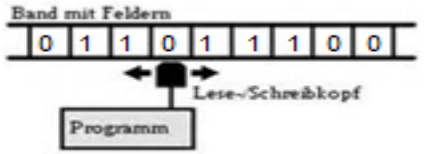
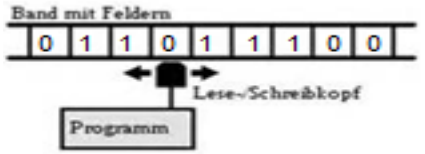


From the problem to the machine program

From the idea to the software

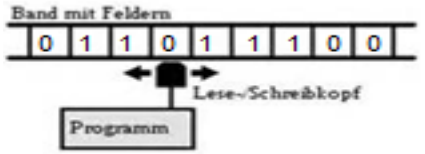
Development of a program



- Software projects are divided into several phases
  → Software Life Cycle

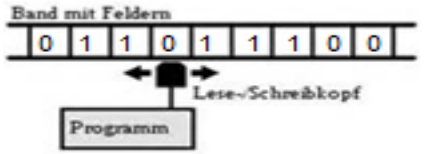
  – Analysis
  – System Design
  – Detail Design
  – Implementation and testing
  – Operation and maintenance

The phases of the Software Life Cycle



1. Problem analysis (Requirements analysis)
   -> is carried out together with the client.
   -> results in a software requirements specification.
2. System design: the tasks are divided into modules or components.
   -> Division into “smaller” problems increases the comprehension.
   -> improve correctness and reliability.
3. Detail Design: (Program specification)
   -> define the data structures.
   -> develop the algorithms.

Development of a program

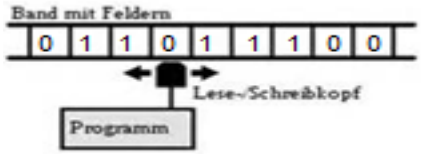


4. Implementation and testing
   - -> Development of programming code.
   - -> Implementation of test cases

5. Operation and maintenance

   While using the software, errors or new requests to the software are found.
   This may cause a return to problem analysis, which creates a cycle.

Compilation vs. Interpretation



Programs are either interpreted or compiled:

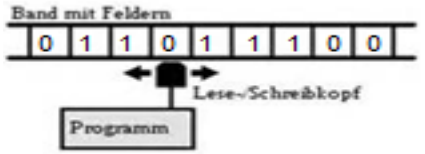
Compilation

Each command of the program written in the higher programming language is translated into a corresponding sequence of machine code and stored in a (binary) file. The application is then executed from this file.

This technique of translation is called compilation.

The program responsible for translation is called a compiler.

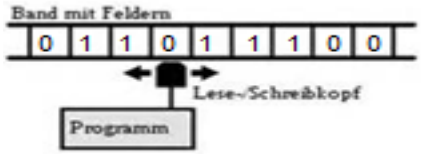
Compilation vs. Interpretation



- Interpretation
  - The commands are translated one at a time into corresponding sequences of machine commands and then executed.
    This technique, in which no new file must be created in machine code, is called interpretation.
  - The program responsible for translating and executing each command is called an interpreter.

Examples: Script languages such as Perl, Python, JavaScript, ...

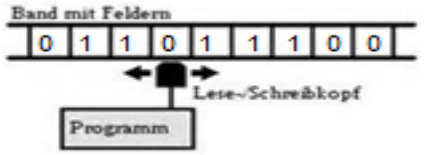
Compilation vs. Interpretation



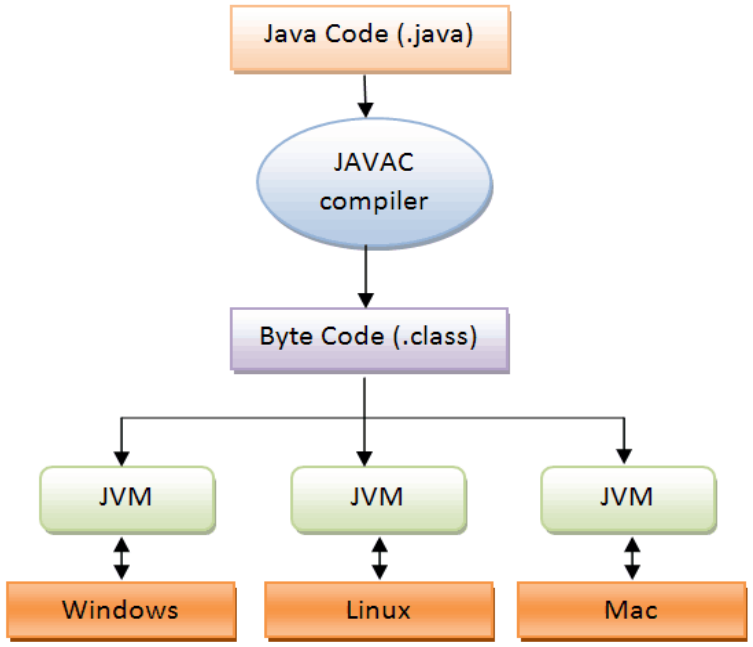
Compilation and interpretation in comparison:

- During compilation, the entire program is translated into machine code, so you have two files:
  - the program code in the higher programming language, which is readable by humans and but can't be performed on the machine, and
  - the machine program (machine code), which can be carried out on the machine.
- In case of interpretation, each command of the program is executed step by step after translation. As the translation takes time, interpreted applications are not as performant as compiled ones.

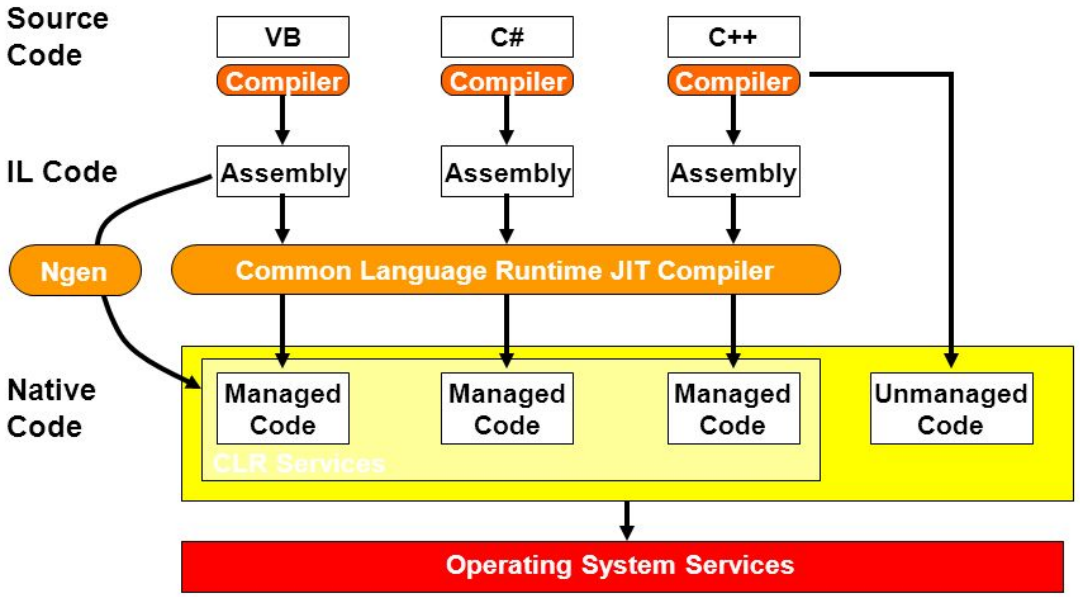
Programming Tools



Modern programming languages, such as Java, C#, . . . use a mixture of both methods. Such programs are translated into virtual machine code (e.g. Java byte code or IL code).

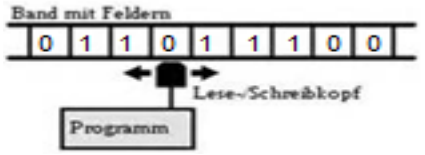


Java Virtual Machine (JVM)



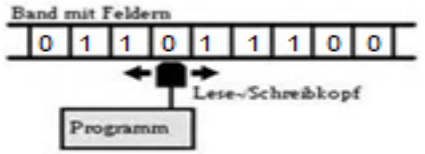
.Net Common Language Runtime

Programming Tools

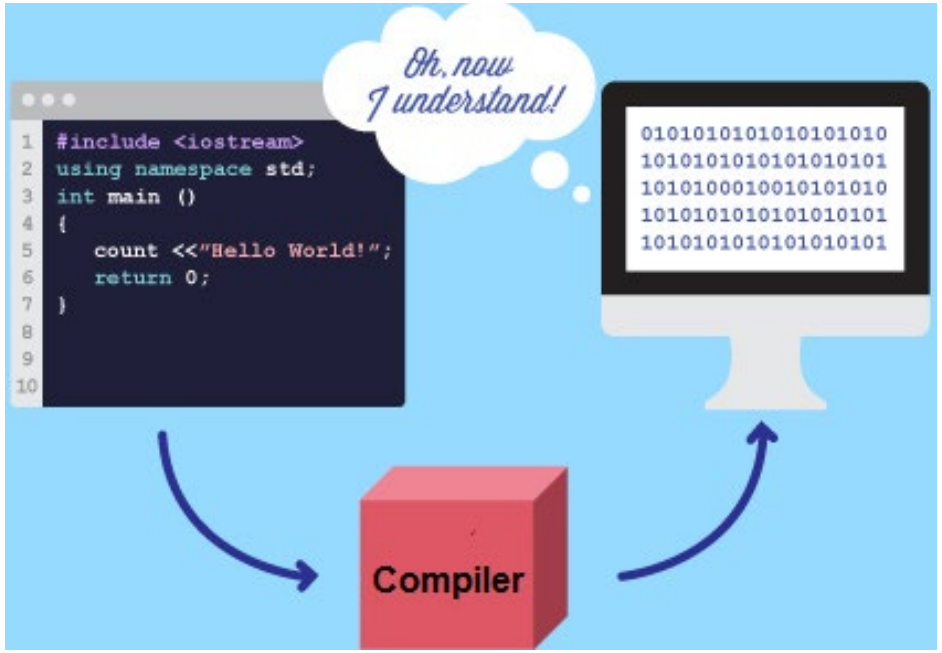


- Virtual machine code is first interpreted by a virtual machine and is only converted into machine code at run time.
- Such programs can run on any system that has a corresponding virtual machine installed.
- In addition, each command can be checked directly during execution (e.g. for allowed memory or hard disk access).
- These benefits usually compensate for a slightly higher runtime.

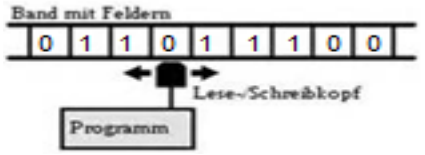
The Compiler



For the computer to run a program (source code), it must first be converted (translated /compiled) into machine code.

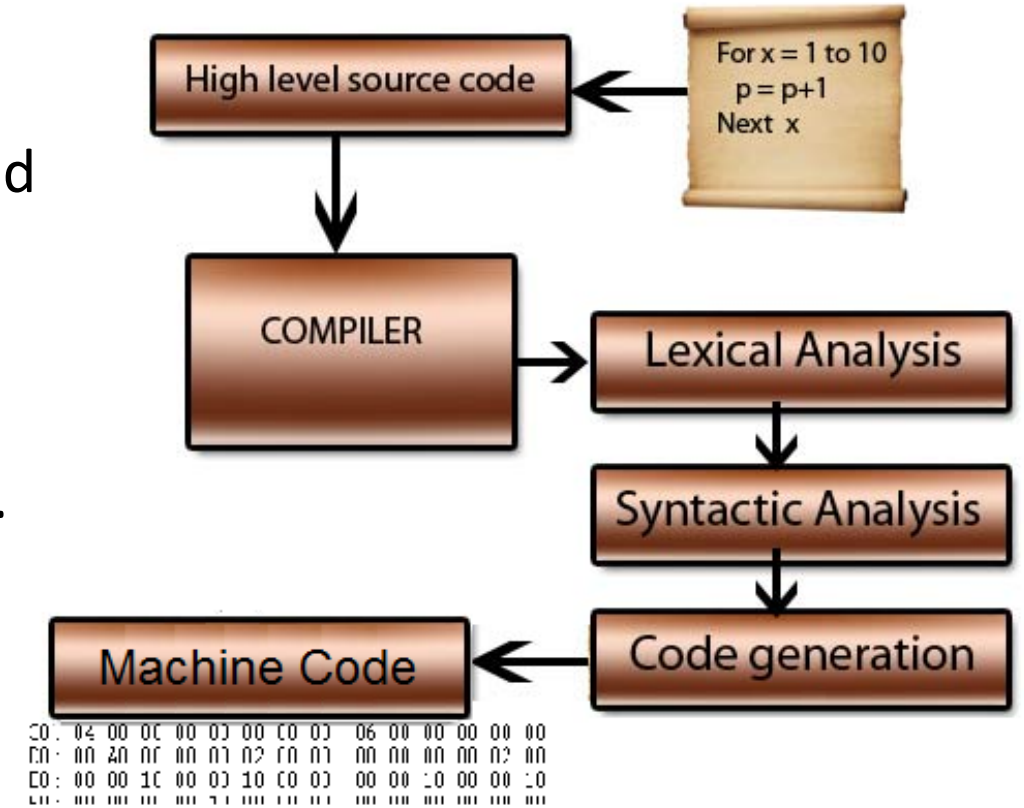


The Compiler

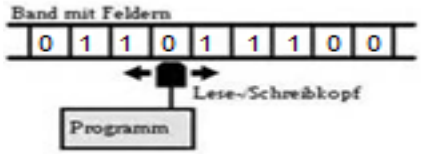


Compilation consists of the following steps:

1. *Analysis (Lexical and Syntactic)*
   The source program is checked and broken down into its components. An intermediate representation, a so-called parse tree, is generated.
2. *Synthesis*
   The desired target program (machine code) is generated from the parse tree.



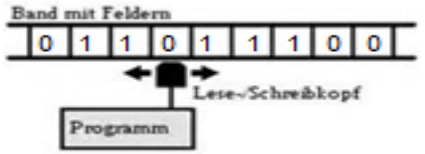
The Compiler



Special features of compilation

- The program can consist of several modules (program files), all of which are to be compiled individually before they are "bound together" with the linker.
- If the machine code uses library routines, the code must be linked to the appropriate libraries.

Debugging



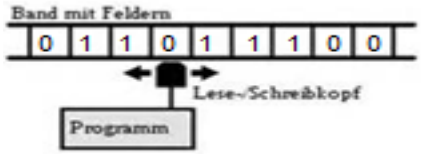
Debugging

…provides an easy way to view the running code.

… allows you to go through the program code step by step and show the assignment of all variables.

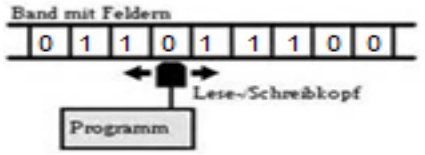
… is used to search for programming errors.

Higher programming languages



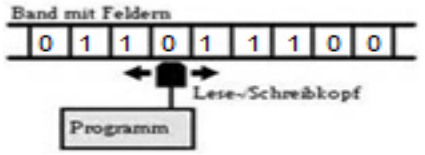
- In machine code, each step must be at the level of address registers and data registers (machine instructions).
- A programmer must be able to focus on solving the application problem.
- For this reason, the first programming languages such as FORTRAN and ALGOL emerged in the late 1950s.
- Higher programming languages should make it possible to specify a problem solution in a subject-specific notation (in the problem space).

Higher programming languages



- The compiler can convert the source code into machine code.
- Higher programming languages are intended to facilitate the implementation of solutions and are therefore also referred to as problem-oriented programming languages.
- Applications for programming languages include data management tasks (business objects) or technical-scientific tasks (statistics, research, data mining, ...)
- Today, there are hundreds of programming languages. They differ in the means of expression for various problem solutions.

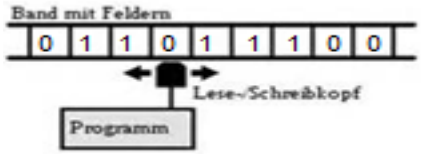
Higher programming languages



Programming languages are assigned to different generations:

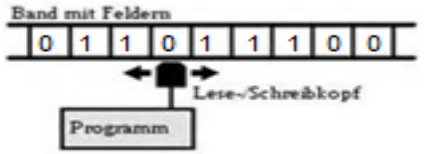
	Example	Characteristic
1G	Machine Code	Binary Code
2G	Assembler Code	Symbolic instructions
3G	Imperative programming languages (Fortan, Cobol, …)	Hardware independent
3G+	Object oriented programming language (Pascal, Java, C#, …)	Structured / object oriented, (abstract data types)
4G	Declarative programming languages (Prolog, SQL, XSL, …)	Rule based
…		

Higher programming languages



- Higher programming languages are from the 3rd generation languages.
- The concepts of these languages include elements such as
  - data types and variables for storing data,
  - Operators and expressions for logical linking of data,
  - Control or control instructions in the form of branches and loops (program flow control).
  - …

OO-Program = Data + Algorithm

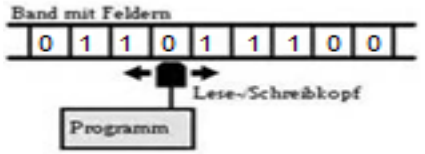


In its most general form, object-oriented programs consist of:

- data (objects, information) and
- operations or algorithms on the objects.

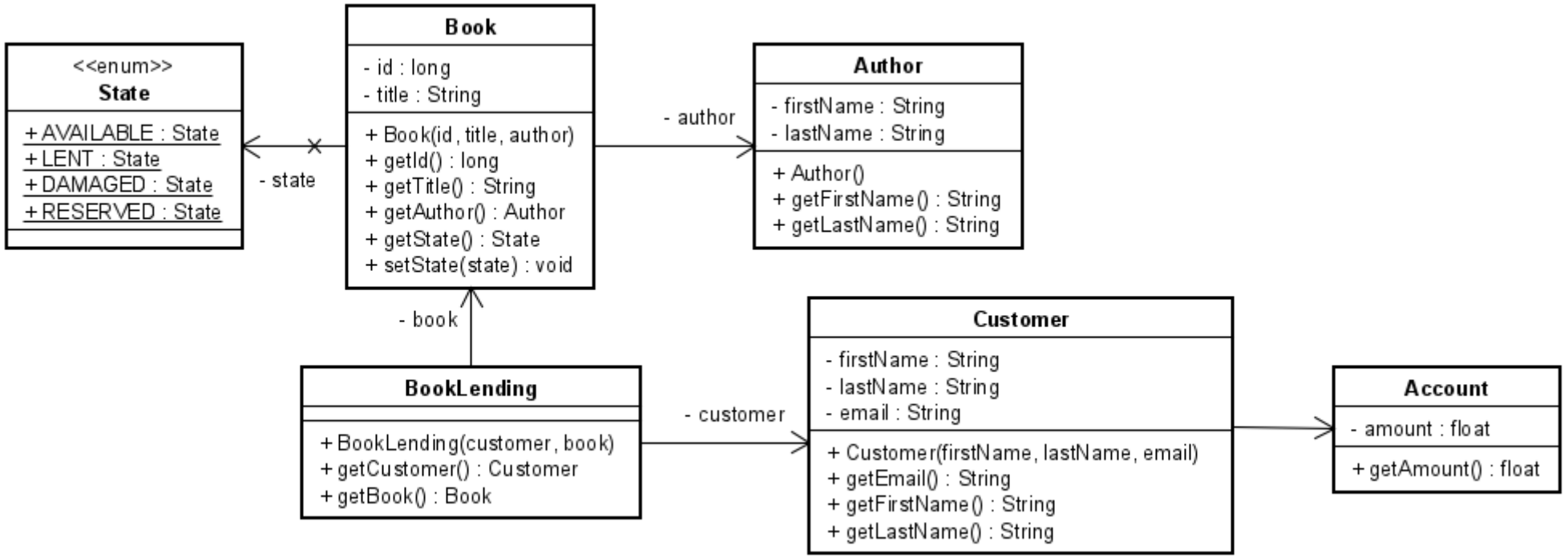
  The operations cause the data to be brought from an original state (initial or input value) to a final state via any intermediate values that may be necessary.

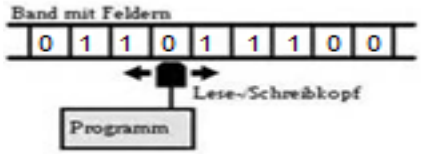
OO-Program = Data + Algorithm



Example: Excerpt of a class diagram for a public lending library (books and movies).

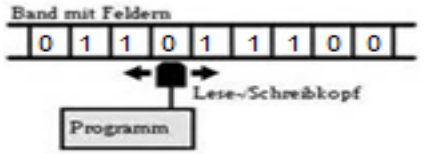
As soon as a customer lends a book, a certain amount is withdrawn from its account and the book state changes from available to lent.





From the specification to the code

Specification of a task

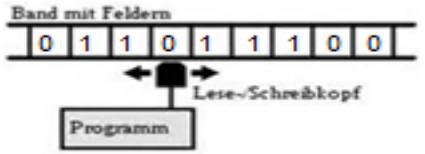


Before starting the coding, the problem to be solved must be specified.

A specification is a complete, detailed and unambiguous description of the problem.

- Complete: all relevant information is known, there are no open points.
- Detailed: the specification is so precise that it can be implemented step by step.
- Unambiguous: criteria are given which define whether a solution is correct.

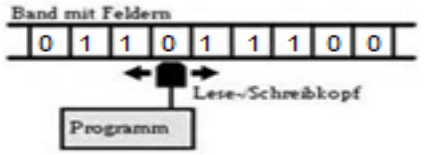
Algorithms



- After a problem is specified, a solution must be designed.

- Since the solution is to be executed by a computer, each step must be prescribed precisely and step by step.
- This is done by an algorithm.

The term *algorithm*



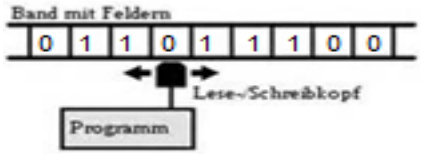
Informal characterization:

An algorithm is a detailed and explicit directive for the resolution of a problem, precisely formulated, presented in a finite description and effectively executable.

Example characterization:

- The total amount of the invoice consists of the individual amounts of the flight, the accommodation (hotel), meals (half or full board) as well as the booked additional services (excursions and wellness treatments).
- In the months of January to May, a 25% discount is granted on the accommodation.

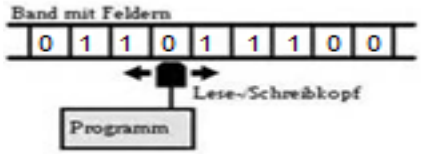
Algorithm



Example:

- Get the transaction from the database.
- Read from the database the price p_1 for the flight included in the booking.
- Read from the database the price p_2 for the number of hotel nights booked.
- Read from the database the price p_3 for the number of booked meals per day (half or full board).
- If one or more excursions are included in the booking, read the price for each included excursion p_{a1} to p_{an} from the database.
- If one or more wellness offers are included in the booking, read the price for each included offer p_{w1} to p_{wm} from the database.
- If the trip does not take place in the months of January to May, sum up all prices p_i received in this way.

Formulation of an algorithm

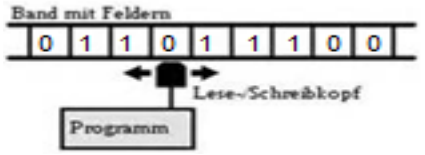


For the formulation of algorithms, we use **variables**.

- Variables are memory slots to which a name is given in a program. The value of variables can usually be changed during the program's runtime.

- Constants are variables that are initially assigned a value that can`t be changed later.

Formulation of an algorithm



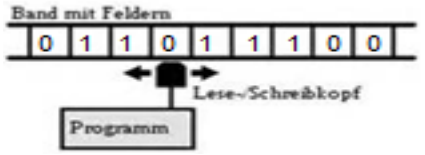
The formulation of an algorithm can be done in natural or formal language or graphically.

Algorithms can be described using structured text:

```
Algorithm LargestNumber
  Input: A list of numbers L.
  Output: The largest number in the list L.
```

```
  if L.size = 0 return null
  largest ← L[0]
  for each item in L, do
    if item > largest, then
      largest ← item
  return largest
```

Formulation of an algorithm



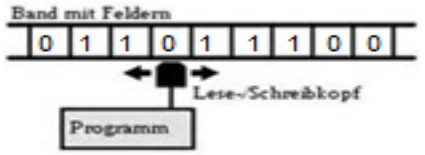
To find the largest number in a list of numbers:

- Set the first value of the list as candidate for the largest value.
- If the next value is larger than the actual candidate, set this value as new candidate for the largest value.
- Repeat this for all values of the list.
- Return the found value as result of the algorithm.

```
Algorithm LargestNumber
  Input: A list of numbers L.
  Output: The largest number in the list L.
```

```
  if L.size = 0 return null
  largest ← L[0]
  for each item in L, do
    if item > largest, then
      largest ← item
  return largest
```

Formulation and presentation of an algorithm



As a Python function, for example, this looks like this

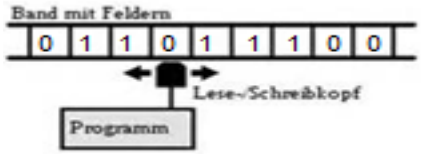
```
def findMaximum(aList):
    largest = aList[0]
    for element in aList:
        if element > largest:
            largest = element
    return largest
```

Loop across all items in the list:
If the found value is greater than the previous largest, then the found value becomes the new candidate.
In the end, the content of the largest variable is displayed as a result (→ return largest)

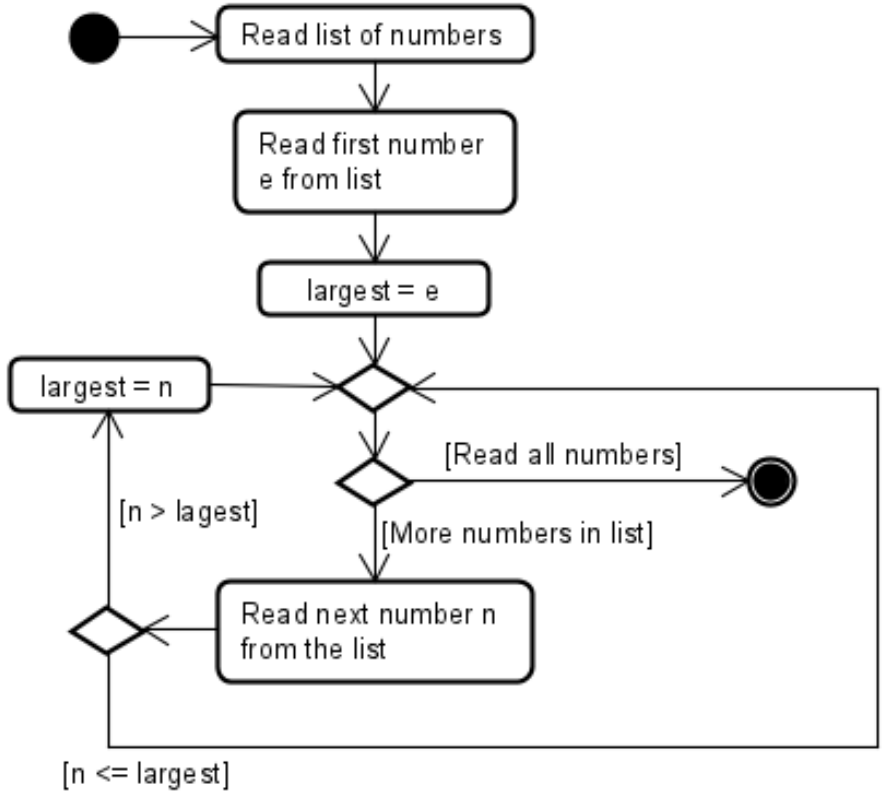
```
numbers = [3, 4, 6, 8, 11, 13, 2, 17, 9, 12, 1]

print(findMaximum(numbers))
```

⇨ 17

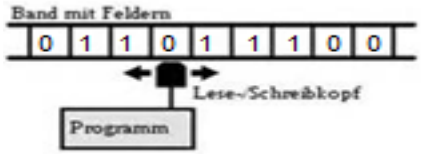
Formulation and presentation of an algorithm



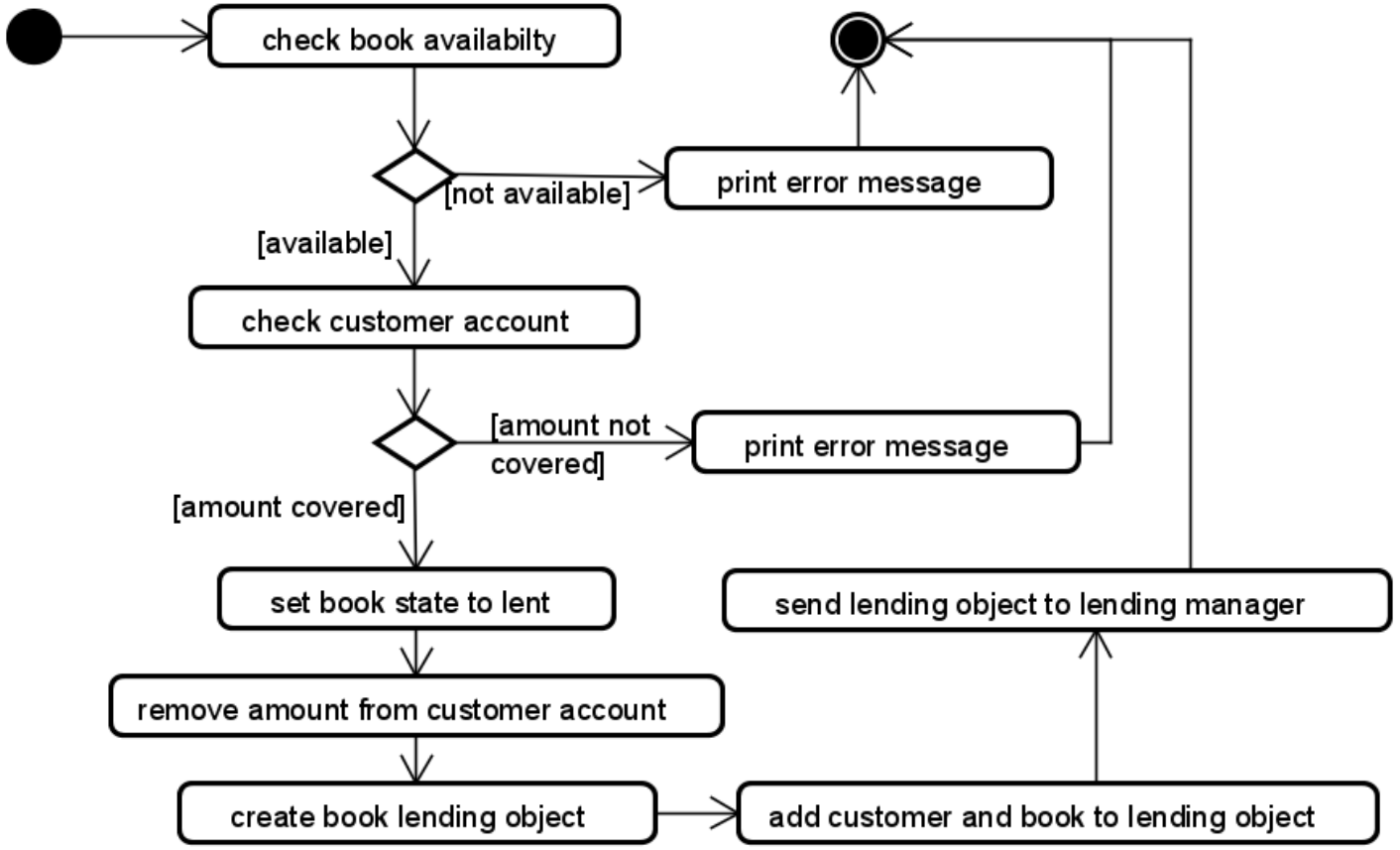
Formulation of a “find maximum value” algorithm by an activity diagram

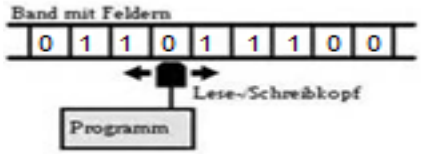


Lending Algorithm



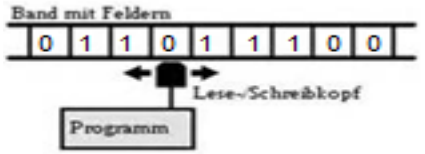
The process of “lending a book in the public library”.





Boolean algebra

Boolean algebra



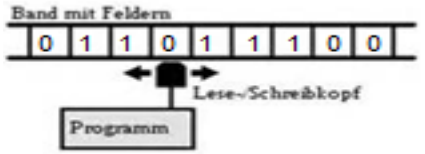
George Boole

- English mathematicians of the 19th century
- Formal view of digital structures

Boolean algebra has only two values: 0 (False) and 1 (True).

- Is the basis for today's computer hardware.
- Defining decisions → if (condition) .
- The result of a condition is True or False.
- In Boolean algebra there are three operators: and, or and not

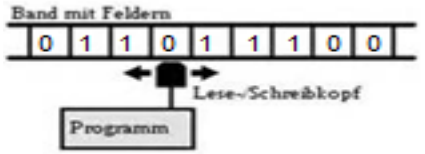
The *or* operator



- Or operator is a logical sum:
  - The result of an Or operation is 1 (True), if at least one of the two variables in the term a or b has value 1.

a	b	a or b
0	0	0
0	1	1
1	0	1
1	1	1

The *or* operator

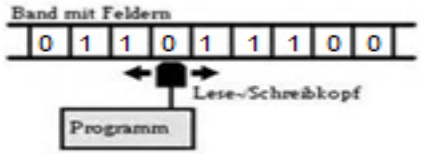


An overall condition formed with the Or operator is true if (at least) one of the individual conditions is true.

Example: the following python function returns True if two of the three numbers x, y, z are equal.

```
def twoEqual(x, y, z):
    if x == y or x == z:
        return True
    if y == x or y == z:
        return True
    return False
```

The *and* operator



- The And operator is also called a logical product.
- The result of an And operation is 1 if both variables in a and b have a value of 1.
- Two conditions associated with the And operator only are true if both individual conditions are true.

a	b	a and b
0	0	0
0	1	0
1	0	0
1	1	1

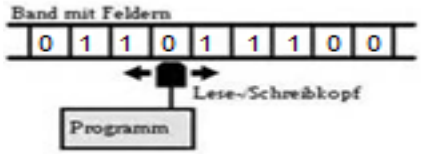
(it's Friday) and (it‘s raining) and (the sun shines)

not impossible but rare

(the person is younger than 6) and (the person is older than 50)

cannot be fulfilled

The *and* operator

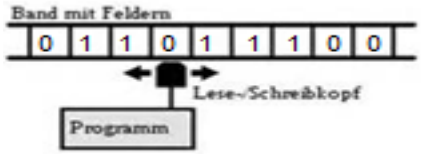


An overall condition formed with the and operator is true if all of the individual conditions are true.

Example: the max function returns the largest of the three numbers x, y, z.

```
def max(x, y, z):
    if x >= y and x >= z:
        return x
    if y >= x and y >= z:
        return y
    if z >= x and z >= y:
        return z
```

The *not* operator



- The not operator turns True to False and vice versa (inversion).

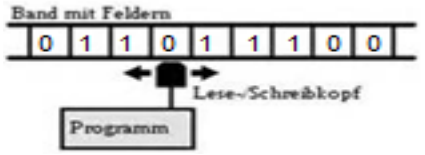
- The result of a not operation is 1 if the corresponding variable has value 0. If the variable has a value of 1, the result is 0.

a	not(a)
0	1
1	0

not(the person is younger than 6) or (the person is older than 50)

is True for all persons older than 6

The *not* operator



A condition formed with the not operator is True if the inner condition is False.

Example: the function allDifferent returns True if the three numbers x, y, z are all different.

```
def allDifferent(x, y, z):
    if not(x == y) and not(x == z) and not(y == z):
        return True
    return False
```

```
if  x != y and x != z and y != z:
```