

# Programmeren (Ectrie)

## Lecture 1: Introduction

Tommi Tervonen

Econometric Institute, Erasmus University Rotterdam

After this course, you should be able to:

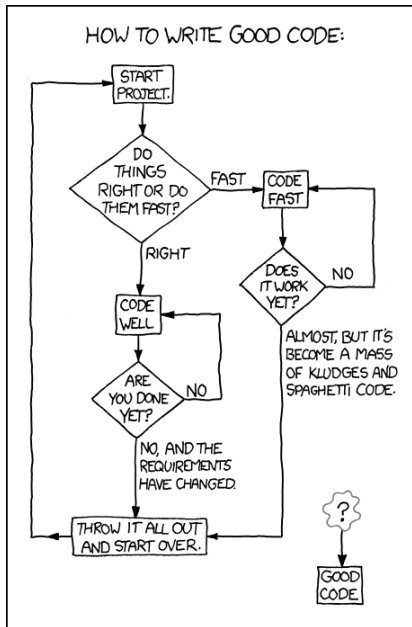
- Program econometrical models in Matlab
- Understand core concepts of imperative programming
- Explain what happens when your Matlab code is executed
- Understand what is an efficient algorithm
- Code efficient algorithms in imperative programming languages

- 7 lectures
  - Theoretical contents
  - Provide background for the exercises
- 14 exercise sessions
  - 10 large exercises: some are done in pairs
  - Come to exercises to ask questions and get help with your code
  - Due to this being a 2nd year course, 70% attendance is required
- 6 exercise lectures
  - For discussing previous weeks exercises
  - The first exercise lecture is not held, and example answer to the last exercise will be posted online

- 4 ECTS = 112h
- 7 lectures = 14h
- 14 exercise sessions = 28h
- 6 exercise lectures = 6h
- Exam = 4h
- $\Rightarrow$  Independent programming 60h = 7,5h/w

- Exercises: 50% (each 5%)
  - Individually unless stated specifically that they are to be done in pairs
  - Exercises will be published in BB after Monday's lecture (or if 2: the second latest after Tuesday's exercise session)
  - 1st, 2nd and 3rd weeks: 2/w, then 1/w
  - **Strict** deadline on Sundays @ 23.59
  - Submission via BB: *only* the source file(s). Include a comment in the beginning with your name(s) and student number(s)
- Written exam: 50%
  - Essay questions

# Making the exercises



```
boolean done=false;
boolean understood=false;
while(!understood) {
    understood = readLN()
    && readExercise();
}
while(!done) {
    done = code();
    if (!done) {
        getHelp();
    }
}
```

# Help! I can't code!

- 1 Read exercise & LN
- 2 Go to exercise sessions and get help
- 3 Code @ home
- 4 Get frustrated
- 5 Go to exercise sessions and get help
- 6 Code @ home
- 7 Get frustrated
- 8 Code @ home
- 9 Get frustrated
- 10 Go to Tommi's office
- 11 Get frustrated
- 12 Send Tommi email with topic [FEB22012] Help!

- All exercises will be automatically checked against each other *and* web resources
- Do not submit anything you haven't written yourself
- Do not submit anything that is not your idea
- The teaching assistants will not give you answers in the tutorials: they will merely help you find the answer
- “But I could've solved this problem myself, it was just faster to google the solution”



Tommi Tervonen	Lectures	H10-23
Alexander Hogenboom	Exercises	H10-21
Damir Vadic	Exercises	H7-13
Bruno Jacobs	Exercises	-

- Also: you! Participate in course discussion forums in BB to get and provide help with the exercises

Inleiding programmeren:

- Variables and methods
- Program flow
- Decisions and branching
- Control structures
- Bitwise operators
- Arithmetic operators
- Scoping

## L1 Introduction

- Practicalities
- Programming paradigms
- Scripting languages
- Introduction to types

## L2 Computing

- Numerical representation
- Introduction to complexity theory
- Insertion sort

## L3 Memory organization

- Matrix representation
- Matrix multiplication

## L4 Program correctness

- Side effects
- Pre- and post-conditions
- Loop invariants

## L5 Linear data structures

- Arrays, stacks and queues
- Linked lists

## L6 Nonlinear data structures

- Trees
- Heap
- Heapsort

## L7 Sorting & searching

- Mergesort
- Quick sort
- Binary search

- LN-TT-22012-1, available @ <http://smaa.fi/tommi/courses/prog2/> and (hopefully) in print version from the student association, loosely based on a very selected set of material from:
  - Knuth: The Art of Computer Programming (vols 1-3)
  - Cormen, Leicerson, Rivest: Introduction to Algorithms
  - Goulb, van Loan: Matrix Computations
  - Wikipedia
- Matlab book can be useful to own
- All course material is posted in <http://smaa.fi/tommi/courses/prog2/>, and also in BB (but easier to find in the actual course page)

- The exercise sessions will be guided with Matlab
- You can do most of the exercises with R, Python, or even Octave (though visualization in Octave sucks)
- Other courses require “fluency” in Matlab

Q?

“The competent programmer is fully aware of the strictly limited size of his own skull; therefore he approaches the programming task in full humility, and among other things he avoids clever tricks like the plague.”

E.W. Dijkstra



- Programming paradigms refer to the philosophy behind designing programming languages
- When you know to program with 1 language of a paradigm, others of the same paradigm are easy to learn (mostly just syntax)

- 1 Procedural / imperative paradigm (C, Pascal, Matlab, R, Fortran, Algol, Python)
- 2 Object-oriented paradigm (Java, Smalltalk, C++ partially)
- 3 Declarative paradigm, including
  - Functional programming (ML, Lisp, Haskell, Erlang, Scala, Scheme)
  - Logic programming (Prolog)

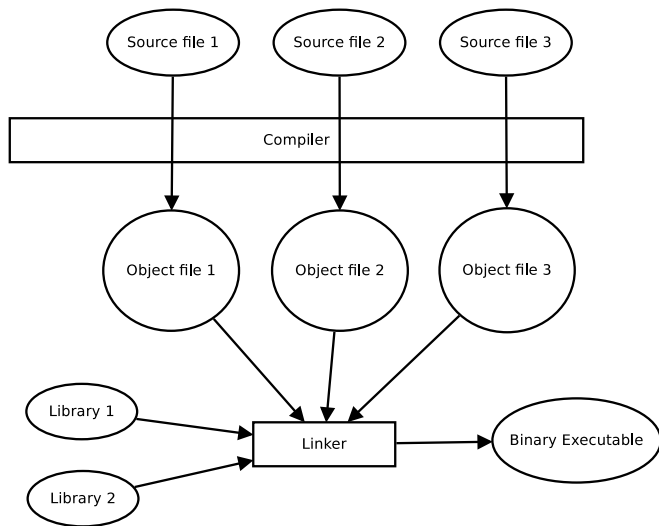
# OO vs Procedural

Object-oriented	Procedural
Design classes that communicate	Design global methods
Abstract Data Types	Data structures
Suitable for large programs	For “small” programs
Access control in language	Programmer has full access

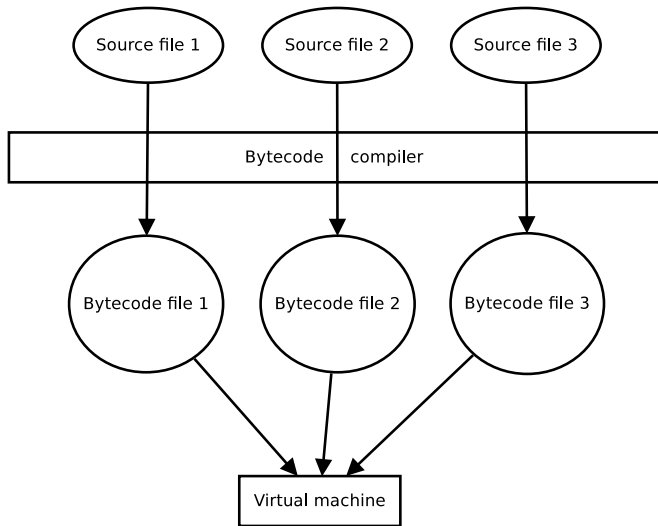
- Both are part of imperative paradigm: control flow consists of *statements* that change the state of the program
- $x = 2;$
- Imperative paradigm makes program correctness hard to prove, as  $x = 2 \neq x \leftarrow 2$

- Before source code can be executed, it needs to be *compiled* into an executable format
- The compilation can be made
  - 1 Completely in advance to a binary executable (fast)
  - 2 Partially in advance to bytecode to be executed in a virtual machine (Java, quite fast and portable)
  - 3 Run-time (slow but allows easy “modify & execute” cycles)

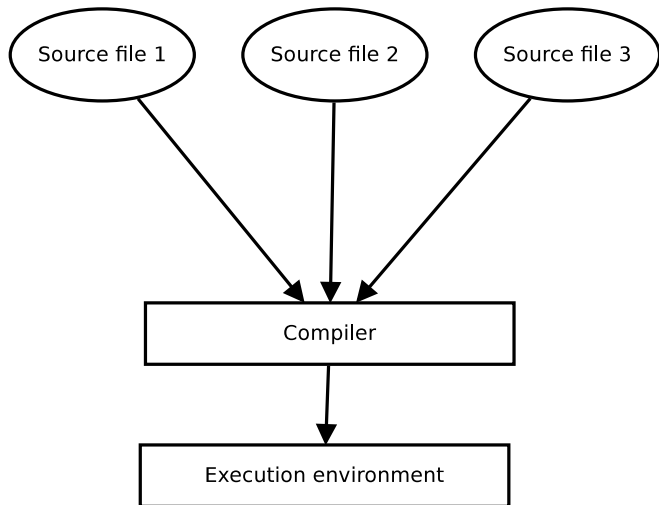
# Fully compiled languages (e.g. C)



# Bytecode compiled languages (e.g. Java)



# Runtime compiled languages (e.g. Matlab)



- In scripting languages the instructions are compiled run-time into execution statements
- Slow, as less optimization can be made
- In languages of statistical / scientific computation, you have to understand what happens “under the hood” to make efficient *and* correct code



- Typing systems form the core of programming languages - they allow construction of abstractions
- Differences in electric currency → bits → numbers → characters → objects

# Strong and weak typing

Strong typing: each variable has a type associated with it

```
int x = 2; // ok  
x = 3; // ok  
x = 's'; // error
```

Weak typing : a single variable can be assigned varying types of values

```
y = 3; % ok – no type declaration required  
y = 't'; % ok
```

- Matlab is a weakly typed language, and the following are valid expressions:

```
x = 1;
```

```
y = '1';
```

```
z = x + y;
```

- Now  $z = ?$

- Get your copy of LN from student association
- Check the newest exercises in BB
- Make sure you understand the exercises
- Familiarize yourself with Matlab
- Get coding.

(nothing to do? try <http://projecteuler.net>)