LibreLogo - Handbook

Turtle Geometry

Andreas R. Formiconi

Versione 0.1

January 2018

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

Contents

1.1 Aknowldegments

First af 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 contributes 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 syntonic 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 versione 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 [1]. 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 [6] and a more classroom oriented one by Lakó Viktória [8]. Some of the examples presented in this work have been inspired by those in Lakó Viktória's book, but the perpespective 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* [7]. 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.

---

[1]The commands dictionaries can be downloaded from https://help.libreoffice.org/Writer/LibreLogo_Toolbar

8

Contents

1.3 Acknowledgments

First af 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 contributes 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 developped 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 syntonic 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[2]. 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 [6] and a more classroom oriented one by Lakó Viktória [8]. 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* [7]. 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.

---

[2]The commands dictionaries can be downloaded from *https* : //help.libreoffice.org/Writer/LibreLogo$_T$oolbar

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, Inskape 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) [7]:

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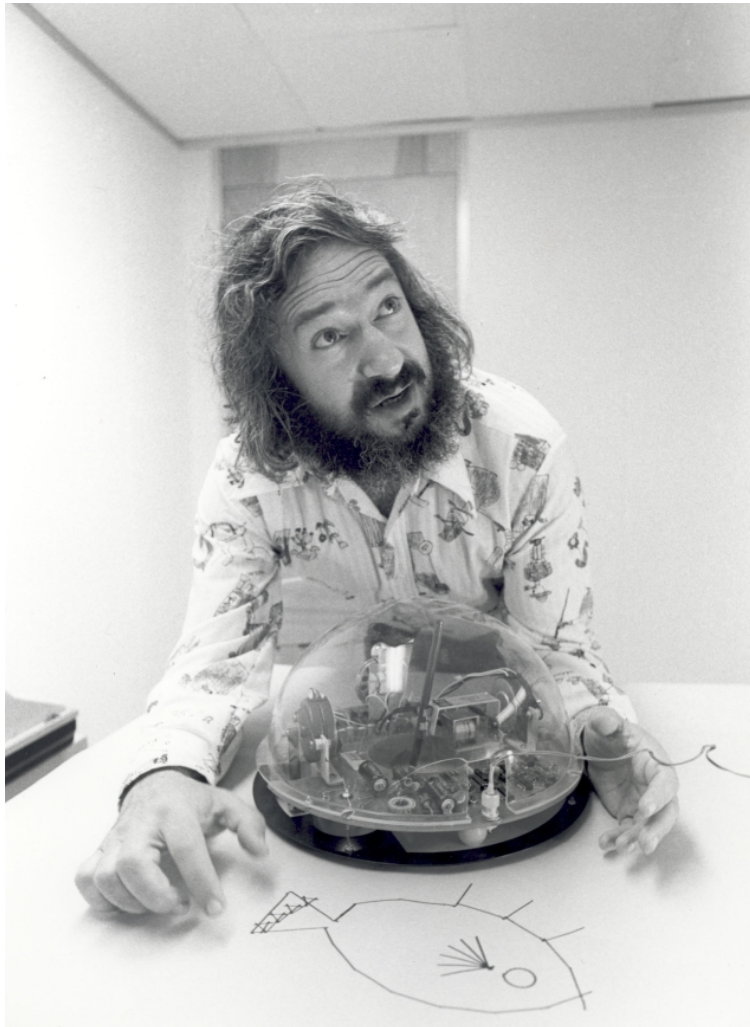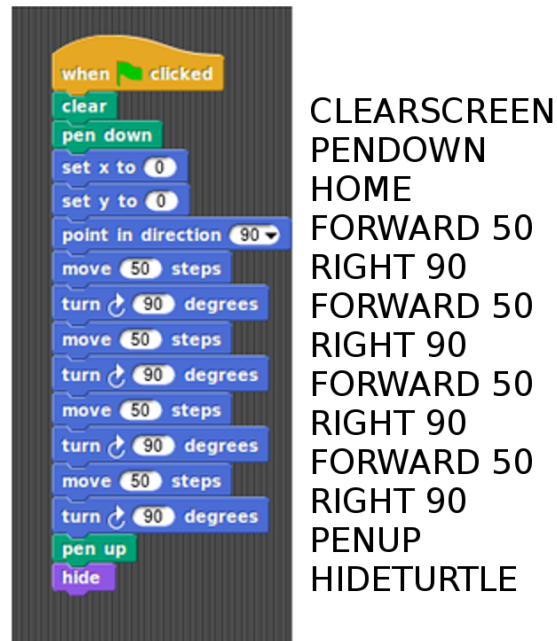
Figure 2.1: Seymour Papert shows one of the first versions of Logo, when it was kind of a robot for draving.

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 inabundance, 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 [3]. 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 [5]. 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 [2]. 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" [4]

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[1], 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[2]:

- 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

---

[1] 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.

[2] 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* ³ 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, he further improved 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**:
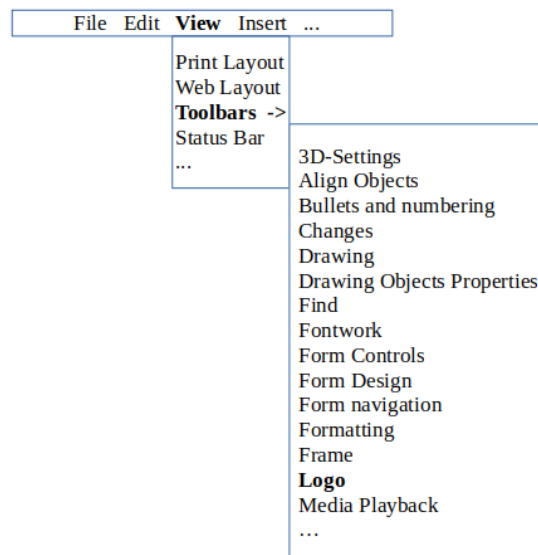
Once this is done, you must close the program and relaunch it to see, among the other toolbars also the LibreLogo one:

where the icons have the following meanings:

---

³Di *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 [4] 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.

---

[4]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[5]. 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.

---

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

Chapter 3

The fear of math

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, *Mathofobia*, 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[1]; that the initial motivation, based on a serious and difficult revisitation of the way young people are introduced to math, has ended up diluted in the pot

---

[1]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 Wathsapp 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, Finlandâs 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 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.

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.

3.1   *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 neoliberistic 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.

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 mathemathics. 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 [7] (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[2]. 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 where 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 then 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 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

---

[2][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.

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. A 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 *appropriable*. 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

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 LibreoOffice 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[1]:

FORWARD 100

---

[1]I used uppercase characters just for clarity, LibreLogo is "case insensitive"

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

Chapter 5

Drawing

5.1 Moving the Turtle around – Drawing - Using variables

5.1.1 Basic commands

The program allows to create graphics through the movements of a "turtle" who obeys specific commands. Let's just see an example. Open a new text document and write this command:

HOME

You'll see that the turtle just appeared in the middle of the screen with its head up:

You'll see that the turtle just appeared in the middle of the screen with its head up:

Now, add another command (from now on I'll write in bold just the new commands that are being introduced):

HOME
FORWARD 100

The turtle has moved, drawing a line; this is the way you can draw by giving commands to the turtle. Let's now write the following instructions:

HOME
FORWARD 100
RIGHT 90
FORWARD 50
RIGHT 90
FORWARD 50
RIGHT 90
FORWARD 50
RIGHT **90**
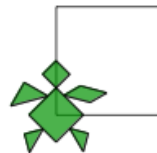
We have drawn four sides, and at the end of each side we turned right by 90°, thus obtaining a square. Just two words about the peculiarity of LibreLogo. The sequence of instructions we wrote is a bit of code, it's a software. We wrote it in a particular kind of language, which is Logo's, and is it also very simple, but it's a software just like any other. Usually the software is written in special documents using simple text editing and context editing and it's saved that way. Then, the computer executes them. The ways these operations are being executed vary a lot, depending on the kind of languages and the context. Nowadays there are hundreds of different languages that are used for the most different goals. The peculiarity of LibreLogo is that the software is written in a document and the turtle "works" on the same document, leaving its mark in a graphic way. So, one obtains both the code and its graphic result all together in one document. The graphics can be selected with the mouse and, eventually, may be transferred to various contexts. For example, I created the above pictures by playing with the turtle in another document, then I selected the graphics and I brought them here. Another notation: a group of instructions made to operate in sequence is called script, expression that we will use profusely.

The interaction between Logo and LibreOffice goes beyond that. It's evident that when we write this command

FORWARD 50

the number 50 expresses the length of the path that the turtle needs to make. You can verify this at once by changing the value and watching what the turtle does instead.

But what does that 50 stand for? They are typographic points: 50 pt. One point is 0.35 mm[1]. LibreLogo understands unity measurements, so you can write 50, 50pt, 50mm, 50cm, 50in (inches), 50 " (" stands for inch). They are all different lengths, of course. Let's use mm, for example:

```
CLEARSCREEN
HOME
FORWARD 50mm
RIGHT 90
FORWARD 50mm
RIGHT 90
FORWARD 50mm
RIGHT 90
FORWARD 50mm
RIGHT 90
```
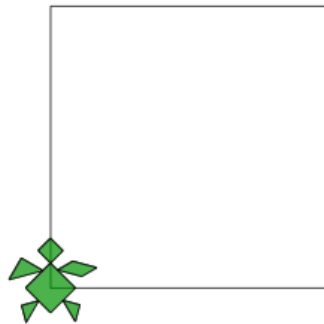
The square is obviously bigger because the previous one was 50 pt = 17.6 mm. Let's try and make the drawing more complicate, imagining to draw a house. The turtle is located on the bottom left angle, looking upward. Firstly, we need to make it go up to the angle on the top left. Let's try, and at the same time, let's take advantage of the fact that the instructions can be conveniently grouped on the same line, if we need it:
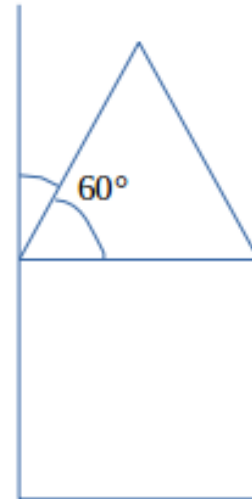
```
CLEARSCREEN
HOME
FORWARD 50mm RIGHT 90
FORWARD 50mm RIGHT 90
FORWARD 50mm RIGHT 90
FORWARD 50mm RIGHT 90
FORWARD 50mm
```

---
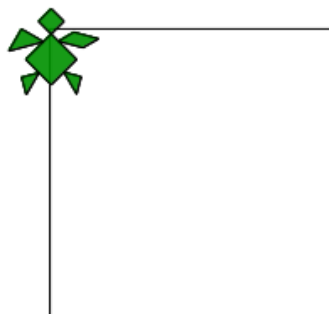
[1]The correct definition is: 1 pt = 1/72 inches, where 1 inch = 25.4 mm. Therefore 1 pt = 2.54/72 = .352$\overline{7}$ mm

There are no strict rules to group instructions on the same line, but it can be useful to read the code easily. It's vital to get things smooth because, as the code keeps growing, it can rapidly become complicated and all the tricks to make it nice and simple are useful.

Now we need to build the roof of the house. We can do this by drawing an equilateral triangle over the square, its base coinciding with the upper side of the square. Being equilateral, the other sides of the triangle will have to be 50mm long as well. So, in order to make the left side of the roof, the turtle will need to move 50mm, but before that, it needs to change direction. How much? Since the internal angles of an equilateral triangle are 60°each, the turtle needs to detour by 90°-60°= 30°on the right. Once it has reached the top, drawing the left side of the roof, it will have to turn right by 120°in order to draw the right side. Finally, it will turn by 30°on the right to align itself to the wall of the house.

```
CLEARSCREEN
HOME
FORWARD 50mm RIGHT 90
FORWARD 50mm RIGHT 90
FORWARD 50mm RIGHT 90
FORWARD 50mm RIGHT 90
FORWARD 50mm RIGHT 30
FORWARD 50mm RIGHT 120
FORWARD 50mm RIGHT 30
```
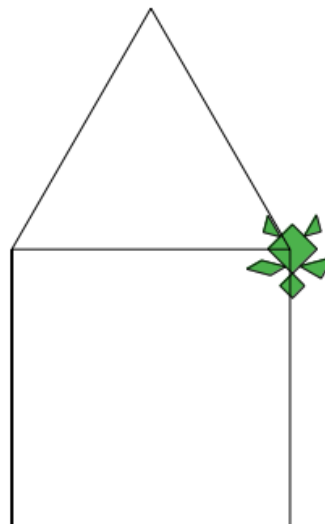
Now, let's suppose we want to draw a window in the middle of the wall. To do that it's necessary to introduce two new commands - PENUP and PENDOWN - that allow to move the turtle without drawing.
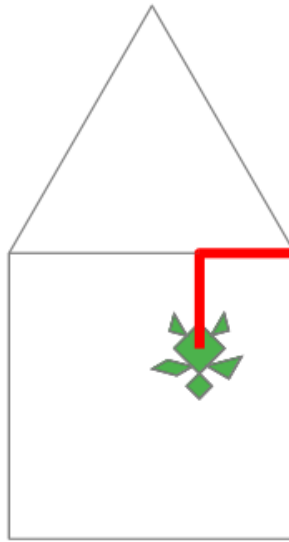


```
CLEARSCREEN
HOME
FORWARD 50mm RIGHT 90
FORWARD 50mm RIGHT 90
FORWARD 50mm RIGHT 90
FORWARD 50mm RIGHT 90
FORWARD 50mm RIGHT 30
FORWARD 50mm RIGHT 120
FORWARD 50mm RIGHT 120
PENUP
FORWARD 50mm/3 LEFT 90
FORWARD 50mm/3
```
we underlined red the path that was made without leaving a mark.


```
CLEARSCREEN
HOME
FORWARD 50mm RIGHT 90
FORWARD 50mm RIGHT 90
FORWARD 50mm RIGHT 90
FORWARD 50mm RIGHT 90
FORWARD 50mm RIGHT 30
FORWARD 50mm RIGHT 120
FORWARD 50mm RIGHT 120
PENUP
FORWARD 50mm/3 LEFT 90
FORWARD 50mm/3
PENDOWN
FORWARD 50mm/3 RIGHT 90
FORWARD 50mm/3 RIGHT 90
FORWARD 50mm/3 RIGHT 90
FORWARD 50mm/3 RIGHT 90
```
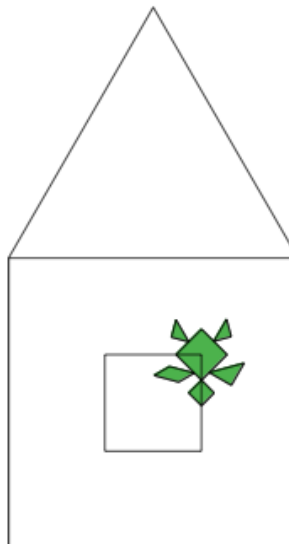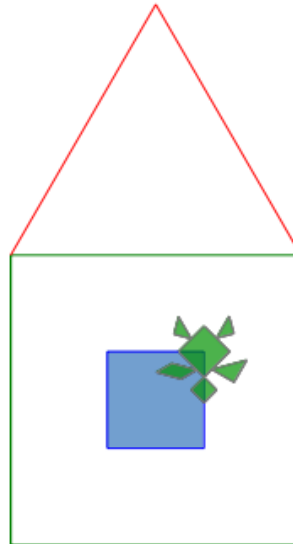
Let's try and enrich the drawing further, i.e. using colours.

```
CLEARSCREEN
HOME
PENCOLOR "green "
FORWARD 50mm RIGHT 90
FORWARD 50mm RIGHT 90
FORWARD 50mm RIGHT 90
FORWARD 50mm RIGHT 90
FORWARD 50mm RIGHT 30
PENCOLOR "red "
FORWARD 50mm RIGHT 120
FORWARD 50mm RIGHT 120
PENUP
FORWARD 50mm/3
LEFT 90
FORWARD 50mm/3
PENDOWN
PENCOLOR "blue "
FORWARD 50mm/3 RIGHT 90
FORWARD 50mm/3 RIGHT 90
FORWARD 50mm/3 RIGHT 90
FORWARD 50mm/3 RIGHT 90
PENCOLOR "gray "
```

And why not colour the inside as well?

```
CLEARSCREEN
HOME
FORWARD 50mm RIGHT 90
FORWARD 50mm RIGHT 90
FORWARD 50mm RIGHT 90
FORWARD 50mm RIGHT 90
FORWARD 50mm RIGHT 30
FILLCOLOR "yellow " FILL
FORWARD 50mm RIGHT 120
FORWARD 50mm RIGHT 120
PENUP
FORWARD 50mm/3
LEFT 90
FORWARD 50mm/3
PENDOWN
FILLCOLOR "red " FILL
FORWARD 50mm/3 RIGHT 90
FORWARD 50mm/3 RIGHT 90
FORWARD 50mm/3 RIGHT 90
FORWARD 50mm/3 RIGHT 90
FILLCOLOR "green " FILL
```

In this case as well we used the grouping of instructions:

FILLCOLOR "green " FILL. With the first one we establish which colour to use (FILLCOLOR "green ") and with the second one we proceed to colour the picture - it comes natural to gather them together in one instruction.

The instruction FILL does two things, actually: it closes one picture and it fills it with a colour. Let's try:

FORWARD 30mm RIGHT 90
FORWARD 30mm RIGHT 90
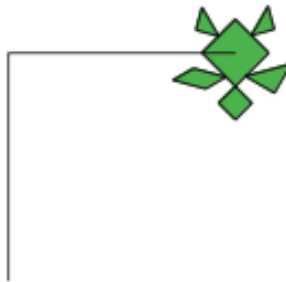
This way we have drawn an open picture. If we want to make a triangle, we may let the turtle draw the third side. Alternatively, we may close the picture with FILL, as seen before:

FORWARD 30mm RIGHT 90
FORWARD 30mm RIGHT 90
FILL
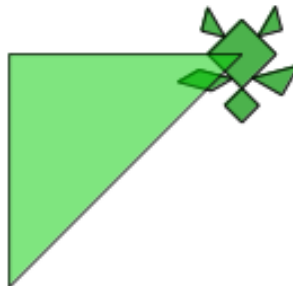
This way, the picture has been closed and coloured. We may even close it without colouring it, using the instruction **CLOSE**:

```
FORWARD 30mm RIGHT 90
FORWARD 30mm RIGHT 90
CLOSE
```

It's interesting to notice that, both with FILL and CLOSE, the picture is closed without moving the turtle. So, if we keep adding further moves, they will take place from the turtle's position, that in the previous examples has remains with its head pointing downward. Let's try and set the two previous code blocks one after the other:

```
FORWARD 30mm RIGHT 90
FORWARD 30mm RIGHT 90
FILL
FORWARD 30mm RIGHT 90
FORWARD 30mm RIGHT 90
CLOSE
```

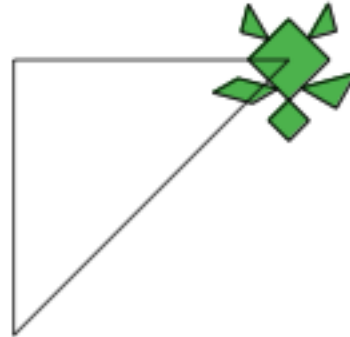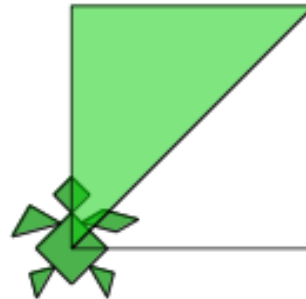After closing and colouring green the upper triangle, the turtle, with the following FORWARD instructions, procedes to draw two sides of the lower triangle. The instruction CLOSE makes close this second triangle without colouring it. What if one would want to remove the turtle once the picture is finished? Easy: just add at the end the instruction **HIDETURTLE**. Try!

5.1.2   RGB codes for making colors

In the previous examples we used codes to express colours, for instance "red". It's possible to name 24 colours:

BLACK
SILVER
GRAY
WHITE
MAROON
RED
PURPLE
FUCHSIA
GREEN
LIME
OLIVE
YELLOW
NAVY
BLUE
TEAL
AQUA
PINK
TOMATO
ORANGE
GOLD
VIOLET
SKYBLUE
CHOCOLATE
BROWN
INVISIBLE

Actually the available colours are many more. They can be expressed through the so called RGB code, that stands for Red, Green, Blue. In computer graphics colours can be expressed as a combination of three main colours, which are precisely red, green and blue. All the others may be expressed by mixing and dosing properly these three basic colours. In order to do that, the intensity of every colour is expressed with a number that goes from 0 to 255 and they are mixed by writing for instance [255,0,0] for red, [0,255,0] for green or [0,0,255] for blue. All the other colours are obtained by varying the parameters inside the square brackets. With [0, 0, 0] one obtains black and with [255, 255, 255] white. This code is used in any of the commands that allow to specify the colour. For example

PENCOLOR [45, 88, 200] FILLCOLOR [255, 200, 100]

Just play for a while with these numbers and see which colours come out.

The colours wheel shown above has an oddity: there are two elements with

different name but the colour seems the same. Found them? They actually differ, but because of a fourth attribute, which is transparency. Let's see while playing along.

To make things easier, we'll anticipate a new command. In the previous examples we already drew a square, using the instructions FORWARD and RIGHT, conveniently combined. In reality, in order to draw the main geometric shapes, in Logo there are pre-established instructions, that help us write the code synthetically. The square is one of these. In order to make a 50mm side square you write:

CLEARSCREEN
HOME
SQUARE(30mm)

This way the turtle draws a square, then places itself in the cetre with its head upward (we didn't choose the colour, but it came out "automatically", by default). Let's use this instruction, first to draw a square that we'll colour red, then we'll move a bit right down, to draw a second square, making it overlap a little to the previous one, then we'll colour it white.

CLEARSCREEN
HOME
FILLCOLOR [255, 0, 0]
SQUARE(50mm)
FORWARD -25mm
RIGHT 90
FORWARD 25mm
FILLCOLOR [255, 255, 255]
SQUARE(50mm)
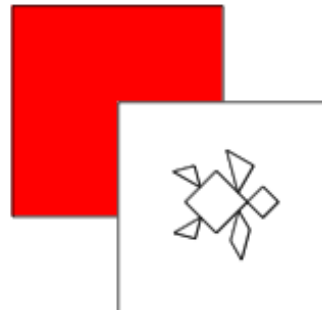
Good, now the white square overlapped to the red one. A foreseeable result: the colours overlap because they are opaque. A painter would say that they "cover". Actually, with RGB codes in Logo you can assign a fourth parameter: the "transparency", that you may want to give to a certain colour. With a value 0 you give complete opacity, whilst the value 250 makes it completely transparent. With values between 0 and 250 you can obtain different levels of transparency. So, let's try and make the white square transparent, adding the fourth parameter equal to 255:
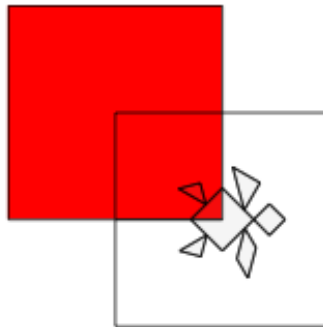
```
CLEARSCREEN
HOME
FILLCOLOR [255, 0, 0]
SQUARE(50mm)
FORWARD -25mm
RIGHT 90
FORWARD 25mm
FILLCOLOR  [255,   255,   255,
255]
SQUARE(50mm)
```

The white square has become transparent. If, instead of FILLCOLOR [255, 255, 255, 255] we would have used FILLCOLOR "invisible ", then we would have obtained the same result. That's the difference between "white" and "invisible", in the colour wheel we saw at the beginning. One last remark. Looking closely, you can see that in these pictures our turtle have different colours. In fact, the turtle is coloured with the same colour we selected to fill the last picture, maybe with a different transparency. After drawing the square light green, the turtle came out a shade darker. After the white square, it came out light grey and after the invisible colour it came out dark grey. This behaviour does not affect the results we want to obtain because, should we want to use the graphics we are producing in any possible way, in the end we'll just need to add the instruction HIDETURTLE.

5.1.3   More commands

With these basic commands seen so far, it is possible to produce a large amount of graphics. In Logo however there are commands that allow to shorten the picture of some basic forms. We already anticipated the instruction SQUARE that allows to build a square at once. The others are RECTANGLE, CIRCLE and ELLIPSE. Let's try the following commands:

```
CLEARSCREEN
HOME
SQUARE 50
CIRCLE 50
```

You'll see how drawing two pictures in sequence has the effect of overlapping them and making the symmetry centre coincide, and how the turtle always repositions itself on that centre. It's also evident how the topic[2] of command SQUARE represents the square's side and the topic of command CIRCLE represents the circle's diameter that we want to build.

Try and practise with the commands SQUARE and CIRCLE, for instance by building this locomotive. You may practise to respect the given proportions, like those in the example, where the side of the bigger square is two or three times bigger than the smaller squares and the circles' diameter is the same length of the side of the smallest square. You'll find the solution on the next page, but first try to work it out on your own.

One important thing to understand with the code is that the same goal can be achieved in many different ways. There's no absolute criterion to establish the best procedure. Therefore there is no "right answer". One procedure may be better than another on a certain point of view: clarity of the written code, fastness of execution, total memory used from the code, management of possible resources etc. It can happen that a very good code under one of these terms may result a very bad one on another.

---

[2]by "topic" it is intended the value that a command requires to be operated. There can be more than one topic - we'll see some examples

```
COLOR "red "
SQUARE 60
PENUP BACK 15 RIGHT 90
FORWARD 45
LEFT 90 PENDOWN
SQUARE 30
PENUP FORWARD 25 PEN-
DOWN
FILLCOLOR "black "
SQUARE 20
PENUP BACK 40 PENDOWN
CIRCLE 20
PENUP LEFT 90 FORWARD
60 PENDOWN
CIRCLE 20
HIDETURTLE
```

The square and the circle need just one parameter to be executed. Instead, the rectangle and ellipse need two parameters. For the rectangle, the two parameters are the length of the long side and the short side. The instruction to build the rectangle is the following:

RECTANGLE [60,40]

This command produces a rectangle 60 points large and another of 40. In comparison to the square and the circle's cases, this command has another peculiarity: in this case, in order to give the two needed parameters, we resorted to the writing [**60,40**]. This is a "list" of values. It's a way of considering a bunch of values as a whole, a list in fact, and the way to obtain this result is to list the values, split them with commas and close them between square brackets. The lists may be useful in various circumstances, not only in this case, but we'll see how later.

Esercizio: try to make a figure like the one below
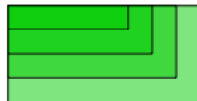
In this case as well, try out on your own. Then, scroll down the page to see a possible way of solving it.

```
RECTANGLE [40mm, 20mm]
PENUP FORWARD 2,5mm LEFT 90
FORWARD 2,5mm RIGHT 90 PENDOWN
RECTANGLE [35mm, 15mm]
PENUP FORWARD 2,5mm LEFT 90
FORWARD 2,5mm RIGHT 90 PENDOWN
RECTANGLE [30mm, 10mm]
PENUP FORWARD 2,5mm LEFT 90
FORWARD 2,5mm RIGHT 90 PENDOWN
RECTANGLE [25mm, 5mm]
HIDETURTLE
```

Another picture that can come in handy is the ellipse. To put it simple, the ellipse is a flattened circle. It's perfectly in the spirit of the Turtle's geometry to get help by using physical activities references: probably the best way of using modern technologies is to keep on using the traditional ones, making the most out of both by integrating them. Nothing better than to get a little help from Emma Castelnuovo [1], who writes about the emerging of the ellipse while studying isoperimetrical triangles with same base. Here is the whole writing, in order to respect Emma's valuable didactic intent:

> Another mathematical subject, as the study of isoperimetrical triangles with the same base, brings us to ponder what we have in front of us. The material is, even this time, a piece of string.
>
> In order to build triangles with same base and same perimeter, we'll do it this way: let's fix two nails - A and B - on a table over which has been laid a sheet of paper; AB will be our triangles base. Let's tie both ends of a string to the nails, bearing in mind that the string needs to be wider than the AB side. Let's make sure, with the aid of a pencil, that the string stays stretched and... let the pencil lead us.
>
> This pencil, guided by the string, will draw on the paper an oval shaped curve: it's an ellipse. The points A and B are the focuses of the ellipses. Therefore: the isoperimetrical triangles tops with same base are on an ellipse. A geometry problem led us to the drawing of the ellipse. With the same string we can make a more or less "flat" ellipse, depending on the distance between the points A and B. We can obtain a circle, if these two points coincide: the circle, actually, is a particular type of ellipse.
>
> We find the ellipse again on the street, after dealing with it in geometry problems, when we "step" over it (a street sign's shadow makes an ellipse). In our hectic lives, we rarely pause to notice the shadow that the sun or a lamp cast over an object. Yet now a geometry activity prompts us to look closely and it's indeed the comparison between the shadow cast from the sun or a lamp that arouses our observation skill.
>
> Let's take, for instance, two pencils vertically laid on a table. If they are lit by the sun, their shadow tend to be parallel; on the contrary, if they are lit by a lamp, then their shadows divert.
>
> Hence the mathematical study of affine transformations and projective transformations, to reach perspective, art, the way one looks at

a painting, at history. It's a small geometry problem that challenged
to observe and to ... take a look around.
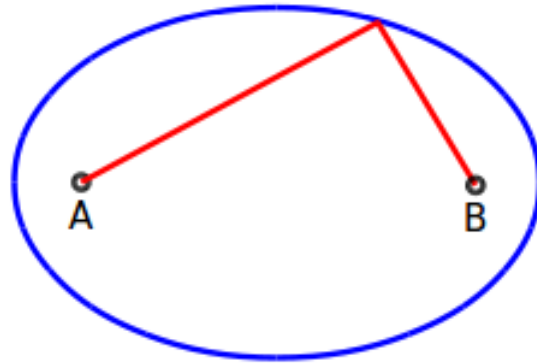
In LibreLogo the ellipse is drawn with the command

ELLIPSE[40, 20]

Obviously the ellipse does not have a fixed diameter, like the circle. For
this reason, to be defined it requires two parameters: the two axis, bigger and
smaller. In the example the ellipse has the major axis equal to 60 points and the
minor equal to 40 points. It's possible to combine various forms and adjust their
parameters to obtain a variety of effects. For instance, a circle inscribed in a
square can be obtained also by starting from the instructions to draw rectangles
and ellipses:

```
RECTANGLE [60, 60]
ELLIPSE [60, 60]
```

Try to draw a figure like this one:

Possibile answer:

```
ELLIPSE [120, 80]
PENUP FORWARD 10 PEN-
DOWN
ELLIPSE [90, 60]
PENUP FORWARD 10 PEN-
DOWN
ELLIPSE [60, 40]
PENUP FORWARD 10 PEN-
DOWN
ELLIPSE [30, 20]
PENUP FORWARD 10 PEN-
DOWN
```

These instructions also allow to use other parameters to obtain rectangular and ellipsis variables. In the case of rectangles it's possible to adjust a third parameter in order to make the vertexes rounded:

RECTANGLE [60, 50, **10**]

Now, let's suppose a friend just told us about this opportunity, but he can't remember anything else. Can we control the "roundness" of the vertexes? Probably with that third parameter, our friend told us to assign it a 10 value, but how does it work? This is a great example to highlight the crucial tool of software developers: experimentation. So, in this case, we could make some attempts, that we might summarize like this:

```
FILLCOLOR "invisible "
PENCOLOR "green "
RECTANGLE [200, 150, 0]
PENCOLOR "black "
RECTANGLE [200, 150, 10]
RECTANGLE [200, 150, 20]
RECTANGLE [200, 150, 30]
RECTANGLE [200, 150, 40]
RECTANGLE [200, 150, 50]
RECTANGLE [200, 150, 60]
RECTANGLE [200, 150, 70]
RECTANGLE [200, 150, 80]
RECTANGLE [200, 150, 90]
RECTANGLE [200, 150, 100]
PENCOLOR "red "
ELLIPSE [200,150]
```
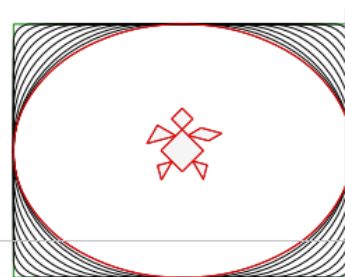
This way we worked out how to control rounded rectangles. We also realized that, playing with the third parameter, we can go from the extreme case of a normal rectangle to the real ellipse. Let's see now the possible variants for the instruction ELLIPSE.

ELLIPSE [100, 50]

ELLIPSE [100, 50, 0, 90, 1]

ELLIPSE [100, 50, 0, 90, 3]

ELLIPSE [100, 50, 0, 90, 2]
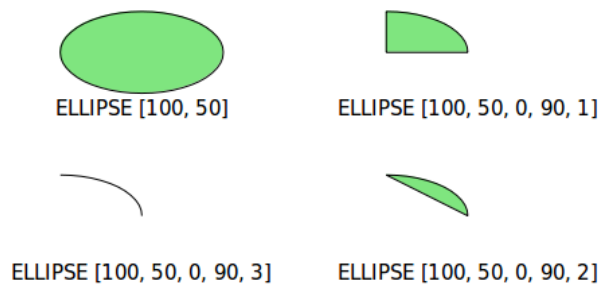
So in the instruction ELLIPSE we can use 3 additional parameters, to draw just one area of the ellipse. The first two represent the initial and final angle, expressed in degrees, that limit the area. In the above example, having chosen 0 and 90 we established to draw the first quadrant of the ellipse. The fourth parameter determines whether to draw one sector of the ellipse (1), one segment of the ellipse (2) or just an arch (3).

5.1.4   Variables

The instructions we have seen so far allow us to do many things: we learned to move the turtle anywhere around the screen, to make it draw or not, we saw how to control the colour of the line and the filling of figures. We may get the impression that, to make graphics, we won't need anything more. Instead we just scratched the surface of potentialities of a programming language, even if for educational purposes, as in Logo and its derived. We'll introduce the main characteristics as we go. The first of which, that we need at once to get going, is the concept of "variable". So far we used various instructions that require "arguments". The argument is the value that an instruction may require to be executed. For instance the instruction FORWARD wouldn't make any sense without an argument that represents the distance that the turtle needs to cover. The expression FORWARD 50 means that the turtle has to move forward by 50 points; 50 is the value of the argument. There are also instructions that require more than an one argument, for example the RECTANGLE and ELLIPSE. Basically, in all the examples seen before we always used numerical arguments for all the instructions. Actually, every language allows to understand an important

generalization of the use of "variables". They are symbolic names that can be assigned to convenient numerical values. Let's try and execute the following code:

```
CLEARSCREEN
HOME

LATO = 100
ANGOLO = 90

FORWARD LATO
LEFT ANGOLO
FORWARD LATO
LEFT ANGOLO
FORWARD LATO
LEFT ANGOLO
FORWARD LATO
HIDETURTLE
```

[297.86456692913384, 421.11496062992126]
OK   Cancel
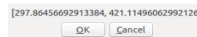
We have drawn a square, but instead of using directly the value 100 as an argument of the instructions FORWARD, first we assigned the value 100 to the variable named "SIDE" and then we used this as an argument of all the instructions FORWARD. The utility of this method is evident: let's suppose I am not satisfied of this square's dimensions and that I want to try other values of the side. Well, I just need to change the value 100 into the instruction SIDE=100, changing it for example with SIDE=150. Feel free to experiment! You may even want to change the value of an ANGLE, trying out different values... For those who studied algebra basics, they will certainly have recognized the concept of variable, that in that discipline is used widely to make simple symbolic calculus. They will also remember that the concept of variable is used in various ways, to express quantities that are considered variables - for instance a dependant variable in function of other independent variables - then quantities that we assume constant, finally quantities that assume the meaning of parameters, that are like constants we may be interested to change from time to time. In any case, all these quantities are represented in a symbolic way. Actually, those who had time to deepen algebra studies, will know that the concept of variable is passable of a series of generalizations. No fear, this is not a Maths class in disguise, or maybe it is, in a way: after all Logo represents Seymour Papert's goal to make Maths more accessible. But what we are doing here does not require special attitudes or skills. We are just introducing one of the possible generalizations, that we are going to need at once. The generalization we are proposing concerns the concept of the turtle's position on the screen. The position along aline is determined by a simple number - for instance the position along a road: "Road work at km...". A different case is that of the position on a surface. In a sat nav, a familiar example, you can give the position in geographic terms, but this has to be given through two values: latitude and longitude, denoting the parallel and the meridian. In order to sink a boat at a naval battle board game, you must give two coordinates, for instance b7, where "b represents the column and 7 the line". Likewise to identify cells on a spreadsheet, and so on. Our turtle also needs two values in order to identify a specific place on the sheet, that we can imagine as the x and the y of the turtle in the space of the page. So, the way to express this concept in the turtle's world (and not only that!) is the following:

```
P=[200, 300]
PRINT P
```

If you execute this code, LibreLogo prints the "value" [200, 300]. Obviously we chose these numbers randomly, just to make an example. The aim is to show how to represent in a symbolic way a position, that actually is expressed by two numbers. In algebra this kind of variable is called a "vector". There is also another way to "isolate" the single elements inside a vector. Let's describe this with this example:

```
P=[200, 300]
PRINT P
PRINT P[0]
PRINT P[1]
```

If you execute this fragment of code, first the "value" [200, 300] is printed, then the value 200, then 300. Therefore it's clear that with P [0] one obtains the first element of the position vector, which contains the number 200, and with P [1] the second element, which contains the number 300. Therefore, this is how much is needed to go and see how you can control even more easily the position of the turtle on the screen.

5.1.5   The page space

We already saw various commands to move the turtle, but all of them have the goal of drawing. Actually you can make movements with the "lifted pen" (PENUP command) but it also can be useful to "jump" directly to any position on the screen, or point to a particular direction. It's all about, in other words, to choose a position or a direction in absolute terms and not relatively, related to the present position and direction, just like it's done for instance with instructions like FORWARD or LEFT. Hence the need to use spacial references as a couple of coordinates for the position on the sheet and an angle for the direction. To understand how these references work, let's introduce and use two new instructions at once: **POSITION** and **HEADING**. Moreover, the instruction **PRINT** is helpful to know the current value of the position and of the direction. In fact, these two commands **POSITION** and **HEADING**, can be used with and without parameters. When they are used without parameters, they provide the current values. In fact, if I open a new document in Writer and execute the command

```
PRINT POSITION
```

I obtain the following answer:

[297.86456692913384, 421.11496062992126]

OK   Cancel

The two numbers between brackets represent the coordinates x and y of the position in the space of the page: 298 e 421^3 respectively. Since we have just opened the document and that at the beginning the turtle is placed in the centre, we can assume that these coordinates represent the centre of the page. However, in order to have the complete control of the situation, we need to know exactly the extension of the image's space. Now, the coordinates of the upper left angle are [0, 0], where the first number represents the coordinate x and the second the y, whilst those in the lower right angle are easily obtained by printing the value of the special variable **PAGESIZE**, that LibreLogo uses to maintain the page's dimensions. In this moment my version of Writer is set for pages A4 and, consequently, by executing the instruction

PRINT PAGESIZE

we obtain:

[595.303937007874, 841.8897637795275]

OK   Cancel
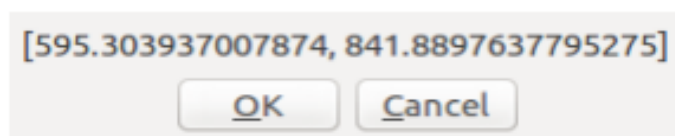
These numbers, rounded up, 596 and 842, represent the dimensions of a A4 sheet expressed in "points" at the density of 72 DPI/footnoteThe acronym DPI stands for dots per inch. The value of DPI depends on the physical support over which an image is supposed to be represented. Since an inch is 2.54 cm, 72 DPI density corresponds to $72/2.54 = 28.3$ points for cm, or 2.83 points for

---

^{3}we approximated the two numbers to four significative digits, that are adequate to determine the position on the sheet, to our goals. In the following note we'll give a short explanation of the unit of measurement used for these numbers

mm; just to have a more familiar reference. When in Writer you choose the measurement unit "points" (instead of cm or inches), these refer to the density of 72 DPI just cited. In Writer, the measurement unit can be changed by clicking Tools-¿Options-¿LibreOffice Writer-¿General. you can choose between mm, cm, inches, pica, points.. In mm it will be 210 and 297 mm. Let's recap the situation with the following picture.
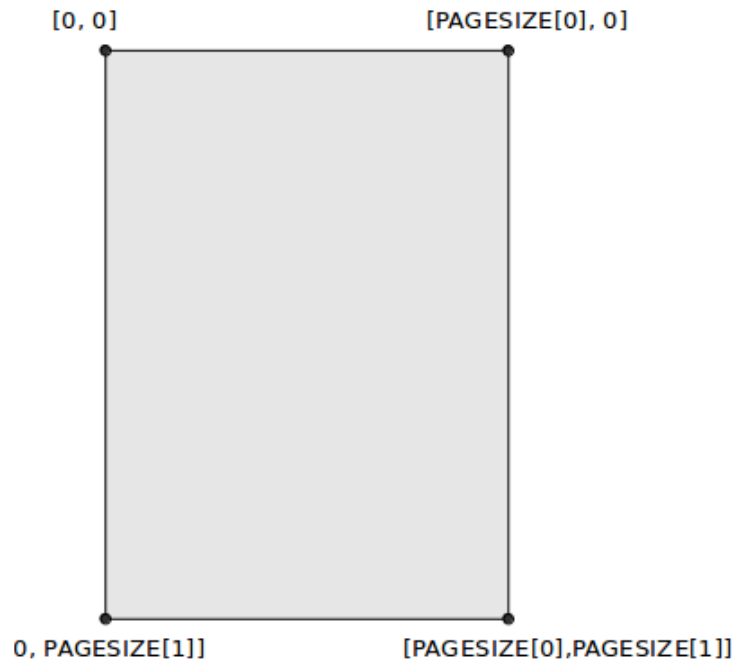
Every time we want to use absolute positions we can refer to this scheme that gives us the orientation of the referral system on the sheet and its dimensions. In the next chart we can see the equivalent numeric values:

[0, 0]	[0, 0]
[PAGESIZE[0], 0]	[596, 0]
[0, PAGESIZE[1]]	[0, 842]
[PAGESIZE[0], PAGESIZE[1]]	[596, 842]

Reconnecting to the previous paragraph, PAGESIZE is a variable treated as a constant inside LibreLogo because it contains the dimensions of the page. Besides it's that particular kind of variable that is called vector, because it's composed by more elements, precisely two, the two dimensions of the page:

PAGESIZE[0] is the width and PAGESIZE[1] the length. Then how can we send the turtle to a more precise position?

[0, 0]   [PAGESIZE[0], 0]

[350, 320]

[298, 421]

, PAGESIZE[1]]   [PAGESIZE[0],PAGESIZE[1]]
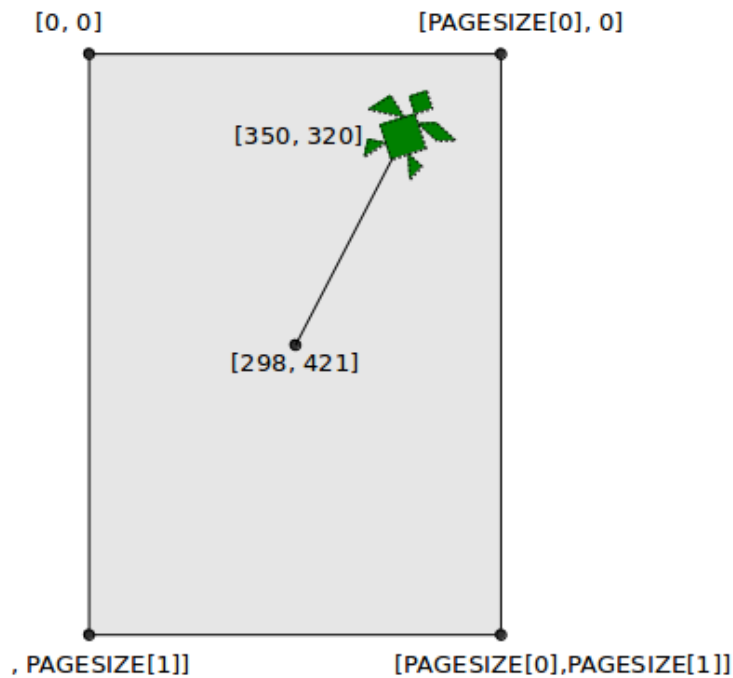
With the instruction **POSITION [350, 320]** the turtle moves directly to the coordinates point **POSITION [[350,320]**, starting from the point where it is, in this case from the centre of the page. As can we use the instruction PO-SITION to control the position, likewise we can use the instruction HEADING to control the direction where the turtle is heading. Claiming the instruction HEADING without any parameter we obtain the current position. By using another parameter, for instance HEADING [30], we make the turtle rotate by 30°. In the next picture we show the orientation of the reference system.

HEADING 0

HEADING 30

30°

HEADING 90
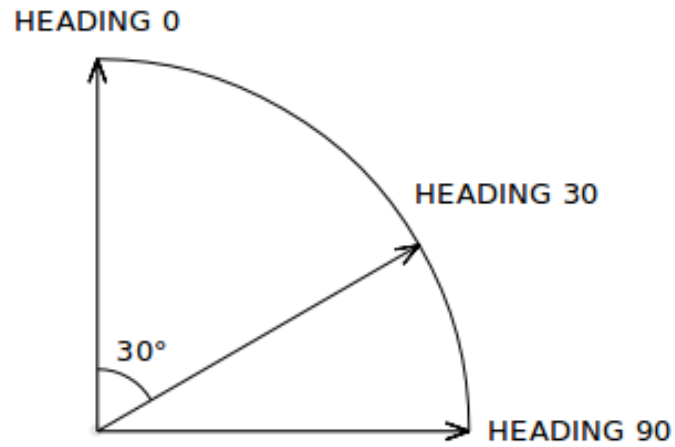
When we open a new document, or after the instruction HOME, the turtle points towards the upper side of the sheet, and this direction corresponds to 0°.

5.1.6   more graphics commands

It's possible to control other aspects of the drawing as well, in addition to the colour.

PENWIDTH (pen thickness)

With the command **PENWIDTH** we determine the thickness of the line:

---------------------- PENWIDTH 0

———————————— PENWIDTH 1

━━━━━━━━━━━━ PENWIDTH 2

━━━━━━━━━━━━ PENWIDTH 3

━━━━━━━━━━━━ PENWIDTH 4

━━━━━━━━━━━━ PENWIDTH 5

━━━━━━━━━━━━ PENWIDTH 6

━━━━━━━━━━━━ PENWIDTH 7

━━━━━━━━━━━━ PENWIDTH 8
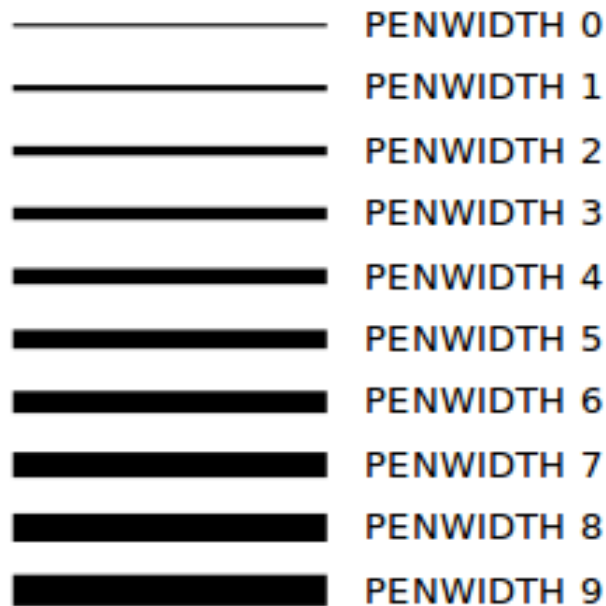
━━━━━━━━━━━━ PENWIDTH 9

PENJOINT (joining of segments)

With the command **PENJOINT** we control the vertexes form:

Let's draw a triangle, for example, in the following way:

```
PENWIDTH 5
FORWARD 40 RIGHT 120
FORWARD 40 RIGHT 120
FORWARD 40 RIGHT 120
HIDETURTLE
```

We can alter the vertexes overlay by giving the instruction PENJOINT and an appropriate argument. Le possibilities are the following:
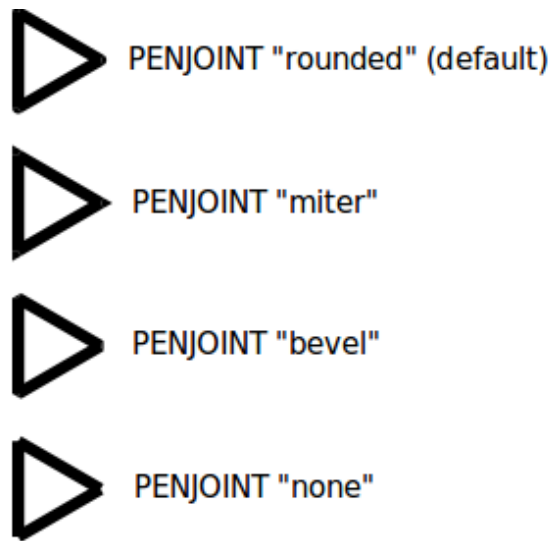
The argument "rounded" (default) mustn't be written in the command: we added it to point out that that is the standard behaviour if anything else is specified. If however we just used one of the other options, then it has to be used explicitly the command **PENJOINT "rounded "**, to obtain rounded vertexes. The argument "miter" in carpentry means to cut off a frame's straight edges in order to sharpen the vertexes. "Bevel" means rounded vertexes.

PENCAP (end of segments)

With this command you can control the ends of a segment:
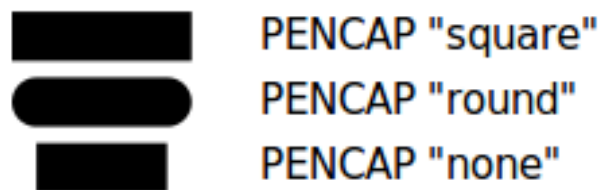
To comment this command behaviour, and also to recap code writing in Logo, let's analyze the code that produced this drawing, omitting, for the sake of simplicity, the part that produces the writing on the right. To understand more easily, let's orient ourselves with cardinal points, so orientated:
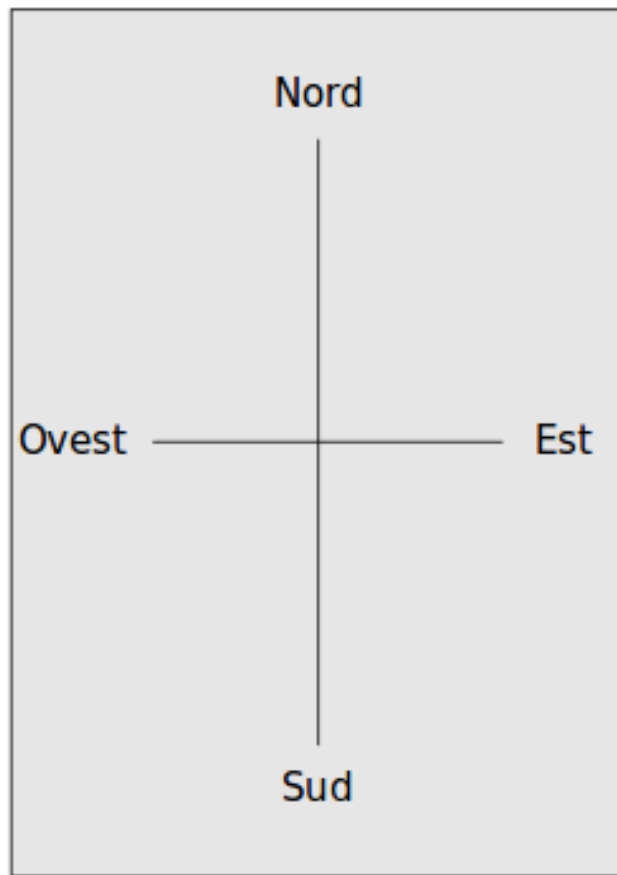
Let's see the following code:

```
CLEARSCREEN                    ; I cancel the sheet (just the graphic part)
HOME                           ; starting point: turtle in the center, pointing North
HIDETURTLE                     ; I hide the turtle
PENWIDTH 15                    ; I set the thickness of the line at 15 pt

RIGHT 90                       ; I rotate 90\degree  to the right so the turtle can
                               ; points towards East so that
                               ; to mark from left to right

PENCAP  "square "                       ; I set the mode "squares ends"
FORWARD 40                     ; I draw 40 pt of line
                               ; (Ovest -> Est)
PENUP                          ; I lift the pen
RIGHT 90 FORWARD 20            ; I turn right by 90\degree  (pointing South) and
                               ; I decrease by 20 pt
LEFT 90 BACK 40                ; I turn left (pointing East) and I get back
                               ; back  (East -> West) by 40 pt
PENDOWN                        ; I put down the pen

PENCAP  "round "               ; I set the mode "rounded ends"
FORWARD 40                     ; I draw 40 pt of line
                               ; (West -> East)
PENUP                          ; I lift the pen
RIGHT 90 FORWARD 20            ; I turn right by 90\degree  (pointing South) and
                               ; I decrease by 20 pt
LEFT 90 BACK 40                ; I turn left (pointing East), I get back
                               ; back ( East -> West) by 40 pt
PENDOWN                        ; I put down the pen

PENCAP  "none "                ; I set the mode "rounded ends"
FORWARD 40                     ; I draw 40 pt of line
                               ; (West -> East)
PENUP                          ; I lift the pen
```
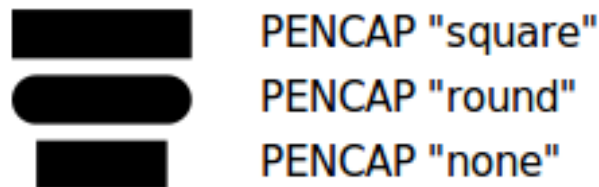
We highlighted the instructions that realize three traits, here reported:

Here, it's clear that in reality all three are drawn with the same length of 40 pt, and instead they don't seem to be the same length. This is to understand how the command PENCAP works. The line drawn without specifications for the ends ("none effect) is 40 pt long. So those with "round" and "square" effects" come out longer. It's clear that roundings are added to the normal length and that the "square" effect is obtained by straightening the roundings.

I's evident that this is just a marginal aspect. We took the opportunity to

show the extreme sophistication of LibreLogo, to get used to move in the sheet and think graphically.

PENSTYLE (dashing of segments)

With this instruction you can define the continuity of the trace, to produce dashed lines of many kinds:

PENSTYLE "solid"

PENSTYLE "dotted"

PENSTYLE "dashed"

It's also possible to adjust the trace. For example with the instruction **PENSTYLE [3, 1mm, 2, 4mm, 1mm]** we obtain the trace:

These are the rules:

- Parametro 1: number of points

- Parametro 2: length of points

- Parametro 3: number of traits

- Parametro 4: length of traits

- Parametro 5: length of spaces

- Parametro 6: optional, if it's worth 2 then the rectangles are forced to squares
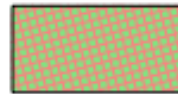
FILLSTYLE (cross-hatching)

The instruction FILLSTYLE 1, prior to the drawing of a figure, determines the trait of it. The numeric parameter determines the trait style, in the following way:
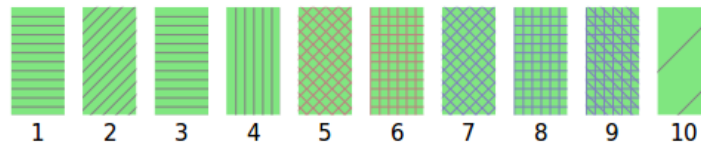
Here as well, it's possible to personalize the trait scheme, using additional parameters:

FILLSTYLE [2, "red ", 3pt, 15°]

5.1.7   Conclusions

With these instructions the part of our LibreLogo exploration dedicated to drawing comes to a conclusion. We learnt the main commands to move the turtle through the sheet, both to draw and to simply move. We learnt to move around just like a real turtle would, or the way we would do when we follow a path downtown, following a certain path, imagining a "forward" in front of our current position, a "backward", a right and a left. But we also saw how to move with a "teleportation", as if setting our car's nav sat would instantly propel us into that place, without having to go through the ends of the Earth; or how the knight moves in a chess game, jumping directly to its destination square distant 1+2 or 2+1 positions. We saw how to code colours to decorate both the lines and the surfaces. With all this bit, it seemed to us we could manage everything on the sheet, but then we suddenly discovered some instructions to draw directly the main geometrical figures: squares, rectangles, circles and ellipses, with some variables, like sectors, segments and arches of ellipses (or circles), or rectangles with rounded edges. So we introduced the first of the fundamental elements that characterize a real programming language: the concept of variable, with which we can use generic literary symbols to design specific quantities - distances, angles and more - without the concern of assign them precise numeric values; an essential characteristic that confers the language a potential

similar to what stands at the foundation of algebra, with symbolic calculus. We haven't explored all the potential that lies under the concept of variable, but just pointed out what was needed to outline the movements on the sheet based on the use of spatial coordinates. To do so we had to consider a particular type of variable that is the vector, made itself by more numbers - two if it's the case of a "position vector" on a surface. Finally we saw that we can use many other commands to determine the specific graphics of the trait (colour, width, ends, junctions) and of the surfaces (colour, dotted lines). At this point, again, we may seem to have at our disposal a powerful instrument to produce graphics to insert in documents. And actually many things can be undoubtedly done with the commands we have seen so far. In fact, from the programming language point of view, we only saw a tiny part of its potential. These constructs, at the base of all programming languages, confer extraordinary capacities of flexibility and generalization to the countless kinds of software we all know, and unwittingly use in any kind of device.

Chapter 6

Life chaos

Now we want to give the Turtle the ability to take decisions but first let's make the scenario more interesting.

So far we have depicted a sort of deterministic vision of computer programming. In our context this means giving the Turtle clear commands - do this, do that. Once the program is written the game is over. No matter the complexity, the drawing is frozen in the code. Thus, there is no place for randomness in the computer? Yes and no. A technical explanation for this answer would be too complex here. In a first approximation we can say that, no, a computer cannot produce true randomness but a sort of pseudo-randomness, thanks to appropriate mathematical tricks[1].

The Turtle understands the RANDOM command:

- X = RANDOM 100 ; gives back a random float[2] number (0 ¡= X ¡ 100), that is equal or greater than 0 and smaller than 100

- X = RANDOM "abcde" ; gives back a random letter among a, b, c, d, e

- X = RANDOM [1, 2, 3] ; gives back a random element among 1, 2 and 3 - you can also mix different items, for instance RANDOM [1, "pippo", 3.14]

In the aforementioned examples the random choices are memorized in the variable named X. Of course you can choose whatever name you prefer, or even use the instructions in some different way. For instance, try to play with the different use of the RANDOM command by means of the following code:

---

[1]Basically, in order to produce randomness one has to be able to generate random numbers, by means of so called random number generators. In reality, they generate periodic sequences, in the sense that after a certain number of numbers the same initial sequence is started again. The trick consists in using algorithms that produce extremely large periods so that one never reaches the end of the sequence by means of successive number extractions.

[2]In computer science a float number is a number with decimal digits. For instance, 3.14 is a float number, 18 is an integer number.

```
HOME
CLEARSCREEN                                        51.54394381549091
                                                   89.24543102328022
PENUP                                              70.94070676116884
FORWARD 300                                        47.784948686323844
                                                   24.377983435289906
                                                   64.78090220434123
REPEAT 10 [                                        30.994487344375155
    LABEL RANDOM 100                               70.74044100746335
    BACK 12                                        22.47845983092507
]                                                  47.425130347678 43
```

With this code the Turtle first goes next to the page top, then writes a column of ten successive random numbers between 0 and 100, printing them by means of the LABEL instruction. Of course you can change whatever you like in this piece of code - it's always good to experiment.

Now, let's take a basic piece of code we are using several times, with variants:

```
TO QUADRATO
    REPEAT 4 [
        FORWARD 100
        RIGHT 90
    ]
END

QUADRATO
```

This is the code to draw a square, as we have seen at the beginning of chapter 5. In chapter **??** we investigated how to transform this code to get any kind of regular polygons. In a somewhat more complex variant, in chapter **??**) we will explore more general cycles properties and in chapter **??** we will discover the natural way of drawing a circle in the Turtle Geometry perspective, put in a practical didactic perspective: here just the code, to point out the similarity with the previous one.

```
TO CERCHIO
    REPEAT 360 [
        FORWARD 1
        RIGHT 1
    ]
END

CERCHIO
```

Now, we have all the ingredients to give life to our Turtle. Let's inject a dose of randomness in the previous code. Within the repeating cycle, first we are going to tell the Turtle to move forward by a random amount between 1 and 2. In order to achieve this effect we write the instruction FORWARD RANDOM(1) + 1. Here we are telling the Turtle to move forward by a quantity equal to RANDOM(1) +1. The command RANDOM(1)[3] renders a random number

---

[3]At the beginning we showed a slightly different syntax: RANDOM 1 instead RAN-DOM(1). The first one is the standard LibreLogo version reported in the official LibreLogo Toolbar manual. The second one is an alternative version which is based on the underlying

between 0 and 1, thus, if we sum 1 to this, we obtain a random number which is comprised between 1 and 2. Second, we tell the Turtle to turn by a random amount, say between -45 and 45 degrees. We can achieve this result simlarly, by means of the instruction LEFT RANDOM(90) - 45, since with RANDOM(90) - 45 we get a random number between -45 and 45 degrees.

Here we have the whole piece of code and its result:

```
TO RANDOMMOVE
    REPEAT [
        FORWARD RANDOM(1) + 1
        RIGHT RANDOM(90) - 45
    ]
END

RANDOMMOVE
```

If these code modifications sound weird to you, just play with it. That is copy the code in a new LibreOffice document, try to run it, then try to modify the various numbers and reflect upon what's happening.

But be aware: if you are going to run repeatedly the same piece of code you will not be able to reproduce this drawing, or better: the probability of reproducing the same drawing is extremely low. So what? How can we be happy of producing unpredictable results by means of a machine conceived for making complex and accurate calculations? Well, actually, we are very glad because with this simple code we open a windows on the huge and crucial world of scientific simulation. Nowadays, in every scientific field the simulation is an essential investigation tool, the unique possible to face the overwhelming complexity of natural phenomena. However, nowadays does not mean right today. My graduation thesis in physics, in the seventies, was heavily based on the use of computer simulation of radiation scattering within the human body. A too complex context to be faced with the tools of mathematical analysis. In physics computer simulations make use of so called Monte Carlo Methods, a broad class of computational algorithms that rely on repeated random sampling to obtain numerical results. For instance, in my thesis work, I simulated a source of radioactivity within the body, by reproducing the emission of photons, thus invoking a random number generator for any random step, such as direction of emission, interaction with matter, direction of deflection and so on.

But simulations are not restricted to the domain of physics, on the contrary, the more complex the phenomena the more simulations may turn out to be the unique possible investigation tool. In the context of biological disciplines we talk about *in silico* experiments, referring to the fact that they are realized through calculations done with digital computers that run on silicon chips - the expression is an allusion to the latin phrases *in vivo*, *in vitro* and *in situ*,

---

python code by means of which LibreLogo has been written. In the example we are discussing this version is preferable because it gives more clarity. Actually, with the expression RANDOM 1 + 1 it is not clear if we are willing to sum 1 to the result of RANDOM 1 or if we want to call RANDOM 2. By using RANDOM(1) + 1 there is not such an ambiguity.

commonly used in biology. Nowadays *in silico* experiments include molecular biology, genetic assays, tumour growth, dermatology, bone remodelling, organ failure, clinical trials, just to mention some. In medicine, for instance, *in silico* studies are used to discover new drugs because it is faster and costs much less. In biology they are used to study to formulate behaviour model of cells and in genetics to analyze gene expression[4].

But what are we simulating with our piece of code? The obvious answer is a turtle wandering randomly. Of course, if we imagine to observe an animal from the height in its environment, the impression will be that of a random movement. In this sense our code could be considered a simulation of the animal behaviour, even if a very poor one, since our randomness assumption is based on the fact that we ignore what the animal is actually doing. Animals move because they are looking for food or a mate, for instance. Or, perhaps, the simulation could be more appropriate for a drunk hanging around in the night, looking for a nonexistent happiness. The distribution of food for a given specie of animals in its natural environment could be modeled, in some extent, but the inner struggle of a man is much more difficult to translate in mathematical terms.

But what does it mean exactly by "modeling something", such as food distribution, in the context of our example? Yes, apparently, we injected a drop of life in our code, reproducing a randomness which we all experience in our life, in the shape of good or bad luck. However, despite such strong perception of uncertainty, we all know that our capability of observing the outside world and of tacking decisions, is crucial, as well. In substance, our turtle is totally passive! It just have some energy to proceed and to turn: no smell, no sight, no decisions. This is the point: we need a way to let the Turtle have the ability to take decisions!

Before going on learning how to let the Turtle take decisions, we diverge a bit from the LibreLogo environment to have a fast look to the fascinating world of *multiple turtles*. We have to give up to LibreLogo for a while because it does not allow to run more turtles simultaneously, at least in the current version. It is plenty of Logo versions out there, and some of them allow to run more then one turtle at a time. As we have already said, the reason for using LibreLogo as the first but not trivial introduction to Logo relies in its extremely *low floor*, which is crucial in primary school.

---

[4]In molecular biology "gene expression" means the way a set of genes determines the functioning of the cell at the macromolecular level

Chapter 7

Taking decisions

Now we want to give the Turtle the ability to take decisions but first let's make the scenario more interesting.

So far we have depicted a sort of deterministic vision of computer programming. In our context this means giving the Turtle clear commands - do this, do that. Once the program is written the game is over. No matter the complexity, the drawing is frozen in the code. Thus, there is no place for randomness in the computer? Yes and no. A technical explanation for this answer would be too complex here. In a first approximation we can say that, no, a computer cannot produce true randomness but a sort of pseudo-randomness, thanks to appropriate mathematical tricks[1].

The Turtle understands the RANDOM command:

- X = RANDOM 100 ; gives back a random float[2] number (0 ¡= X ¡ 100), that is equal or greater than 0 and smaller than 100

- X = RANDOM "abcde" ; gives back a random letter among a, b, c, d, e

- X = RANDOM [1, 2, 3] ; gives back a random element among 1, 2 and 3 - you can also mix different items, for instance RANDOM [1, "pippo", 3.14]

In the aforementioned examples the random choices are memorized in the variable named X. Of course you can choose whatever name you prefer, or even use the instructions in some different way. For instance, try to play with the different use of the RANDOM command by means of the following code:

---

[1]Basically, in order to produce randomness one has to be able to generate random numbers, by means of so called random number generators. In reality, they generate periodic sequences, in the sense that after a certain number of numbers the same initial sequence is started again. The trick consists in using algorithms that produce extremely large periods so that one never reaches the end of the sequence by means of successive number extractions.

[2]In computer science a float number is a number with decimal digits. For instance, 3.14 is a float number, 18 is an integer number.

```
HOME
CLEARSCREEN                                    51.54394381549091
                                               89.24543102328022
PENUP                                          70.94070676116884
FORWARD 300                                    47.784948686323844
                                               24.377983435289906
                                               64.78090220434123
REPEAT 10 [                                    30.994487344375155
   LABEL RANDOM 100                            70.74044100746335
   BACK 12                                     22.47845983092507
]                                              47.425130347767843
```

With this code the Turtle first goes next to the page top, then writes a column of ten successive random numbers between 0 and 100, printing them by means of the LABEL instruction. Of course you can change whatever you like in this piece of code - it's always good to experiment.

Now, let's take a basic piece of code we are using several times, with variants:

```
TO QUADRATO
   REPEAT 4 [
      FORWARD 100
      RIGHT 90
   ]
END

QUADRATO
```

This is the code to draw a square, as we have seen at the beginning of chapter 5. In chapter **??** we investigated how to transform this code to get any kind of regular polygons. In a somewhat more complex variant, in chapter **??**) we will explore more general cycles properties and in chapter **??** we will discover the natural way of drawing a circle in the Turtle Geometry perspective, put in a practical didactic perspective: here just the code, to point out the similarity with the previous one.

```
TO CERCHIO
   REPEAT 360 [
      FORWARD 1
      RIGHT 1
   ]
END

CERCHIO
```

Now, we have all the ingredients to give life to our Turtle. Let's inject a dose of randomness in the previous code. Within the repeating cycle, first we are going to tell the Turtle to move forward by a random amount between 1 and 2. In order to achieve this effect we write the instruction FORWARD RANDOM(1) + 1. Here we are telling the Turtle to move forward by a quantity equal to RANDOM(1) +1. The command RANDOM(1)[3] renders a random number

---

[3]At the beginning we showed a slightly different syntax: RANDOM 1 instead RAN-DOM(1). The first one is the standard LibreLogo version reported in the official LibreLogo Toolbar manual. The second one is an alternative version which is based on the underlying

between 0 and 1, thus, if we sum 1 to this, we obtain a random number which is comprised between 1 and 2. Second, we tell the Turtle to turn by a random amount, say between -45 and 45 degrees. We can achieve this result simlarly, by means of the instruction LEFT RANDOM(90) - 45, since with RANDOM(90) - 45 we get a random number between -45 and 45 degrees.

Here we have the whole piece of code and its result:

```
TO RANDOMMOVE
    REPEAT [
        FORWARD RANDOM(1) + 1
        RIGHT RANDOM(90) - 45
    ]
END

RANDOMMOVE
```

If these code modifications sound weird to you, just play with it. That is copy the code in a new LibreOffice document, try to run it, then try to modify the various numbers and reflect upon what's happening.

But be aware: if you are going to run repeatedly the same piece of code you will not be able to reproduce this drawing, or better: the probability of reproducing the same drawing is extremely low. So what? How can we be happy of producing unpredictable results by means of a machine conceived for making complex and accurate calculations? Well, actually, we are very glad because with this simple code we open a windows on the huge and crucial world of scientific simulation. Nowadays, in every scientific field the simulation is an essential investigation tool, the unique possible to face the overwhelming complexity of natural phenomena. However, nowadays does not mean right today. My graduation thesis in physics, in the seventies, was heavily based on the use of computer simulation of radiation scattering within the human body. A too complex context to be faced with the tools of mathematical analysis. In physics computer simulations make use of so called Monte Carlo Methods, a broad class of computational algorithms that rely on repeated random sampling to obtain numerical results. For instance, in my thesis work, I simulated a source of radioactivity within the body, by reproducing the emission of photons, thus invoking a random number generator for any random step, such as direction of emission, interaction with matter, direction of deflection and so on.

But simulations are not restricted to the domain of physics, on the contrary, the more complex the phenomena the more simulations may turn out to be the unique possible investigation tool. In the context of biological disciplines we talk about *in silico* experiments, referring to the fact that they are realized through calculations done with digital computers that run on silicon chips - the expression is an allusion to the latin phrases *in vivo*, *in vitro* and *in situ*,

---

python code by means of which LibreLogo has been written. In the example we are discussing this version is preferable because it gives more clarity. Actually, with the expression RANDOM 1 + 1 it is not clear if we are willing to sum 1 to the result of RANDOM 1 or if we want to call RANDOM 2. By using RANDOM(1) + 1 there is not such an ambiguity.

commonly used in biology. Nowadays *in silico* experiments include molecular biology, genetic assays, tumour growth, dermatology, bone remodelling, organ failure, clinical trials, just to mention some. In medicine, for instance, *in silico* studies are used to discover new drugs because it is faster and costs much less. In biology they are used to study to formulate behaviour model of cells and in genetics to analyze gene expression[4].

But what are we simulating with our piece of code? The obvious answer is a turtle wandering randomly. Of course, if we imagine to observe an animal from the height in its environment, the impression will be that of a random movement. In this sense our code could be considered a simulation of the animal behaviour, even if a very poor one, since our randomness assumption is based on the fact that we ignore what the animal is actually doing. Animals move because they are looking for food or a mate, for instance. Or, perhaps, the simulation could be more appropriate for a drunk hanging around in the night, looking for a nonexistent happiness. The distribution of food for a given specie of animals in its natural environment could be modeled, in some extent, but the inner struggle of a man is much more difficult to translate in mathematical terms.

But what does it mean exactly by "modeling something", such as food distribution, in the context of our example? Yes, apparently, we injected a drop of life in our code, reproducing a randomness which we all experience in our life, in the shape of good or bad luck. However, despite such strong perception of uncertainty, we all know that our capability of observing the outside world and of tacking decisions, is crucial, as well. In substance, our turtle is totally passive! It just have some energy to proceed and to turn: no smell, no sight, no decisions. This is the point: we need a way to let the Turtle have the ability to take decisions!

Before going on learning how to let the Turtle take decisions, we diverge a bit from the LibreLogo environment to have a fast look to the fascinating world of *multiple turtles*. We have to give up to LibreLogo for a while because it does not allow to run more turtles simultaneously, at least in the current version. It is plenty of Logo versions out there, and some of them allow to run more then one turtle at a time. As we have already said, the reason for using LibreLogo as the first, even if not trivial, introduction to Logo relies in its extremely *low floor*, which is crucial for using it in primary school.

---

[4]In molecular biology "gene expression" means the way a set of genes determines the functioning of the cell at the macromolecular level

```
clear()
invisible()

def path(t: Turtle, delay: Int) = runInBackground {
    t.setAnimationDelay(delay) {
    repeat(10000) {
        t.forward(random(1) + 1)
        t.right(random(90) - 45)
    }
}
}
val t1 = newTurtle(0, 0)
t1.setPenColor(red)
val t2 = newTurtle(0, 0)
t2.setPenColor(orange)
val t3 = newTurtle(0, 0)
t3.setPenColor(yellow)
val t4 = newTurtle(0, 0)
t4.setPenColor(green)
val t5 = newTurtle(0, 0)
t5.setPenColor(blue)

path(t1, 50)
path(t2, 50)
path(t3, 50)
path(t4, 50)
path(t5, 50)
```
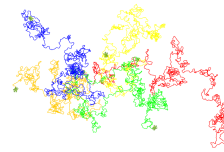
In this piece of Kojo Logo code there are instructions that do not belong to the standard set of Logo commands, and precisely those that allow for controlling different turtle simultaneously. However, the standard Logo commands are pretty similar between the two systems. In the appendix in chapter 8 there is a little vocabulary.

Chapter 8

Appendix 2

LibreLogo and Kojo instructions are quite similar. Logo is not case-sensitive, you can write commands lower or uppercase, indifferently. I usually show examples written in uppercase because it seems to be easier for beginners. Instead, Kojo is case-sensitive and commands are written in "camel-case" style, for instance penUp().

Here a short instructions comparison list between LibreLogo and Kojo commands is given.

Action	LibreLogo	Kojo
Move forward by 100	FORWARD 100	forward(100)
Move back by 100	BACK 100	back(100)
Turn left by 90°	LEFT 90	left(90)
Turn right by 90°	RIGHT 100	right(90)
Raise the pen	PENUP	penUp()
Lowers the pen	PENDOWN 100	penDown()
Moves to new position of coordinates (100,200) drawing a line	POSITION [100,200]	changePosition(100, 200)
Clears the screen and goes home	CLEARSCREEN HOME	clear()
Hides the Turtle	HIDETURTLE	invisible()
Shows the TURTLE	SHOWTURTLE	visible()

At the following link you can find an ebook with a list of the Kojo turtle commands: http://www.kogics.net/kojo-ebooks#quickref.

Bibliography

[1] Emma Castelnuovo. *L'officina matematica – ragionare con i materiali*. Edizioni La Meridiana, Molfetta, 2 edition, 2008.

[2] 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`, 2015. Accessed: 2017-08-13.

[3] 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.

[4] 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`, 2015. Accessed: 2017-08-13.

[5] 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`, 2015. Accessed: 2017-08-13.

[6] László Németh. Esempi di uso librelogo (ungherese). `http://www.numbertext.org/logo/logofuzet.pdf`. Accessed: 2017-08-13.

[7] Seymour Papert. *Mindstorms, Children, Computers, and Powerful Ideas*. Basic books, New York, 2 edition, 1993.

[8] Lakó Viktória. Manuale di comandi di librelogo. `http://szabadszoftver.kormany.hu/wp-content/uploads/librelogo_oktatasi_segedanyag_v4.pdf`. Accessed: 2017-08-13.