, and taking it elsewhere, in a clean page simply to see it clearly, or in some other document where it must be integrated.

<sup>5</sup>A closer look at the distinction between bitmap and vector images can be found at <http://iamarf.org/2014/02/23/elaborazione-di-immagini-tre-fatti-che-fanno-la-differenza-loptis/>

## Chapter 3

# La genesi

### 3.1 La paura della matematica

The fundamental motivation for the genesis of Logo lies in the still unresolved question of mathematical teaching. The Logo language was conceived by Papert precisely to try to solve this age-old problem for which he had also coined a precise name, *Mathophobia*, to describe the widespread antipathy towards this subject. Papert's interest in this issue has accompanied his entire working life, which extended along the second half of the 20th century. His contribution was exceptional, both in terms of theoretical elaboration from the pedagogical point of view, and of the creativity that led him to devise a language specifically to bring children closer to mathematics. His profound competence in both mathematical-informatics and pedagogy makes his work unique and explains his rare ability to propose concrete solutions.

The feeling is that not much has changed, since the 1980s, at least on average<sup>1</sup>; that the initial motivation, based on a serious and difficult revisitation of

---

<sup>1</sup>This statement hides a world of perplexity. What has changed? Perhaps that is what a mere average does not express. The school to which Papert referred is probably more similar to the one I attended (I grade in 1960). At that time, the panorama was probably much more even. "Beat him if he doesn't understand why he's a goose!" recommended the mother of a classmate of mine to the teacher. Parents were allies of that school system, in a educational vision that could be coercive and punitive, but that ran through all kinds of schools and all social strata. There were no "parent coaches" or "unionist parents". Families worked hard, school was hard. There wasn't yet any "free time". The school was more brutal, perhaps unfair, the pedagogy simple, but the picture was clearer. At least in the rural province of the 60s where I lived. Nowadays complexity reigns supreme. The categories intersect. The debates explode, amplified by the media, at a microscopic level (groups of parents in Whatsapp or Facebook) and at a macroscopic level (press, television, etc.). My personal experiences are schizophrenic: my contacts with the world of teaching represent a fascinating picture of commitment, study and experimentation; but private stories and the stories of acquaintances are populated by obsolete and superficial didactic practices. Variability is amazing. Where is the average? Frankly, I cannot assess it, but the dispersion is certainly much wider than it once was. The picture is complicated by international investigations, marked by scientific rigour but that may turn out to be fatuous. For some years, Finland's polar star has shone in the sky of OECD PISA assessments, particularly about mathematics. But at the same time you can stumble upon a number of complaints from Finnish academics about a collapse in mathematical skills: it seems that Finnish students have become good at PISA mathematical testing but have worsened in mathematics! Reading Giorgio Israel's post "The Bluff of Finnish Mathematics" (<http://gisrael.blogspot.it/2011/05/il-bluff-della-matematica-finlandese.html>), which summarizes these complaints, we discover that learning models are trivially utilitarian

the way young people are introduced to math, has ended up diluted in the pot of "coding", sort of Disneyland, superficially exciting for some, object of derision for others; that Papert's message, in some ways extreme and provocative, certainly to be decoded with respect to a changed context, is largely mistaken; that everything is defeated by the failure of Logo, restricted to a minority of experimental circles, without having revolutionized anything, in contrast with Papert's legitimate expectations; that invoking the magic of mathematics to introduce young people to a domain commonly considered "cold", is a dream conceivable only by a mathematician, somewhat 'idealistic. In other words an utopia.

Logo failed we said. It failed in Papert's initial intentions. It is not widespread in the schools and it is not adopted as a standard way to support math and science learning. But it did not fail in the sense of not having left a trace, quite the contrary. There are many versions of Logo around the world, some of which became important tools for educational investigation, for example in the field of simulation of complex biological systems. And it is always from Logo that the sprawling world of visual block languages took its cue, first and foremost Scratch. Logo and Scratch are not in opposition. In a way, Scratch derives from Logo and "contains" many of its functionalities. Many of the things you can do in Logo can also be done in Scratch. But Scratch is much more oriented to the building of "you own videogame" or to the storytelling. The problem, however, is that this wider range of possibilities, deployed in a context of poorly technological educated teachers, has ended up dispersing the original educational intentions of Logo. One of the intentions of this manual is to recover the original "mathematical flavor" of the coding practice at school. In the next section we are going to comment what Papert called "Mathophobia", a kind of illness that Logo was intended to fight.

### 3.2 *Mathophobia: The Fear for Learning*

Seymour Papert chose to title the second chapter of *Mindstorms* "Mathophobia: The Fear for Learning". Its starting point is the schizophrenic split between humanities and science, a division that is deeply built in the language, the worldview, the social organization and the educational system. A division that in the last 30-40 years of neoliberalistic drift, has further widened. For instance, in the universities, since the 1980s, academics struggle in the competition for getting their researches funded. Unfortunately, the other side of the coin is that almost nobody cares about teaching, or very little. The career of a professor does not depend on the quality of his teaching but almost only on the quantity of her or his scientific outputs. The consequences are too bad. Academics tend to transform in managers when they do research and public civil servant when they teach: dynamic entrepreneurs on one hand and (strong) conservatives the other. In that way, even in the humanities we see a technical drift of the academic role. The mission of teaching, which in a sense is the "humanistic side" of the job is reduced to a kind of Cinderella. Thus, the dichotomization between scientific and humanistic is even stronger and unbalanced.

---

and far from Papert's ideas. Where will the truth be? In short, confusion reigns supreme and one seriously wonders whether one should not resign oneself to considering it just inevitable.



The issue is not that of some proper balance but to break the line between the two cultures. Papert looked at the computer as a force to dampen the distinction. The practice of coding was thought as a way to introduce a more humanistic mathematics and to exploit some scientific reasoning in the humanities. It is in this context that Papert talked about a *Mathland*, where mathematics would become a natural vocabulary, with the idea that we could change not only how we teach math but even the way in which our culture thinks about knowledge and learning.

Papert claims that his arguments are not limited to the learning of math but concern the attitude to learning in general. The word *mathophobia* suggests two associations. One is the widespread dislike of mathematics. The other derives from the stem "math", that in ancient greek means learning in general sense. Thus, if children begin by being skillful spontaneous learners, later on they "learn" the fear of learning and not only of mathematics. Ironically, it seems that the more you get instructed the more you fear learning.

Children learn thousands of words before entering the first grade. Less obvious to many people is the fact that kids learn a great deal of mathematics as well. Among these preschool knowledges there are for instance notions such as the volume conservation of liquids in vessels of different shape, or the independence of the total number of objects from the order in which they have been counted. Papert called this

Piagetian learning, a learning process that has many features the schools should envy: it is effective (all the children get there), it is inexpensive (it seems to require neither teachers nor a curriculum development), and it is humane (the children seem to do it in a carefree spirit without explicit external rewards and punishments).

As a matter of fact, the practices of mathematics teaching largely underestimates the Piagetian learning, imposing formal knowledges that turn out to be mostly dissociated from the former spontaneous notions. The consequences for the future adults are heavy. The loss of the child's positive attitude towards learning is a very common phenomenon of adult ages and it does not concern only mathematics.

Deficiency becomes identity: "I can't learn French, I don't have an ear for languages;" "I could never be a businessman, I don't have a head for figures."

Some 80% of my Primary Education students declare themselves as "distant from math". Many claim to have difficulties with technologies as well, despite they are supposed to be "digital natives".

The notion that there are smart people and dumb people for a given activity is widespread. It is extremely difficult to eradicate the prejudices about one's own attitudes. The point is that the accepted beliefs about mathematical aptitude do not follow from the available evidence. In order to reinforce the concept Papert recasted the argument as follows [6] (p. 43):

Imagine that children were forced to spend an hour a day drawing dance steps on the squared paper and had to pass tests in these

“dance facts” before they were allowed to dance physically. Would we not expect the world to be full full of “dancophobes” ? Would we say that those who made it to the dance floor and music had the greatest “aptitude for dance”? In my view it is no more appropriate to draw conclusions about mathematical aptitude from children’s unwillingness to spend many hundreds of hours doing sums.

School constructs aptitudes:

Consider the case of a child I observed through his 8th and 9th years. Jim was a highly verbal and mathophobic child from a professional family. His love for words and for talking showed itself very early, long before he went to school. The mathophobia developed at the school. My theory is that it came as a direct result of his verbal precocity. I learned from his parents the Jim had developed an early habit of describing in words, often aloud, whatever he was doing as he did it. This habit caused him minor difficulties with parents and preschool teachers. The real trouble came when he hit the arithmetic class. By this time he had learned to keep “talking aloud” under control, but I believe that he still maintained his inner running commentary on his activities. In his math class he was stymied: he simply did not know how to talk about doing sums. He lacked a vocabulary (as most of us do) and a sense of purpose. Out of this frustration of his verbal habits grew a hatred of math, and out of the hatred grew what the tests later confirmed as poor attitude.

For me the story is poignant. I am convinced that what shows up as intellectual weakness very often grows, as Jim’s did, out of intellectual strengths. And it is not only verbal strengths that undermine others. Every careful observer of children must have seen similar processes working in different directions: for example, a child who has become enamored of logical order is set up to be turned off by English spelling and to go on from there to develop a global dislike for writing.

Papert’s idea was that we could use computers as vehicles to escape from the situation of Jim or that of children loving logic but with kind of dyslexic problems. In both cases they are victims of our culture’s sharp separation between the verbal and the mathematical. He imagines a *Mathland* where Jim’s love and skill for language could be mobilized to serve his formal mathematical learning instead of opposing it, whereas for the other kind of children, the love for logic could nourish the interest in linguistics. The prevailing teaching methods give mathematics learners limited possibilities to make sense of what they are learning. Consequently, children are forced to follow the worst model for learning mathematics, which is rote learning, where material is meaningless. It is what Papert called a *dissociated* model.

Well into a year-long study that put powerful computers in the classrooms of a group of “average” 7th graders, the students were at work

on what they called “computer poetry”. They were using computer programs to generate sentences. They gave the computer a syntactic structure within which to make random choices from given lists of words. The result is the kind of concrete poetry we see in the illustration that follows<sup>2</sup>. One of the students, a 13 year old named Jenny, had deeply touched the project’s staff by asking on the first day of her computer work, “Why were we chosen for this? We’re not the brains”. The study had deliberately chosen children of “average” school performance. One day Jenny came in very excited. She had made a Discovery. “Now I know why we have nouns and verbs,” she said. For many years in school Jenny had been drilled in grammatical categories. She had never understood the differences between nouns and verbs and adverbs. But now it was apparent that the difficulty with grammar was not due to an inability to work with logical categories. It was something else. She had simply seen no purpose in the enterprise. She had not been able to make any sense of what grammar was about in the sense of what it might be *for*. And when she had asked what it was for, the explanations that her teachers gave seemed manifestly dishonest. She said she had been told that “grammar helps you talk better.

INSANE RETARD MAKES BECAUSE SWEET SNOOPY SCREAMS  
 SEXY GIRL LOVES THATS WHY THE SEXY LADY HATES  
 UGLY MAN LOVES BECAUSE UGLY DOG HATES  
 MAD WOLF HATES BECAUSE INSANE WOLF SKIPS  
 SEXY RETARD SCREAMS THATS WHY THE SEXY RETARD  
 THIN SNOOPY RUNS BECAUSE FAT WOLF HOPS  
 SWEET FOGINY SKIPS A FAT LADY RUNS

In fact, tracing the connection between learning grammar and improving speech requires a more distanced view of the complex process of learning language than Jenny could have been given at the age she first encountered grammar. She certainly didn’t see any way in which grammar could help talking, nor did she think her talking needed any help full stop therefore she learnt to approach grammar with resentment. And, as is the case for most of us, resentment guaranteed value. But now, as she tried to get the computer to generate poetry, something remarkable happened. She found herself classifying words into categories not because she had been told she had to but because she needed to. In order to teach a computer to make strings of words that would look like English, she had to teach it to choose words of an appropriate class. What she learnt about grammar from this experience with a machine was anything but mechanical or routine. How learning was deep and meaningful. Jenny did more than learn definitions for particular grammatical

---

<sup>2</sup>[NdR] I reported this example because we will find it again, in different form, among the more advanced applications of Logo. Interestingly, in the 70s you needed a powerful computer and an advanced research staff, nowadays you can do the same thing with a simple implementation of Logo in a PC.

classes. She understood the general idea that words (like things) can be placed in different groups or sets, and that doing so could work for her. She not only “understood” grammar, she changed her relationship to it. It was hers, and during her year with the computer, incidents like these helped Jenny change her image of herself. Her performance changed too; her previously low to average grades became “straight A’s” for her remaining years of school. She learned that she could be a brain after all.

Papert put his argument in a very strong way: often children cannot understand what are math and grammar for because they perceive adult explanations as double talk.

It is easy to understand why math and grammar seem to make sense to children when they fail to make sense to everyone around them and why helping children to make sense of them requires more than a teacher making the right speech or putting the right diagram on the board. I have asked many teachers and parents what they thought mathematics to be and why it was important to learn it. Few held a view of mathematics that was sufficiently coherent to justify devoting several thousand hours of a child’s life to learning it, and children sense this. When a teacher tells a student that the reason for those many hours of arithmetic is to be able to check the change at the supermarket, the teacher is simply not believed. Children see such reasons as one more example of adult double talk. The same effect is produced when children are told school math is “fun” when they are pretty sure that teachers who say so spend their leisure hours on anything except this allegedly fun-filled activity. Nor does it help to tell them that they need math to become scientists - most children don’t have such a plan. The children can see perfectly well that the teacher does not like math anymore than they do and that the reason for doing it is simply that it has been inscribed into the curriculum. All of these erodes children’s confidence in the adult world and the process of education. *And I think it introduces a deep element of dishonesty into the educational relationship.*

It is important to keep in mind the difference between mathematics - a vast domain of inquiry whose beauty is rarely suspected by most laymen - and *school math*. The latter is a kind of social construction, that is a set of mathematical topics determined by a succession of specific circumstances. A process that do not guarantees, *per se*, the achievement of an optimal result. It reminds the story of the QWERTY keyboard layout. QWERTY represents the first five keys in the upper rows. This arrangement has no rational explanation but only an historical one. It was introduced because the keys of the first typewriters tended to jam. So they were arranged to reduce collisions by separating the keys that followed one another most frequently. The technology of typewriters improved rapidly and in a few years the jamming problem was no more an issue but the QWERTY arrangement stuck. At this point too many people were fluent with the QWERTY layout and the production of typewriters was too far away to make a step back for redesigning a more rational layout, for instance by grouping the most used keys together.

The QWERTY problem is a good example of how consolidated habits may not necessarily be the best choice. Like the QWERTY layout, school math was shaped in a different historical context. In the same way, this idea of mathematics has dug itself deeply and, even nowadays, for most people it is inconceivable that math could be also something else. I remember a well known professor of calculus who, at the beginning of the first year of the mathematics curriculum, exhorted his students to forget what they had learned in high school since math was something different.

Turtle geometry was conceived to fit children and, first of all, to be *appropriate*. We could describe this concept by means of some principles. First, the *continuity principle*: new mathematical knowledge has to be continuous with the existing one, the one kids have before going to school. Then the *power principle*: new knowledge must empower learners to realize personally meaningful projects, that could not be done without it. Finally, the *principle of cultural resonance*: new concepts must make sense to kids in their social context. Ironically, even in adults social context: we should not inflict on children something we have not thoroughly understood and, unfortunately, with "classic basic math" this is the case.



## Chapter 4

# II LOGO

This chapter is an introduction to the use of Logo. It is addressed mainly to teacher students, trying to show how to initiate kids to the Logo practice. Interestingly, having proposed this matter to several hundreds of teacher students, I realized how teaching Logo to kids is very similar to teaching it to grown up students. And, in average, during the first approach kids perform even better! We often discuss it with those students that claim having had major difficulties talking to the Turtle. The majority of them experience a kind of "resurrection" after some initial struggling: "At the beginning I didn't understand anything but then... what a marvel!" But few never free themselves from the bad mood, at least within the two and a half weeks duration of the lab. Talking with these students is always very interesting because they are amazed when I tell them that, most of the times, 9 year old kids get along with the Turtle quite well. When trying to dig into this paradox, it appears that kids grasp quite naturally the playful aspect of the situation. On the other side, as we have pointed out in the previous chapter, adults easily get trapped in their own prejudices: I don't have a head for numbers, computer and me are very far away, I never went along with technologies. Especially the last one is a statement I heard lots of times, and, amazingly, from true digital natives. That is, a person who may have thousands of Facebook contacts, candidly claims of not getting along with technologies.

Having stressed that, please, do face the Turtle with the most playful spirit possible, starting from scratch. Since I suppose you know how kids tackle a new game, try doing just the same. Having stressed that, please, do face the Turtle with the most playful spirit possible, starting from scratch. Since I suppose you know how kids tackle a new game, try doing just the same. Now, provided you downloaded LibreOffice and activated the Logo toolbar, as explained in the first chapter, open a new text document. You are in front of a white sheet: you could write a letter, a shopping list, a poem or whatever you like. Try to write the following<sup>1</sup>:

FORWARD 100

What do you see? Forget your "limits" and everything you think about your

---

<sup>1</sup>I used uppercase characters just for clarity, LibreLogo is "case insensitive"

relationships with math, technology and so on. Look at the result you have obtained and compare it with what you have written. Ask yourself questions. Experiment with the Turtle about the possible answers that come in your mind. Before trying something new, write the following commands before the previous one:

```
CLEARSCREEN
HOME
```

With the first one you cancel the previous drawing, with the second one you send the Turtle in its "home" position, that is at the center of the sheet with its nose pointing upside.

```
CLEARSCREEN
HOME
FORWARD 50
```

Then, again, press the "Run" button. What's changed?

Now, in order to explore better what the Turtle can do, I tell you a couple of other commands: RIGHT and LEFT. Not difficult to imagine what are these commands for, isn't it? But something is missing, in order to be able to use it, what? Well, try to guess and experiment. Please, reflect on what's really going on with the Turtle, when you apply these commands. Then, just play to see what you can do. Remember, as you were I child...

The reason for I'm insisting so much about what children can do is that I had several opportunities to work with them and with the Turtle. really a few say it's too difficult, most just play and get excited to let the small animal create funny things. Once, trying to draw a house a girl came out with kind of a bizare castle. I praised her, commenting on how marvelous was this drawing, she was so proud. Then, when I came back to her, having considered the works of other kids, I found her in tears, sobbing desperately: she had accidentally erased the commands in a way that I have been no more able to recover. I felt guilty for not having taught her how to save the work once in a while.

Therefore, if you start creating some more complex stuff you like, don't forget to save the document once in a while. It's so easy.

In a past version of this book this chapter was much longer. Following the experience I have gathered during the last couples of years, both working with teacher students and with kids, I prefer to stop here, inviting you to freely explore the possibilities. From here you can go in several directions. If you want specific indications and examples for using the basic commands, or you want to discover more powerful instructions, you'll find them in the next chapter and following ones. In chapter ?? you can discover the possibilities of explorations by reading the story of Marta, a former teacher student of mine. Or, if you want to read about a fascinating way about introducing kids to the drawing of a circle, go and read chapter ??, well, the first part of it, otherwise you'll begin flying...



# Bibliography

- [1] Lewis Colleen. How programming environment shapes perception, learning and goals: Logo vs. scratch. [http://ims.mii.lt/ims/konferenciju\\_medziaga/SIGCSE'10/docs/p346.pdf](http://ims.mii.lt/ims/konferenciju_medziaga/SIGCSE'10/docs/p346.pdf), 2015. Accessed: 2017-08-13.
- [2] Weintrop David. Comparing block-based, text-based, and hybrid blocks/-text programming environments in introductory computer science classes. <http://dweintrop.github.io/papers/Weintrop-diss-4pager.pdf>. Accessed: 2017-08-13.
- [3] Weintrop David e Wilensky U. To block or not to block, that is the question: Students' perceptions of blocks-based programming. [http://dweintrop.github.io/papers/Weintrop\\_Wilensky\\_ICER\\_2015.pdf](http://dweintrop.github.io/papers/Weintrop_Wilensky_ICER_2015.pdf), 2015. Accessed: 2017-08-13.
- [4] Weintrop David e Wilensky U. Using commutative assessments to compare conceptual understanding in blocks based and text based programs. [http://dweintrop.github.io/papers/Weintrop\\_Wilensky\\_ICER\\_2015.pdf](http://dweintrop.github.io/papers/Weintrop_Wilensky_ICER_2015.pdf), 2015. Accessed: 2017-08-13.
- [5] László Németh. Esempi di uso librelogo (ungherese). <http://www.numbertext.org/logo/logofuzet.pdf>. Accessed: 2017-08-13.
- [6] Seymour Papert. *Mindstorms, Children, Computers, and Powerful Ideas*. Basic books, New York, 2 edition, 1993.
- [7] Lakó Viktória. Manuale di comandi di librelogo. [http://szabadszoftver.kormany.hu/wp-content/uploads/librelogo\\_oktatasi\\_segedanyag\\_v4.pdf](http://szabadszoftver.kormany.hu/wp-content/uploads/librelogo_oktatasi_segedanyag_v4.pdf). Accessed: 2017-08-13.