TOOL - PROGRAMMING EXPLAINED TO A YOUNG AUDIENCE

DIGITAL CONTENT CREATION > 3.4 PROGRAMMING

TARGET GROUP	AGE GROUP	PROFICIENCY LEVEL	FORMAT	COPYRIGHT	LANGUAGE
Facilitators	N/A	Level 1	Activity sheet	Creative Commons (BY-SA)	English, French

This document provides background information for the facilitator. It details basic notions on programming.

General Objective	Knowledge acquisition
Preparation time for facilitator	less than 1 hour
Competence area	3 - Digital content creation
Name of author	BSF Belgium
Resource originally created in	French

WORKSHOP DIRECTIONS

1 Introduction

This roadmap will give you simplified explanations on a series of ideas on programming. It aims to give the basics to a facilitator who may not be familiar with the subject.
You can explore these ideas with your group in different ways, for example :

- As a link between different activities (the workshops concerned will be linked below)
- With an audience already familiar with these ideas, via a discussion where notions are suggested by participants and informed further by you
- With an inexperienced group of over 12s, via a basic discussion using guided questions
- With an inexperienced group of under 12s, via a more facilitator-led explanation but in which interactivity is still maintained by asking them for examples and leaving them space to ask questions

2 Are computers intelligent?

Activities such as Robot game or Scratch Jr help to understand an essential idea: computers do not **think** . They don't reflect. They are not intelligent. They can't speculate or guess. They cannot understand. They only do one thing: **execute** instructions that we give them. If we give them wrong instructions, they will follow them anyway.

What is then is their advantage ? Answer: in the **complexity** of what they can achieve (make very enormously difficult and complex calculations), in the **speed** of execution (in 2020, a simple office computer can make more than 10 billion operations per second!), and that they never get tired (unlike humans).

Since they don't think, we need to think for them, for example by finding solutions by sorting elements efficiently. Next, we give them instructions that will tell them what to do, and they will do it rapidly and without thinking.

It is therefore important that the instructions we give them are very clear, precise and error-free. If not, we will encounter **bugs**, which is what happens when the result obtained is not the one expected, meaning there is an error in the **program**.

3 Did someone say program?

A **program** is a collection of instructions to be followed by the computer systematically, one after the other, until it reaches the end. By doing this, it will complete a task or solve a problem.

The task can be of many natures: opening a website, playing music, sending a document, reacting to a mouse click, making a character move in a video game, searching for a word in a text, sorting data, calculating the trajectory of a spaceship, etc.

The person who writes these instructions is called a **programmer** or **developer**. These instructions are written using one or several programming languages.

4 What is a programming language?

Computers cannot understand human language. They understand only **binary** (a way of presenting information in 1s and 0s using electricity). Humans can ‘speak’ binary only with difficulty and impracticality. We have therefore invented **programming languages**, easily manipulated by humans, which are translated to binary to be understood by computers.

A programming language is therefore a vehicle for **linking** between the programmer (person who writes programs) and the computer. Through a programming language, orders – **instructions** – can be given to the computer to tell it what to do (play a video, open a website, etc.) and how to react (to a mouse click, to a keypress, etc.).

Human languages have evolved through practice, i.e. in the creation of new words, the disappearance of obsolete words, the mixing of languages, etc. As for programming languages, they have evolved through technological **progress**. Different IT languages have **different** results: managing internet sites, making smartphone apps, controlling robots, etc. In general, programming languages use English words to

express simple precise things. The closer a programming language is to binary, the more it is known as ‘ **low level**‘.

The further away from binary a language is on the other hand, the more it is known as ‘**high level**‘. In Scratch, we can choose the human language in which we will program, therefore we use high level language when using Scratch.

5 More information on programming languages

Human languages are grouped by ‘families’ : Germanic languages, Romance languages, etc. This is depending on their origins and their common characteristics. Programming languages also have ‘families’ based on **paradigms** : their methodologies allowing them to program using different styles. Different programming languages can represent things in various ways and we use them differently, somewhat like human languages.

With German we use declensions while in French we don’t; in Dutch verbs are often put the end of the sentence, etc. For example, in the **block** programming language family, instead of writing instructions word by word, we use predetermined block that we assembly end to end. This is used by Scratch and Blockly, for example.

This paradigm was invented to make learning programming easier and is mostly designed for children or beginners. Using the **object oriented** programming language family, we can represent objects and concepts which can then be manipulated. Java, Python and C++ are examples of these. This paradigm facilitates the representation of processes and is used in many areas, from video games to business management tools.

In the **web** programming language family, we find a range of very different languages which are used to create and operate websites. The most basic, HTML, works using tags which describe where things should be located (text, images, frames, links, etc.) on a webpage. Every site uses it! Often, developers add CSS, which allows more complex formatting (colours, fonts, etc.).

These days, nearly every site adds Javascript too, also object oriented language, which means that things can be manipulated (images can be moved, videos and animations can be integrated, etc.) to add functionality to sites and make them feel more dynamic.

6 What are algorithms then ?

The majority of activities aiming to teach programming start by explaining the concept of an algorithm. This is for good reason: programs are collections of algorithms, sometimes very complex. But not all algorithms are programs…

Actually, an **algorithm** is a series of ordered instructions leading to the completion of a task. They come in many forms, the most common being cooking recipes. A recipe establishes a series of parameters (ingredients), relates a series of steps (cut, knead, cook, etc.) and leads to the completion of a task (preparing a dish).

IT programs are just one of writing, transmitting and using algorithms. These are used to perform highly varied tasks such as sorting, searching, sending data and many more.

And the **code** ? Coding is in a certain way storing information (for example, the text you are currently reading is coded in English). When we communicate using a method known only by a small number of people, this can be known as a secret code.

In IT, we are constantly storing, manipulating and decoding information, including instructions in programming languages. This is why we often call programs ‘code’ and programmers ‘coders’.

7 Some further basic ideas in programming

There are a number of concepts that are common to practically every programming language which you will discover as you complete the Scratch tutorials.

These are **control structures** : elements of code that mean that the feeding of instructions to the computer can be controlled and nuanced.

Conditions

Very often, our actions depend on several parameters. If the weather is good, you might not take a jumper when you go out. Verifying a condition (‘if…then…’) means that only a part of the code will be executed when a certain condition or conditions are respected. It is also possible to introduce alternative code which would be executed when certain conditions are not used (‘si…then…else [i.e. otherwise]).

Loops

When using Scratch or participating in certain offline workshops, you will often need the same instruction to be repeated several times. So as to not write the same code over and over , we can run a loop which will instruct that one part of the code be executed several times (‘repeat…’). The number of times the code will be executed can depend on a condition (‘repeat…until…’), be fixed (‘repeat x times’) or be infinite (‘repeat forever’) and more.