Module	**Programming, Algorithms and Data-Structures**
Code	V5_1
Degree Program	Master of Science in Life Sciences (MSLS)
ECTS Credits	5
Workload	150h: 45h Lecture (3 Lessons/W), 30h Exercises (2 Lessons/W), 75h Self-study
Module Coordinators	**Name** Dr. Manuel Gil **Phone** +41 (0)58 934 57 44 **Email** manuel.gil@zhaw.ch **Name** Dr. Stefan Höck **Phone** +41 (0)58 934 55 74 **Email** stefan.hoeck@zhaw.ch **Address** ZHAW Zürcher Hochschule für Angewandte Wissenschaften Life Sciences and Facility Management Einsiedlerstrasse 31 CH-8820 Wädenswil
Lecturers	• Dr. Manuel Gil • Dr. Stefan Höck
Entry Requirements	• Basic knowledge in arithmetic • Basic handling of a computer • No prior experience in programming is required.
Learning Outcomes and Competences	• Understanding the concept of computation and computational problems • Definition of the model of computation and motivation of computational problems • Acquire modular and algorithmic thinking to classify and solve such problems • Knowledge of basic algorithmic paradigms and corresponding data-structures • Basic navigation in an Unix environment • Ability to create small (to medium-size) programs • Syntax of the prominent programming language *Python* • Core knowledge to self-teach new languages
Module Content	The content of the module is conceptually divided in two parts: theory and practice. The theoretical part includes the sections *Computational Model* and *Algorithms and Data-structures*, whereas the practical part covers *Unix Environment* and *Programming*. The actual topics in the four sections, however, have crosslinks and dependencies. Thus, the topics will be built-up in combination and according to their connections.

	• *Computational Model*: A certain theoretical background greatly facilitates learning to program and is a pre-requisite for good programming. Such a background is introduced in this section.
	  ○ *Algorithm: G*eneral concept of procedures or formulas for solving a problem
	  ○ *Boolean algebra:* Whereas in elementary algebra expressions denote mainly numbers, here they denote the truth-values *false* and *true*. The main operations of Boolean algebra are the conjunction (*and-operation*), the disjunction (*or-operation)*, and the negation (*not-operation).*
	  ○ *Programming structures:* The *flow of a program* is controlled through *loops*, *conditional statements*, and *modules* (logical units for sub-tasks), which in turn rely on Boolean algebra.
	  ○ *Computational complexity:* Different Algorithms can solve the same problem with different efficiencies (concerning time and memory used). Based on simple examples, we learn to quantify the efficiencies.
	  ○ *Floating point arithmetic* and *numerics:* Natural numbers, like the numbers 0 or “18 billion billions” can be represented in a computer. Real numbers that have an infinite number of digits (like the square-root of 2, or Pi) cannot be represented exactly. They have to be approximated and one has to control the rounding errors introduced.
	• *Algorithms* and *Data-structures:* In this section we learn about fundamental problems (and their solutions) encountered when data is processed.
	  ○ *Sorting, searching:* When we have a set objects, for example of numbers or names, we would like to bring them in an order and be able to search them efficiently.
	  ○ *Data-structures (strings, lists, tuples, dictionaries trees, graphs):* Objects (or data) can be organised (or structured) in a linear order, or in hierarchical way, or be associated to each other according to other criteria. Such an organisation is called a *Data-structure.*
	  ○ *Recursion* and *Dynamic programming:* Recursion is one of the central ideas of computer science. It is a method where the solution to a problem depends on solutions to smaller instances of the same problem. During recursion, there may exist cases where same sub-problems are solved multiple times. Dynamic programming is a memorisation technique to store the results of already solved sub-problems.
	• *Unix environment:* Unix is a family of operating systems (like the Microsoft Windows family). Apple’s OS X, for instance, is Unix based. From a programmer's perspective, Unix is characterised by a modular design. It provides a set of simple tools (or commands), each performing a limited, well-defined function. The tools can be combined to perform more complex tasks. In this section we learn to use Unix.
	  ○ *Basic shell navigation:* The shell is a user-interface to direct commands to Unix. We will learn to move, copy, delete, edit, search, archive files; to make directories; to start, end, pause programs and to control priorities; and more.
	  ○ *File manipulation:* Make directories, move, copy, delete, edit, search and archive files.
	  ○ *Process management:* Start, end, pause programs.
	• *Programming:* Building on the theoretical knowledge we will learn to write actual programs.

	○ *Pseudo-code:* Before an algorithm is written in a programming language, it can be designed and formulated in a language intended to be read by a human rather than a machine. Such languages have similar structural conventions to programming languages but offer more freedom. ○ *How to comment code:* In computer programming, a comment is a human-readable annotation in the source code of a computer program. They are added with the purpose of making the source code easier to understand. ○ *Memory and its management:* There are different regions of a computer's memory (referred to as *registers*, *heap,* and *stack).* ○ *Data types:* All complex data in a program has to be built from basic *data types* like, for example, numbers, boolean (i.e. true and false), characters, arrays (or vectors). ○ *Syntax of Python:* Python is the programming language used in this course. The syntax is a set of rules that defines how a Python program will be written and interpreted. ○ *Procedural programming* is a programming paradigm, which we will focus on in this course. A procedure is a sequence of computational steps, providing modularity. Such a procedure can be called by a name at any point in a program, including by other procedures. ○ O*bject orientation:* Procedures and data structures are intimately related. The object oriented paradigm makes this relation explicit. It bundles procedures and data structures together into objects. In the beginning, when we program procedurally, we will already be using object oriented features. Later in the course, we will define our own objects. • *Practical part:* Here we will write small to medium size programs in the programming language Python.
Teaching / Learning Methods	• Basic theoretical knowledge is acquired during the lectures. • The theoretical knowledge is applied and solidified in the practical exercises. • The exercises are tied to programming assignments, which are mostly carried out as self-study • The students are required to read a portion of the literature as self-study (tutorials, original papers, reference manuals). Depending on the number of students, discussions in seminar-style are conceivable.
Assessment of Learning Outcome	• Programming assignments during the semester (30%) • Written exam, if number of students is >15; Oral Exam otherwise. Exam counts 70%.
Bibliography	Pointers to literature will be provided on Moodle.
Language	English
Comments	–
Last Update	02.04.2025