10. Schweizer Tag für den Informatik unterricht

Kreativität entfalten mit Informatik

Workshop 5
Exploring the land of powerful mathematical ideas with Logo's Turtle

Andreas R. Formiconi

5 Februar 2019

Contents

1 Foreword

This document is intended to support the "Exploring the land of powerful mathematical ideas with Logo's Turtle", workshop N. 5, to be held at the "10. Schweizer Tag für den Informatik unterricht", on 5th February 2020. The text follows the succession of topics which will be proposed in the workshop. By means of this text, participants will be able to deepen these topics at their will — I will be available for discussion and help by email at arf@unifi.it.

For all the topics mentioned here, workable examples are provided at $http://iamarf.ch/Codice-LOGO/$.
Just download the ZIP file, expand it and browse through the available folders.

2 Introduction

These notes are about the use of Turtle Geometry at the primary (chapters 4-6) and secondary school (chapter 6-9), in the framework of Bruner's concept of *spiral curriculum*[1] [Bruner, 1960]. It is addressed to teachers, trying to show how to initiate kids to Turtle commands for exploring math and other sciences. This text is largely inspired to the work and vision of Seymour Papert, a mathematician, computer scientist and pedagogist who got his PhD with Jean Piaget. Nowadays, Papert is known mainly for being the inventor of the educational programming language Logo. However, the Logo language is only a tool that Papert conceived to foster a different way to teach sciences and to foster self-reflection: learning while reflecting on its own learning. We live in times of very rapid change, too rapid. Papert's profound thought has been largely forgotten today, except in a limited number of places, such as the ABZ group. Many of current "coding" practices seems to have much to do with a sort of instructive game, but the connection between such practices and sound scientific concepts seems to be largely absent. This text represents an effort to fill this gap. In particular, by showing a series of exercises, spanning from the primary school level to the last years of secondary school, we try to recover and discuss Papert's concepts of *syntonic learning* and *powerful ideas*. Papert was used to say that the *powerful idea* concept is so important that the most powerful idea is that of *powerful idea* itself! Throughout the article you will find gray text boxes intended to comment on powerful mathematical ideas, concepts or approaches that are evoked by the exercises. In appendix 1 (pag. 86) an index of gray boxes is provided.

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"[2]. 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 some never get rid of 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, nine 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 but thoughtful aspect of the situation. On

[1] A spiral curriculum is one in which there is an iterative revisiting of topics throughout the course. It's not about their mere repetition. Instead, it requires the deepening of it , with each successive encounter building on the previous one. This concept dates to Jerome Bruner:

> I was struck by the fact that successful efforts to teach highly structured bodies of knowledge like mathematics, physical sciences, and even the field of history often took the form of a metamorphic spiral in which at some simple level a set of ideas or operations were introduced in a rather intuitive way and, once mastered in that spirit, were then revisited and reconstrued in a more formal or operational way , then being connected with other knowledge, the mastery at this stage then being carried one step higher to a new level of formal or operational rigour and to a broader level of abstraction and comprehensiveness. The end state of this process was eventual mastery of the connexity and structure of a large body of knowledge...

[2] The Turtle will be the protagonist of our activities. We'll clarify it in the following.

the other side, 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 the older digital natives. That is, people having thousands of Facebook contacts, candidly claim of not getting along with technologies!

Having stressed that, please, do face the Turtle with childish curiosity, starting from scratch. Since I suppose you know how kids tackle something new, try doing just the same. In the following, (chapters 4-6) we are going to describe a first Logo exploration identifying ourselves with our kids. What is written is what we could say to our kids, or students. However, here and there, we will insert text boxes dedicated to reflections that we do as teachers, mainly emphasizing important implicit aspects carried by the Logo activities. For example, powerful mathematical ideas, as Seymour Papert [Papert, 1993] called them, i.e. fragments of scientific thought that one day may help build more formal mental devices. Of course these concepts have not to be mentioned explicitly with kids but it is important that teacher are aware of them.

In chapter 3 the software tools which can be used in the workshop or in subsequent work, together with appropriate links, are mentioned. The concept of free software is presented with a specific focus on educational use. Finally, some instructions on activating LibreLogo are given. An introduction to Papert's *syntonic learning* is given at the beginning of chapter 4; then, a careful step by step guide to drawing the first simple geometrical figures is given. In the final part the reader is introduced to the magic of creating its own new commands. Chapter 5 deals with the concept of variable and how it can be used in new commands. *Syntonic learning* and an interesting example of a powerful mathematical idea hidden behind a simple activity — drawing a circle — are at the heart of Chapter 6. In the second part of this chapter the reader is invited to continue playing with "round things" by perturbing the circle, thus joining Bernoulli's wonder for the "spira mirabilis". In the same chapter the concept of circularity is explored further by means of polygonal approximations. If these last considerations are more oriented to the secondary level, even more are those of chapter 7, where the reader is shown how it is possible to analyse some basic physics problems in a computational way. Even if all the problems reported here can be coded with Logo[3], in this chapter we have focused on the use of Python, which is more appropriate for this type of exercises, being also perfectly adequate for the secondary level. The last example about the simulation of the orbit of Halley's comet goes somewhat beyond the scope of this text but it serves to show how these educational programming languages have *low floor and high ceiling*, i.e. they are quite easy for beginners but, at the same time, they allow one to fly very high! In chapter 8 we discover randomness in the digital context, even if at a very basic level, by letting the Turtle playing the turtle. Finally, through the curious construct of *recursion*, we have a look at the fascinating world of fractal forms.

[3]Both the Logo and the Python version can be found among the downloadable materials. The Logo examples are given for the LibreLogo environment but they can be reproduced in XLogo easily. The Python programs have been tried in TigerYjthon.

3 The tools we are going to use

We are going to use mainly the LibreLogo programming language, which is a version of Seymour Papert's Logo implemented as a plugin of the Writer word processor available in the LibreOffice suite. LibreLogo is available by default in Writer since version 4.0 of LibreOffice. It has been written in Python by László Németh[4] [Németh, 2014].

Alternatively, I will give also examples of using the programming resources provided by the Ausbildungs- und BeratungsZentrum für Informatikunterricht ABZ of ETH, directed by Prof. Juraj Hromkovič. Both the educational materials as well as the programming tools are excellent. The XLogo [Hromkovič, 2014] comes both as a web service (XLogoOnline) and as a downloadable software for all three usual operating systems: Windows, OS X and Linux[5]. XLogo and LibreLogo commands are very similar. We will comment on any differences in the examples presented in this text. In chapter 7 we will also give examples of using the TigerJython Python environment provided by the ABZ[6] both as software or web service. For the exercises proposed in this chapter Python is more appropriate.

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 co-director with Marvin Minsky, a prominent 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.

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.

Figure 1: Painting owned by ABZ at ETH — Author: Ingrid Zamecnikova

[4]The documentation can be found at *http* : *//librelogo.org*. From there a description of all the LibreLogo commands can be downloaded in 33 different languages at *https* : *//help.libreoffice.org/Writer/LibreLogo_Toolbar*

[5]Link available in
http : *//www.abz.inf.ethz.ch/primarschulen − stufe − sek − 1/unterrichtsmaterialien/*
In the same web page a short manual is available in various languages.

[6]Link available in
https : *//webtigerjython.ethz.ch/*
In the same page there are links to an online tutorial, documentation and examples.

Since 1964 he was a researcher at the MIT Artificial Intelligence laboratory where, in 1967 he got the position of co-director with Marvin Minsky, a prominent 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.

3.1 Free software

Free software is a beautiful reality. Actually it is incredibly widespread but, at the same time, so few people know something about. Few people know that fragments of free software are also incorporated into industrial consumer products, such as telephones and even washing machines! But here we are interested in certain types of free software that can be used in school contexts, because of their relevant ethical and educational characteristics. Probably, the best known free software is the Linux operating system, but there are many more: LibreOffice, analogous of MS Office, Gimp for manipulating digital bitmap images, Inkscape for vector images, Audacity for audio recorded files, OBS for streaming and screencasting, Shotcut for video cutting and many others.

It is important here to understand what makes free software different. Basically, free software is characterized by four essential freedoms:

- The freedom **to run** the program as you wish, for any purpose
- The freedom **to study** how the program works, and change it so it does your computing as you wish
- The freedom **to redistribute copies**, so you can help your neighbor
- The freedom **to distribute copies of your modified versions** to distribute copies of your modified versions to others. By doing this you can give the whole community a chance to benefit from your changes

I emphasized two statements in red — "so you can help your neighbor" and "By doing this you can give the whole community a chance to benefit from your changes" — because they express ethical concepts instead of mere technical facts. This is exactly what makes free software different from proprietary software[7] and open source software[8].

Free software is good in educational contexts because you are implicitly teaching:

- the value of collaboration
- that to do something for other people is good
- that to crack proprietary software is bad because it is against the law; better using free software so that you are helping the developers communities as well: more users → better software

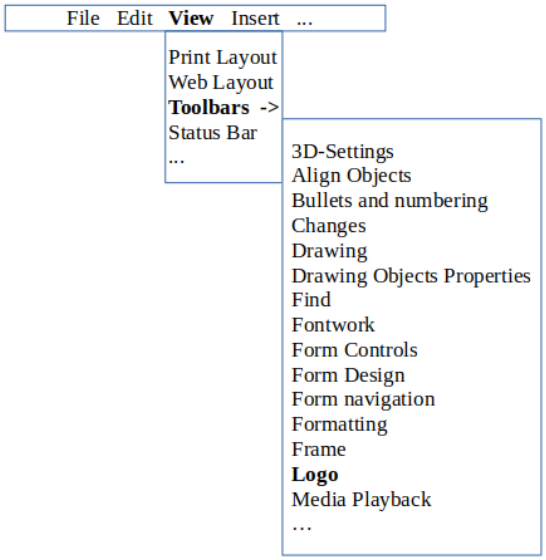
[7] Proprietary software is the software you may download and install in your computer but you have to pay for the license to use it with full options. What you download is the so called executable software, not the relative *source code*, i.e. you can run the software but you cannot read the its instructions code to know how the functionalities are realized.

[8] With open source software the code is available, like with free software, but there are no other specifications, not to mention ethical statements.

- that free software is something for everybody and not only for those that con afford it; that's important for public schools
- that you can save money for the public benefit in a perfectly legal way

3.2 Activating LibreLogo

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:

We are going to show down here the meaning of these icons, however remember that only the "play" (green arrow) and "stop" (red square) buttons will be used in all our work.

	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.

4 Do not ”teach”: be maieutic!

4.1 Before talking to the Turtle...

Reflection — Syntonic learning

Always keep in mind that our brain evolved to optimize a multisensorial, olistic kind of learning. We evolved to learn through our bodies. Not as Sir Ken Robinson used to joke about: — Professors use their bodies mainly to bring their heads to congresses! —

And this sort of holistic learning is particular important for kids. Thus, various kind of “unplugged activities”, involving physical games or practices are appropriate before facing the cold world of touchscreens or other electronic devices. Seymour Papert talked about *syntonic learning*:

> This term [syntonic learning] is borrowed from clinical psychology and can be contrasted to the dissociated learning already discussed. Sometimes the term is used with qualifiers that refer to kinds of syntonicity. For example, the Turtle circle is body syntonic in that the circle is firmly related to children's sense and knowledge about their own bodies.

The following pictures show some examples of possible practices.

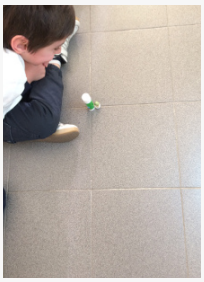


Figure 2: If I would be the puppet...

Figure 3: Pay attention to the path...

Figure 4: One small step, turn a little bit, again and again...

Figure 5: "Drawing in the flour"...

These pictures were shot by maestra Antonella Colombo, a former student of mine, now teaching in the primary school of Paderno d'Adda. She made great work in her classroom, based on our discussions of the concept of syntonic learning.

Very powerful mathematical idea — Isomorphism

We go back here to Seymour Papert for this insight: syntonicity evokes the concept of *isomorphism*. It is a very general and fundamental concept that appears in several areas of mathematics. In simple words, an isomorphism is a correspondence between two different worlds that preserves sets and relations among elements. The fact that one sees the correspondence between a square "drawn with one's own body" — for instance by walking along some square figure visible on the floor — and the square drawn by the Turtle on the computer screen, is a significant step towards abstraction, a step towards the concept of *isomorphism*.

4.2 First steps

Provided you downloaded LibreoOffice and activated the Logo toolbar, let students open Writer LibreOffice. They find themselves in front of a white sheet: you could write a letter, a shopping list, a poem or whatever you like. Of course, one has to say something, especially at the beginning. But refrain to give instructions, as much as possible — Today we are going to make something new... let's open this program (LibreOffice) — Then, everyone is faced with a blank page with pretty familiar text formatting commands at the top. Give them time (always give time...) to realize where they have landed. Do not explain that this is a word processor, so and so. Just suggest to try something, for instance typing something and let emerge and realize that this is is nothing else that one

of those programs for writing text, possibly assignments or similar stuff — So what? What are we here for? — Try creating a sense of mystery. At a certain point, you may propose to type a specific word: for instance[9]: FORWARD. Discuss the meaning of this word in English, recalling the translation into their native language. Then, give this small instruction — Go over there, to the left, and press the green arrow... — Two things will happen: a green object appears, could it be an animal, with head and legs? A sort of turtle? Yes, a turtle actually, the protagonist of our story and we will call it "the Turtle" in the following. But also a small window appears, telling that there is an error. It is good to stumble upon an error immediately. It's an expedient to begin creating a feeling of confidence with the error: the error as an opportunity of getting precious information. — Well — you may admit — I forgot something! — and tell them to add a number, FORWARD 100, and press the green play command again...

```
FORWARD 100
```

Listing 1: The first move...

There is a drawing: the Turtle at the top of a vertical segment. When working with kids, it is good to present the Turtle as a friend, who may help them to do something new. Thus, the Turtle is a kind of living being, not so much because it resembles a turtle but because it's able to talk, even if in a peculiar way. We write FORWARD 100 and the Turtle draws a segment, i.e. we write a command and the Turtle answers by doing something related to the command — a dialogue is going on. What one should do is to let kids have the feeling of being playing with the Turtle, following their innate attitude to set one's own goals and to explore what is amazing them. Of course this needs time but true learning deserves time - no tricks are available in this respect. And, of course again, to achieve this simple drawing you have to tell kids the command to write but you do not need to say something like - "Now I tell you how to draw a segment". Just tell the the command and wait...

Probably someone will propose something: —- let's change the number...

[9] I used uppercase characters just for clarity, LibreLogo is "case insensitive". Also XLogo is case insensitive but it's better to use lowercase because in this way the environment provides highlighted syntax. Instead, in TigerYjthon lowercase must be used because the environment is case sensitive.

what else can we do... — Wait for ideas or questions like these ones, always keep waiting till explorations are going on. Then add something new, RIGHT 90 for instance. However, at this point, you may introduce two handy commands for deleting previous drawings and sending the Turtle home at any new try:

```
CLEARSCREEN
HOME
FORWARD 100
RIGHT 90
```

Listing 2: The turn command

We got the same drawing but the Turtle turned right by an angle of 90°. However, do not explain what 90 means, let them find out for themselves.

Powerful idea — Breaking the problem down into smaller parts
This is one of the commonly quoted elements of the so-called "computational thinking". There is a lively debate about the exact definition but, to tell the truth, several of its aspects have always been sound habits of a good scientist. This is the case of "breaking the problem down into smaller parts". I'm quoting it right here, because of how the issue may emerge when teachers are "doing coding" with devices such as the Bee-Bot or similar ones. Sometimes, when planning the coded path with text or cards, they interpret turning commands as steps, thus confusing two different acts: turning should not involve movement, as well as stepping should not involve turning. This will be clearer when we will comment on the "state" powerful idea, later on. Here it's worthwhile to point out the importance to break down a problem or a process in smaller clearly defined sub-parts.

The reason for I'm insisting so much about what children can do is that I had several opportunities to play Turtle with them. Really a few say it's too difficult, most just play and get excited to let the small animal create funny things. Too often we forget how kids crave hard. Once, trying to draw a house a girl came out with kind of a bizarre castle. I praised her, commenting on how marvellous was this drawing, she was so proud. Then, when I came back to her, after having considered the works of other kids, I found her in tears, sobbing desperately: she had accidentally erased the commands in a way that I was no

more able to recover. I felt guilty for not having taught her how to save the work once in a while.

By the way, 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.

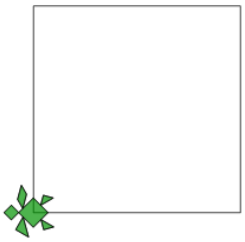
4.3 Towards first geometrical shapes

Most probably someone will come out saying: — Let's do a square! — or — Let's do a triangle... — or something else. Just go and follow their proposals, even if here we are going to show the example of a square. If nobody comes out with an idea, you may help: — Why don't we try to make a ...?

Let's go on with the square. Often kids achieve this easily, landing to something like:

```
CLEARSCREEN
HOME
FORWARD 100
RIGHT 90
FORWARD 100
RIGHT 90
FORWARD 100
RIGHT 90
FORWARD 100
```

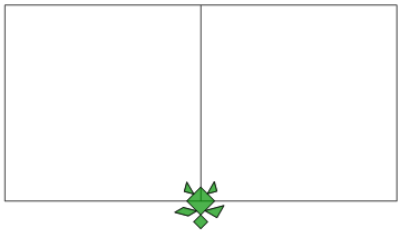
Listing 3: The first figure



The first of your kids who will achieve this result will rejoice, of course, but is the task really finished? At first glance yes, since the square is there, however, the point is more intriguing than that. In order to work properly, that is, in order to use the principle of dividing a job in smaller parts, the Turtle should end always in its starting position, after having drawn a geometrical figure. What we mean is that if we are invoking two times the drawing of a square, the Turtle should draw the same square, superimposing the second to the first one. Instead, in this case, see what happens if we try to repeat two times the same code:

```
CLEARSCREEN
HOME
FORWARD 100
RIGHT 90
FORWARD 100
RIGHT 90
FORWARD 100
RIGHT 90
FORWARD 100

FORWARD 100
RIGHT 90
FORWARD 100
RIGHT 90
FORWARD 100
RIGHT 90
FORWARD 100
```

Listing 4: What does it really mean that a square is "finished"?



This is quite different from the expected results. The problem is that our code for drawing a square is difficult to use repeatedly, since we have to reflect on the landing position of the Turtle to foresee where the next square will be drawn. These considerations drive us towards the definition of the important concept of *state* of a system.

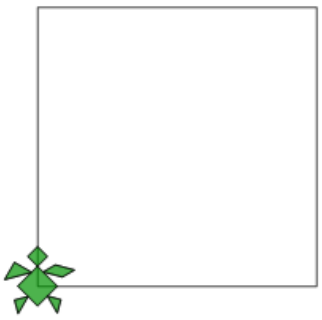
Powerful idea — State of a system

It's very natural to think about the "state of the Turtle" since by knowing its position we do not know everything: we also need to know where the Turtle is looking. That is, in other to know everything about the Turtle we need to know both its position and direction. Therefore, when drawing a square, instead of just talking about its initial position, we should specify position and direction, which makes three numbers: two spatial coordinates in the plane and a direction angle. Thus, the Turtle allows us to employ, in a practical way, the concept of "state" of a system, which is a crucial concept in all basic sciences. For instance, in classical mechanics, the state of system is given by the knowledge of spatial coordinates and speed components of all its particles. In the world of Turtle Geometry (we'll talk about later on) the Turtle state is given by its two spatial coordinates and the direction angle it is pointing to.

Returning to our square, it is easy to see that what is missing is a last turn right — I added numbers at the beginning of lines, in order to refer them more easily:

```
1  CLEARSCREEN
2  HOME
3  FORWARD 100
4  RIGHT 90
5  FORWARD 100
6  RIGHT 90
7  FORWARD 100
8  RIGHT 90
9  FORWARD 100
10 RIGHT 90
11
12 FORWARD 100
13 RIGHT 90
14 FORWARD 100
15 RIGHT 90
16 FORWARD 100
17 RIGHT 90
18 FORWARD 100
19 RIGHT 90
```

Listing 5: We added RIGHT 90 instructions at lines 10 and 19



Now, if we we repeat this piece of code a second time, the Turtle will draw a second square, perfectly superimposed to the first one. Later on we will see the advantage of such behaviour in an important case.

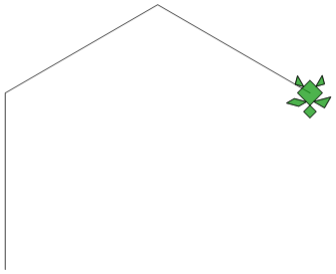
4.3.1 And why not a triangle?

This is a typical goal, often finalized to build the roof of a house, on top of the previous square. Usually both kids and grownups are pretty confident to achieve it after having been successful with the square experience. The idea is to work on the square code by changing the number of sides — pretty easy — and, of course, the angle, to obtain an equilateral triangle. In the majority of cases, without distinction of age, people decide to substitute 90°with 60°, because the internal angles of the equilateral triangle are equal to 60 degree. So, what's the result?

Try to think about this before moving on to the next page. Maybe you could use this page to make a pencil drawing...

```
CLEARSCREEN
HOME
FORWARD 100
RIGHT 60
FORWARD 100
RIGHT 60
FORWARD 100
RIGHT 60
```

Listing 6: We wanted to draw a triangle but...



Quite disappointing, isn't it?

Reflection — Scientific knowledge process

The process of building scientific knowledge is iterative: first imagine a theoretical description, then try to devise an experimental confirmation, finally go back to refine the theory if necessary. The triangle example is an example of how naturally the process is activated with Logo. In this case we know the theory, which says that the internal angles of an equilateral triangle are equal to 60°. This is correct of course. What's wrong is how we applied this knowledge: the angle we specify in the RIGHT instruction is the *deviation angle* from the current direction of the Turtle, not the internal angle of the figure we are trying to build. The deviation angle is supplementary of the internal angle we want to produce, that is $180 - 60 = 120°$. Once they see the disappointing result, people realize the error immediately and go to write the correct code.

4.4 Making things easier...

Reflection — Math for making things easier

A lot of math it's about simplifying complex things. For instance, by going back in you memory, to the rigorous definition of irrational numbers, you get an idea of this: in order to deal with the notion of irrational number (just a number!), say π, you need two classes of infinite numbers, the totality of those that are less than π and all those that are greater then π.

In the following, we are going to propose two code constructs which, in different ways, simplify things, encapsulating some sorts of complexity.

Of course, we are introducing these constructs at this point but you may have reasons to introduce them at different points, according to your contexts. The power of the logo is best exploited by leveraging children's exploration skills. Thus, decisions like this one, i.e. when to introduce a new piece of information, depends on how the class is working. If fruitful exploration are going on, just let them go. If someone is lagging behind,

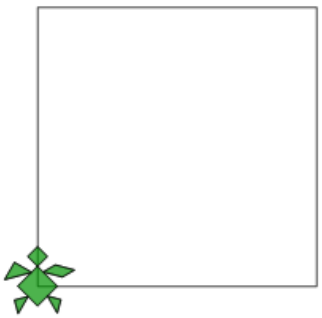
try to let the most advanced ones help them, thus fostering collaborative working as well.

4.4.1 Repetitions

In order for the Turtle to draw a square, some typing was necessary. In particular, we had to repeat a couple of commands four times. This isn't terrible, but imagine if you had some more complex piece of code and wanted to repeat it a lot, say hundreds or thousands of times - this can happen when you are more familiar with coding. It would be painful to type all this stuff, or even simply impossible. Fortunately, Turtle is able to understand some repetition commands. The most common is the following:

```
CLEARSCREEN
HOME
REPEAT 4 [
   FORWARD 100
   RIGHT 90
]
```

Listing 7: In order to execute a sequence of instructions more than once we use the REPEAT command: all instructions within square brackets are repeated.



When the Turtle finds, say, a REPEAT 100 instruction, it will execute all the commands listed among the two square brackets, [and], for one hundred times. Instead of typing one hundred times the same sequence of instructions, it is sufficient to enclose them within the square brackets specifying the number of repetitions.

4.4.2 Creating new commands

As far as we have seen so far, the Turtle executes the instructions while "reading" it or, if you prefer, while "listening" to it. However, the Turtle not only executes the instructions one after the other, like an automaton, but it is also able to memorize many commands at once. Let's explain this with an example.

```
CLEARSCREEN
HOME

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

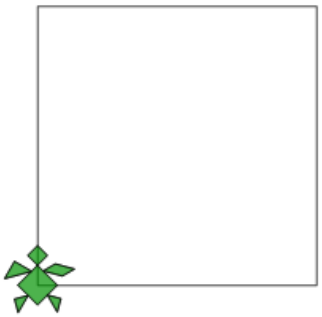
Listing 8: Defining the "SQ" new command

When the Turtle finds an instruction beginning with the word TO, followed by a name, it stops executing the successive instructions, as usual. Instead, it "sits and listens to the commands", learning them one by one without doing nothing else, until it meets the END command. What's going on when you try to execute this code in LibreLogo? Nothing! However, if we think about it, there is nothing to execute in this code, actually, except for the first two commands, CLEARSCREEN and HOME.

The following code shows how to exploit this ability of the Turtle.

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


CLEARSCREEN
HOME

SQ
```

Listing 9: Defining and applying the new "SQ" command. We colored in blue our new command.



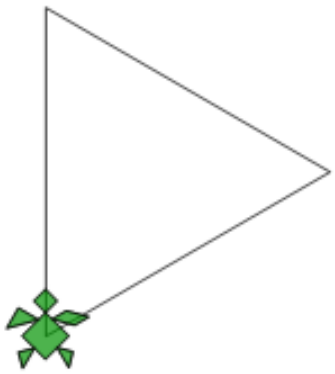
Instruction number 1 is composed by two parts: TO tells the Turtle to sit down and listen and SQ is the name we can use as a new command. This means that, if the Turtle finds this new command, it will remember all the instructions memorized between TO and END, executing them in sequence. That's why we have put the instruction SQ at the end of the code. More precisely, between instructions 1 and 6 the turtle sits and listens - nothing happens. Starting with instruction 7 the turtle will execute all instructions, one at a time.

Since we have put the code at page 20 between the TO and END commands, we got a square. Probably, sooner or later someone will try to define other new

commands, for instance the triangle, again.

```
TO TR
   REPEAT 3 [
      FORWARD 100
      RIGHT 120
]
END


CLEARSCREEN
HOME

TR
```

Listing 10: The "TR" new command



Reflection — Encapsulating functionality in new commands: modular thinking

As soon as students grasp the meaning of the TO...END construct, they feel a sense of empowerment. First because one can encapsulate even a very complex sequence of instructions in just one command and, second, because one can extend and customize the Logo commands set limitlessly. This perceptions lead easily to a modular way of thinking, which means trying to simplify a large process in a limited number of blocks that may be combined easily and, possibly, even reused in other contexts. This is an extremely useful step towards a scientific way of thinking, by means of which more complex tasks can be simplified, reducing the occurrencies of errors and improving communication.

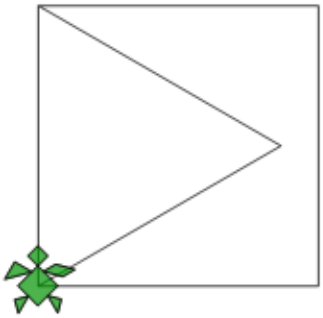
As a consequence of this achievement, the level of self-determined goals is raised for instance for drawing new figure by combining previously made pieces. A typical example is that of a house composed by a square and a triangle.

4.4.3 Another classic — drawing the house

So, let's try to make a house. It should be easy: first draw the square and then put the triangle at the top. Very often people write a piece of code of the following kind...

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

TO TR
   REPEAT 3 [
      FORWARD 100
      RIGHT 120
   ]
END


CLEARSCREEN
HOME

SQ
TR
```

Listing 11: The "usually wrong" first house...



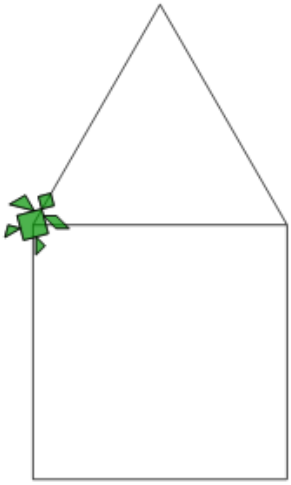
But this is not what it was intended. One realizes that some more reflection is needed and that something has to be done between the two new commands, SQ and TR. In order to solve the problem you must put yourself in Turtle's feet: where is it and in which direction is pointing it, after having drawn the square? And from this *state*, which is the fist move when tackling the triangle? When reflecting on these questions, it is clear, that before starting to draw the triangle, the Turtle should move to the upper left corner of the square, by means of a FORWARD 100 instruction. By doing so, the drawing of the triangular roof will begin at the top of the square, which is good. However, in this way we achieve just a translation of the triangle and our house will look like if it would be opened like a can. For this reason, before starting to draw the roof we should let the Turtle rotate right by an angle of... well you should discover it by yourself!

Reflection — Recalling the process of building scientific knowledge while drawing the house

Just to recall, again, about the iterative nature of building scientific knowledge: first devise a theoretical description, then try to devise an experimental confirmation, finally go back to refine the theory if necessary.

This is the result you should have achieved:

```
1  TO SQ
2     REPEAT 4 [
3        FORWARD 100
4        RIGHT 90
5     ]
6  END
7
8  TO TR
9     REPEAT 3 [
10       FORWARD 100
11       RIGHT 120
12    ]
13 END
14
15
16 CLEARSCREEN
17 HOME
18
19 SQ
20
21 FORWARD 100
22 RIGHT 30
23
24 TR
```



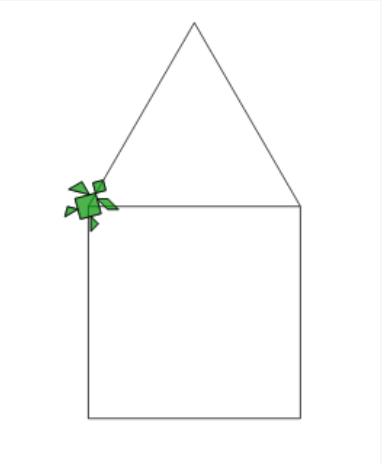
Listing 12: The correct first house...

In instruction 19 the Turtle draws the square, then with 21 and 22 it achieves the correct status (position and orientation) to start drawing the triangle by means of instruction 24.

Reflection — Again on modular thinking while drawing the house

When you find out you can draw an object like that, you feel a sense of empowerment: this new language is extensible, at your will. You can build new blocks of functionalities and combine them freely. Once the house is there, it comes spontaneous to take one more step: perhaps we could put everything in a new command, HOUSE for instance. Try to reflect on the following code, which draws the same house:

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

TO TR
   REPEAT 3 [
      FORWARD 100
      RIGHT 120
   ]
END

TO HOUSE
   SQ
   FORWARD 100
   RIGHT 30
   TR
END


CLEARSCREEN
HOME

HOUSE
```



Listing 13: The "HOUSE" COMMAND

Where the names of new commands are emphasized: **SQ**, **TR** and **HOUSE**. Now we can send the Turtle around, disseminating houses everywhere!

5 Making things smaller and larger

5.1 The Turtle is able to learn symbolic names

It's time to introduce another magic of the Turtle. It was fun to draw many houses so easily, but what if we wanted to create smaller and larger houses? Here you have the new trick: the Turtle is able to use *symbolic names*. Try this:

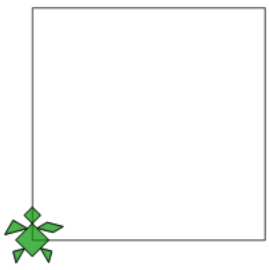
```
SIDE = 100
FORWARD SIDE
```

Listing 14: "SIDE" is a parameter that has value 100

We used a new name here, SIDE, assigning a number to it. In this way the Turtle knows that, every time it encounters the name SIDE, it must use the number 100 instead. This is very useful when you have to use many times the number 100 in different instructions. For instance, if go back to our first way to draw a square, we can write the code in the following way:

```
CLEARSCREEN
HOME
SIDE = 100
FORWARD SIDE
RIGHT 90
FORWARD SIDE
RIGHT 90
FORWARD SIDE
RIGHT 90
FORWARD SIDE
RIGHT 90
```

Listing 15: Using the SIDE parameter



Thus, if we want to make a smaller square we have to change just the value of SIDE.

5.1.1 In case you are using XLogo

In case you use XLogo the code for assigning symbolic names is somewhat different. For instance, in order to assign the value of 100 to SIDE one has to type make "SIDE 100:

```
home
make "SIDE 100
forward :SIDE
right 90
forward :SIDE
right 90
forward :SIDE
right 90
forward :SIDE
right 90
```

Listing 16: How one should write the previous code in the XLogo environment

Therefore, in order to run the same code in XLogo one has to change accordingly the assignment instruction and substitute :SIDE in place of SIDE, everywhere.

Reflection — The door to algebra

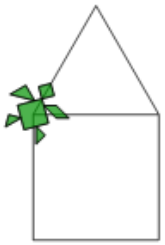
As it is well known, algebra is based on symbolic representation of numbers. Algebra is about generalization. The use of symbolic names in place of numbers in Logo creates the mental device for manipulating symbolic representations of numbers. A crucial step towards mathematical thinking. We used the expression "symbolic names" so far, in the following, depending on the context, we will call them *variables* or *parameters*.

5.2 The Turtle is even smarter: passing parameters to new commands

We have seen how to create new commands, for instance the HOUSE one. Let's suppose we would like to draw houses of different sizes. See here how we can do that:

```
TO SQ SIZE
   REPEAT 4 [
      FORWARD SIZE
      RIGHT 90
   ]
END

TO TR SIZE
   REPEAT 3 [
      FORWARD SIZE
      RIGHT 120
   ]
END

TO HOUSE SIZE
   SQ SIZE
   FORWARD SIZE
   RIGHT 30
   TR SIZE
END


CLEARSCREEN
HOME

SIZE = 50
HOUSE SIZE
```

Listing 17: How to draw houses of different sizes with the same command using the SIZE parameter



The possibilities are unlimited. Look what a primary teacher student was able to do by exploiting the use of parameters in new commands:

The possibilities are endless, especially if we consider that we may give as

many parameters as we want to a new command. Try to play with this:

```
TO RECT A B
REPEAT 2 [
   FORWARD A
   RIGHT 90
   FORWARD B
   RIGHT 90
]
END

CLEARSCREEN
HOME

A = 100
B = 200
RECT A B
```

Listing 18: A new command with two parameters

6 Doing round things

6.1 Syntonic learning

Sooner or later someone will come out saying: — Why not making something round, like a circle? Remember figures 2, 3, 4 and 5 on page 12? The playful exercises shown in these figures were followed by lively discussions about letting the Turtle produce the same geometrical shapes. Maestra Antonella reported a piece of these discussions among her eight year old kids:

> ...
> Girl 1 — With always LEFT the Turtle turns on itself but it doesn't move!"
> Boy 1 — We must change the numbers...
> Boy 2 — We must bend!
> Girl 1 — Yes, I said it, we have to keep turning left!
> Girl 2 — ... but if we keep just turning left we get just a small point...
> Girl 1 — I got it: we have to make FORWARD 1 LEFT 1!
> Girl 2 — ... and keep doing it...
> Boys — Wow! It's turning in circle.
> Girl 1 — But it's not a true circle, just a piece...

Teacher Antonella took the opportunity to introduce the REPEAT command.

> Boy 2 — Well, we have to repeat it the right number of times, but how many? That's difficult...
> Boy 1 — Let's try, 10... oh, too small
> Girl 1 — We must do it much longer, let's try 300...
> Boy 2 — No, 355!
> Girl 3 [10] — Oh that's really easy: the right number is 360.
> Boy 1 – Mmh... no wonder... you're a grown-up!

Reflection — Syntonic learning described through a dialogue

The story of Antonella, shown here by her shots and the dialogue among her kids, is a remarkable example of Papert's pedagogic ideas: activation of syntonic learning through physical activities and fostering of self-determined goals, freedom of exploring. Everything here is about discovering the rules to achieve a personally significant goal instead of teaching rules to solve non significant problems.

Let's now rework figure 4 in this way:

[10] That day some older girls were hosted in Antonella's classroom because their teacher was temporarily missing.

Figure 6: A small step and turn a little bit, a small step and turn a little bit...

This is a kind of "syntonic picture", in that it connects a specific physical action with a piece of code, i.e. an abstract symbolic representation of that action. The correspondence is nice: the girl has to make a small step, in order to remain on the yellow ring. At the same time she has to turn the forward feet a little bit, for the same reason. This helps her to imagine that the Turtle should receive commands with small numbers, FORWARD 1 and LEFT 1, to let it drawn something circular.

Powerful mathematical idea — Differential equations

"Powerful ideas" is a typical expression of Seymour Papert, together with the conception of learning environments that act as "incubator for powerful ideas". We mean ideas that are pre-existent, not belonging to the learning environment, but that are activated by such an environment. At the light of nowadays neuroscientific knowledge of learning, we could define the concept even more precisely: the activation of a powerful idea by an appropriate environment turns out in the creation of a neuronic configuration, sort of a first path in a bush, that in future may host a new mental device. The powerful idea is now activated in an unwittingly way but in future it may flourish in a thoroughly definite concept.

This is exactly the case of the syntonic experiences of drawing circles we have just described. The key point here is that in the FORWARD 1 LEFT 1 code there isn't any reference to global attributes characterizing

the concept of circle: no mention of a center, no mention of a radius. The Turtle is given just a "local rule" and nothing else. Then we get a circle. Is there something wrong? Is there something inaccurate? Something not "enough mathematical"? No, this kind of "local description" is perfectly legitimate in mathematics: it falls into the world of *differential equations*. The following is the differential equation of the circle:

$$\left\| \frac{d\vec{T}}{d\vec{S}} \right\| = k, \tag{1}$$

where k is called the *curvature* of the circle and it is given by $k = 1/r$, where r is the radius. From the mathematical point of view, equation 1 is a *differential equation* because it's about variations, relating them to other quantities. In our case, talking in intuitive terms, equation 1 tells us how much the direction $\vec{T}$ changes for a given variation of the position $\vec{S}$ along the trajectory, that is the girl's step length, and this amount is given by curvature k, which is constant in the circle case.

The following diagram shows the "small spatial displacement", $d\vec{S}$, and the "small direction change", $d\vec{T}$.

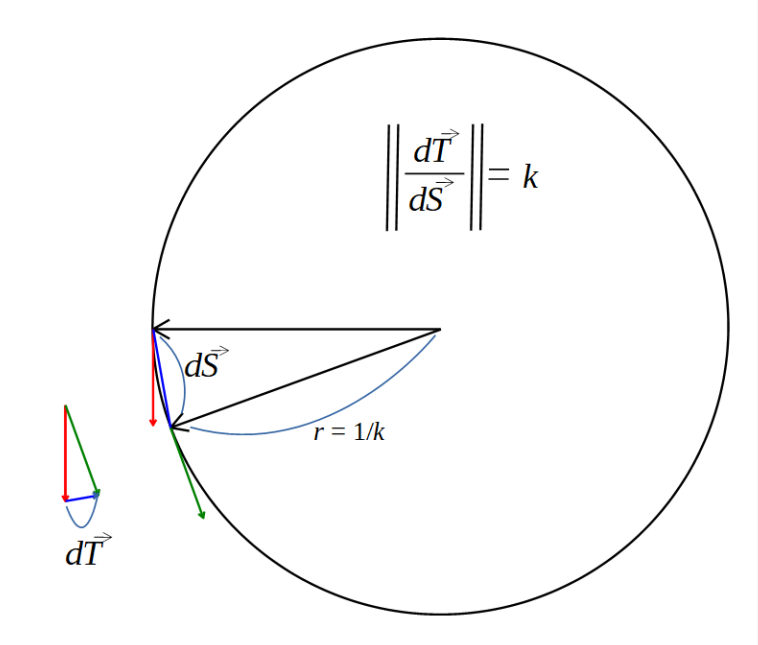


Figure 7: "Syntonic math"

The important concept is that, in a circle, the variation of the direction changes in a constant way while moving along the trajectory. This

constance is expressed by the constant value of the curvature k. In the following picture the relevant elements of the diagram are superimposed to the step of the girl. Here the constance of the curvature is maintained by the girl's effort of making regular steps and regular bending of the forward foot.

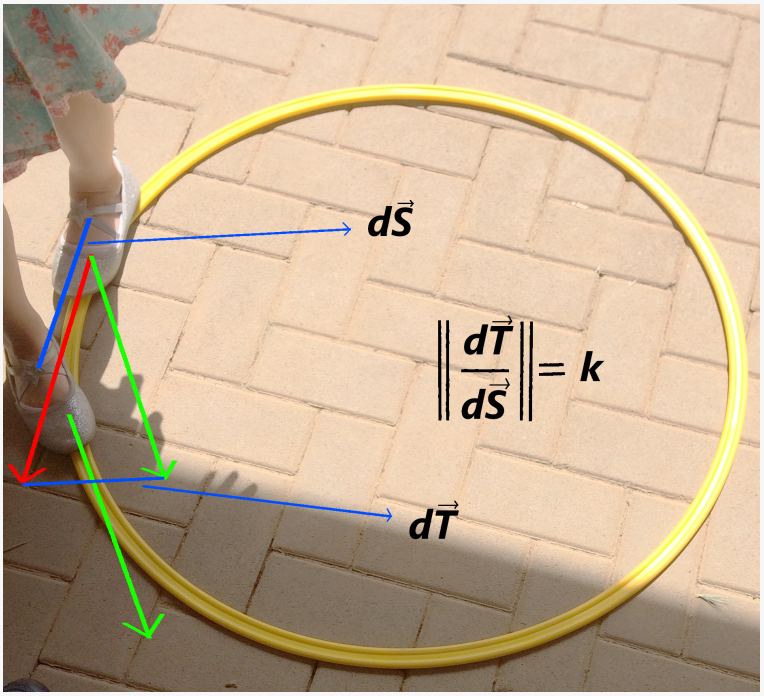


Figure 8: "Syntonic math"

Of course, our girl is not supposed to know anything about differential equations and vectors[a]. However, by means of this activity, she has discovered a new way to look at a geometrical figure, a way lending to the concept of differential equation. A perspective where a figure (eventually a physical phenomenon) is builded by means of a precise behaviour which is "purely local", without the need od any synthetic description of the final figure. Differential equations are the fundamental mathematical tool need in all fields of science.

Within a spiral curriculum this is a first step in a series of successive explorations of the circle, that will take place later on in its development path. Let's explore some of the successive steps, leading to other powerful matematical ideas.

[a] In exact mathematical terms, $\vec{T}$ and $\vec{S}$, represent vectorial quantities, having direction and intensity, $\frac{d\vec{T}}{d\vec{S}}$ is the derivative of the unit tangent vector $\vec{T}$ with respect to the

position vector along the trajectory, $\vec{S}$. The symbolic representations $d\vec{T}$ and $d\vec{S}$ are not finite quantities *per se*. They can be used in expressions involving the concept of *limit*, the basic tool of mathematical analysis for embedding the infinity or the infinitesimal in finite quantities. For instance the so called derivative $\frac{d\vec{T}}{d\vec{S}}$ is the result of a *limit* operation, where the imaginary very small step, at *limit*, is approaching 0. Thus we get a finite quantity, expressing the local rate of change of a variable, $\vec{T}$, with respect to another, $\vec{S}$. Later on, when talking about successive approximations, the same kind of concepts will been evoked.

6.2 Spira mirabilis

Figure 9: Jacob Bernoulli's headstone,
in Basler Münster.

This is the headstone of Jakob Bernoulli (1654-1705), which can be seen in the Basler Münster. Jakob Bernoulli studied the spiral extensively. He was fascinated by its mathematical properties, which he described in a treatise entitled *Spira Mirabilis*. He was so impressed that asked to have the spiral carved on its headstone with the inscription *Eadem Mutata Resurgo*: although changed, I

keep arising the same.

Figure 10: The wrong spiral engraved in the tombstone: Archimede's instead Bernoulli's one...

Unfortunately, the sculptor carved the wrong spiral. Bernoulli's wonder was lifted by the logarithmic spiral, not the Archimedean one, which can now be seen on his tombstone. The difference is crucial, since the two spirals grow in fundamentally different ways.

Let's go back to figure 6 on page 31: by keeping doing a small step and turning a little bit regularly, we approximate a circle. But what happens if, for instance, we reduce step size? Well, we get a kind of spiral, which shape depends on how we reduce the step length. These effects can be easily explored in Logo.

```
TO SPIRA STEP ANGLE
  REPEAT 400 [
    FORWARD STEP
    RIGHT ANGLE
    STEP = STEP + 0.075
  ]
END

TO SPIRB STEP ANGLE
  REPEAT 100 [
    FORWARD STEP
    RIGHT ANGLE
    STEP = STEP + STEP * 0.035
  ]
END
```

Listing 19: SPIRA: Archimedean - SPIRB: Bernoulli.

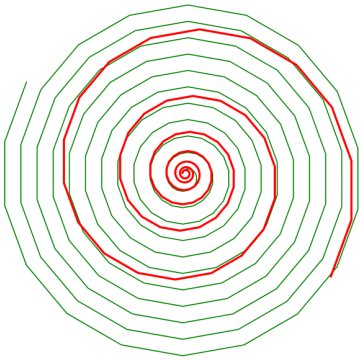


Figure 11: Archimedean (green) and Bernoulli's (red). Both have been called with STEP=1 and ANGLE=20.

Both programs, SPIRA and SPIRB, differ from that of a circle by the fact that they have an instruction (N. 5 in SPIRA and N. 13 in SPIRB) where the parameter STEP is increased by a certain amount at every step. The parameters have been arranged to show the basic difference between the two spirals. The two programs differ for instructions 5 and 13, where the STEP parameter is increased, but in SPIRA (Archimedean) it is increased by the fixed amount 0.075 whereas in SPIRB (Bernoulli) it is increased by STEP*0.035, that is by a quantity proportional to STEP. This means that in the former case the coils grow by constant amounts whereas in the latter the coils become larger and larger at every turn. The Bernoulli spiral is called *logarithmic* and can be found in a great number of natural shapes. Among these there are the ammonite fossils.

```
TO SPIRB STEP ANGLE
  REPEAT 300 [
    FORWARD STEP
    LEFT ANGLE
    STEP = STEP + STEP * 0.005
     ]
END

RIGHT -90
SPIRB 1 2.5
```

Listing 20: SPIRB with parameter adjusted to approximate the ammonite curvature.

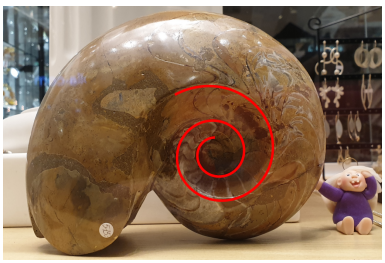


Figure 12: Bernoulli's spiral superimposed to an Ammonite fossil shot from a shop in the Bahnhofstrasse area, in Zürich.

Powerful mathematical idea — Linear and exponential growth

By playing with these spirals students get in touch with the essence of linear and exponential variations.

The equation of Archimedean spiral, expressed in polar coordinates, is

$$r = a + b\theta \tag{2}$$

This means that the radius r grows proportionally with the angle θ.

The equation of the Bernoulli's logarithmic spiral is:

$$r = ae^{k\theta} \tag{3}$$

In this case r grows exponentially with angle θ.

How much of these explanations should be provided to the students, depend at which point of the spiral (!) curriculum they are. What is important is to get in touch early with the linear-exponential dichotomy.

Powerful mathematical idea — Self-similarity → fractals

Among the mathematical properties of the logarithmic spiral that fascinated Bernoulli so much there is also the so-called self-similarity. Let's add a shift θ_s to the angle θ, which means rotating the spiral by an angle θ_s.

$$r = ae^{k(\theta+\theta_s)} = ae^{k\theta}e^{k\theta_s} \tag{4}$$

This means that by rotating the spiral we just change the scale, by a factor $e^{k\theta_s}$, while the shape remains exactly the same, i.e. $e^{k\theta}$. Vice versa, if we multiply the spiral by a scale factor, S

$$r = Sae^{k\theta} = ae^{k\theta+\ln S} = ae^{k\left(\theta+\frac{\ln S}{k}\right)} \tag{5}$$

we obtain the same spiral, just rotated by an angle $\frac{\ln S}{k}$.

The property of mantaining the same shape while changing scale is extremely common in nature and it is the key element of fractal shapes — we are going to see a very simple example, later on.

6.3 Golden spiral

The Bernoulli's logarithmic spiral is related to the so-called *golden spiral*. More precisely, the *golden spiral* is a particular type of logarithmic spiral, where the rate of growth of the spiral depends on the *golden ratio*, one of the magic number of mathematics, ubiquitous in nature. Its value can be calculated with $\phi = \frac{1+\sqrt{5}}{2}$. It's approximated value is 1.618[11].

There is a nice approximation of the *golden spiral*, which can be constructed with paper, pencil, compass and scissors — its' good to mix technologies. You can proceed as follows:

1. Cut a square of paper, for instance with side of 10 cm

[11]The *golden ratio* is an irrational number, therefore it cannot be represented exactly with a finite number of digits.

2. Draw a quarter of circle inscribed in the square, from one corner to its opposite
3. Cut another square with side $10/1.62 = 6.17$
4. Again, draw a quarter of circle inscribed in this smaller square, from one corner to its opposite
5. Place the small square adjacent to the previous one in such a way that the first quarter of circle continues with the first one, without interruption
6. Repeat steps 3-5 until you can manage the progressively smaller squares

You will get something of this kind:

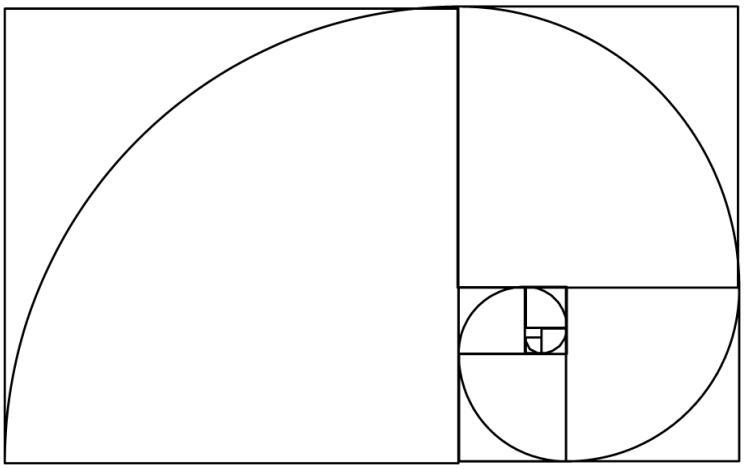


Figure 13: Golden spiral: with Logo but a paperwork as well...

This is a good approximation of a *golden spiral*, which in turn is a *logarithmic spiral*, or a *spiralis mirabilis*, in Bernoulli's words. These figure was made with Logo. As usual when programming, the result can be achieved in many ways. You can try on your own to reproduce with Logo the physical construction of the spiral.

6.4 POLY

Let's go back to the triangle and square codes:

```
TO TR SIZE
   REPEAT 3 [
      FORWARD SIZE
      RIGHT 120
   ]
END

TO SQ SIZE
   REPEAT 4 [
      FORWARD SIZE
      RIGHT 90
   ]
END
```

Listing 21: Let's look at what the triangle and square codes have in common...

The two codes are similar. What are the differences? And what do the codes have in common?

The differences are:

- the command names, TR and SQ
- the number of repetitons, 3 and 4
- the turning angle, 120 and 90

In a nutshell, the turning was done "in three goes" or "in four goes".

What else can we explore here?

Let's try to think a bit in the mathematicians way, i.e. by making things as simple as possible in order to squeeze the essence:

```
TO POLY SIZE ANGLE
   REPEAT [
      FORWARD SIZE
      RIGHT ANGLE
   ]
  END
```

Listing 22: The POLY code

The POLY program is an important object of the so-called Turtle Geometry. We do not go into complicated details ere, it suffices to think about it as the

"Turtle World". By playing around with POLY you may discover some pretty and also relevant facts about Turtle World, geometry and computers. But more importantly, you learn to explore, formulating questions, setting and achieving goals on your own.

If you run this code, for instance in this way:

```
TO POLY SIZE ANGLE
   REPEAT [
      FORWARD SIZE
      RIGHT ANGLE
   ]
  END

POLY 100 120
```

Listing 23: The POLY code

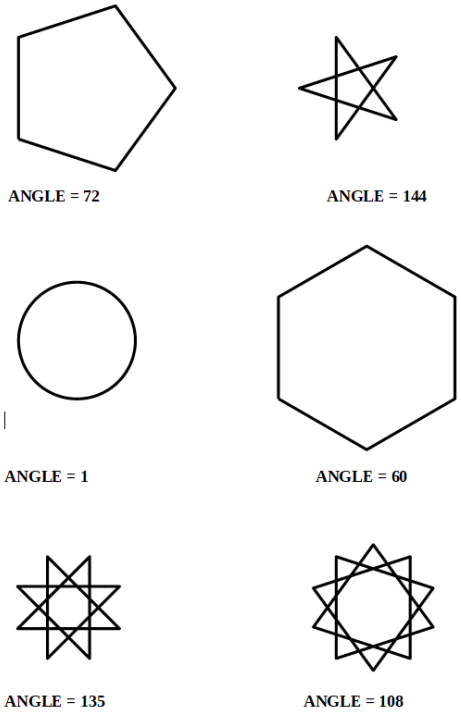
The Turtle will draw a triangle, which is not a great news, since we already learned that by drawing sides with 120°turns one gets an equilateral triangle. The first observation is that, once the triangle has been drawn, the Turtle is keeping drawing it, again and again. We'll come back on this aspect later on. For now, to stop the Turtle, click on the red STOP button, in the LibreLogo toolbar[12]. Instead, let's go straightforward to play, taking advantage from the super concise writing, where the POLY code is reduced to its essential pattern: no numeric parameter hanging around, just the ANGLE and SIDE, which are the important parameters, especially the ANGLE one. By playing with different values of SIZE and ANGLE parameters, think about POLY as a kind of magic box for creating new objects. Try reflecting on the different effects you get by changing SIZE or ANGLE. Which are more interesting? Which kind of figures are coming out? Just polygons? Or also something else?

In order not to spoil the results for those willing to experiment by themselves, we show some outcomes in the following page.

[12]Similarly, yo can stop the process both in the XLogo and the TigerYjthon environments, by means of the red STOP button, more or less in the same position, at the top left in the window.

```
TO POLY SIZE ANGLE
   REPEAT [
      FORWARD SIZE
      RIGHT ANGLE
   ]
  END

POLY SIZE ANGLE
```

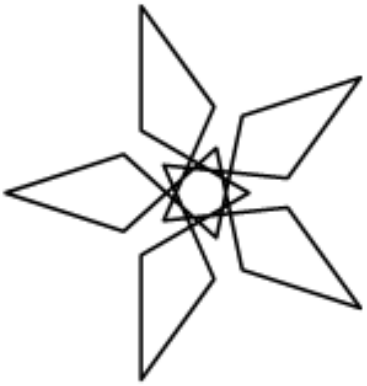
Listing 24: The SIZE parameter is chosen to accomodate the whole picture



This is just an example, we invite the reader to experiment on its own. It is fun to explore the variety of shapes and also how they grow. One can also go on tinkering with the content of POLY, inserting other instructions to see what happens. Like this, for instance:

```
TO POLY SIDE ANGLE
   REPEAT [
      FORWARD SIDE/3
      RIGHT 60
      FORWARD SIDE/3
      LEFT 120
      FORWARD SIDE/3
      RIGHT 60
      FORWARD SIDE/3
      RIGHT ANGLE
   ]
END

POLY 100 144
```

Listing 25: An example of POLY variation, with ANGLE = 144



In all these cases, it is interesting how some figures only need a few cycles to be completed while others need many more iterations. The question arises naturally: how many cycles are needed in POLY in order to "close" the figure, for a given value of ANGLE? Probably, you alredy found that for certain figures the answer can be found rather easily. Basically, among the previous examples we found two kind of objects: usual geometric shapes (polygons, circle) and sort of stars. When ANGLE = 120, we get the triangle, with 90 the square, with 72 the pentagon. Since 120 x 3, 90 x 4 and 72 x 5 all make 360 we can conlude that, to draw a polygon with N sides, the Turtle has to make 360 / N turns. This is a rule derived empirically, which means that we may not be sure that it is universally true. However, the *Total Turtle Trip Law*[13] holds true:

Total Turtle Trip Law *If a Turtle takes a trip around the boundary of an area and ends up in the state in which started, i.e. same position and same heading, then the sum of all turns will be 360°.*

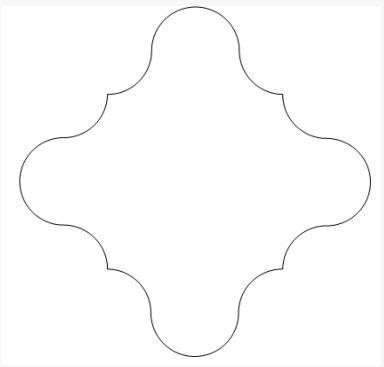
We used the word "law" because it reminds us of something that is always true, in every condition, like the laws that we must respect in our everyday life. In mathematics there are axioms, or postulates, upon which whole theories are founded; and theorems, i.e. statements which are demonstrated to be true starting from axioms. In physics we have principles, of energy conservation, of dynamics and so on. In Turtle Geometry there are some theorems which hold true: the Total Turtle Trip Theorem (the proper name) is the first one.

[13] This is Papert's enunciation [Papert, 1993, pag. 76:] of the Total Turtle Trip Theorem. We are using here the word "law", which is more appropriate when talking to kids.

Powerful idea: Concept of "law"

By playing with the constance of a results, independently from the path, we are developing the important conceptual device of *invariant*, such as laws or principles in physics, axioms or postulates in mathematics, which are the pillars needed by any kind of scientific theory. It is a way to give a structure to the narration of science. With children a little older, one can try to build some strange figure.

```
1
2  CLEARSCREEN
3  HOME
4  HIDETURTLE
5
6  T =0
7  REPEAT 4 [
8     FD 100
9     RT 90
10    T = T +90
11 ]
12
13 PRINT T
14
15 CLEARSCREEN
16 HOME
17 PENUP
18 LEFT 90
19 FORWARD 100
20 PENDOWN
21 RIGHT 90
22
23 T =0
24 REPEAT 4 [
25    REPEAT 90 [
26       FORWARD 1
27       LEFT 1
28       T = T  - 1
29    ]
30    REPEAT 180 [
31       FORWARD 1
32       RIGHT 1
33       T = T  + 1
34    ]
35    REPEAT 90 [
36       FORWARD 1
37       LEFT 1
38       T = T -1
39    ]
40     RIGHT 90
41     T = T + 90
42 ]
43
44 PRINT T
```



Listing 26: A distorted square, to show invariance of total turn

For instance here first we drew a square, accumulating in the variable T the deviation angles, printing the final value at the end, which turns out to be 360, of course. Then a new figure is drawn, which is the same square

but with highly distorted sides, again always accumulating the deviation angles in T, negative for left turns and positive for right turns. At the end the final value of T is printed, which is, again, 360. The idea is to let them create challenges: let's see if with this crazy shape it's still true...

Powerful idea: Concept of integration

In the previous example we were accumulating a quantity along a path, by summing all deviation angles while the Turtle was travelling. This is the basis of a fundamental operation of mathematical analysis: the integration. In the previous example we have proposed a simple code using for instance an instruction such as T = T + 90, to add the turning angle of 90 to the current value stored in the variable T. However, with kids that are still not sufficiently familiar with programs, this work can be done in parallel, by reading the code and, at the same time, taking note of the values with paper and pencil, summing them by hand.

Of course, we have not to talk about integration explicitly with the students — this would be a nonsense. It is the process, the direct experience that matters. The opportunity of creating a new mental device, which one day will help to accommodate the formalisation of the concept.

Reflection — The limits of the machine (and of theory): how can the execution of a program unintentionally become a never-ending story?

This is the title of a section (2.4) of Juraj Hromkovič's book, "Algorithmic adventures" [Hromkovič, 2009, pag. 62]. It is perfect here. It may seem strange: according to common sense, computers are associated to deterministic computational procedures. Sort of complicated technology but clearly defined and behaving predictably. In reality they are complex, and therefore tricky machines, both from the practical and theoretical point of view. Well, we are already aware of the absence of a halting condition in our POLY program, this was not an error. It is a known imperfection, but imperfections (and errors) are useful in learning. The POLY "defect", that it doesn't stop itself, may turn out in an opportunity to reflect upon the machine behaviour. For instance, what it does actually mean that the computer (the Turtle) is doing nothing? Or, what's going on when it appears to be stuck?

When we run POLY we saw something interesting: after a certain number of cycles, the figure was finished but the program was still running. The two things do not coincide. It is almost always easy to see when the figure is finished, starting from the time where the Turtle begins to travel over lines that were already drawn. But try to hide the Turtle, before starting POLY, by means of the HIDETURTLE command (same command in XLogo, hideturtle, and hideTurtle() in TigerYjthon). At the beginning the invisible Turtle draws the figures and you see it clearly, but once the figures is completed, there is no way to see what the Turtle is doing, since it is invisible! You know that is there, because you put the HIDETURTLE command, so you know the magic, but imagine that you did not know it. What would you conclude? That the task is finished? Not exactly. At

first glance yes, but after a while you will notice something wrong. For instance, if you try to start the code again, you cannot do so and the reason is that the invisible Turtle is still busy. If you want to start a new run you have first to stop the current one. It is exactly what we did by playing with POLY. But what is worthwhile to note here is that, once the figure is finished *the system appears to be stuck*. What we learn thus is that when we say that the computer is stuck it is actually working, only we don't see it. In our case the Turtle is keeping doing something useless, we now it perfectly. But it is important to know that, when running a program, wrong conditions causing a similar condition are quite common, even with very famous (and expensive) software.

It is not to say that software designer are incapable, they are very good, generally. Instead, there are serious reasons for this state of affairs. A first issue is that software systems are extremely complex nowadays. It's often impossible to foresee all the potential problems. However the reality is even harsher: there is no way to build automatic testing methods (algorithms, technically speaking) for assessing the correctness of a program. This is a strong statement, in the mathematical sense, meaning that it has been demonstrated theoretically that there is no way to perform task such as the following ones [Hromkovič, 2009, pag. 155]:

- Is a program correct? Does it fit the aim for which it was developed?
- Does a program avoid infinite computations (endless repetitions of a loop)?

We aren't computer scientists, nor software engineers, and only a very small number of our students will take one of these careers. The point here is the attitude towards these complex machines and technology in general. An attitude aware of the limits and of the unavoidable risks of errors or misbehaviours. A more watchful attitude towards technologies is crucial in education, nowadays, if we want to educate aware citizens instead of passive consumers.

For the curious — Trying to find a stopping condition for a (simple) program

Probably, at this point it is clear that, when ANGLE is a submultiple of 360, the figure is a regular polygon with N=360/ANGLE sides. Therefore, in these cases the number of repetitions in POLY needed to close the figure is N. But in all other cases? When ANGLE is not a submultiple of 360 or ANGLE $>$ 360?

Three years ago a primary teacher student sent me a letter — it was not an assignment — where, starting with the description of an exploration with Logo, written "as if I were a child", she ended up with a mathematical conjecture: is there a way to establish *a priori* how many iterations are needed to close these figures? Actually, what she was exploring, was the behaviour of a kind of POLY program. More precisely, she was playing with a POLY structure with the construction of a house inside it! If you like, you can find the detailed story in my MOOC about cod-

ing: *https* : *//federica.eu/l/martas_story_the_turtle_total_trip_theorem* — the MOOC is free: in order to access all the lessons you have to make just a free account — truly free: no cost, no advertising, no tracing.

The solution to this problem comes from Turtle Geometry. It is based on some theorems, where the fundamental one is a generalised version of the Turtle Total Trip Theorem we have seen before. It is the *Closed-Path Theorem* [Abelson and diSessa, 1980, pag. 24]:

Theorem 1 *The total turning along any closed path is an integer multiple of 360°.*

Where total turning is an intrinsic property of a path. It does not depend on where the path starts, or how it is oriented. The total turning of a path is summarized by the integer that multiplies 360. That integer is called the *rotation number* of the path. It is interesting to try evaluating the rotation numbers for different paths. Here you have some examples:

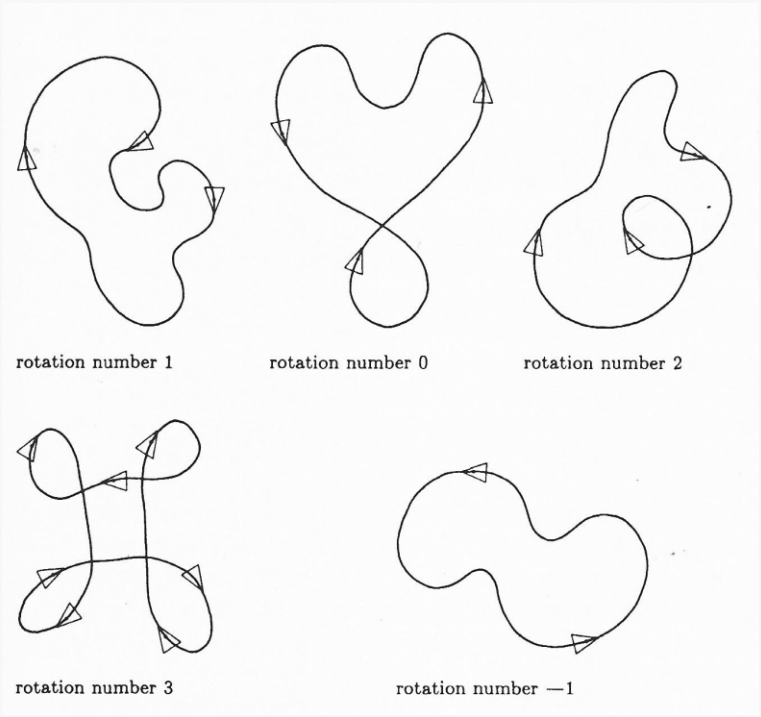


Figure 14: Close path examples and relative rotation numbers

The general outcome is that the number of cycles needed in POLY to close a figure is given by

$$n = \frac{\text{LCM(ANGLE/360)}}{\text{ANGLE}} \tag{6}$$

where ANGLE is the input angle to POLY and LCM(ANGLE/360) is the least common multiple between ANGLE and 360.

The exact derivation of this formula can be found in [Abelson and diSessa, 1980, pag. 24-32], the answer to Marta's conjecture in [Abelson and diSessa, 1980, pag. 32-36] and a synthetical derivation in the above mentioned MOOC lesson.

Thus, this was for the curious, but some of the readers could find material to propose some interesting explorations to somewhat older kids, here.

6.5 Successive approximations

We have seen that with POLY we can draw regular polygons and "stars". Let's write a version of POLY for drawing just polygons. At this point it's easy:

```
1 TO POLYG SIDE N
2    REPEAT N [
3       FORWARD SIDE
4       RIGHT 360/N
5    ]
6 END
```

Listing 27: A new version of POLY with SIDE and N, number of sides, as input parameters

This is a sort of "educated" POLY, which knows how to stop. This version is limited to regular polygons[14] but this is exactly what we want right now. In this version we have changed one input parameter: instead of ANGLE we have put N, the number of sides. Then, within the body of the program we calculate the turning angle as 360/N.

Now we can play with POLYG focusing on regular polygons. The game is simple: try drawing polygons with an increasing number of sides. Well, from time to time you will need to readjust SIDE, otherwise with large number of sides the figure will exceed the page. What will you see from a certain point?

Let's try, for instance, to superimpose the decagon (10 sides) with the icosagon (20 sides):

[14] It would be possible to write an "educated" general POLY program but we would need to use the result given in equation 6, therefore we should write the code for calculating the least common multiple.

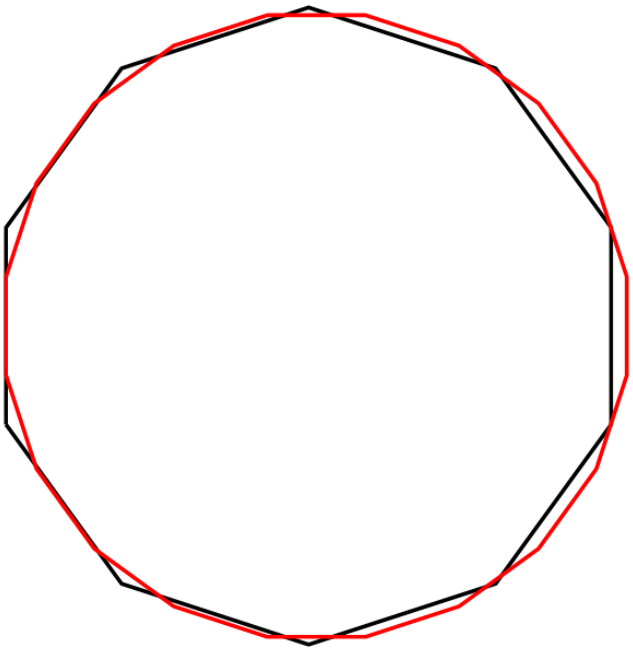


Figure 15: 10-sided and 20-sided polygons

Do you see how the icosagon is "more round" with respect to the decagon? Now let's compare the icosagon (20 side) with the triacontagon (30 sides)[15]

[15]In case you wonder I know these awkward names... not at all! I' m finding the names here: *https* : *//en.wikipedia.org/wiki/List_of_polygons*.

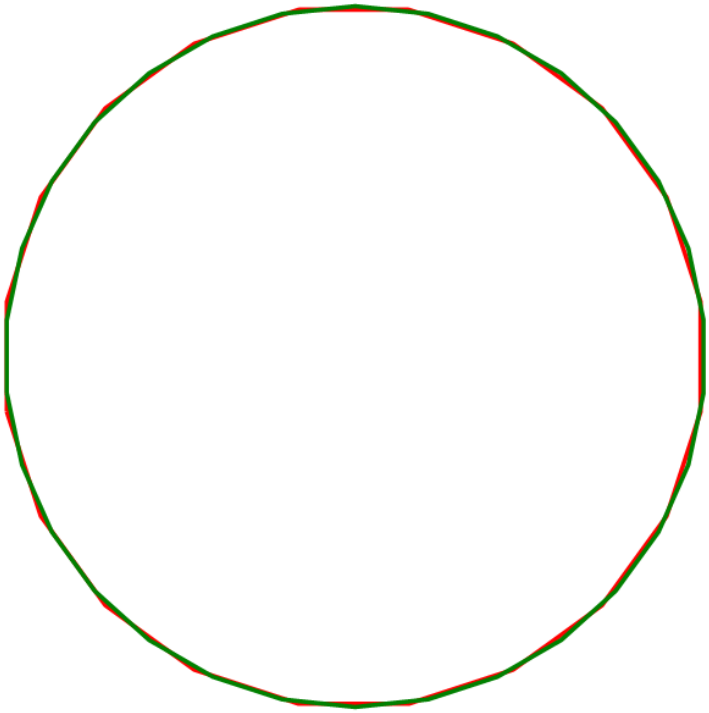


Figure 16: 20-sided and 30-sided polygons

Here we see that, in turn, the 30-sides polygon is "more round" than the 20-sides one. Now, let's exaggerate, comparing the 30-sides polygon with the 360-sides polygon — 360 1°turns!

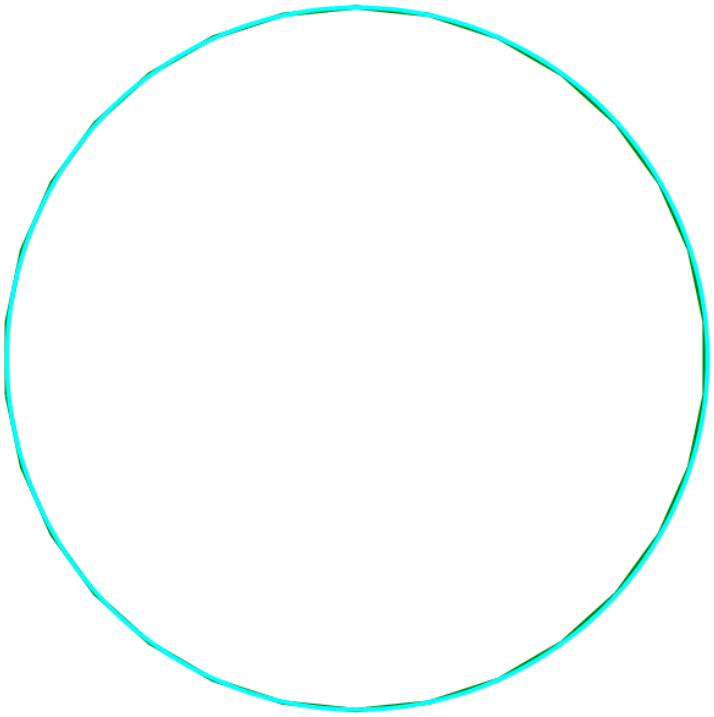


Figure 17: 30-sided and 360-sided polygons

The differences are minimal and, even if you zoom in deeply the image the 360-sided polygon appears to be "round". This happens because the length of the sides are comparable with the pixels, thus we cannot see them any more. Therefore, in Logo we can assume that a circle can be drawn by means of an instruction like REPEAT [FORWARD 1 RIGHT 1]. Recently, a smart 16 years old student told me promptly: "Prof. you are claiming we are drawing a circle that way, but it cannot be, it is still a polygon, even with many very small sides!". He was perfectly right. The discussion arised by this remark was extremely fruitful, leading to a crucial consideration: we can choose an extremely high number N to build an N-sided polygon, and this will be "rounder", even with respect to our last 360-sided polygon, but still, it will not be a true circle. Nobody's stopping us from picking an even bigger number to draw an even rounder polygon, but still a polygon! Thus, are we able to approach the circle as much as we want? Yes! Will we be able to reach the circle? No, unless we have the eternity!

Powerful mathematical idea — Successive approximations

Considerations of this kind are the roots of reasoning for handling the infinite and the infinitesimal in mathematical analysis. Limits, derivatives, integrals, approximation by infinite series and other mathematical devices allows us to solve all the basic problems of science, by tackling the concept of infinite. The experience of approximating a circle, as we have shown, will turn out to be extremely useful for those students who will face a STEM path. By the way, once posed in rigorous formal terms, this is one of the possible definitions of a circle.

The idea of a thought process, composed by perfectly defined steps but "requiring eternity", falls in the realm of the considerations made in the note *a* on page 33. It is by means of the *limit* tool of mathematical analysis that infinity and infinitesimal may be "packaged" in finite and manageable expressions. Again, this is not about telling your students all this, but it is important to be aware of what is behind these Logo exercises.

The ability to manage the concept of infinity and infinitesimal is of crucial importance in all domain of science. Hromkovič, in his book *Algorithmic adventures* devotes an entire chapter to a brilliant description of the concept of infinity and why *infinity is infinitely important in computer science* [Hromkovič, 2009, pag. 73].

7 Physics lab

7.1 Free falling body

This chapter is aimed at those involved in learning or teaching physics at school, particularly classical mechanics. We are therefore no longer in primary school. Since the idea here is to reproduce a sort of virtual physics laboratory, we will introduce some new commands, in addition to the basic Logo command, to better focus on the relevant concepts.

Let's consider a free-falling body in proximity of earth surface. If you are a student at her or his first contact with these matters, probably you have been told that there is a formula describing the motion of the falling body, which can be written as

$$y(t) = y_0 + v_0 t + \frac{1}{2} g t^2 \tag{7}$$

where y is the position of the body at each time instant t, y_0 is its initial position, that is when we open the hand to let it fall, v_0 is its initial velocity, 0 if you just let the body fall, and g is the gravity acceleration constant in proximity of earth surface, its value being 9.8 m/s^2. Depending on the curriculum, the teacher method and your age, you may be told this formula derives from the second principle of dynamics, which states that

$$\mathbf{F} = m\mathbf{a} \tag{8}$$

where $\mathbf{F}$ is the force applied to the body (gravity in this example), m is the body mass and a its acceleration[16]. To understand this derivation you have to master the basic tools of mathematical analysis, i.e. integration in this case. Most probably you learned these formulas by heart, in order to be able to solve the exercises and pass the exams. Which, unfortunately, is not enough for the thorough and substantial understanding of the matters. Logo can help us, let's see how.

In the following figure, on the left, we show the relevant part of Logo code for simulating the free fall of a body, on the right you have the output of the program, where the position of the body at successive equal interval of times is drawn with a small green circle.

[16]Later on in your studies, you will learn that equation 7.1 is a differential equation of the second order, since it's involving variations of variations: acceleration is the variation of velocity whereas velocity is the variation of space. This is written as $\frac{d}{dt}\frac{dy}{dt} = mg$. By integrating this equation two times with respect to time we obtain equation 7.

```
1
2  YPOS = 0.0 ; initial position
3  VEL = 0.0 ; initial velocity
4  ACC = 9.8 ; constant acceleration
5  DT = 0.5   ; time interval
6
7  WHILE YPOS < 500 [
8    VEL = VEL + ACC*DT ; update velocity
       for next point
9    YPOS = YPOS + VEL*DT ; update position
       for next point
10   PENDOWN
11   CIRCLE 5
12   PENUP
13   FORWARD VEL*DT
14   ]
```

Listing 28: Logo code for a free falling body experiment

There are a couple of new commands here. First we made a loop by means of a WHILE instruction instead of a REPEAT one. In particular we used

WHILE YPOS < 400

When using a REPEAT we have to specify a certain number of repetitions, for instance REPEAT 10. Here we have the expression YPOS < 400 instead of the number of repetitions. What does it mean? The expression is a so-called *condition*: the WHILE command will repeat the sequence of instructions between the square brackets, [...], until the *condition* is satisfied, i.e., until the value of the variable YPOS is less then 400. The first time that YPOS will result to be greater or equal than 400 the loop the Turtle will get out of the loop, executing the following instructions, eventually. The second new command is CIRCLE. This is a ready-made LibreLogo command: CIRCLE R, where R is the radius of the circle. We used it just for brevity. In LibreLogo there are such ready commands for the most common figures. We did not used them so far because for the first basic explorations of Turtle Geometry it is better to do things "by hand". Nobody's stopping us from building our own MYCIRCLE R program, starting from our basic REPEAT 360 [FORWARD 1 RIGHT 1] code — it could be a good exercise. Here we go straightforward to the physics

concepts but before discussing them we suggest to try the Python version. We used here the ABZ's TigerJython Python environment[17] Python is more suited for these kind of programs and it is also perfectly adequate for students facing the first physics issues.

```
1  from gturtle import *
2
3  makeTurtle()
4  clearScreen()
5
6  setPos(0,200)
7  setHeading(180)
8
9  YPOS = 0.0  # initial position
10 VEL = 0.0   # initial velocity
11 ACC = 9.8   # constant acceleration
12 DT = 0.5    # time interval
13
14 while YPOS < 400:
15     VEL = VEL + ACC*DT      # update
       velocity for next step
16     YPOS = YPOS + VEL*DT    # update
       position for next step
17     penDown()
18     dot(5)
19     penUp()
20     forward(VEL*DT)
```

Listing 29: Python code for a free falling body experiment. Steps are increasing because of uniform acceleration.

Let's comment this program in detail. Instruction N. 1 imports all the resources needed by Python to manage a Turtle. It's a common praxis in Python, so as to load the necessary resources only. Instruction N. 3, makeTurtle(), creates a so called Turtle instance, in substance a new Turtle listening to your commands. By means of instructions N. 6-7 we send the Turtle at the top of the page to drop it on the ground, at the bottom of the page. That's trivial. Instead, instructions 9-10 are important because they set the so-called *initial conditions*. This is the first crucial step for solving any physics problem. In the algebraic formulation of the problem, namely in equation 7, the initial conditions consists in the values of the initial position y_0 and initial velocity v_0. You will realize (perhaps) the meaning of the values when you will have to solve some problems but here it is more direct: you have to decide at once where to place the Turtle and which initial velocity you have to assign to it. Not only

[17] Link available in
https : //webtigerjython.ethz.ch/

that, you have also to decide which unit of measurement you should use. In other words, you have to build a thorough knowledge of the concept by using it at once. Similarly, you have to fetch the correct value for the acceleration of gravity in instruction N. 11.

Powerful concept in physics — Initial conditions

The setting of initial conditions is the first essential step needed to solve any problem in Physics. Initial conditions are embedded in formulas taught in secondary schools but, at this age, the capability of understanding all the implications which are synthesised in formulas is rare — building the mental device for understanding mathematical formulas requires time, especially if physical meaning is involved. The problem is not trivial and not a novel one: a famous physicist, Enrico Persico, master of Enrico Fermi, wrote an interesting paper in 1956 [Persico, 1956], wondering about the good formal preparation of some students but their poor comprehensions of the core matters[a]:

> Why does this girl, who is not stupid, but who finds it so difficult to describe a capacitor, once put on writing formulas [Maxwell equations], runs like a locomotive?

The computational formulation of physical problems favours the dynamic perception of physical phenomena, with respect to the conventional algebraic formulation, as it has pointed out in a very thoughtful paper by Sherin [Sherin, 2001]. It's not about dropping algebraic description of physical phenomena. Instead, it's about integrating both approaches.

[a]The PDF of the paper can be found among the materials in // *http* : *//iamarf.ch/Codice − LOGO/*

In instruction N. 12 we set the *time interval*. This is a necessary parameter because continuous quantities become discrete, when solving problems numerically. That is, when using time, we have to divide it in many equispaced intervals, and refer to all quantities — position, velocity... — evaluating them once for each time interval, for instance every second. The length of the interval has to be chosen by means of a trade-off: many short intervals describe better what's going on but the computation and other burdens may turn out to be too high.

Powerful concept in physics — Computational vs algebraic approach

When facing problems from the numerical point of view, one is obliged to enter the anatomy of phenomena, once again. Appreciating the numerical way to face problems, besides the algebraic one, is essential to shape adequately the scientific culture of a student. Nowadays, the computational face of all sciences is equally important with respect to the classical descriptions. Moreover, entire new crucial fields are totally numerical. All technologies are based on numerical solutions of mathematical problems — ubiquitous computed tomography images represent the numerical solution of the tomographic problem, based on the inversion of the Radon Transform. Thus, more precisely, technologies are based on a fine blend

of analytical formulations and numerical methods. It is important to give students the possibility to dive into numerical exploration.

Instructions N. 14-20 solve the problem. What does it mean "solving a problem" here? From the classical point of view, it means to find the function $y(t)$, given the physical circumstances. Equation 7 is the solution to problem , given the acceleration g and the initial conditions y_0 and v_0. In the numerical way, solving the problem means to calculate the value of the position YPOS for a sufficient number of time points and, in our case, representing them graphically. In the numerical way the phenomenon is simulated, actually. Instruction N. 14 setup a loop which will last until the position YPOS will keep being less than 400. For each cycle, first the velocity for the next time point is evaluated with VEL = VEL + ACC * DT (instr. N. 15), based on the definition of acceleration. Once we have the new value of VEL, we can find out the new value of position: YPOS = YPOS + VEL * DT, according to definition of velocity (instr. N. 16). Finally, dot(5) draws a point in the current position and forward(VEL*DT) send the Turtle in the next point. The figure shows the expected uniformly accelerated motion.

The reader is invited to play with this code, trying for instance to calculate more data points or to change the acceleration, i.e. who would the body fall on the Moon? And on Jupiter?

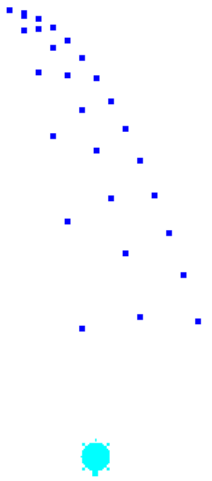
All these codes can be found in the downloadable ZIP file at
$http://iamarf.ch/Codice-LOGO/$.

7.2 Free-falling body with constant acceleration and horizontal velocity component

In this and the following examples we are going to show some variations on the previous case. We leave to the reader the pleasure to discover the differences. We will only quote the added effects.

Here we show the same free-falling body situation but with a constant horizontal velocity component. Three trajectories are shown, for three different values of the gravity acceleration constant. This physics exercise allows to see how a mathematical objects — the parabola — stems from a natural phenomenon. It is interesting to explore the effect, in the parabola equation $ax^2+bx+c=0$, of the a parameter, whose role in this context is played by the acceleration gravity constant G.

```
from gturtle import *
from math import *

makeTurtle()
clearScreen()


def PARAB(ACC):
    setPos(0,200)
    setHeading(180)

    YPOS = 0.0
    VEL = 0.0
    DT = 1.

    while(YPOS < 500):
        VEL = VEL + ACC * DT
        YPOS = YPOS + VEL * DT
        penDown()
        dot(5)
        penUp()
        forward((VEL/2-1) * DT)
        left(90)
        forward(10 * DT)
        right(90)

PARAB(5)
PARAB(9.8)
PARAB(30)
```



Listing 30: Free-falling body with a constant horizontal velocity component. Three trajectories are shown, for three different values of the gravity acceleration constant.

7.3 Free-falling body with air resistance

Here we have an air resistance component, RES, which we express by means of a force which is proportional to the velocity VEL, by means of constant KA. The acceleration, or better said, the deceleration caused by this force will be RES/M, where M is the body mass. Try for instance to simulate a thicker atmosphere...

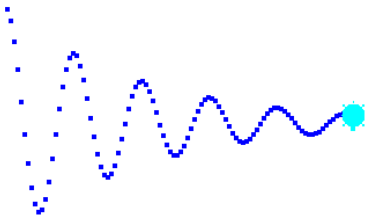
```
from gturtle import *

makeTurtle()
clearScreen()

setPos(0,200)
setHeading(180)

YPOS = 0.0
VEL = 0.0
ACC = 9.8
DT = 0.5
RES = 0.0

G = 9.8
M = 10.0
KA = 2.0

while YPOS < 350:
    RES = VEL * KA
    ACC = G - RES / M
    VEL = VEL + ACC*DT
    YPOS = YPOS + VEL*DT
    penDown()
    dot(5)
    penUp()
    forward(VEL*DT)
```

Listing 31: Python code for a free falling body experiment with air resistance. Steps are becoming constant because the balancing between gravity and air resistance.

7.4 Body hanging from spring

We maintain here the air resistance component proportional to body velocity, but we attach it to a spring. This effect is achieved by adding an elastic force component, proportional to the displacement with respect to the equilibrium position. The intensity of this force is given by constant K. In this version we also put a horizontal displacement to obtain the effect of plotting the motion with respect to time. For simulating just the motion you can set as comment the last three instructions, by placing a # character at the beginning of the lines.

```
from gturtle import *
from math import *

makeTurtle()
clearScreen()

setPos(0,200)
setHeading(180)

YPOS = 0.0
VEL = 0.0
ACC = 9.8
DT = 1.0
RES = 0.0

G = 9.8
M = 10.0
KA = 0.5
K = 1.0

repeat(100):
    RES = KA * VEL
    SPR = K * YPOS
    ACC = G - RES / M - SPR / M
    VEL = VEL + ACC * DT
    YPOS = YPOS + VEL * DT
    penDown()
    dot(5)
    penUp()
    forward(VEL * DT)
    # set as comments the following
    instructions
    # if want to see oscillating along y
    -axis
    left(90)
    forward(3*DT)
    right(90)
```

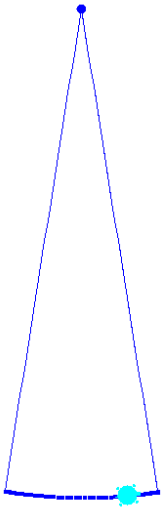


Listing 32: Python code for a body hanging from a spring in presence of air resistance.

7.5 Pendulum

Here a little bit of trigonometry is needed. There is something to improve in this code but still a starting point...

```
from gturtle import *
from math import *

makeTurtle()
clearScreen()

penUp()
forward(290)
right(180)
penDown()

# physical context

G = 9.8/1000/2.857  # constant acceleration
                    # expressed in points/sec^2
L = 500             # Pendulum length

# Where are we starting?

PHY = -pi/20        # initial angle

# Whith which velocity?

OMEGA = 0.0         # initial angular velocity
DT = 1.0            # integration time interval

# Position the Turtle in an appropriate location

dot(10)
setHeading((pi + PHY)/pi*180)
forward(L)
back(L)
setHeading((pi - PHY)/pi*180)
forward(L)
penDown()

repeat(200):

# how much do I have to turn the moving foot?

    setHeading((pi/2-PHY)/pi*180)

    OMEGA = OMEGA - G*sin(PHY) * DT
    PHY = PHY + OMEGA * DT

# mark current location

    penDown()
    dot(5)
    penUp()

# do next step

    forward(OMEGA*L * DT)
```



Listing 33: Pendulum, something to improve...

7.6 Body free-falling from space

This is the same case of the free-falling body but we have to drop the assumption of constant gravity acceleration, because as long as we move away from Earth the gravity force is fading. This means that we have to use explicitly Newton gravity law. Are you able to find out in which line of code we are using Newton's law?

```
from gturtle import *

makeTurtle()
clearScreen()

setPos(0,-200)
setFillColor("blue")
dot(70)
setFillColor("lime")
XPOS0 = getX()
YPOS0 = getY()

setPos(0,200)
setHeading(180)
hideTurtle()

XPOS = getX()
YPOS = getY()
VEL = 0.0
DT = 2.0
KG = 9800

repeat(500):
    DY =  YPOS-YPOS0
    print(DY)
    if(DY != 0.):
        VEL = VEL + KG / DY**2 * DT
    YPOS = YPOS - VEL * DT
    penDown()
    dot(5)
    if(YPOS-YPOS0 < 10):
        break
    penUp()
    moveTo(XPOS,YPOS)

setPenColor("red")
dot(20)
```

Listing 34: Body free-falling from space

7.7 Calculating the orbit of Halley's comet...

To tell the truth, I hesitated to add this section, being afraid of scaring the reader. Actually, it is somewhat outside the scope of this text but, I believe, it could also be fascinating. Read it just if you are curios.

In order to adhere to Papert's statement about educational programming languages that should have both a "low floor and high ceiling", with this example we show that even rather complex tasks can be done with these educational programming languages. Therefore, among the downloadable software, you find also a LibreLogo version of this program. Here we are going to comment the

Python version[18].

But another reason to go through this example is to see how similar are the structures of the simple algorithm we have seen for drawing a circle and that of drawing the orbit of a celestial body, despite the remarkable difference in complexity. The reason of such similarity stays in the same differential local nature in which the two problems are posed. We show this fact by sketching the two algorithms with the so-called *pseudo code*, which is not an executable code but one that allows to grasp the essence of the algorithms:

Data: (small) step length and (small) turning angle
Result: plot of the circle
go to starting point;
while *circle is not closed* **do**
    step forward;
    turn right;
    plot the step;
end

Algorithm 1: Drawing a circle

Data: astronomical data about Halley's comet
Result: plot of the orbit
initialization;
while *orbit is not closed* **do**
    calculate acceleration in current point;
    given this acceleration value calculate velocity;
    given this velocity value calculate position of next point;
    move to next point;
    plot the step;
end

Algorithm 2: Calculating Halley's comet orbit

[18]As far as the listing at page 64, we reported the whole content of the Halley-RK-4-AU-90-sharable.py file you find among the downloadable materials. For instance, also the initial free software declaration, just as a reminder. A note about the expression "program" we used throughout the listing. In all the "Einfach Informatik" volumes published by ABZ, for instance in "Einfach Informatik — Programmieren — Sekundarstufe I"' [Hromkovič and Kohn, 2018a], the word "command" (*Befehl*) is used for designing sequences of instructions which can be invoked by means of a single name for being executed altogether. This choice makes sense since, when starting with Logo, it is very appropriate to give the kids the idea of building their own self-made commands, because with the Turtle these encapsulated pieces of code are usually new commands for the Turtle. However, in this case it is somewhat weird to call "command" the instructions used to make an interpolation, for instance. Hromkovič and Tobias [Hromkovič and Kohn, 2018b, pag. 45] raise the point about the variety of names which have been used to call these objects, for instance "routines", "procedures", "subprograms", "subprograms", "methods", "functions". They do not mean exactly the same things, depending on historical context and on some specific behaviour. For instance, in Python jargon they are usually called "functions". However, there also the tendency to assume that "functions" calculate a value and renders it. Well... in this listing we used the term "subprograms".

The previous example of a body free-falling from space is opening the way to the evaluation of bodies orbits in space. Adding an initial horizontal velocity component lend us to the simulation of orbits. Let us sketch the problem in the context of Newton's gravity, first in algebraic notation and, of course, limiting us to the two-dimensional problem. As far as the algebraic approach is concerned, here we just pose the problem. The analytic solution of the differential equations is a higher education level problem. However the algebraic approach serves as starting point for the programmed numerical solution.

To solve the problem we have to start from the second principle of dynamics

$$\mathbf{F} = m\mathbf{a} \tag{9}$$

where the force is given by Newton's law

$$\mathbf{F} = -\frac{GMm}{r^2}\frac{\mathbf{r}}{r}. \tag{10}$$

G is the universal gravitational constant, M is the mass of the largest of the two bodies, m the smaller one, r is the distance between the two masses, between the barycenters of the two bodies to be precise. Bold symbols are vectors, the others are scalars. First we have to derive the acceleration of the smaller body — the larger one will stay still, assuming that its mass is negligible with respect to that of the sun, i.e. $M \gg m$

$$\mathbf{a} = -\frac{GM}{r^2}\frac{\mathbf{r}}{r}, \tag{11}$$

and then one should solve the differential equation

$$\frac{d}{dt}\frac{d\mathbf{r}}{dt} = -\frac{GM}{r^2}\frac{\mathbf{r}}{r}. \tag{12}$$

Of course, this goes beyond our aims here. However we are going to use this relation as a basis for the numerical approach. Let' see the listing.

```
# Halley-RK-4-AU-90-sharable.py

# Copyright 2020 Andreas Robert Formiconi (arf@unifi.it)

# Program distributed under the terms of the GNU
# General Public License

# This program is Free Software: it can be redistributed
# and modified under the terms of the GNU General Public
# License published by Free Software Foundation, in version 3
# or one of the following ones. The text of the license is
# accessible in
# <https://www.gnu.org/licenses/licenses.en.html>.

# Calculation of the orbit of a celestial body around the sun
# by  numeric integration of the motion equations by Newton
# gravitation law. The problem is posed in two dimensions and
# assumes that there are no perturbations from other bodies.
# The code is adjusted to solve the case of a highly eccentric
# orbit like that of Halley's comet.
```

```
21
22 # The Turtle plays the role of the comet, The planet is at the
23 # center, which in TigerJython has coordinates (0,0), where
24 # the unit of measurement is the "point" (p).
25 # Warning: if you run this code as it is, be patient: the
26 # trajectory takes a minute or so to appear, on the right,
27 # since a lot of points have to be calculated, in order to
28 # achieve reasonable accuracy...
29
30 from gturtle import *
31
32 makeTurtle()
33 clearScreen()
34 showTurtle()
35
36 # Global variables, i.e., variables that are "visible" in both
37 # the main program and within each subprogram, such as
38 # NEWTON, STP and so on.
39
40 # global GG, Dt, DX, DY, XPOS0, YPOS0, XPOS, YPOS,
41 # XVEL, YVEL, XACC, YACC
42
43 # In the following, subprograms in which specific functions
44 # have been encapsulated: NEWTON calculates the acceleration
45 # at a given point, STP evaluates the next step, WRITEPOINT
46 # plots the point as long as they are calculated
47
48 # ************************************************
49 # Subprogram NEWTON: calculates the acceleration at point of
50 # coordinates X, Y returning two acceleration components
51 # ACCX and ACCY
52
53 def NEWTON(X,Y):
54     global GG, Dt, DX, DY, XPOS0, YPOS0, XPOS, YPOS, XVEL, YVEL
       , XACC, YACC
55     DX = (X-XPOS0)
56     DY = (Y-YPOS0)
57     R2 = (DX**2 + DY**2)
58     R = sqrt(R2)
59     XACC = - GG / R2 * DX / R
60     YACC = - GG / R2 * DY / R
61
62
63 # ************************************************
64 # Subprogram STP: calculates the next step with the
65 # Runge-Kutta interpolation of the fourth order.  This
66 # interpolation represents a fairly sophisticated way to
67 # calculate the trajectory points it is needed to reduce
68 # the approximation errors inherent to the calculation of a
69 # continuous function in a discrete set of points. The
70 # algorithm is taken from W.H. Press et al, Numerical
71 # Recipes - The Art of Scientific Computing, Cambridge
72 # University Press, 1992, pp. 704-708.  The coordinates
73 # of the new position, XPOS and YPOS, and the speed at
74 # that point, XVEL and YVEL, are returned.
75
76 def STP():
77     global GG, Dt, DX, DY, XPOS0, YPOS0, XPOS, YPOS, XVEL, YVEL
       , XACC, YACC
78
```

```
79      NEWTON(XPOS, YPOS)
80      KX1 = Dt * XACC
81      XVEL1 = XVEL +KX1 / 2.
82      KY1 = Dt * YACC
83      YVEL1 = YVEL +KY1 / 2.
84      XPOST = XPOS +XVEL1 * Dt / 2.
85      YPOST = YPOS +YVEL1 * Dt / 2.
86
87      NEWTON(XPOST, YPOST)
88      KX2 = Dt * XACC
89      XVEL2 = XVEL +KX2
90      KY2 = Dt * YACC
91      YVEL2= YVEL +KY2
92      XPOST= XPOS +XVEL2 * Dt / 2.
93      YPOST= YPOS +YVEL2 * Dt / 2.
94
95      NEWTON(XPOST, YPOST)
96      KX3 = Dt * XACC
97      XVEL3 = XVEL +KX3
98      KY3 = Dt * YACC
99      YVEL3= YVEL +KY3
100     XPOST= XPOS +XVEL3 * Dt / 2.
101     YPOST= YPOS +YVEL3 * Dt / 2.
102
103     NEWTON(XPOST, YPOST)
104     KX4 = Dt * XACC
105     XVEL4 = XVEL + KX4
106     KY4 = Dt * YACC
107     YVEL4 = YVEL +KY4
108
109     XVEL= XVEL + (KX1 + 2 * KX2 + 2 * KX3 + KX4) / 6.
110     YVEL= YVEL + (KY1 + 2 * KY2 + 2 * KY3 + KY4) / 6.
111
112     XPOS= XPOS + (XVEL1 + 2 * XVEL2 + 2 * XVEL3 + XVEL4) * Dt /
         6.
113     YPOS= YPOS + (YVEL1 + 2 * YVEL2 + 2 * YVEL3 + YVEL4) * Dt /
         6.
114
115
116 # ************************************************
117 # Subprogram WRITEPOINT. It does two things:
118
119 # 1) it sends the Turtle to the next point of the trajectory
120 # by means of a POSITION instruction - that's how you are
121 # creating the drawing
122
123 # 2) it writes in a file all the relevant values of each point
124 # of the trajectory: two position, speed and acceleration
125 # components - that is the solution of the motion problem.
126 # These data can be used successively for creating graphical
127 # representations with other software or for further
128 # processing.
129
130 def WRITEPOINT():
131     global GG, Dt, DX, DY, XPOS0, YPOS0, XPOS, YPOS, XVEL, YVEL
        , XACC, YACC
132
133     moveTo(XPOS, YPOS)
134
135 # ************************************************
```

```
# ************************************************
# Main program

# First of all, the physical constants involved are calculated
# in the M.K.S. system (Meter, Kilogram, Second), with the
# exception of the distances of the orbits because, given the
# enormous values at stake, the AU (Astronomical Unit) is
# more handy, where 1 AU = 1.495978707 x 10^11 meters.
# One AU corresponds to the average distance between the
# earth and the sun. Thus, for example, when it is found that
# Halley's aphelion is about 35.08 AU, it means that the
# maximum distance of the comet from the sun is equal to
# about 35 times the distance between the earth and the sun.
# In the end, however, all distance measurements are
# transformed into "points" for the purposes of the graphic
# representation.

G = 6.67E-11             # (N*m^2/Kg^2)  Gravitation constant
Ms = 1.99E30             # (Kg) Mass of the sun

Dp = 200.0               # Aphelion expressed in points
rAf = 35.08              # Aphelion (AU)
Dt = 0.001               # Integration interval (time)

K = Dp/rAf               # Scaling factor: number of points/AU
GAU = G / 1.496E11**2    # Gravitation constant expressed in AU
Gp = GAU * K**2          # (N*p^2*Kg^2)
GG = Gp * Ms             # Gravitation constant inclusive of the
                         # solar mass (to reduce the number of
                         # multiplications in cycles)
eps = 0.967              # Halley's comet orbit eccentricity

clearScreen()

setFillColor("yellow")  # sun color
setPenColor("black")
dot(5)                   # Given the size of the comet's orbit
                         # the sun cannot be in scale

# These are the coordinates of the center of the page, which
# we assume to be at the origin of the reference system and
# that we make coincident with sun position.

XPOS0 = getX()         # origin coordinates (centre of page)
YPOS0 = getY()

  hideTurtle()           # better hiding the Turtle: too slow
    ...

# Determining initial conditions

# Let us place the comet at its initial position

setPos(XPOS0 + rAf*K, YPOS0)

XPOS = getX()
YPOS = YPOS0

# Initial velocity of the body, which is given by II Kepler law
    ,
```

```
194 # once the eccentricity and the solar mass are given
195
196 XVEL = 0.0
197 YVEL = sqrt(GG/(rAf*K)*(1-eps))
198
199 setPenColor("blue")
200 setPenWidth(1)
201
202 # Here begins the cycle for drawing the trajectory.
203 # The points of the trajectory are calculated via subprogram
204 # STP, then they are drawn by the WRITEPOINT subprogram,
205 # which takes also care of writing data (position, speed,
206 # acceleration) in a file. However, WRITEPOINT is invoked
207 # only once a while, since the trajectory is calculated in a
208 # very large number of points, in order to evaluate the path
209 # with reasonable accuracy and this number would be far to
210 # large for drawing purposes.
211 # In this implementation, which is adjusted to reproduce
212 # strongly eccentric orbits, like that of Halley's comet, the
      code
213 # uses only one point out of 10000 for drawing the path.
214 # This feature is controlled by means of the nWrite counter.
215 # A flag (yes/no variable), yIsNegative, is then used to
216 # check that the orbit will be drawn only once.
217
218 nWrite = 0
219 yIsNegative = False    # Flag one orbit only
220
221 while not ( yIsNegative and (YPOS-YPOS0) > 0):
222     nWrite = nWrite + 1
223     if not yIsNegative and (YPOS-YPOS0) < 0:
224         yIsNegative = True
225     STP()
226     if nWrite == 1:
227         WRITEPOINT()
228     if nWrite == 10000:
229         nWrite = 0
230
231
232 print("Done!")
```

Listing 35: Calculating the orbit of Halley's comet

Here you have the result, as plot on the TigerYjthon graphic output window. Of course, you could also print the point coordinates or save them on a file for further elaborations or other graphic representations. The point on the left is the sun.

Let's go on through the program listing. There are three subprograms: NEWTON (Inst. 53-60) for calculating acceleration, based on equation 10, STP (Instr. 76-113) for calculating position (XPOS, Ypos) and velocity (XVEL, YVEL) in the next point[19], WRITEPOINT moves the Turtle to the next point

[19]In this program we use a particular interpolation to reduce the approximation errors inherent to the calculation of a continuous function in a discrete set of points. We use here

drawing the path at the same time. Let's rewrite here the pseudo code of page 69 to show the role of the different subprograms:

Data: astronomical data about Halley's comet
Result: plot of the orbit
1 initialization;
2 **while** *orbit is not closed* **do**
3 | calculate acceleration in current point (NEWTON);
4 | given this acceleration value, calculate velocity (STP);
5 | given this velocity value, calculate position of next point (STP);
6 | move to next point (WRITEPOINT);
7 | plot the step (WRITEPOINT)
8 **end**

Algorithm 3: Calculating the orbit of Halley's comet

The main program begins at Inst. 137. Instructions 153-166 set the problem data: sun mass (or whatever), aphelion (largest distance from sun) and eccentricity of orbits, integration time interval. The initial conditions are set between instructions 184 and 197. The Turtle (comet) is started at the aphelion, at the far right in the figure, where the velocity vector is vertical and directed upwards and vertical[20]. As far as the velocity module is concerned, it can be determined by Kepler II law and ellipse properties:

$$v = \sqrt{\frac{GM}{r_a}(1-\epsilon)} \tag{13}$$

$$r_a = a(1+\epsilon) \tag{14}$$

where a is the length of the semi-major axis, ϵ is the eccentricity of ellipse and r_a is the aphelion.

Finally, the core of the main program between instructions 221 and 229. It's just a loop for calling STP anf WRITEPOINT, with some controls to stop once the orbit is closed.

By changing the appropriate astronomical data, this program can be used to simulate orbits for other two-bodies systems, provided the mass of the smaller one is negligible with respect to the other. It could be interesting to look for such data and experiment with the code.

the Runge-Kutta interpolation of the fourth order [Press et al., 2007, pagg. 704-708]. We do not need to enter into the details of this interpolation here.

[20]This holds for the TigerYjthon environment, where the Cartesian origin is in the center of the image space, x axis is positive towards right and y axis towards the top. In the case of LibreLogo, the same condition holds except for the y axis which is positive towards the bottom of the page.

8 Is there a place for randomness in computers?

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, is there 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.

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 random extractions, 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.

Thus, they appear as true random numbers. In LibreLogo the Turtle understands the RANDOM command:

- X = RANDOM 100 ; gives back a random float number[21] ($0 \leq X < 100$), 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

X = RANDOM [1, “pippo”, 3.14]

But randomness is also embedded in the special default variable ANY, for instance with

PENCOLOR ANY

you are going to use a random pen color.

Let’ take our original program POLY (pag. 41) again, and change it by choosing randomly direction of steps and turning angles, instead passing them as parameters:

[21] 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.

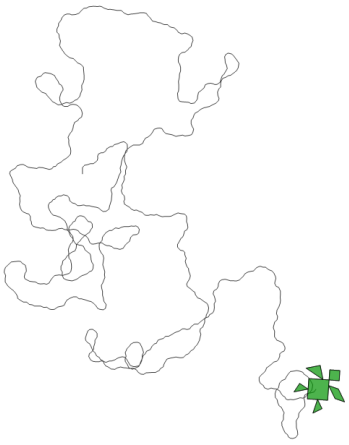
```
TO RAND
   REPEAT [
      FORWARD RANDOM(10)
      RIGHT RANDOM(360)
   ]
END
```

Listing 36: POLY in random version

The structure is the same of POLY, apart the absence of the parameters, but the behaviour is totally different! What's doing here the Turtle? Well, the Turtle is doing the turtle, even if a bit crazy one. It's a game, it's a simulation. You can play with it. In the previous code the steps are randomly chosen between 0 and 10, whereas the turning angles between 0°and 360°. These values make the Turtle's behaviour hectic, let's give it a tranquillizer...

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

Listing 37: RAND with different random choices



With FORWARD RANDOM(1) + 1 we are choosing step lengths between 1 and 2. With RIGHT RANDOM(90) - 45 we are restricting the deviations to angles comprised between -45°and 45°— after all, who have seen a turtle

turning by 180°all of a sudden?

You can try the same code but I could easily bet that your turtle didn't travelled exactly along the same pattern as mine. To tell the truth, the probability that you get the same picture is not exactly zero but it is extremely low: I'm pretty safe...[22].

Powerful concept — Randomness in science

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 window on the huge and crucial world of scientific simulation. Nowadays, in every scientific field the simulation is an essential investigation tool, the only one possible to face the overwhelming complexity of natural phenomena.

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*, commonly used in biology.

Nowadays *in silico* experiments include molecular biology, genetic assays, tumor growth, dermatology, bone remodelling, organ failure, clinical trials, just to mention a few. 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 analyse gene expression (In molecular biology "gene expression" means the way a set of genes determines the functioning of the cell at the macromolecular level).

Beyond the realm of simulations, the understanding of the role of randomness in nature is one of the most relevant achievement of science. Mathematicians have been building the edifice of statistics since the 18th century, the branch of mathematics needed to manage randomness. During the 20th century randomness bursts into several fields of physics, for instance in statistical mechanics to relate macroscopic properties of gases with their microscopical particle nature, in quantum mechanics to describe the behaviour of subatomic particles, in nonlinear dynamics to describe chaotic degeneration of deterministic systems. The overwhelming complexity peculiar of biological, atmospheric and social systems makes the dealing with randomness an everyday business in these fields. The blending of technologies with social life drives us in the area of data mining. Last but by no way least, computer scientists find in randomness a terrific tool for managing problems otherwise unsolvable [Hromkovič, 2009, pag. 201].

The philosopher and sociologist Edgar Morin describes human condition as living *in islands of certainty in an ocean of uncertainty* [Morin, 1977]. Generally, school curricula do very little to give a correct perspective of our contemporary knowledge of the world, where the uncertainty plays a

[22]Further considerations and examples can be found in lesson 7 of the MOOC *https* : *//federica.eu/l/the_turtle_does_the_turtle*

crucial role. Therefore, the scientific vision which is given is skewed towards a falsely deterministic description dominated by one-answer-only problems. Which is wrong.

9 Recursion and fractals

9.1 Recursion

We keep this topic schematic, just to give an idea and because playing with recursion and fractals means having to do with the idea of infinity, which is important to build a scientific culture, as we already pointed out. In the downloadable materials you will find more examples to play with and some further considerations can be found in chapter 8 of the MOOC[23]. Those willing to explore the fascinating world of fractal and chaos can browse the almost 1000 pages of the beautiful book "Chaos and Fractals" [Peitgen et al., 1992].

Everyone knows how two opposing mirrors generate an amazing fugue of images . Mirror number 1 can do only one thing: reproduce the scene in front of it. Even mirror number 2 can do only the same thing, but in doing so it also reproduces mirror number 1, including the scene it contains, which in turn reproduces the scene in mirror number 2 and so on, *ad libitum*. It is a phenomenon that strikes because it allows us to peek into the infinite, normally inaccessible to human experience. That is recursion.

```
1 TO RECURSION
2   RECURSION
3 END
4
5 RECURSION
```

Listing 38: Minimal recursion program

What's going on here? Nothing: program RECURSION keeps calling itself — a perfect representation of self-referentiality! In this code fragment the Turtle is executing just one command: RECURSION. Actually, if you try to write this code in LibreLogo and you run it, nothing will happen except, after a while, a message box will appear telling you "Program terminated: maximum recursion depth (1000) exceeded." This is a kind of safety measure, because recursive programs may put the computer in trouble, if a stopping criterium is not provided. Every time a procedure is called, the system will allocate some memory needed for its activities. If you let the machine go on with this process, an infinite quantity of memory will be claimed, which is not the case of your computer, of course.

Let's make this silly program a little bit more interesting.

[23] *https://federica.eu/l/recursion_growth_fractals* — to access lessons 2-10 you have to make an account: free, no advertising

```
TO RECURSION D
  CIRCLE D
  RECURSION D+1
END

RECURSION 1
```

Listing 39: Less minimal recursion program

This is more fun, try it: what do we get? You should see a balloon growing. And you'll need a stop rule, because this code will make the balloon go over the page: recursion programs need a stopping rule. For instance in this way:

```
TO RECURSION D
  IF D < 100 [
    CIRCLE D
    RECURSION D+1
]
END

CLEARSCREEN
RECURSION 1
PRINT ``Done!''
```

Listing 40: Recursion program with stopping rule

To create a stopping rule we need an instruction able to evaluate a given condition. In LibreLogo we can do that with the IF command together with a condition, which in this case could be "D $<$ 100". Instruction N. 2 says that if D is less than 100 then instructions 3 and 4, within the square brackets, will be executed, otherwise no. In this case, we get out from program RECURSION and instruction N. 10 is executed.

In TigerYjthon this program would look like that:

```
1  from gturtle import *
2
3  makeTurtle()
4  clearScreen()
5  hideTurtle()
6
7  def rec(d):
8      delay(100)
9      if(d < 100):
10         dot(d)
11         rec(d+1)
12
13 rec(1)
14 print("Done!")
```

Listing 41: Recursion Python program with stopping rule

Apart from syntactical differences, here we added a delay at instruction N. 8 because in TigerYjthon the program runs much faster and the growth of the balloon would not be visible.

So far, what could these simple and unhelpful examples be used for? Considering that they could also easily be made with simple loops? Why bother with these fanciful but strange constructions if loops can do the same? Actually, it's true, everything[24] that can be done with recursion can also be done with loops. However, there are problems that lend themselves easily to a recursive description, for instance *fractals*.

9.2 Fractals

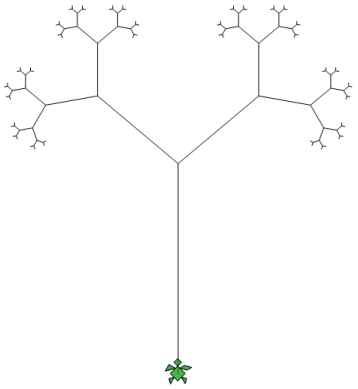
A fractal is a never-ending geometrical pattern, i.e. infinitely complex patterns that are self-similar across different scales. They became popular, even in mathematical circles, in the 1980s, thanks to the work of Benoît Mandelbrot. In fact, the basic idea of the fractal existed in mathematics since the 19th century. For instance, the Cantor set, a set of numbers (we give an example later on) which is fundamental in many branches of mathematics, was published by Georg Cantor in 1883 [Peitgen et al., 1992, pag. 67]. However, before Mandelbrot, Cantor set and other strange entities were regarded as exceptional objects, as "mathematical monsters". Mandelbrot merit was to have shown that these "extreme shapes", which now are called *fractals*, represents actually the normality, instead of the exception, showing that in Nature the are countless examples of fractal structures. Thus the title *The Fractal Geometry of Nature*.

Fractals look the same at different scales, that's what self-similarity means. We already encountered self-similarity, in the Bernoulli's, *spira mirabilis*, that is the logarithmic spiral (pag. 35 and following), and this is an over-represented shape in nature too. To get an idea about growing fractals, a very basic example may be appropriate, for instance the stick tree.

[24] The topic is not trivial, experts may discuss at length on this subject. In our context "everything" is ok.

```
TO TREE LL
  IF LL > 2 [
  FORWARD LL
  LEFT 50
  TREE LL/2
  RIGHT 100
  TREE LL/2
  LEFT 50
  BACK LL
  ]
END


TREE 200
```

Listing 42: Simple fractal tree in Logo



The TREE procedure is recursive because it calls itself, twice. The fact that the calls are two is related to the recurring bifurcated structures that represents the "basic idea" of this tree. To tell the truth, there are nothing else than bifurcations in this tree, which is simply made of bifurcations, the unique differentiation being the spatial scale. Let's go into some details. First, the program is called in instruction N. 14: TREE 200. Therefore, when TREE is entered, the value of LL is checked if it is less than 2 (Instr. N. 2). Since this time it's value is 200, execution goes ahead. With instruction N. 3 the turtle goes forward by LL=200 points and turns left by 50°, thus drawing the trunk and turning itself left. Then, instead of keeping drawing something it calls TREE (N. 5), but passing it a value of LL/2. Let us refrain from diving into this TREE call and assume to have it done. In N. 6 the turtle turns right by 100°, in N. 7 calls TREE LL/2 again, then in N. 8 turns left by 50°and comes back along the last branch. In the home page of the downloadable materials you can see the animation of the tree growth: $http://iamarf.ch/Codice-LOGO/$.

Another simple example is that of the Cantor set:

```
TO CANTOR(x, y, l)
    IF l > 0.1 [
        PENUP
        POSITION [x, y]
        PENDOWN
        HEADING 90
        FORWARD l
        y = y + 20
        CANTOR(x, y, l/3)
        CANTOR(x+l*2/3, y, l/3)
    ]
END

CANTOR 50 300 500
```

Listing 43: Cantor set in Logo

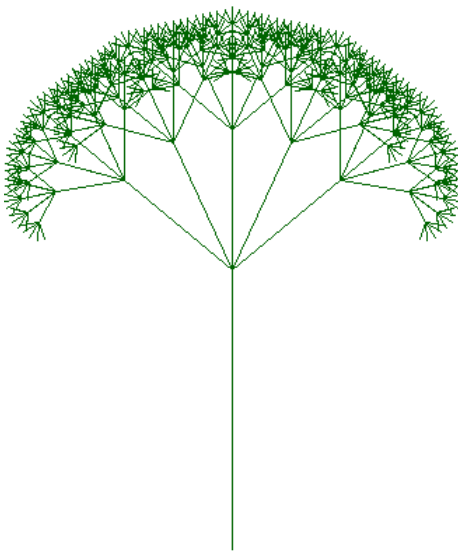
Figure 18: The representation of the Cantor set after 7 recursions

Here in INSTRUCTION n. 4 we used the special LibreLogo command POSITION [x, y], which sends the Turtle directly to the point of coordinates (x, y). The properties of the Cantor set have profound and complex implications but the construction is very simple: take an interval — the typical reasoning is made on the interval between 0 and 1 — divide it in three parts and cut away the middle part. Do the same on the two remaining segments, and do it again and again. It is impossible to produce an accurate representation because the size of the segments decrease rapidly and they become substantially invisible. However the process is very simple and we have no trouble imagining it going on *ad libitum* but the result is quite complex. Going on in the process it seems that a lot of void is produced, as if the segments will disappear. However, it is also obvious that while the segments become smaller and smaller they also multiply infinitely. Thus, apparently, we tend towards an infinity of isolated points. Well, these are not mathematical rigorous considerations but thinking on this line help stretching the mind.

This sort of balancing between infinity and infinitesimal can be revisited with the previous stick tree. Let's rewrite the code in slightly more complex way. We use here a Python version because the program runs faster[25].

[25] We're actually jumping between the Logo and Python programs. It's a good exercise, try translating some small programs from one language to another yourself.

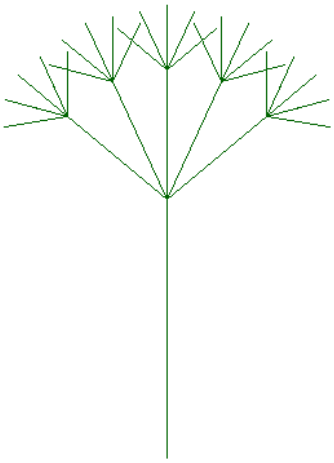
```
from gturtle import *
from math import *

makeTurtle()
clearScreen()
hideTurtle()

def tree(ll,aa):
    if(ll > 10):
        forward(ll)
        left(aa)
        tree(ll/2,aa)
        right(aa/2)
        tree(ll/2,aa)
        right(aa/2)
        tree(ll/2,aa)
        right(aa/2)
        tree(ll/2,aa)
        right(aa/2)
        tree(ll/2,aa)
        left(aa)
        back(ll)

setPenColor("dark green")
setPos(0,-100)
ll = 200 # First branch length
aa = 50 # Deviation angle

tree(ll,aa)
```



Listing 44: Stick tree with more branches.

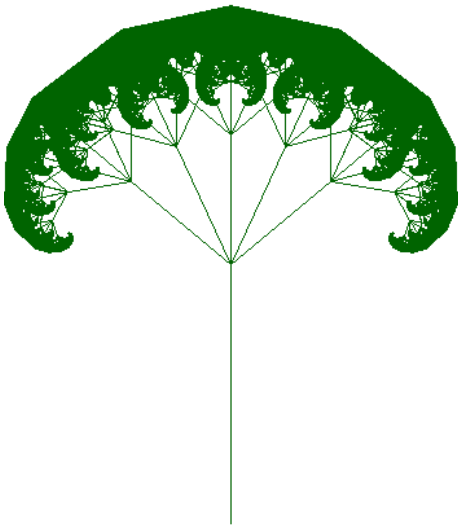
Two things have changed since the previous Logo version: 1) we have added the "aa" (angle) parameter to the "tree" subprogram and 2) rather than generating two branches at each recursion level, five branches are now generated. Instead we mantained the scale factor 1/2, i.e. at each recursion the tree suprogram is called with the branch length "'ll" divided by two. The result is nicer, it looks like a dandelion, or a pine tree. But apart from that, this version allows another interesting reflection about infinity and infinitesimal, similar to the one we did for the Cantor set. In subprogram "tree" we have inserted a stop rule which requires that the length of the branches should not be less than 10. Since we started with ll =200, we have obtained four recursion levels with values ll = 100, 50, 25 and 12.5.

It is obvious that if we had set the threshold at a higher level, we would have obtained a tree with less branches but also smaller, such as this one, where we set the threshold at 30 obtaining only two additional levels: ll = 100, 50. On the other hand, by reducing the threshold, allowing the creation of smaller branches, the tree will become more intricate but also higher. But what will happen if we allow the recursion process to continue indefinitely? By adding more and more branches will the tree become infinitely tall, or will the progressive reduction in branch size compensate for the explosion of the tree? Which one of these two opposite tendencies will prevail?



You can try by yourself, it's very easy: you have just to change the stopping rule at instruction N. 9. For instance, we try now with "if(ll >' 0.1):'

```
1  from gturtle import *
2  from math import *
3
4  makeTurtle()
5  clearScreen()
6  hideTurtle()
7
8  def tree(ll,aa):
9      if(ll > 0.1):
10         forward(ll)
11         left(aa)
12         tree(ll/2,aa)
13         right(aa/2)
14         tree(ll/2,aa)
15         right(aa/2)
16         tree(ll/2,aa)
17         right(aa/2)
18         tree(ll/2,aa)
19         right(aa/2)
20         tree(ll/2,aa)
21         left(aa)
22         back(ll)
23
24 setPenColor("dark green")
25 setPos(0,-100)
26 ll = 200 # First branch length
27 aa = 50 # Deviation angle
28
29 tree(ll,aa)
```



Listing 45: Stick tree with more branches and deeper recursion.

In this case the recursion levels are ten: ll = 100, 50, 25, 12.5, 6.25, 3.125,

1.56, 0.78, 0.39 and 0.19. The vegetation over there is thick, indeed! But is this tree really higher? Perhaps. Let's try to compare them by placing a 400 points ruler at the center of each tree.

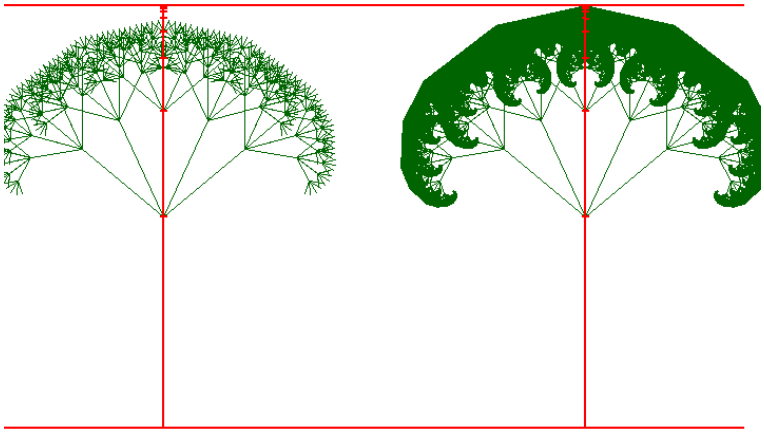


Figure 19: Transition from four to ten recursion level. Distance between the two horizontal bars is of 200 pixels. On the vertical bars, tick marks beginning of successive branches, proportional to 1/2, 1/4, 1/8...

Yes, the tree continues to grow with the number of recursions, but at a progressively slower rate. Will this growth remain confined below a certain limit, when we push recursion forever? Computer experiments could be perfected, but only mathematics can be expected to provide a definitive answer. The structure of our tree is suited to get this answer from a very well known formula. At each recursion, five equispaced branches are generated and the central one has no deviation angle with respect to its originating branch. Thus, at the center we have all the successive branches aligned on a vertical line reaching the top of the tree, and this is the tree height. The successive branches are reduced by a factor 2, at every recursion. Thus, in the case of the first tree the length of the central vertical line is given by

$$H = 1/2d + 1/4d + 1/8d + 1/16d + 1/32d \tag{15}$$

where H is the total height and d is the length of the first branch, the trunk, $d = 200$ in our code. Mathematics give us the tools to manage the infinity. We cannot let our computer do infinite recursions but we can write:

$$H = \sum_{n=1}^{\infty} \left(\frac{1}{2}\right)^n d = d \sum_{n=1}^{\infty} \left(\frac{1}{2}\right)^n \tag{16}$$

This is a geometrical progression, for which

$$\sum_{n=1}^{\infty} a^n = \frac{1}{1-a} \tag{17}$$

holds true if $a < 1$. Therefore in our case we have

$$H = \frac{1}{1-\frac{1}{2}} = 2d \tag{18}$$

Well, it is this value we used to draw the red rulers. Now we know that the recursion process will make a dense tree crown but the total height will remain equal to $200 \times 2 = 400$ points. This is another example of an infinite process giving rise to a finite result.

Reflection — Fractals: infinity in finite → towards calculus

How much of the above considerations should be made with students depends on their age, attitudes and context of the class. What matters is taking the opportunity to play with the concept of infinity. The example of the tree that grows *ad libitum* while remaining limited at the same time, opens the way to the understanding of concepts such as *limits*, *derivatives*, *integral*, i.e. *calculus*.

Our tree is very simple, indeed, but it is possible to create shapes which are very realistic. You can explore some more complex fractal in chapter 8 of the MOOC we quoted at page 74. Among the downloadable materials you find the codes to produce the following examples.

Figure 20: Examples of fractal plants

9.3 Fractals and randomness

The previous natural-looking shapes were created by working on fractal growth patterns. However, we can use another ingredient to simulate natural shapes, which is randomness. Let's take our first simple tree, which is quite geometric, and try to inject some "life" into it.

```
1  TO TREE LL
2    A = RANDOM 50
3    IF LL > 2 [
4      FORWARD LL
5      LEFT A
6      TREE LL*3/5
7      RIGHT A*2
8      TREE LL*3/5
9      LEFT A
10     BACK LL
11   ]
12 END
13
14 TREE 50
```

Listing 46: A livelier tree

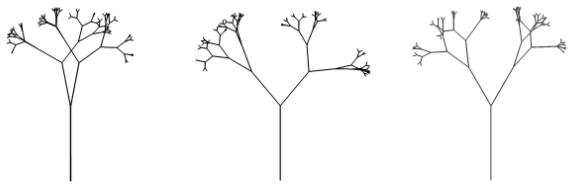


Figure 21: Three different executions of the previous code

The substantial difference with respect to the original code is in instruction N. 2, where the turning angle A is chosen as a random number between 0°and 50°.

We can try adding a random number between -2 and 2 to the step length - in this example 50 at the first iteration.

```
TO TREE LL
  A = RANDOM 50
  L = LL - 2 + RANDOM 4
  IF LL > 2 [
    FORWARD L
    LEFT A
    TREE L*3/5
    RIGHT A*2
    TREE L*3/5
    LEFT A
    BACK L
]
END

TREE 50
```

Listing 47: An even livelier tree

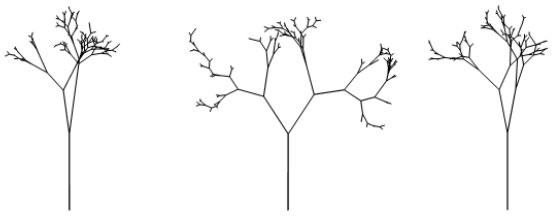


Figure 22: Three different executions of the previous code

It's fun to see what happens when you try to introduce a different amount of randomness into a fractal — a whole world of wonder and opportunity for reflection.

We like to close, again, remembering that Turtle geometry, even if very appropriate to introduce students of different ages to scientific thinking, plays also an important role in science. That's why we included a fascinating paper about computer imagery of plants [Prusinkiewicz et al., 1988] among the materials, where there are several amazing examples of simulated natural shapes. In section 3 of this paper it is told how these shapes can be created by manoeuvring a "LOGO-like" turtle in three dimensions, quoting Turtle Geometry [Abelson and diSessa, 1980].

Exactly the Turtle we have talked about in this short trip.

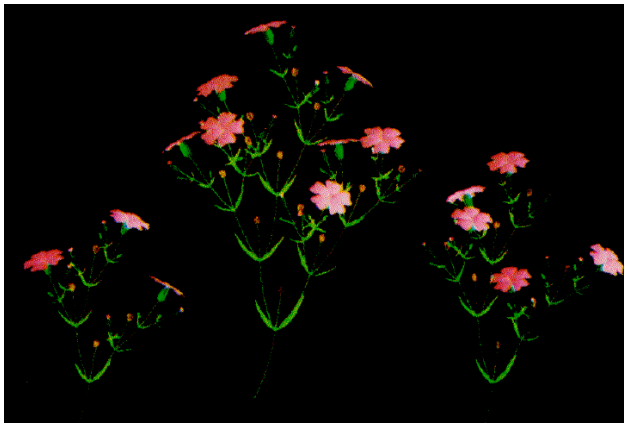


Figure 23: An example of simulated plants from Prusinkiewicz et al [Prusinkiewicz et al., 1988] paper

10 Appendix 1: Index of powerful ideas and relevant reflections

Basically, this is an index of the gray text boxes we dispersed in the article. They were intended to point out powerful mathematical ideas, concepts or approaches that are evoked by the exercises. Concepts that have not to be explicitly described to the students but that teachers should know, in order to understand the potential of the practices and their crucial points.

11 Appendix 2: How to manage graphics in Writer

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 graphics can then be managed like any other LibreOffice graphics. 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 [26] 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. They are different from *raster* or *bitmap*, that consist of a matrix of pixel[27]. 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

[26] 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.

[27] 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/

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.

References

[Abelson and diSessa, 1980] Abelson, H. and diSessa, A. (1980). *Turtle Geometry - The computer as a medium for exploring mathematics.* MIT Press, Cambridge, Massachusetts.

[Bruner, 1960] Bruner, J. S. (1960). *The process of education.* Harvard University Press, Cambridge.

[Hromkovič, 2009] Hromkovič, J. (2009). *Algorithmic Adventures - From knowledge to magic.* Springer, Heidelberg.

[Hromkovič, 2014] Hromkovič, J. (2014). *Einführung in die Programmierung mit LOGO: Lehrbuch für Unterricht und Selbstudium.* Springer Vieweg, Wiesbaden.

[Hromkovič and Kohn, 2018a] Hromkovič, J. and Kohn, T. (2018a). *Einfach Informatik - Programmieren - Sekundarstufe I.* Klett und Balmer, Baar.

[Hromkovič and Kohn, 2018b] Hromkovič, J. and Kohn, T. (2018b). *Einfach Informatik - Programmieren Begleitband - Sekundarstufe I.* Klett und Balmer, Baar.

[Morin, 1977] Morin, E. (1977). *Il metodo 1. La natura della natura.* Cortina Editore, Milano.

[Németh, 2014] Németh, L. (2014). Manuale di comandi di librelogo. `https://help.libreoffice.org/6.3/en-US/text/swriter/librelogo/LibreLogo.html?DbPAR=WRITER#bm1`. Accessed: 2019-12-09.

[Papert, 1993] Papert, S. (1993). *Mindstorms, Children, Computers, and Powerful Ideas.* Basic books, New York, 2 edition.

[Peitgen et al., 1992] Peitgen, H. O., Hartmut, J., and Saupe, D. (1992). *Chaos and Fractals.* Springer, Berlin.

[Persico, 1956] Persico, E. (1956). Che cos'è che non va? *Giornale di Fisica*, 1:64–67.

[Press et al., 2007] Press, W. H., Teukolsky, S. A., Vetterling, W. T., and Flannery, B. P. (2007). *Numerical Recipes – The Art of Scientific Computing.* Cambridge University Press, Cambridge, Masssachusetts.

[Prusinkiewicz et al., 1988] Prusinkiewicz, P., Lindenmayer, A., and Hanan, j. (1988). Developmental models of herbaceous plants for computer imagery purposes. *Computer Graphics*, 22:141–150.

[Sherin, 2001] Sherin, B. L. (2001). A comparison of programming languages and algebraic notation as expressive languages for physics. *Int. Journal of Computers for Mathematical Learning*, 6:1–61.