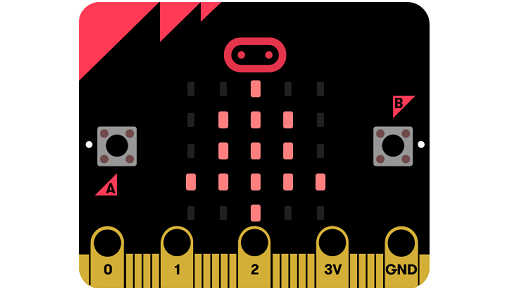
|  |
| --- |
| Macau Edinburgh Exchange Tour (MEET) 2017 |
| Introduction to programming and robotics with Microbit |
| Handbook |

|  |
| --- |
|  |



Contents

[Getting started 2](#_Toc486272589)

[Introduction to programming with Python 2](#_Toc486272590)

[Brief History of Computers 2](#_Toc486272591)

[Programming languages 2](#_Toc486272592)

[Python 4](#_Toc486272593)

[Introduction 4](#_Toc486272594)

[Basic concepts 4](#_Toc486272595)

[MicroPython 4](#_Toc486272596)

[Next Steps 4](#_Toc486272597)

[References 4](#_Toc486272598)

# Getting started

Welcome on board joining our journey in the world of Computer Science. In these couple of weeks we promise to do our very best to assist you and guide though challenges you will face. Do not worry if you have not done any programming before because everything will be though along the way. We encourage you to be curious in the upcoming weeks, ask questions for which we will try to find the best answers.

This handbook serves as:

* Supplementary learning material
* Future reference for ideas after the project ends

All the files related our project you can find at course website: ??????????

During our project, we will be using BBC Microbits – ARM-based micro-controller used for computer science education around the world. The safety guide can be found in appendix 1.

# Introduction to programming with Python

## Brief History of Computers

Before the 1920s, *computers* (sometimes *computors*) were human clerks that performed computations. They were usually under the lead of a physicist. Many thousands of computers were employed in commerce, government, and research establishments. Some performed astronomical calculations for calendars, others ballistic tables for the military.

After the 1920s, the expression *computing machine* referred to any machine that performed the work of a human computer, especially those in accordance with effective methods of the Church-Turing thesis. The thesis states that a mathematical method is effective if it could be set out as a list of instructions able to be followed by a human clerk with paper and pencil, for as long as necessary, and without ingenuity or insight.

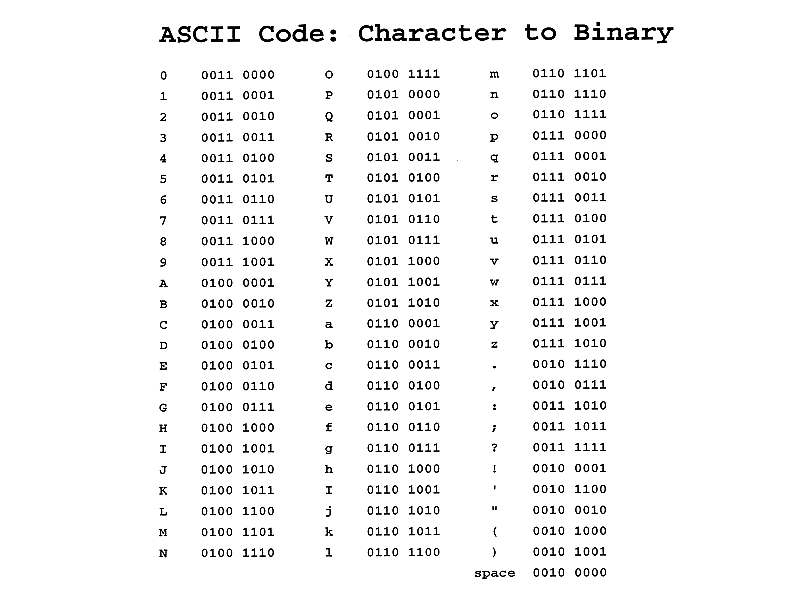
Machines that computed with continuous values became known as the *analog* kind. They used machinery that represented continuous numeric quantities, like the angle of a shaft rotation or difference in electrical potential. Digital machinery, in contrast to analog, could render a state of a numeric value and store each individual digit. Digital machinery used difference engines or relays before the invention of faster memory devices. The phrase *computing machine* gradually gave way, after the late 1940s, to just *computer* as the onset of electronic digital machinery became common. These computers could perform the calculations that were performed by the previous human clerks. Since the values stored by digital machines were not bound to physical properties like analog devices, a logical computer, based on digital equipment, was able to do anything that could be described "purely mechanical." The theoretical Turing Machine, created by Alan Turing, is a hypothetical device theorized to study the properties of such hardware.

## Programming languages

Whenever you give a task for the machine it does not have any clue what to do by itself, the only clue is you. During the time computers have become smarter and now have some understanding of your work but that is just because someone a long time ago said to him that this is the way our world works. For instance, if you ask your school calculator (which by itself is a computer too) to divide 5 by 0 you will get an error. It won’t try to calculate the result because someone (a programmer) told him that division by 0 is not a valid action in the world of mathematics and informed the machine what to do in situation like this.

When communicating with the computer we not only encounter the absence of its understanding but the problem of simple communication to tell it what to do. In our utopia kind of world, we could love for computers to understand human language (English, Cantonese, Swahili etc.). Unfortunately, this kind approach would have two mayor problems:

* **Human spoken languages are ambiguous**

Already in your lifetime you have encountered a situation when the same sentence said for two different people may be interpreted/understood differently: For instance, sentence “Alice saw the Bob with a telescope” could have two meanings:

* Alice saw Bob who was carrying a telescope
* Alice saw Bob using a telescope as a device to do so.

Both interpretations are valid (though first one seems more probable to be the real one from our English understanding). Computer would not know which one to choose in this case. Moreover, this is not the only sentence in English that would have such a behaviour so just listing what to do in each situation would be labour intensive. Besides that, there would be even greater problems to maintain machines understanding of English because it is an actively spoken human language for which the number of valid words and grammar expressions change constantly. To cope with these changes machine would have actively increase its English understanding along the way, but as mentioned before a machine cannot do it by itself, it needs human intervention.

Figure 1 ASCII character represented with 8 bits

* **Computers by itself cannot understand these languages.**

Since the invention of a first computer, every computer ever build uses binary as its main/native communication language. Binary is a special language represented by two symbols 0 and 1. This language is favourable by the computers because it can easily understand it. Computers understand high voltage in the chip as 1 and low voltage as 0. From these two symbols (similarly to Morse code) more complicated stuff can be built for instance Figure 1. Show how using 8 bits (2 bytes) you can represent all English language characters using ASCII system.

Using these symbols every machine has its own instruction set for which each member is represented with 32 bits (64 bit for newer computers). These instructions say for a computer when executing one or other kind of encrypted program. Instruction describes simple stuff like adding two numbers or storing the result in the memory but by just using these you can already do quite a complicated stuff.

Unfortunately, the biggest drawback of a machine language that it cannot be understood by humans you can easily check what an instruction 01001011010110010010011110010101 does but writing them by itself would be a tedious stuff. For this reason, an assembly languages were introduced. Assembly language is a language which has a one-to-one correspondence with a machine language. The main difference between it and machine language is that it encodes the meaning of the instruction which cannot be seen in machine language. The translation from an assembly language to machine language is done by a special program called assembler in a process of assembling (program is making other programs, is it not cool?)

Assembly language was a short-term solution. It was realized that just to produce a simple task like taking two numbers from the user input adding them up and printing the result takes a bunch of assembly instructions, hence the new level of abstraction was introduced to encapsulate the common stuff done by everyone. This led to the development of High-level programming language such as C, FORTRAN, and Pascal which are still used for today. The translation from high level language such as C to assembly language such as MIPS is done by yet another program called compiler in the process called compiler (seems familiar?)

Of course, the abstractions and solutions given by one language sometime did not offered the solutions needed for everyone but more specialized and convenient programming language are created every day. Currently there are thousands of programming language suited more for one or other purpose and given different kind of functionality with its own trade-offs. Currently the most popular and widely used programming languages in the world are Java, C++, and Python which we are going to explore in our project!

## Python

### Introduction

Python is a powerful high-level, object-oriented programming language created by Guido van Rossum. It has simple easy-to-use syntax, making it the perfect language for someone trying to learn computer programming for the first time. It is a general-purpose language. It has wide range of applications from Web development (like: Django and Bottle), scientific and mathematical computing (Orange, SymPy, NumPy) to desktop graphical user Interfaces (Pygame, Panda3D). The syntax of the language is clean and length of the code is relatively short. It's fun to work in Python because it allows you to think about the problem rather than focusing on the syntax. At the same time, it is a very powerful language. One of the most notable things created using Python is TensorFlow library produced by Google™ which is currently the most widely used machine learning platform in the world which power things like google translate and google self-driving car technology.

To write Python code on any computer and to compile it you need to install Python on your device. You can do it by going to [www.python.org](http://www.python.org) and following instructions for a specific operating system you are using. After installing Python on your device, you will need a text editor in which you will be writing your code to compile. The one that we will be using is Mu which is designed for programming with “micro:bit” . You can download it from <codewith.mu>. In the real-world applications other, more sophisticated text editors such as ATOM or even IDE’s (integrated development environments) like PyCharm or Visual Studio should be used.

If you will have any problems installing Python on your device just let tutors know and we will help you out.

In the appendix 1 you can a getting starting guide which explains how to connect “micro:bit” to your computer.

### Basic concepts

All the material related basic concepts in Python with examples and references can be found in <askpython.com>

### MicroPython

For a learning material about MicroPython we will be using an official MicroPython tutorial for “micro:bit” found in <http://microbit-micropython.readthedocs.io>

### Next Steps

There are plenty resources to expand your understanding and skills in Python. Here we give an example of the view most popular resources to do so:

* <docs.python.org> official Python tutorials created by Python Software Foundation
* Learn Python the Hard Way: A very Simple Introduction to the Terrifyingly Beautiful World of Computer Code by Zed A. Shaw (free .pdf available online)
* Python programming tutorial in the single video by Derek Banas available at YouTube.

# Introduction to Robotics with “Bit:Bot”