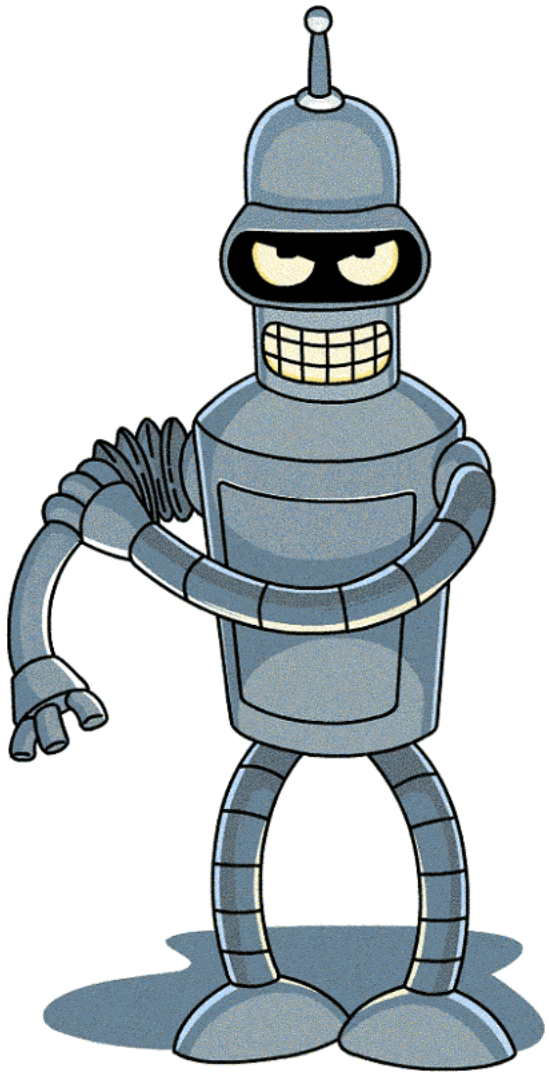
PGR103 Objektorientert programmering

Schedule

- Dynamic (we take the time you need to learn the important concepts!)
- Talking plus hands on ☺
  - Introduction
  - Simple programs
  - Conditions and loops
  - Arrays and methods
  - IDE
  - **Classes and objects**
  - **Object orientated programming**
  - **Inheritance**
  - Information hiding
  - Interfaces
  - More…

Practical assignments

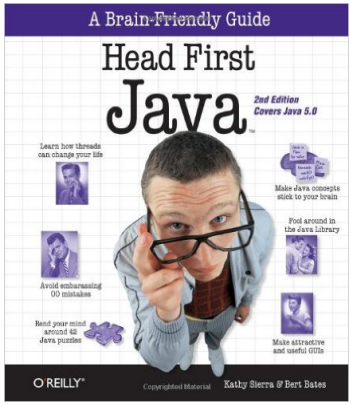
- Will follow on the fly!



Readings (English)

Kathy Sierra, Bert Bates (2005) **Head First Java** (Englisch), O'Reilly and Associates;

- This book covers object oriented programming, so there is a gap in the first part. For this I recommend **Java 8 in Action: Lambdas, Streams, and functional-style programming 1st Edition**



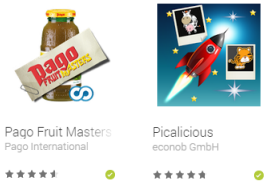
Java Documentation

- Java API Doc
  - http://docs.oracle.com/javase/8/docs/api/
- Java Tutorials
  - http://docs.oracle.com/javase/tutorial/

How should I learn Java?

1. Learn to have fun programming. It makes it easier.
2. Invest time in the Java Tutorials and the readings.
3. Go to the course.

Motivation – Why Java?



Motivation

- It’s necessary for research & development
  - Grand Challenge projects, prototypes
- Projects for multimedia production, ie. Processing
- Games, apps, etc.

What is “programming”?

... describing the solution of a problem in such an exact way, that a computer can solve the problem.

Cp. recipes, manuals, etc.

Programming is

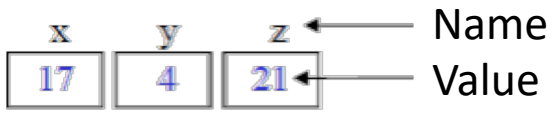
- a creative process
- an engineering skill
- a complex task if you want to do it right.

What is a program

program = data + commands

Data

- Set of address-able memory cells



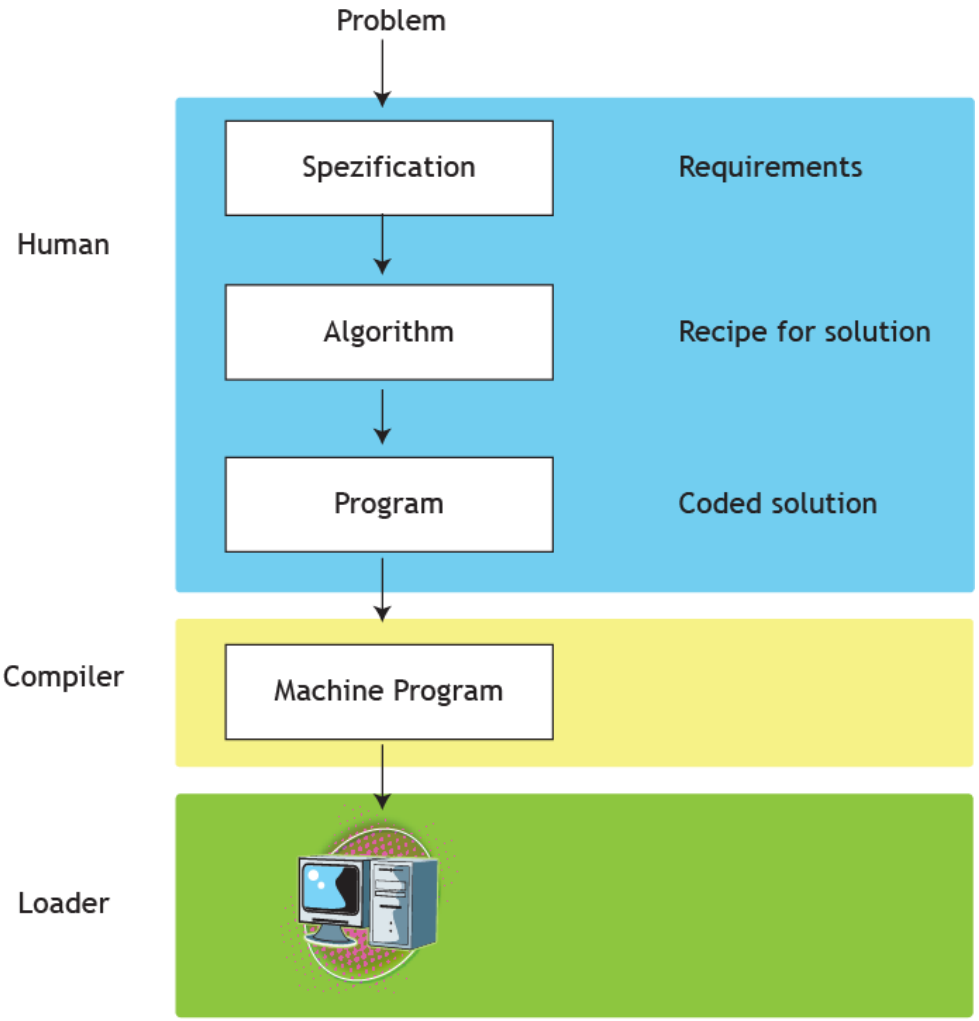
- Data is stored in binary format, eg. 17 = 10001
- Binary format is universal
  - numbers, text, image, audio, ...
- 1 Byte = 8 Bit
- 1 word = 4 Byte (typically)

Commands

- Operations on memory cells

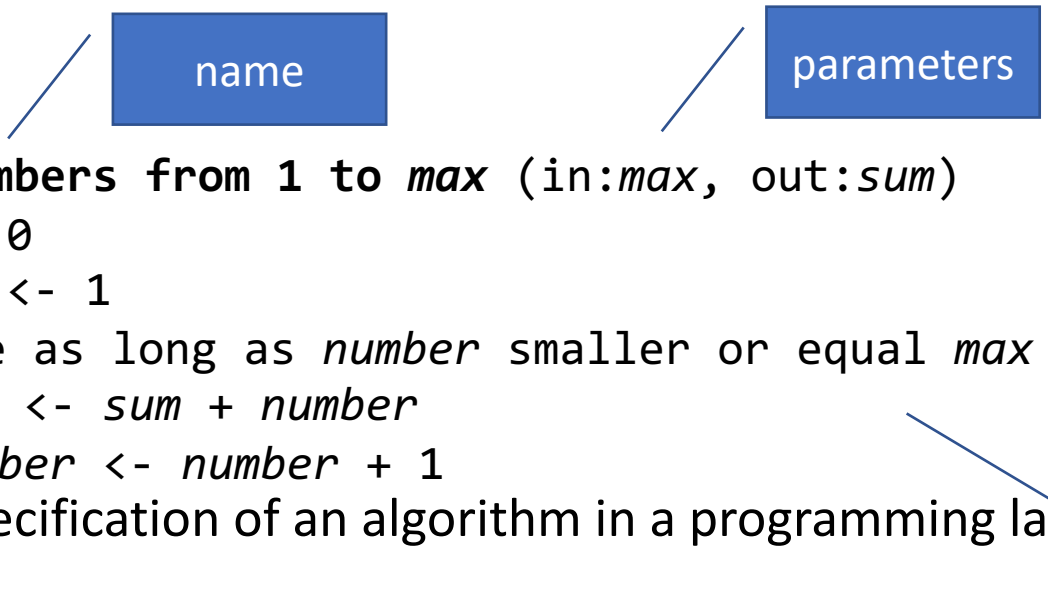
Machine language			Programming Language
$ACC \leftarrow x$	//	load memory cell x	$z = x + y;$
$ACC \leftarrow ACC + y$	//	add memory cell y	
$z \leftarrow ACC$	//	store result in memory cell z	

How to create a program?



Algorithm

- Precise, step by step solution to a problem

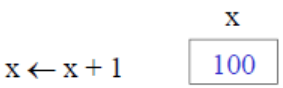


- program = specification of an algorithm in a programming lan

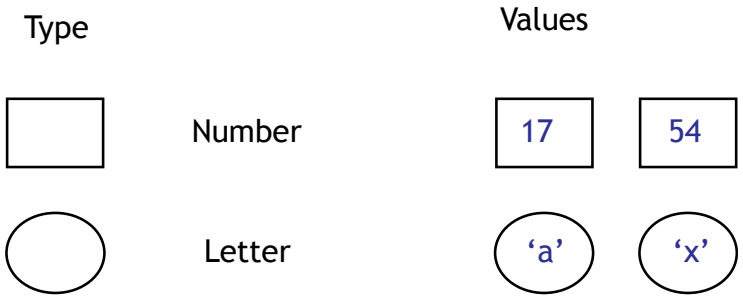
Variables

- Variables are named container for values.

- Values can change

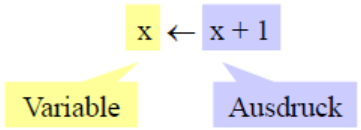


- Variables have a data type
  - which is the set/range of values allowed for a variable.



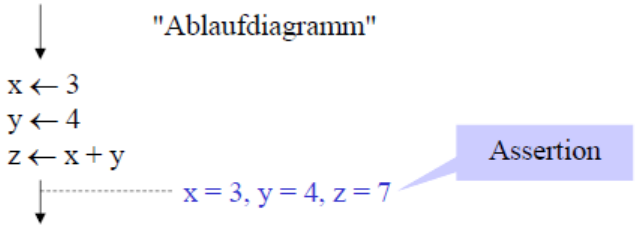
Statements

- Assignement



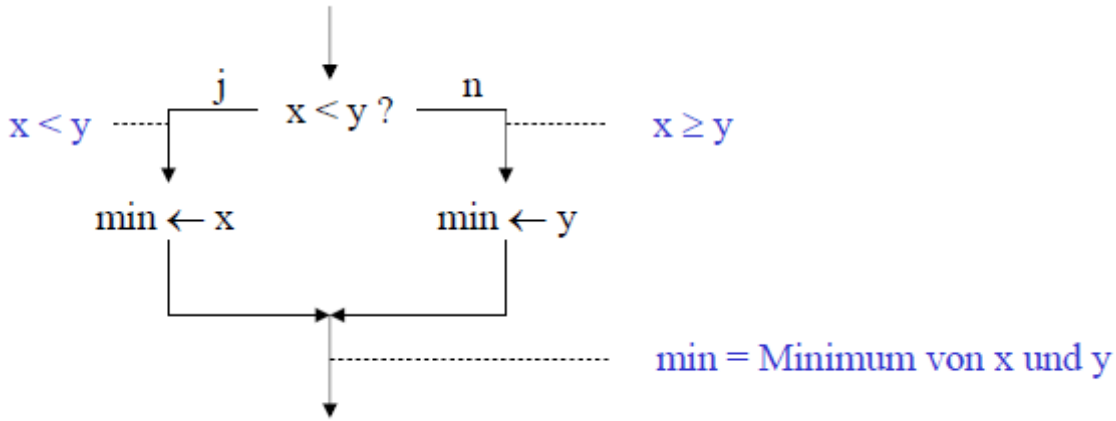
1. compute value
2. assign result to variable

- Sequence of statements



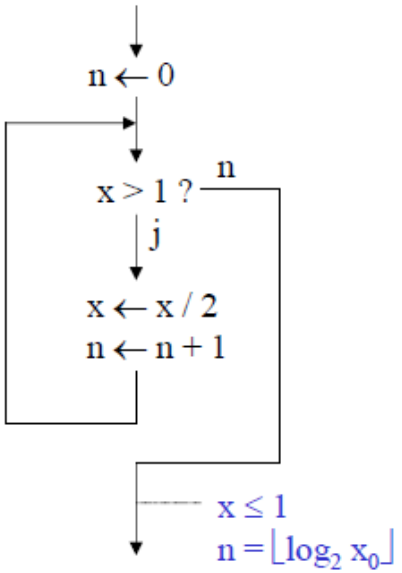
Statements

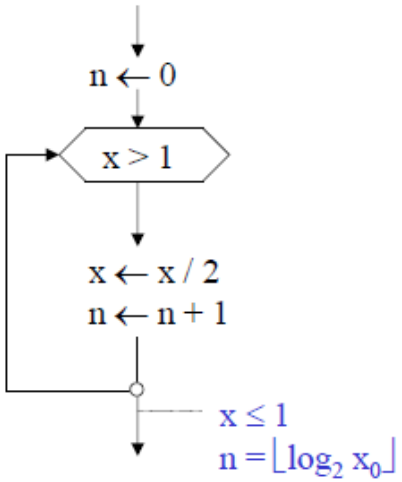
- Condition / Choice



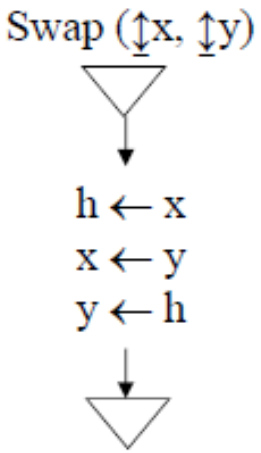
Statements

- Iterations, Loops





Example: swap values



proof of concept

x	y	h
~~3~~ 2	~~2~~ 3	3

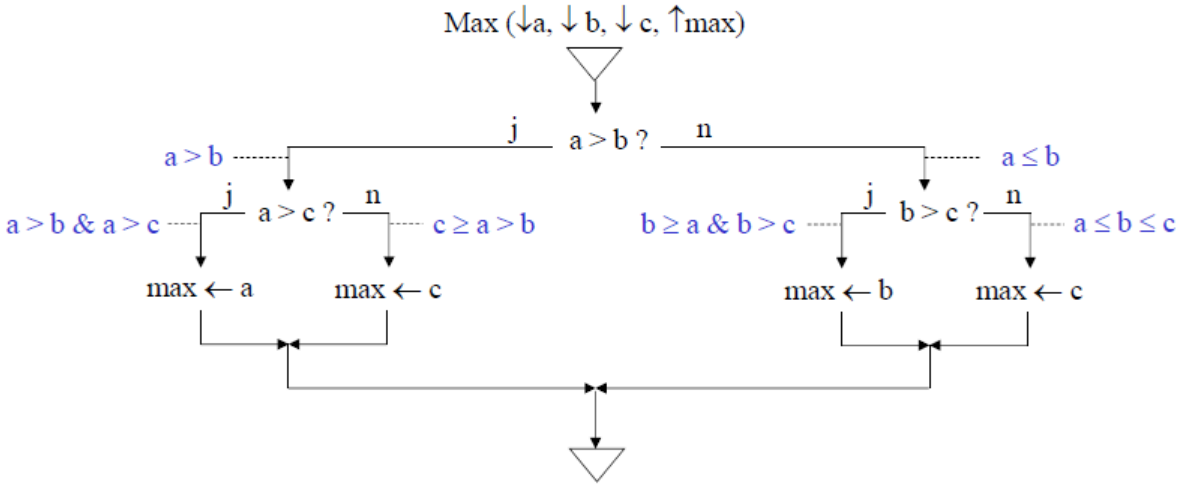
Example: swap values

```
int x = 10;
int y = -5;
int h;

println(x);
println(y);

h = x;
x = y;
y = h;

println(x);
println(y);
```

- Source Code for Processing
- Processing is „like Java“

- int … data type
- ; … ends a statement
- println() … function for printing text on screen.

Example: maximum of three numbers



Example: maximum of three numbers

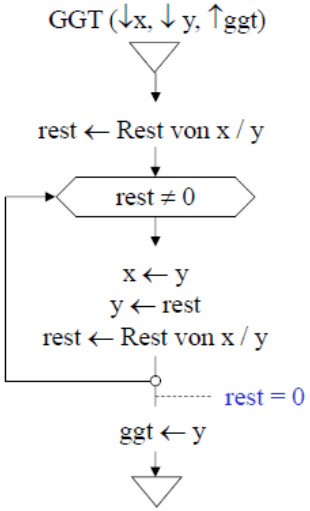
```
int a = 11;
int b = 12;
int c = 13;
int max;

if (a<b) {
  if (b<c) {
    max = c;
  } else {
    max = b;
  }
} else {
  if (a<c) {
    max = c;
  } else {
    max = a;
  }
}

println(max);
```

- Source Code für Processing

- if (test) {..}
- else {..}

Example: Euclidean algorithm

- Greatest common divisor (ggt) of two numbers.



proof of concept

x	y	rest
~~28~~	~~20~~	8
~~20~~	~~8~~	4
8	4	0

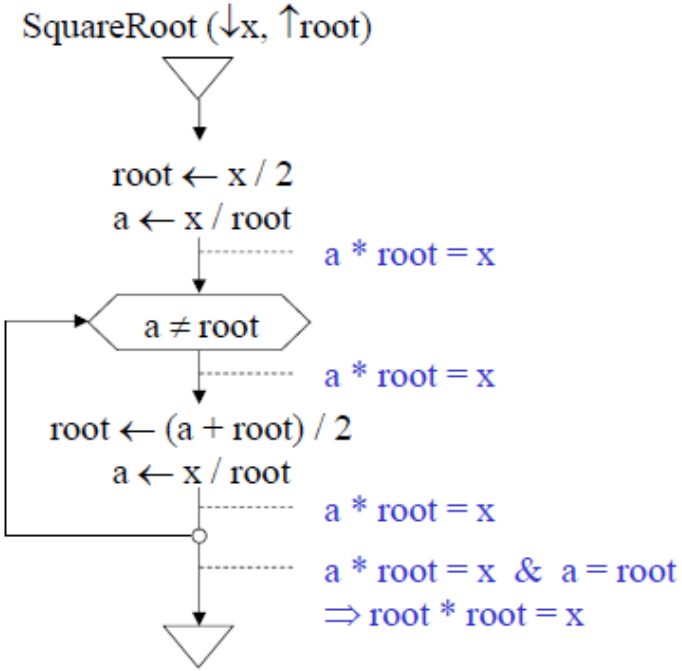
Why does this work?
(ggt divides x) & (ggt divides y)
-> x = i*ggt, y = j*ggt, (x-y) = (i-j)*ggt
-> ggt divides (x-y)
-> ggt divides (x-q*y)
-> ggt divides rest of x/y
-> ggt(x,y) = ggt(y, rest)

Example: Euclidean algorithm

```
int x = 21;
int y = 14;

int rest = x % y;

while (rest != 0) {
  x = y;
  y = rest;
  rest = x % y;
}

println(y);
```

- Source Code for Processing
- While (test) {..}
- % … modulo

Example: square root



proof of concept

x	root	a
10	~~5~~	~~2~~
	~~3.5~~	~~2.85714~~
	~~3.17857~~	~~3.14607~~
	~~3.16232~~	~~3.16223~~
	3.16228	3.16228

Example: square root

```
float x = 10;

float root = x / 2;
float a = x / root;

while (a != root) {
  root = (a + root) / 2;
  a = x / root;
}

println(root);
```

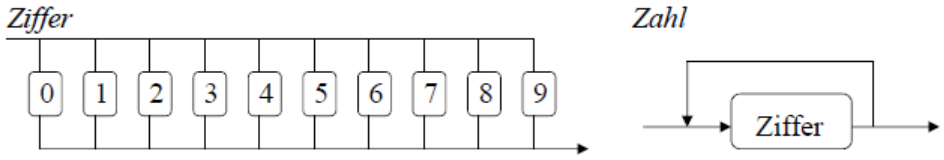
- Source Code for Processing
- float … data type
- / … Division
- Hint: Don‘t test float on equality!
  - |a-root| < 0,00001

Specification of programming languages

- Syntax
  - rules to build sentences
  - e.g. assignment = variable <- statement
- Semantics
  - Actual meaning of sentences
  - e.g.: compute statement and assign result to variable.

Specification of programming languages

- Grammar
  - Set of syntax rules
  - eg. grammar for discrete positive numbers.
    - numeral = "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9".
    - number = numeral {numeral}.



EBNF (Extended Backus-Naur-Form)

Examples

- *Grammar for floating point values*
  - number= numeral {numeral}.
  - float = number"." number["E" ["+" | "-"] number].
- *Grammar for If-statements*
  - IfStatement = "if" "(" Statement")" Statement ["else" Statement].

Usage	Notation
definition	=
concatenation	,
termination	;
termination	. [1]
alternation	|
option	[...]
repetition	{ ... }
grouping	(...)
terminal string	" ... "
terminal string	' ... '
comment	(* ... *)
special sequence	? ... ?
exception	-

src. Wikipedia

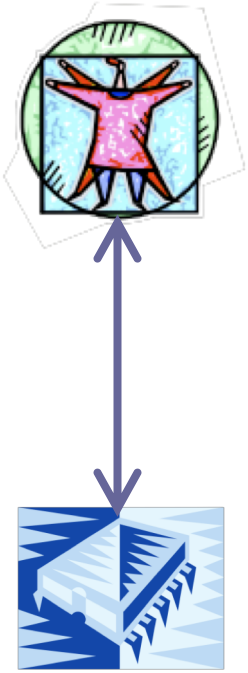
Programming Languages

- Formal languages that can be translated to machine language with a program.
  - A program is a „text“ written in a formal language
- There are a lot of different languages
  - Java, Python, C, C++, Objective C, Pascal, Modula, Perl, Basic, C#, JavaScript, Dart, Erlang, LUA uvm.

Programming Languages

- Compiler: program is translated
  - by a program
  - to machine code
  - Eg. C, C++
- Interpreter:
  - program is executed step by step by another program
  - Eg. Python, Ruby, JavaScript, Perl, LUA

Specification of Algorithms



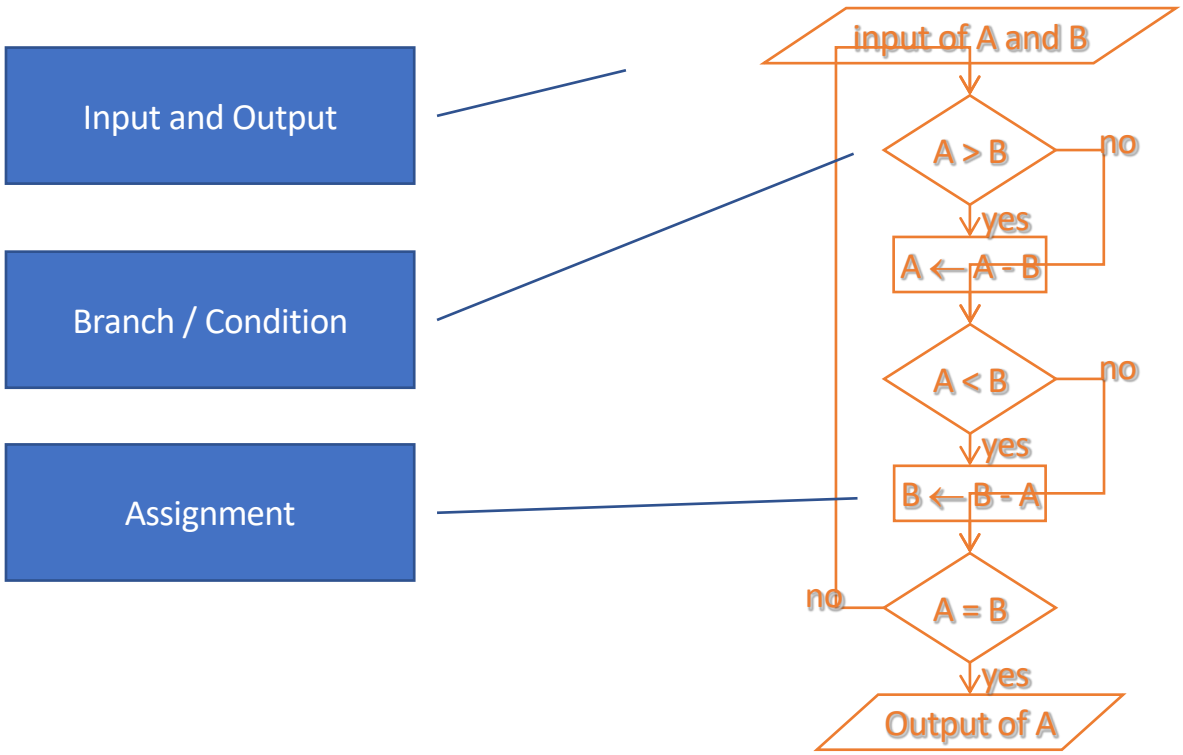
Graphical or verbal notation
Higher programming languages (like Java)
Assembly languages
Machine code
Hardware, electric signals

Verbale Notation

- Description in natural language

```
Euclidian Algorithm ggT(A, B)
  0.  Input of A and B
  1.  If A larger than B, then subtract B from A  and assign
  the result to A.
  2.  If A smaller than B then subtract A from B  and assign
  the result to B.
  3.  If A is not equal B then go to step 1
  4.  The result is A (or B)
```

Flowchart


Input and Output
Branch / Condition
Assignment
input of A and B
A > B
no
yes
A ← A - B
A < B
no
yes
B ← B - A
no
A = B
yes
Output of A

Flowchart

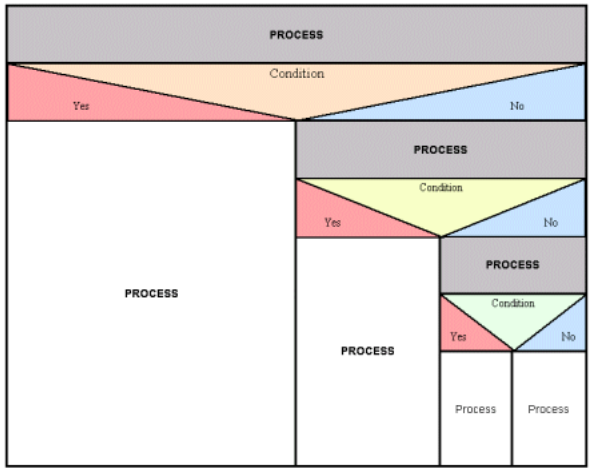
Contra

- Often unstructured, no formal framework.
- Not good for working in teams, hard to read for others
- Hard to update and revise.

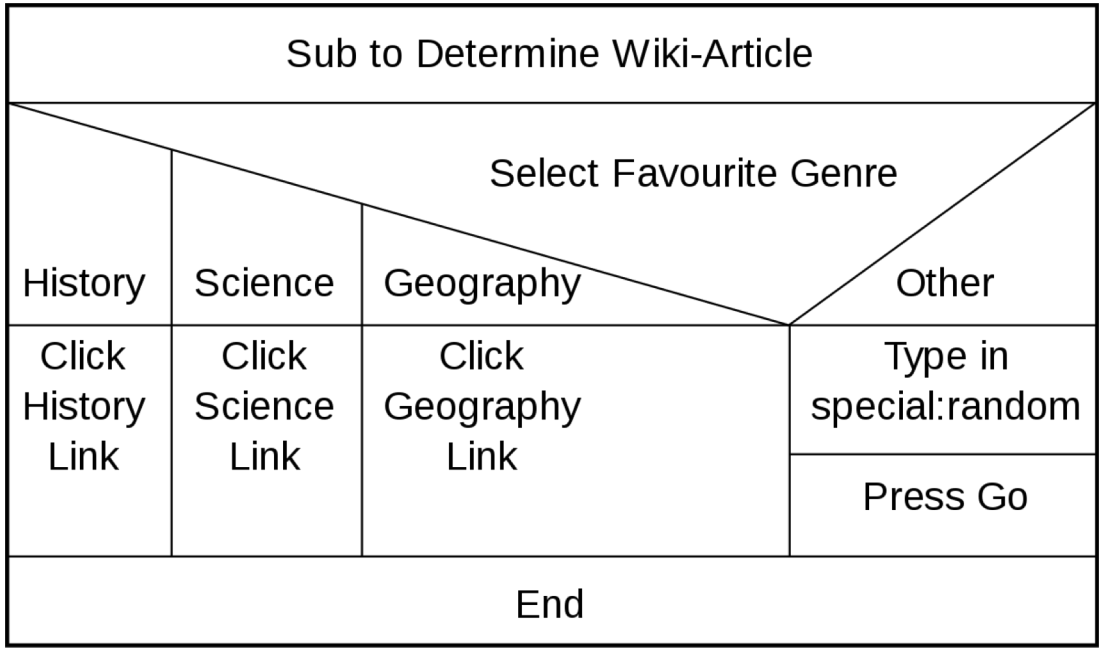
Nassi-Shneiderman-Chart

- More structured due to stronger restrictions.
- Sequence
- Branch / Condition
- + nesting!

- How to draw one:
- http://www.thern.org/projects/nassi-schneiderman/nassi.htm



Nassi-Shneiderman-Chart



Pseudocode

- Semi-formal languages
- Examples:

```
WHILE     A not equal B
    IF A > B
    THEN subtract B from A
    ELSE
            subtract A from B
    ENDIF
ENDWHILE
ggT := A
```

Questions?

- michael@simula.no
- Discord? or slack?
- Other comments, wishes?