Preface

Introduction

Computation is everywhere, but what is computation really? In this lecture we will discuss the power and limitations of computation.

Understanding computation lies at the heart of many exciting scientific and social developments. Computational thinking is more than programming a computer – rather it is thinking in abstractions. Consequently, computational thinking has become a fundamental skill for everyone, not just computer scientists. For example, designing an electronic circuit relates directly to computation. Mathematical functions which can easily be computed but not inverted are at the heart of understanding data security and privacy. Machine learning on the other hand has given us fascinating new tools to teach machines how to learn function parameters. Thanks to clever heuristics, machines now appear to be capable of solving complex cognitive tasks. In this class, we study all these and more problems together with the fundamental theory of computation.

While computation is predominantly an engineering discipline, some of our insights are going to be philosophical. One may even claim that this class is going to discuss all major connections between computation and philosophy.

The weekly lectures will be based on blackboard discussions and programming demos, supported by a script and coding examples. The course uses Python as a programming language. Python is popular and intuitive, a programming language that looks and feels like human instructions.

Python

Python is a general-purpose programming language. This course assumes that students already are familiar with any procedural programming language such as C, C++, Java, JavaScript or Rust. For students familiar with programming, learning Python should be a piece of cake. Python does have a few quirks though:

- Python has little notational clutter such as semicolons or curly brackets: Instead, program flow is directly determined by indentation.
- Python numbers can be arbitrarily big: Python automatically stores a large number in as many bytes as needed.
- In Python, there is no need to declare data types: In principle, the same variable can be assigned a string and later in the code to a number, array

or boolean value.

- Python does not use explicit pointers to data: Python decides automatically whether a variable should be referenced by value or by name.
- Likewise, copying data structures is a bit tricky in Python: Copying arrays (or more complex data structures) require the copy (or deepcopy) function.
- Python has automatic garbage collection, so manual memory management is not necessary.
- Python code is usually not compiled, but executed ("interpreted") directly. This is one of the reasons why Python is slower than C/C++.

The two pages of the Python Cheat Sheet should provide all the necessary information to read and write Python programs. On top of that, there is a ton of information about Python on the web, in the form of introductions, text, videos, examples, tutorials, etc.

Have fun!