Notes and references

You will find here comments and readings for each activity.

General resources

- The official `Python` documentation contains tutorials and explanations of each function.
  docs.python.org/3/
- *Wikipedia* is a reliable source for learning more about some concepts (mainly projects) but the level is not always suitable for a high school student.
- *Internet* and in particular the forums often have answers to questions you ask yourself!
- The most experienced among you can participate to the *Euler project* which offers a list of mathematics-with-computer puzzles.
  projecteuler.net

1. Hello world!

Learning a programming language can be very difficult. It's hard enough to learn alone and it is not uncommon to stay blocked for several hours for a stupid syntax error. You have to start small, don't hesitate to copy code written by others, be persistent and quickly ask for help!

2. Turtle (Scratch with Python)

The ideal is to master *Scratch* before attacking `Python`. `Python` offers a *Turtle* module that works on the same principle as *Scratch*. Of course instead of moving blocks, you have to write the code! It is a good exercise to transcribe into `Python` all the activities you can do with *Scratch*.

3. If ... then ...

We really start the programming with the "if/else" test. The computer therefore acts in one way or another depending on the situation. It is no longer an automaton that always does the same thing. In addition to the keyboard entry, we can start having interactive programs.
The classic syntax errors are:

- forget the colon after `if condition:` or `else:`,
- incorrectly indent the blocks.

These errors will be reported by `Python` with the line number where there is a problem (normally your editor places the cursor on the wrong line). On the other hand, if the program starts but does not do the right thing, it is surely the condition that is incorrectly formulated. It is more complicated to understand the right condition: a little logic and reflection with paper and pencil are welcome. Some editors also allow a step-by-step execution of the `Python` program.
There is also the test

```
if ... elif ... elif ... else ...
```

which allows you to run the tests in sequence and which is not covered here. Understanding `if ... else ...` is enough.

4. Functions

Quite quickly it is necessary to understand the structure of a computer program: the program is broken down into blocks of definitions and simple actions. Simple actions are grouped into intermediate actions. And at the end the main program is just to perform some good functions.
The arguments/parameters of a function are a delicate learning process. You can define a function by `def func(x):` and call it by `func(y)`. The `x` corresponds to a dummy mathematical variable. In computing, we prefer to talk about the *scope* of the variable which can be local or global.
What we can remember is that nothing that happens within a function is accessible outside the function. You just have to use what the function returns. It is necessary to avoid the use of `global` in the first instance.
Last but not least, it is really important to comment extensively on your code and explain what your functions are doing. If you define a function `func(x)`, plan three lines of comments to (a) say what this function does, (b) say what input is expected (`x` must be an integer? a floating point number ? positive?...), (c) say what the function returns.
It is also necessary to comment on the main points of the code, give well-chosen names to the variables. You'll be happy to have a readable code when you come back to your code again later!
A good computer scientist should check that the parameter passed as an argument verifies the expected hypothesis. For example if `func(x)` is defined for an integer `x` and the user transmits a string, then a nice warning message should be sent to the user without causing the whole program to fail. The commands `assert`, `try/except` allow to manage this kind of problem. For our part, we will refrain from these checks assuming that the user/programmer uses the functions and variables in good intelligence.

5. Arithmetic – While loop – I

We will only use Python3. If you don't know which version you have, type 7/2: Python3 returns 3.5 while Python2 returns 3 (in version 2, Python considered that the division of two integers should return an integer).
With Python3 it is clearer, a/b is the usual division (real numbers) while a//b is the Euclidean division between two integers and returns the quotient.

6. Strings – Analysis of a text

Handling strings allows you to do fun activities and leave the mathematical world a little bit. You can code quizzes, programs that chat with the user... Strings are especially a good introduction to the notion of list, which is an essential tool later on.

7. Lists I

Python is particularly flexible and agile for using lists. "Old" languages often only allowed lists containing a single type of element, and to browse a list you always had to do this:

```
for i in range(len(mylist)):
    print(mylist[i])
```

While:

```
for element in mylist:
    print(element)
```

is much more natural.
Sorting a list is a fundamental operation in computer science. Would you imagine a dictionary containing 60 000 words, but not sorted in alphabetical order? Sorting a list is a difficult operation, there are many sorting algorithms. The bubble sort algorithm presented here is one of the simplest. Programming faster algorithms in high school is a great challenge: you have to understand recursivity and the notion of complexity.

8. Statistics – Data visualization

If all readers can program the calculations of sum, mean, standard deviation, median... and their visualization, then the objective of this book is achieved! This demonstrates a good understanding of basic mathematical and computer tools.
The tkinter module allows a graphic display. To begin with, you have to copy lines of code that works without asking yourself too many questions, then adapt it to your needs.

9. Files

The files make it possible to communicate `Python` with the outside world: for example, you can retrieve a file of grades to calculate the averages and produce a report card for each student.
The activities on the images are nice and these image formats will be used later, even if this format has the big disadvantage of producing large files. Nevertheless they are standard and recognized by image software (*Gimp* for example). Note that some software write files with only one data per line (or all data on a single line). It is a good exercise to implement the reading of all possible formats.

10. Arithmetic – While loop – II

Arithmetic in general and prime numbers in particular are very important in computer science. They are the foundation of modern cryptography and ensure the security of transactions on the Internet. The algorithms presented here are of course elementary. There are sophisticated techniques to say in a few seconds if a number of several hundred digits is prime or not. However, factoring a large integer remains a difficult problem. A computer shows its power when it handles a large quantity of numbers or very large numbers. With arithmetic, we have both at the same time!

11. Binary I

The first difficulty with binary notation is to make a good distinction between a number and writing the number. We are so used to decimal numeral system that we forgot its origin, 1234 is just $1 \times 1000 + 2 \times 100 + 3 \times 10 + 4 \times 1$. As computers work with 0 and 1 you have to be comfortable with binary numeral system. The transition to binary notation is not very difficult, but it is still better to make a few examples by hand before starting the programming.
Binary writing will be used for other problems on the following principle: you have 4 switches so 1.0.0.1 means that you activate the first and last switch and not the others.

12. Lists II

Lists are so useful that `Python` has a whole efficient syntax to manage them: list slicing and list comprehension. We could of course do without it (which is the case for most other languages) but it would be a pity. We will also need lists of lists and in particular arrays to display beautiful images.

13. Binary II

There are 10 kinds of people, those who understand binary and others!

14. Probabilities - Parrondo's paradox

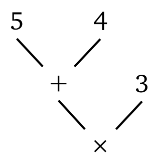
Here is a project with some probabilities and a nice paradox, it's surprising (like all paradoxes) and have been recently discovered. This activity is based on the article "Parrondo's paradox" by Hélène Davaux (*La gazette des mathématiciens*, 2017).

15. Find and replace

Searching and replacing are two actions so common that you don't always realize how strong they are. First of all, it is a good exercise to program the search and replacement operations yourself. There is a sophisticated version of these operations called regular expressions (*regex*). It is a language in itself, very powerful, but a little esoteric.
There is also the mathematics of "find/replace"! The proposed activities are examples that illustrate the main theorem of the article *A complete characterization of termination of* $0^p1^q \to 1^r0^s$, by H. Zantema and A. Geser (*AAECC*, 2000).

16. Polish calculator – Stacks

Polish notation (the exact name is "reverse Polish notation") is another way to write algebraic operations. Once again, the hardest part is to adapt your brain to this change in writing. This allows operations (and their priorities) to be seen in a new light. Another interesting vision is to see an operation in the form of a binary tree:



which represents $(5+4)\times 3$ or in Polish notation $5\ 4\ +\ 3\ \times$.
We tried to postpone as much as possible the change of a global variable in a function, but here this is natural. The use of `global` should be avoided in general.
The notion of stack is a very simple way to structure and access data. However, this is the right way to manage an expression with brackets. The analogy with the sorting station should be illuminating. We will encounter stacks again in the activity on L-systems.
A stack works on the principle of "first in, last out" (*filo*). Another possible data management is on the principle "first in, first out" (*fifo*) as in a queue.

17. Text viewer – Markdown

The purpose of this chapter is twofold: to discover *Markdown* which is a very practical language for formatting a text, but also to understand the justification of a paragraph.
Obviously the *Markdown* language has more tags than those presented here. We could continue the project by doing the justification with fonts of different sizes and shapes, or even create a complete small word processor.

18. L-systems

We come back to the theme “find/replace” but this time with a geometric vision. The figures obtained are nice, easy to program using the turtle, but the most beautiful is to see in live the layout from L-systems. For L-systems defined by expressions containing brackets, we find again the notion of stacks.
The formulas are taken from the book *The algorithmic beauty of plants*, by P. Prusinkiewicz and A. Lindenmayer (Springer-Verlag, 2004) with free access: *The algorithmic beauty of plants*.
The illustrations that begin each part of this book are iterations of the L-system called the Hilbert curve and defined by:

```
start = "X"  rule1 = ("X","lYArXAXrAYl")  rule2 = ("Y","rXAlYAYlAXr")
```

19. Dynamic images

This activity is based on the article “Blurred images, recovered images” by Jean-Paul Delahaye and Philippe Mathieu (*Pour la science*, 1997). This article also looks at the calculation of the number of iterations required to find the original image. The underlying mathematical notion is that of *permutation*: a one-to-one transformation of a finite set (here the set of pixels) into itself.

20. Game of life

A great classic of fun computing! There are dozens of sites on the Internet with configurations having incredible properties and many other ideas. But the most fascinating thing is that extremely simple rules lead to complex behaviors that resemble the life and death of cells.

21. Ramsey graphs and combinatorics

Graphs are very common objects in mathematics and computer science. The problem presented here is simple and fun. But what we should remember from this chapter is how the difficulty increases with the number of vertices. Here we do the calculations up to 6 vertices and we can’t go much further with our exhaustive verification method.
A great mathematician Paul Erdös stated that if aliens landed on Earth threatening to destroy our planet unless we could solve the problem of 5 friends/5 foreigners, then by mobilizing all the computers and mathematicians in the world we would manage to get by (we know that the answer is between 43 and 48 people). On the other hand, if the aliens asked us to solve the problem for 6 friends/6 foreigners, then the easiest thing would be to prepare for war!

22. Bitcoin

Before investing all your savings in bitcoins it is better to understand how it works! The activities presented here are intended to present a (very) simplified version of the blockchain that is the basis of this virtual currency. The principle of the blockchain and the proof of work are not so complicated.

23. Random blocks

These random constructions are first of all computer recreations that produce pretty figures, all similar but all different. But they are also the subject of modern and difficult mathematical work. Martin Hairer won the Fields Medal in 2014 for studying the upper boundary of our falling blocks, whose shape is governed by an equation, called “KPZ equation”. A good bonus activity would be to drop blocks of the game *Tetris* instead of small squares.