# 1.

## 1.1 Presentation of the course

main issues:

What are the objectives of this course? How do you get them?

What to do?

Course objectives, parts, methods, evaluation and information and counseling channels.

What are you supposed to do?

Read the text, sign up for Ood.

Indicative Demand Assessment:

Easy.

Indicative work estimates:

Half an hour.

## Course in English

Although, the lectures and this tutorial will be in Finnish, you can take this course in English. All the assignments are in English, too. You can see the list of assignments in the left navigation pane by clicking Exercise results / Pistetiedot. In addition, there is an electronic text book in English available. See the links in the beginning of each chapter in this tutorial to see the weekly reading.

See also the discussion forum and join the conversations in there!

One more thing: remember to enroll the course in Oodi in order to be able to take the final exam. You can choose the language between Finnish (suomi) and Swedish (ruotsi). If you want the examination questions in English, please, select Finnish/suomi, and send an email to address <[cs-a1141@aalto.fi](mailto:cs-a1141%40aalto.fi)> and tell your student number.

**Registration for the course and exam**

WebOodi

Sign up for the course and exams in WebOodi. If you have not yet enrolled, do it right away!

Starting this autumn, enrollment for the course will be the same for the first exam of the course (11.12.2017).

In the course of enrollment, it is possible to choose the language of the exam. The exam questions are only available in one language, that is, in the language in which they are asked when registering. We will notify you individually of the Swedish-language exam where you can take the exam in Swedish. Note, however, that the course material (eg terminology) is only in Finnish and English in the course material. Acquiring Swedish-language literature and terminology in Swedish remains the responsibility of the student.

The above applies only to the first exam in the course. Recruiters must register as before.

**Course Discussion Forum**

Piazza

Take a look at Piazza (see the link on the left navigation bar). Piazza is used as a forum for discussion about course related issues, rumors about team members, questioning tips on exercises, etc.

What is TRAK Y?

In this course, TRAK Y, the key objective is to utilize computer computing capacity and resources efficiently in computer programs.

The course is a precondition for many other courses in a computer science minor subject.

This material combines reading and exercises. With its tasks and links in an electronic textbook, the material covers the entire course content. Printed textbooks can also be used (ask for a suitable reading, for example, in the course newsgroup).

**Course Objectives**

Well said, the course objectives are as follows.

After the course you will be able to define, compare and implement basic data structures and algorithms, and name and select them for eg search structures, organization problems, and network scanning. In addition, you are able to identify and present a more detailed data structure or algorithm, and you can provide examples of their operation. You can also discuss other key data structures and algorithms using the industry's typical terminology.

The aim of the course is to learn how to solve key, often repetitive, computational problems utilizing well-known solution methods without re-inventing the wheel. In addition, we hope that such problem-solving is both fun and useful! It is also a matter of getting to know the programming languages ​​available in the information library libraries and to understand their soul life a bit deeper than the surface.

This course extends and deepens what you learned from the Basic Programming Programming on Y1 and opens up opportunities to participate in new courses that can be more focused on a specific topic.

For more information about the contents of the course, see the course brochure, which is worth a visit.

**Prerequisites**

Prerequisites for the course are CS-A1111 Programming Basic Course Y1 or one of the older courses such as T-106.1200 / 1203/1206/1207/1208/1210 or T.93.211 (Java). If the previous programming experience is in a language other than Python, contact the lecturer (cs-a1141@aalto.fi).

Even though the primary language of instruction in the course is Finnish, then some English language skills are also needed. Many examples and tasks are related to English-language documents. This reflects the fact that programming is an international sector and many related websites and tools are in English. Business software projects are also commonly used in English.

**Course Material and Returns: A +**

The course material and tasks of the course will be published in the A + system, which brings together all the available ancillary systems and materials. The links below can also be found in the A + navigation bar:

**Links**

Course in MyCourses

The course's electronic textbook

The news related to the course comes to Piazza (see the link on the left navigation bar). You can also ask questions about issues such as exercises.

Presemo is used as a lecture tool for lectures. Presemo can ask questions for a lecture in advance. In addition, the lecturer tunes Presemo's questions for a lecture that can be answered by a mobile device or a laptop during the lecture. It is intended to stimulate discussion with lectures and to interact with more traditional "talking heads".

**Partial achievements**

The course consists of two parts:

Exercises that are a versatile collection of tasks (including multiple choice tasks, algorithmic simulation tasks, programming and small project) with some degree of openness depending on the interest and the grade goal. Passing the course requires an approved grade for the exercises, ie at least one.

Exam. At the end of the course an exam is held, which must also be at least one.

Previously made parts

The part of the course does not go to the next course levels. Therefore, it is advisable to reserve sufficient time for the course to be done during one course version. Some of the exercises are the same as last year. If you did at least a grade of three in the fall 2016, you can contact the course lecturer (cs-a1141@aalto.fi), so the old grade can be passed for this year. Increasing the rating (and in any case raising the rating of less than 3) requires re-doing the tasks from the beginning. Exercises done before 2016 (partial exercises) are not accepted at this course.

**Exercise Groups**

Course hour attendants are on call in the training groups. Exercises are voluntary. Exercise groups do not need to register separately. The places and times of the training groups are

Microclass 3 (Maari-C), Maatintalo, Sähkömiehentie 3, Wednesdays 10-18, starting September 13

A140, Information Technology (T7), Konemiehentie 2, Thursdays 14-18, September 14

**Training tours**

Tasks are divided into timed turns.

Each round has a deadline (DL), by which time the rounds must be returned. The first round of DL is 15.09.2017 at midday. The timescales for the next rounds are usually weekly from the previous one. Less than half the points returned after the deadline.

Below is a table of points and deadlines available at different rounds.

Schedule of Exercise Tours

**Exercise ratings**

The final score for all the rounds is not yet known at the beginning of the course, so the score limits are as follows, expressed as percentages of the maximum points. The table should therefore be read so that grade 1 suffices to do almost all A tasks and slightly (10%) B tasks. If A misses too much, do the same with the B score (B tasks are interpreted as A until the required 90% is full). Similarly, the B tasks can be compensated by C tasks. Grade 4 can therefore get, even if you have not done all the A and B missions if you have done enough C tasks. Exact dropouts are reported when they are known.

Rating Level A Level B Level C

1 90% 10% 0%

2 100% 50% 0%

3 100% 100% 0%

4 100% 100% 25%

5 100% 100% 50%

Individual or couple work, not copying

Copying of assignment assignments from other students or elsewhere is not permitted. Likewise, it is forbidden to obtain points by trying to deceive the automated control system.

Discussing tasks and solving them together is allowed and even desirable. However, each student must return the tasks personally. If parsing is done in programming tasks, enter the name and student number of both authors in the source code.

Copying and other malpractice have consequences, and the programs written by students are also being studied in this regard. The course has automatic control to detect plagiarism. If the source code does not clearly indicate the origin of the borrowed code, it can be interpreted as plagiarism.

**Project**

The course has a voluntary project (the project gets C-points but can also be collected from other tasks). The project is done in small groups. The subject of the project and a more accurate schedule are announced in the second lecture.

**Lectures**

The course has individual lectures to discuss in particular the key themes of the course that are challenging by literature.

Tue 12.9.2016 is an opening lecture that introduces course practices, timetable, etc. It will also be treated as part of the first round. You can post questions before the lecture at Presemo.

See the lectures for more detailed times and places in MyCourses. Lectures can be hailed by e-mail or Presemo. At the end of period I will be held a separate project lecture, which will present topics, schedule, requirements,

The course's lectures are held at the Tietotekniikan talo halli T1 at 10-12.

**Staff**

Responsible teacher Ari Archie Korhonen

../\_images/archie.png

Archie is responsible for course planning, learning material, information, and lectures. He can be reached by lectures and by e-mail by agreeing to the Room of the Information Technology House A138.

Email: <cs-a1141@aalto.fi>

Tuntiassistentit

Timo Räsänen, Arttu Tilanterä and Saskia Kivistö. Hour attendants can be found in practice exercises.

**From email**

Do not ask for help with programming assignments by email. Instead, ask for advice in the training groups or in these electronic forums. The course has a lot of channels for programming questions, but e-mail does not belong to them.

**summarization**

The compulsory part of the course is: Exercise and Examination

The total value of the course is calculated as a weighted average of the grades and exam scores.

50% of the course value will be based on the exercise assignments.

50% of the grade comes from the exam.

Exercise groups, discussion forums, lectures, and weekly compilations support your assignments.

The first deadline is soon!

## 1.2 General instructions for exercises

**exercise**

The course exercises are a collection of different tasks

multiple-choice,

algorithm simulation tasks,

programming and

small project

There are some options in the assignment depending on the interest and the grade rating. Exercise tasks are divided into rounds and partial assignments.

Read the study material

The information required in the assignments can be found in this course material and the course's electronic teMultiple choice and algorithm simulation tasks

Tasks are embedded into this tutorial. You can find them by reading a tutorial in order or, alternatively, by clicking on the left navigation bar, point to point data. The list contains all the individual tasks of the course, which by accruing points. Multiple choice and algorithmic simulation tasks have their own guidelines and should be solved with the latest browsers. By completing the task, it automatically receives feedback and points will appear on A +.

**Project**

The subject of the project is announced in the second lecture. The project is done in group work and evaluated in several stages. The final phase of the project is evaluated by peer review, ie the last part of the project is evaluation of other student projects. More on this later.

Since the project is not evaluated automatically, the scores obtained from it are recorded after the evaluation is complete.

**programming Tasks**

Programming tasks are done in the Pyhon programming language. I also evaluate them automatically, but the solutions must be tested by themselves before returning them to ensure their functionality. To do this, a set of ready-made tests will be included. However, you should write more tests yourself.

Each task has its own instructions, but the following is a little more detailed information about, for example, what kind of work environment to implement programming tasks is worth installing.

**The Python version used is 3**

By default Python version 3.4 or later is used. There are currently installed version 3.4 in school machines.

Do not use Python version 2.

Python2 contains some pesky and partially difficult to notice differences in python3.xtbook. Read each material related to the task before you do the job.

**Python documentation and installation instructions**

The Python standard library documentation contains comprehensive instructions on all basic Python functions. There are many examples of Python tutorials in the documentation site that Python can use.

Task: Install the programming environment for yourself and test the program running

To execute the Python code, just a Python interpreter is enough and the code can be written even with a notepad. However, to save time and effort, it is worthwhile using the development environment. Installation instructions for installing the Python interpreter and the Eclipse development environment can be found here.

**unit testing**

Unit testing is a software testing method whereby the activities of the individual program partners are tested, if possible separated from other code, to determine that it meets the requirements set for it. The test part of the program is usually a function, class, or method.

This course uses unit testing to review your exercise assignments. Task packs include short, pre-written tests to test the accuracy of your solution. However, you may want to add more tests to make sure that the solution is correct before you can get it back. The automatic scanning system also includes more tests, and your program needs to survive. However, no automation can, in the general case, with complete certainty test the correctness of your program, so it is ultimately yours!

It is recommended that you perform tasks in the same order as they are tested using test methods. Tests have been attempted to first test the correctness of the simple functions and only the last complicated functions that may depend on the above-described simpler functions.

The unit test typically includes:

Test code, which is usually short

For a word description test, for example, calling certain methods in a particular order, such as: Add (0), Add (1), Delete (0)

Test Feeds

Possibly the status of the program before the test

Expected outputs and possibly after program status test.

Unit testing is typically performed using one of the unit testing libraries that performs the testing and compilation of the results. Libraries generally also work with IDEs, making it easy and quick to run tests when programming. For larger projects, unit tests are automatically performed whenever new code is imported into the project. The benefits of unit testing are the automation provided by the libraries and the assurance of good unit testing for the performance of the tested classes.

单元测试

单元测试是一种软件测试方法，其中测试各个程序合作伙伴的活动，如果可能，与其他代码分开，以确定其符合要求。程序的测试部分通常是函数，类或方法。

本课程使用单元测试来检查您的练习任务。任务包包括简短的预写测试，以测试解决方案的准确性。但是，您应该编写更多的测试，以确保解决方案是正确的，然后才能恢复。自动扫描系统还包括更多测试，您的程序需要生存。然而，在一般情况下，没有自动化可以完全确定地测试你的程序的正确性，所以它是最终你的！

建议您按照使用测试方法测试的顺序执行相同的任务。已尝试测试简单功能的正确性，仅测试可能取决于上述更简单功能的最后复杂功能。

单元测试通常包括：

测试代码，通常很短

对于单词描述测试，例如，以特定顺序调用某些方法，例如：Add（0），Add（1），Delete（0）

测试Feed

可能是测试前程序的状态

预期输出，可能在程序状态测试之后。

单元测试通常使用执行结果测试和编译的单元测试库之一执行。库通常也可以与IDE一起使用，使编程时可以方便快捷地运行测试。对于较大的项目，每当将新代码导入到项目中时，将自动执行单元测试。单元测试的好处是图书馆提供的自动化功能，以及对测试类的性能进行良好的单元测试的保证。

**def** **inc**(a):

**return** a + **1**

**import** **unittest**

**from** **functions** **import** inc

**class** **TestIncrement**(unittest.TestCase):

**def** **test\_inc\_with\_one\_positive\_integer**(self):

"""Test that incrementing 1 by 1 returns 2."""

x = **1**

expected\_result = **2**

returned\_value = inc(x)

self.assertEqual(

expected\_result,

returned\_value,

"Incrementing {0} with 1 should return {1}, not {2}."

.format(x, expected\_result, returned\_value)

)

**if** \_\_name\_\_ == "\_\_main\_\_":

unittest.main(verbosity=**2**)

A unit testing library is imported.

The function of the function inc.

Define the test class TestIncrement, which collects the test class from the unittest library.

Define test mode in the test class. the default library is defined as the default value for all methods that begin with the test run will be performed as tests.

Test methods can be given a comment unless the test operation clearly indicates the name of the test method.

Define the test data, expected return value, and true value returned by inc function when the parameter is a variable x.

The assertEqual method compares the first two parameters assigned to it, in this case expected\_result == returned\_value. If the result of the comparison is false, the method throws the AssertionError exception and the test is counted as unsuccessful.

The last parameter of the assert method is usually a message indicating why the test failed. This is not shown if the test succeeds.

Run the tests.

Learn more about testing

assertEqual is just one of the checklists provided by the unittest library. Other assert methods can be found in the Python documentation.

If the \_\_name\_\_ == "\_\_main \_\_" condition of the above tests ensures that the code underneath it is executed only if the file containing the tests is executed. For example, it is printed on a test file in Run Eclipse or executed in the command line python3 tests.py (in Windows py tests.py). Tests are performed by invoking the main function of the unittest library. The function was given as parameter verbosity = 2, which outputs a list of all executed test functions and a line of comments written under the function definition line to the test results page.

进一步了解测试

assertEqual只是单元测试库提供的清单之一。 其他断言方法可以在Python文档中找到。

如果上述测试的\_\_name\_\_ ==“\_\_main \_\_”条件确保仅在包含测试的文件被执行时才执行其下的代码。 例如，它打印在Run Eclipse中的测试文件上，或者在命令行python3 tests.py（在Windows py tests.py中）中执行。 通过调用unittest库的主要功能来执行测试。 该函数作为参数verbosity = 2给出，它将所有执行的测试函数的列表和在函数定义行下写入的一行注释输出到测试结果页面。

## 1.3 basic Concepts

main issues:

What is an algorithm? What is a data structure?

What to do?

Basic concepts of the course, role variables, basic data structures, simple abstract data types.

What are you supposed to do?

Read the textbook and learn about the online environment of the course. enter a message into the chat room (Piazza). There are also small tasks in the hall. Finally, little is programmed.

Indicative Demand Assessment:

Rather easy.

Indicative work estimates:

About a couple of hours.

Programming Tasks:

Primes

**Textbook**

The course has an interactive electronic textbook in English. Every week, the textbook reads 1-2 chapters, followed by the actual weekly exercises. Weekly exercises have a summary of the reading in Finnish, whereby the terms in Finnish will become familiar. For a summary holiday, you will also find weekly exercises divided into their themes.

The book proceeds in a slightly different order than the summary. You may also read the textbook linearly from the beginning. It is, however, recommended that the textbook first reads the figures given at least, and then it goes to summary and assignments.

This week's reading: textbook chapter 0. Introduction.

What's the difference between a textbook and a summary? Is not that another enough?

The abstract as its name only summarizes the underlined and underlining key areas of the course. It is therefore not a textbook and does not include any issues studying independently on the whole course. So the summary is not enough on its own, but a textbook is also needed.

On the other hand, as a textbook, you can use some other work than the electronic textbook written by Cliff Shaffer. The textbooks in the industry are typically English, so it is difficult to find a comprehensive Finnish-language book.

The good points of the summary are

it is Finnish,

it has some details explained more precisely than the book,

it raises the course exercises into topics and

schedule the course in addition to that

it clearly defines the key topics of the course, which can be focused especially if it does not aim at the highest grades.

Read the book at least the topics carefully, which are discussed in more detail in the summary.

The good aspects of an electronic book are:

it is free, but

comprehensive field of art,

you will need it to solve some of the tasks because the issue is not adequately addressed in the abstract and

it introduces important data structures and algorithms more widely than what is possible within the course 5 credits.

Read the book completely, especially if you are more interested in the subject and aim for the highest grades in the course.

**Data structures and algorithms**

Below is a brief summary of some important basic concepts that are often used later in the course.

The data structure is a collection of information that is logically linked. Data structures include tables, trees, and nets.

The data structure consists of elements (node) or node (s) that are related to each other. Sometimes you can also talk about records. The connection of the embryos to each other can be implicit (eg, the elements of the table are adjacent) or explicit, allowing access to another item by means of a separate reference.

An embryo, node, or record typically contains data (data, payload) that is one of the essential information for an application using the data structure. This information is often divided into fields (eg a single record can include a person's name, address, and phone number as a field). Often, information blocks are referred to by one or more fields (key), which distinguishes an element from other elements, e.g., in search structures or organization.

The data structure is a passive object that only contains data. Algorithms are required to process data, eg searching, updating or organizing information. Algorithm is a policy that defines how the input data is processed. The definition of the algorithm is:

algorithm

The algorithm is a finite queue with unambiguous, finite work executable commands that calculate the function

f:I→Of:I→O, jossa II on syötejoukko, OO on tulosjoukko ja ∀i∈I∀i∈I, algoritmi pysähtyy s.e., o=f(i)∈O

It should be noted that the algorithm is different from the program. The same algorithm can be implemented in many different programming languages, and in the same language there may be slight differences in its implementation because the instructions in the algorithm do not always correspond directly to the programming language sentences. It is, therefore, an abstraction in nature. Similarly, when dealing with data structures, they are often dealt with algorithms without any direct connection to any particular specific application. Consequently, they are also termed conceptually so that the actual payload associated with the data structure is abstractioned. Data structures are often portrayed as embryos and interrelationships (as references) and are separated by keys.

Data structures and algorithms are therefore general-purpose tools and are suitable as basic solutions to many of the future computational problems that arise, either directly or by applying them appropriately. The key features associated with them are the space usage of data structures and the execution time of algorithms that are processed in algorithmic analysis.

应该注意的是，该算法与程序不同。相同的算法可以在许多不同的编程语言中实现，并且在相同的语言中，其实现可能会有轻微的差异，因为算法中的指令并不总是直接对应于编程语言句子。因此，这是一种抽象的本质。类似地，当处理数据结构时，它们经常被处理算法而没有任何直接连接到任何特定的特定应用。因此，它们也在概念上被称为使得与数据结构相关联的实际有效载体被抽象化。数据结构通常被描绘为胚胎和相互关系（作为参考），并由键分隔。

因此，数据结构和算法是通用工具，可以作为直接或适当应用它们的许多未来计算问题的基本解决方案。与它们相关的关键特征是数据结构的空间使用和算法分析中处理算法的执行时间。

**Classification of data structures**

Data structures and algorithms can be classified in many different ways. **One of the most common ways is to classify them according to a computational problem,** i.e. they may relate to, for example:

Sorting or sorting information, which is often a repetitive computational problem in many applications.

The index structure, which can efficiently retrieve data corresponding to a specific key value.

Priority queue, which can effectively find the smallest / largest element of the material.

Graphs that can be used to describe multiple real-world problems and phenomena and network algorithms that can solve these problems algorithmically.

The classification could further be expanded to cover other algorithmic subdivisions based on the above-mentioned algorithms, such as string matching, focusing particularly on computational problems associated with strings and the algorithms and data structures developed for them. Examples might include strings associated with strings and the text of indexing, text consolidation, or even strings or regular searches. The latter has been successfully used in recent years for example. DNA sequencing (Human Genome Project). Algorithms are, however, required in many other areas.

数据结构分类

数据结构和算法可以以不同的方式分类。最常见的方法之一是根据计算问题对它们进行分类，即它们可以涉及例如：

排序或排序信息，这在许多应用中通常是重复的计算问题。

索引结构，可以有效地检索对应于特定键值的数据。

优先级队列，可以有效查找材料中最小/最大的元素。

可用于描述多个现实问题和现象的图形，以及可以在算法上解决这些问题的网络算法。

分类可以进一步扩展到基于上述算法的其他算法细分，例如字符串匹配，特别关注与字符串相关的计算问题以及为它们开发的算法和数据结构。示例可能包括与字符串关联的字符串，索引文本，文本合并，甚至字符串或常规搜索。后者已经在近年来得到了成功的应用。 DNA测序（Human Genome Project）。然而，在许多其他领域中需要算法。

**Another way of classifying data structures is related to their abstraction level.**

An abstract data type (ADT) is an abstract data structure that can be targeted to the operations listed in the type definition. Data in the structure can only be processed through these operations. The operations described in the ADT definition, however, only define the result of the calculation, not how the calculation is performed. For example, for an abstract data type, the search structure is defined by the operations Search (), Add () and Delete (). Operations can be implemented in many different ways based on a variety of internal data structures.

**Implementation** is a concept associated with the implementation of abstract data types. ADT can be implemented in many different ways and with a variety of data structures and algorithms. The way to implement an abstract data type is called **ADT implementation**. For example, the search structure is an abstract data type that can be implemented as a table, a binary search tree, an AVL tree, a tree B, or even a spreading structure. Similarly, the priority string is an abstract data type that can be implemented as a linked list or, for example, as a binary compiler.

**Implementation therefore defines what data structures and algorithms are used to implement an abstract data type**. The most important of these data structures are a **table, a linked list, a binary tree, a general tree, a fragmented tree, and a variety of network implementation methods, such as a sequencer and a dash matrix.** The charts and trees are, however, only special cases of networks, but they are so important and widely used that they should be treated as independent concepts. Implementing data structures and algorithms in a given programming language still offers a variety of variations to implement the same thing. For example, AVL tree rotation can be accomplished either by swapping keys from one node to another or by changing new values ​​for node markers. Implementations are also described in the literature as abstractions, the detailed implementation of which is left to the programmer.

However, all implementations are based on the basic infrastructures provided by programming languages, which can be thought of as embedded elements and their relationships. These infrastructures are not yet associated with any algorithms, such as implementation, and are irrelevant to what data has been stored on embryos. A single embryo can in fact be an arbitrarily complex second basic structure. The basic structures can thus be composed of different compositions. For example, a sequel is a table whose elements are linked lists; A 2-dimensional table (a contiguous matrix) is a table whose elements are 1-dimensional tables,

Since this course seeks to be independent of the programming language, the terminology is also chosen, and also the basic structures are treated as abstractions. In the background, it is assumed that, for example, the above mentioned items and the references between them can be implemented in the current programming language. On the other hand, the description of the implementation can also abstracate information that is not essential to the operation of data structures and algorithms. For example, the element of the list may contain data elements that can be implemented in the object programming languages ​​as objects and in the C language as the struct. Both can include fields where the actual data is stored. However, it is necessary to know only a few of these fields for the operation of the infrastructure. Typically, just references between embryos and e.g. key. The other fields can be abstracted, but they are essential when the structure is applied to one of the actual programming problems.

对数据结构进行分类的另一种方法与其抽象级别有关。

抽象数据类型（ADT）是可以针对类型定义中列出的操作的抽象数据结构。结构中的数据只能通过这些操作进行处理。然而，ADT定义中描述的操作仅定义计算结果，而不是如何执行计算。例如，对于抽象数据类型，搜索结构由操作Search（），Add（）和Delete（）定义。基于各种内部数据结构，可以以许多不同的方式实现操作。

实现是与抽象数据类型的实现相关的概念。 ADT可以以许多不同的方式实现，并且可以使用各种数据结构和算法。实现抽象数据类型的方式称为ADT实现。例如，搜索结构是可以被实现为表，二叉搜索树，AVL树，树B或甚至扩展结构的抽象数据类型。类似地，优先级字符串是可以被实现为链表或者例如作为二进制编译器的抽象数据类型。

因此，实现定义了什么数据结构和算法用于实现抽象数据类型。这些数据结构中最重要的是一个表，一个链表，一个二叉树，一个通用树，一个分片树，以及各种网络实现方法，比如定序器和破折号矩阵。然而，图表和树木只是网络的特殊情况，但它们非常重要和广泛使用，应被视为独立概念。在给定的编程语言中实现数据结构和算法仍然提供了各种变体来实现相同的事情。例如，可以通过将密钥从一个节点交换到另一个节点或通过更改节点标记的新值来实现AVL树的旋转。在文献中也将实现描述为抽象，详细的实现由编程人员来处理。

然而，所有的实现都是基于由编程语言提供的基础设施，可以被认为是嵌入元素和它们的关系。这些基础设施尚未与任何算法相关联，例如实现，并且与胚胎上存储的数据无关。实际上，单个胚胎可以是任意复杂的第二基本结构。因此，基本结构可以由不同的组合物组成。例如，续集是其元素是链表的表;二维表（连续矩阵）是其元素是一维表的表，

由于本课程旨在独立于编程语言，因此也选择术语，基础结构也被视为抽象。在后台，假设例如上述项目和它们之间的参考可以以当前编程语言来实现。另一方面，实现的描述也可以消除对于数据结构和算法的操作不是必需的信息。例如，列表的元素可以包含可以在对象编程语言中实现的数据元素作为对象，并以C语言作为结构体。两者都可以包括存储实际数据的字段。但是，有必要了解这些基础设施的运行情况。通常，胚胎和例如胚胎之间的参考。键。其他领域可以抽象化，但是当将结构应用于实际的编程问题之一时，它们是必不可少的。

There are other ways to classify data structures as well. We often talk about dynamic and static data structures. In addition, many programming languages have separate so-called. unchanged (immutable) and variable (mutable) data structures.

**Dynamic vs. Static**

The data structure is static if the memory space reserved for it is of standard size (memory can be reserved for translation). The data structure is dynamic if the data structure size or format can vary (memory can be reserved during execution).

**Unchanged vs. Changing**

The data structure is immutable if its status does not change during the execution of the program. For example, a tuple is an unchanged collection because once created tuple has the same items permanently; For example, the list is variable (mutable).

**Presentation of algorithms and the role of variables**

It is not always necessary to present algorithms in a programming language such as Pascal, Java, C, or Scheme, because there are easily too many details that are no longer relevant to the whole. The second extremity of the algorithm can be expressed in words. However, it should be noted that the description is sufficiently precise. Often, however, the verbal description is so inaccurate that, for example, analysis of the algorithm is difficult or nearly impossible. For this reason algorithms are often presented as so-called. using a pseudo-code. The purpose of the pseudo code description is to outline the essential operating principles of the algorithm without the presentation of the syntax of the programming language and the precise description of all the subprograms (for example, if the implementation of the subprogram is known and the time required is known).

The essence of the presentation is to describe the order of execution of the commands according to the definition of the algorithm, i.e., individual investment clauses and the structures governing them (consecutive, selection and repetition).

Example: The Eratosthenese Screen (invented about 200 eaa) is an algorithm that searches for all N (N≥2) prime numbers. The algorithm returns Table A

In the indices 0 ... p-10 ... p-1, the prime numbers were found between 2 ... N.

**def** **algorithm\_sieve\_of\_eratosthenes**(N):

**for** (i = **2**; i <= N; i++) /\* i: askeltaja, silmukka \*/

A[i] = i /\* A: säiliö, alkuluvut \*/

**for** (i = **2**; i <= sqrt(N); i++)

**if** (A[i] != **0**)

j = i\*i /\* j: askeltaja, ei-alkuluvut \*/

**while** (j <= N)

A[j] = **0**

j = j + i

p = **0**

**for** (i = **2**; i <= N; i++)

**if** (A[i] != **0**)

A[p] = A[i]

p = p + **1**

**return** A[**0.**.p-**1**]

The for- and the-loops are used in the playback.

The If clause represents the selection (other branches are missing, so they are conditional statements).

The sequential sentences in the same block (marked with indentation) are performed from top to bottom. The human eye easily finds the blocks easily with indentations, so the indentations here have a special meaning!

Note that variables in the pseudo code description are not usually presented. Man is reasonably easy to "guess" the type of the variable based on its name and behavior. For example, the parameter N obtained in this function is readily deduced as an integer.

In the second loop termination condition sqrt (N) decreases the square root of N. Although this is essentially a sub-program call, it has been omitted here because it can be expected to be generally known.

Ranges manipulate variables. For the algorithm, the behavior of the variable during the execution of the algorithm is essential. In this material, we have tried to comment on the behavior of variables by their roles. A dozen roles are sufficient to cover almost all variables in simple programs. Most roles are fixed value (value is not changed after initialization of the variable), step-by-step (goes through values ​​in some systematic manner), and the most recent preserver (the value is the last one passed through a particular set) covering about 70% of all cases. The example container (where the items to be retained can be rearranged) and the navigator (through the data structure) are the (new) roles in particular in the data structures and algorithms.

播放中使用for和the循环。

If子句表示选择（其他分支丢失，因此它们是条件语句）。

同一块中的连续句子（用缩进标记）从上到下执行。人眼容易发现凹陷，所以这里的缩影具有特殊意义！

请注意，伪代码描述中的变量通常不会显示。人很容易根据其名称和行为“猜测”变量的类型。例如，在该函数中获得的参数N容易被推导为整数。

在第二个环路终止条件下，sqrt（N）减小N的平方根虽然这本质上是一个子程序调用，但是由于可以预期这是普遍已知的，所以这里被省略。

范围操纵变量。对于算法，在执行算法期间变量的行为至关重要。在这个材料中，我们试图通过角色来评论变量的行为。几十个角色足以覆盖简单程序中的几乎所有变量。大多数角色是固定值（在变量初始化后，值不会更改），逐步（以某种系统方式执行），最近的保护程序（该值是通过特定集合的最后一个值），占所有案例的70％。示例容器（要保留的项目可以重新排列），导航器（通过数据结构）特别是数据结构和算法中的（新）角色。

## 1.4 Review of Object Programming

Object-oriented programming

This is not a comprehensive teaching material for object-oriented programming and this is not the only self-study material if the topic is not already known. Take a little time and look at the links in the material if you have not used categories and objects in the past.

Classes and their instances (objects, objects) are one way of presenting information and related functionality. In this course, classes are used to describe data structures and therefore, knowledge of the basics of object programming is important. Since the Basic Programming Course on Y1 subjects are moderately dealt with, the purpose of this section is to review the subject as well as provide tips on how to deepen the object-oriented programming skills.

The Y2 Basic Programming Course offers a wide range of Finnish language teaching materials. However, this is not a mandatory preamble of this course. However, it is recommended that you run Y2 either in front of or in parallel with the TRAK Y course.

See also the enclosed abstract English example of the objects: https://en.wikibooks.org/wiki/A\_Beginner%27s\_Python\_Tutorial/Classes

Python's official documentation of classes and objects can also be found online: <https://docs.python.org/3/tutorial/classes.html>

An example of an object

This section shows a really brief and concise review of how things work. If you are still unknown to us, you should definitely look at the contents of the links above.

In this example, the subject of cars and objects is a few of the basic features and functions of cars. If you can easily follow these tutorial examples, you have sufficient Prerequisites for Object Programming for the continuation of the course.