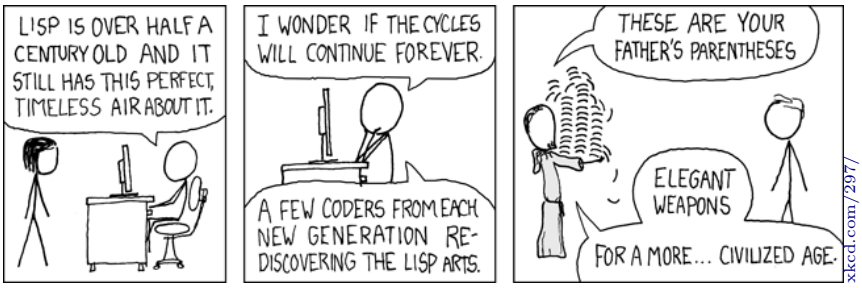
"If you want everything to be familiar, you will never learn anything new."
Rich Hickey, the creator of Clojure

Learn the cheekiest programming language!



MAT 245,340 Poetry of Programming – Puzzle-based Introduction to Functional Programming

v2022.03.28

Level:	200 or 300
Credits:	3
Instructor:	Attila Egri-Nagy
Office:	A3-4
Email:	egri-nagy@aiu.ac.jp

Course Description

The purpose of the course is **to experience the joy of computer programming**. The carefully designed, gradually more challenging problems provide a balance between the sense of difficulty and satisfaction through accomplishment. The course is serious fun that one can turn into a real and vital skill in career planning.

Experiencing the thrill of problem-solving by crafting computer programs used to be a privilege of very few people: professional software engineers, students of computer science, and a handful of hobbyists. Historically, there were reasons for this exclusiveness since programming a computer required an almost complete understanding of its inner workings. However, the situation has changed. Even programming has become user-friendly.

There is a parallel with literacy. We all learn reading and writing, but only a few of us turn into professional writers. Nowadays, it is a frequent thought in education that writing computer programs should be a skill shared by many. Similar to literacy skills, this does not imply that everyone should become a software developer. Actually, in real software development projects, the aforementioned thrilling moments of problem-solving are relatively rare, dwarfed by the more mundane tasks of the profession. Following the literacy metaphor of coding: big software systems are long and dry technical documents, while the short programs developed during the course correspond to expressive little poems. In a sense, this course aims to develop the skill of writing functional program haikus. The poetic

metaphor emphasizes **the creative side of computer programming**. Unlike the myth of 'the single solution', wrongly used in traditional Mathematics education, there is often more than one way of solving a computational problem. How to get to the desired result depends on many choices, allowing room for expressing individual style.

The unusual purpose of the course requires an unconventional choice of language. The Clojure language has an elegant, functional core allowing the students to build powerful solutions by combining a few simple ideas. The following illustration is a piece of code that implements the Caesar-shift cryptosystem and a brute force attack. The course will enable students to read, understand and write programs like this.

```clojure
(ns etudes.caesar-shift
  "Caesar shift and its brute-force attack.")

(def letters "abcdefghijklmnopqrstuvwxyz ")

(defn shift-map
  "A hash-map that maps letters to letters cyclically shifted by n."
  [n]
  (zipmap letters
          (take (count letters)
                (drop n (cycle letters)))))

(defn encrypter
  "A function the encrypts plaintext by an n-shift cipher."
  [n]
  (fn [text]
    (apply str
           (map (shift-map n) text))))

(defn decrypter
  "A decrypter is just an encrypter that shifts back."
  [n]
  (encrypter (mod (- n) (count letters))))

(defn brute-force-attack
  "Calculates all shifts and thus breaking the cipher."
  [ciphertext]
  (map
   (fn [n] ((decrypter n) ciphertext))
   (range 1 (count letters))))
```

Learning Outcomes

The main goal of the course is *to give a direct access to the deepest ideas of functional programming through problem solving*. On the successful completion of this course, in terms of **programming skills**, it is expected that the students will be able to:

1. write snippets of Clojure code for solving logic puzzles and programming problems;
2. read and appreciate code written by others;
3. to use fundamental sequential and associative data structures
4. understand the importance of immutable data structures and consequently the principles of the now pervasive version control systems;

5. have a thorough understanding of recursion and self-reference;
6. and be able to work with higher-order functions (functions working on functions).

In addition, regarding more general **life skills** it is expected that the students will be able to:

7. improve their general problem solving skills;
8. build up emotional intelligence and mental resilience for facing challenges;
9. improve communication skills;
10. and see connections between programming, advanced mathematics and real-world problems.

The course is not planned to provide all the skills necessary for developing complex software applications. However, it will give a fast-track entry point to the post-object-oriented multi-threaded software development world. In case of an IT-oriented career choice, technical knowledge is easier to obtain later, once the deep ideas are comprehended.

Tentative Schedule

WEEK	TOPICS
1	**Historical overview:** Lisp and Clojure. **Introducing the REPL:** interacting with the computer through the command line. Mathematical functions and function composition. Function calls. Clojure as a calculator. **Characters and strings. Predicates.**
2	**List, the most fundamental data collection:** constructing lists, accessing elements, quoting. **Vector as an associative data structure.**
3	**Symbol bindings.** Introducing `def` for binding values to symbols. Quoting revisited. Using `let` statements for temporary bindings. **Defining functions.** Functions that produce text output containing results of some numerical calculations.**Associations:** giving meaning to symbols by `def`. **Associative data structures:** vector, hash-map, hash-set.
4	**Using `apply`:** Unpacking collections for functions with variadic number of arguments; manual function application. **Writing predicate functions.**
5	**Lazy Sequences of numbers:** using `range` for creating sequences of numbers, `take`, `drop`,`take-while`, `drop-while`. **Functional Collection Processing:** `map`, `filter`, `remove`. **Truthy, falsey.**
6	**Decision making:** conditionals `if`, `cond`. **Logic operators:** `and`, `or`, `not`.
7	**Folding:** `reduce`, re-implementing list functions.
8	**MIDTERM TEST**

9	**Hash-map:** a data structure for free associations. **Hash-sets:** collections for membership testing.
10	**Sequence abstraction.**
11	**Recursion:** self-reference, functions calling themselves.
12	**Destructuring:** pattern matching for function arguments. **Point-free style:** excessive use of higher order functions.
13	**List comprehension:** `for`. **Side-effects:** random numbers, printing, `doseq`.
14	CLOJURE **and art:** OVERTONE – collaborative programmable music; QUIL – interactive drawings and animations.
15	**FINAL EXAM** Session for bonus problems.

Textbooks

A tutorial text, reading exercises, practice problems and challenging projects are available in electronic format on the website of this course.

`https://egri-nagy.github.io/popbook/`

There is a YouTube channel for prerecorded lectures.

`https://www.youtube.com/playlist?list=PLI-mrGTUXmHXeKhy6UGdDxIKwM8L4MTbq`

Optional, beginner-friendly textbooks containing more material:

- Daniel Higginbotham: **Clojure for the Brave and True – Learn the Ultimate Language and Become a Better Programmer**, No Starch Press, 2015, ISBN: 978-1-59327-591-4, `http://www.braveclojure.com/clojure-for-the-brave-and-true/`, freely available online
- Carin Meier: **Living Clojure – An Introduction and Training Plan for Developers**, O'Reilly Media, 2015, ISBN: 978-1-4919-0904-1

Software & Resources

CLOJURE can be used from a browser on all platforms without installing any software package, or installed on any laptop or desktop computer. Many software tools are freely available, fully documented and open-source.

- `https://www.clojure.org/` – the official home of the CLOJURE language
- `https://www.maria.cloud/` – coding environment for beginners, the defualt choice for this course
- `https://www.clojuredocs.org/` – community maintained example-based documentation

John McCarthy (1927-2011) computer scientist and cognitive scientist. One of the founders of the discipline of Artificial Intelligence. In late 1950s he invented Lisp, a language famous for its simple syntax. Everything in the language is expressed as a list of symbols. Hence the name, LISt Processing. Clojure is a modern Lisp, featuring built-in associative data structures in addition to lists.

Delivery Format

Guided and independent problem solving and code writing in *computer lab*. Students work on their solutions, the instructor gives comments, hints for the next step when requested. *Group work is encouraged!* Lecturing part is minimized, restricted to short introduction of concepts. Numerous examples for each concept are presented as *live coding*, which can be followed by students on their computers. As the course proceeds, solution walkthroughs will discuss previous assignments. This is not about presenting the 'official' solution, rather exploring the possibilities for different solutions, especially in the light of techniques learned after the assignment.

Assessment Components

Lab work 40% continuous evaluation of the assignment programming work in class; submitting a solution is done by presenting the working code on a computer to the instructor; if a solution is not completed in its due class, it can be finished as a homework assignment and presented later receiving 90% of the marks for that problem; while submitting a solution after the official solution walkthrough gets 80%

Midterm & Final exams 30% each (60% in total) There is a difference for the between on-campus and online exams.

On-campus traditional paper based exam with code reading exercise problems; sample tests are provided and there is a mock exam to get familiar with the format;

Online live coding exercises (paired with the instructor) from a pre-selected collection of problems.

300 Level Topics

Students taking the course at a 300 level have to do an individual project in addition to the assignments. These involve independent exploration of more advanced topics. The project can be one of the following ideas, or chosen by the student.

Parsing, creating programming languages using the InstaParse library https://github.com/Engelberg/instaparse

Logic programming using `core.logic` https://github.com/clojure/core.logic

Go AIs writing a game AI or contributing to the LambdaGo project https://github.com/egri-nagy/lambdago

Requirements

Experience of learning a foreign language, and minimal math background (e.g. regular high school math suffices). The course is built on the premise that **'Anyone can code!'**, which has been proven correct over several semesters. Most importantly, it is expected that students will continuously communicate and share their thought processes with others and the instructor.

Related courses (AILA)

The following courses and the Poetry of Programming can mutually benefit from each other, giving opportunities for synthesizing different knowledge gained from different subjects.

MAT150 College Algebra. This introductory course revisits refines the notions of functions (learnt in high school) and introduces function composition (in preparation for Calculus). Functional programming is also based on these ideas, thus people attending this algebra course learn functional programming as well (may not realize that though). In some sense, mathematics in general and algebra in particular can be considered as a programming language designed for humans.

MAT200 Statistics. Statistics courses are becoming more computational (going beyond the simple use of spreadsheets), therefore programming experience is a big advantage.

MAT230 Igo Math – Natural and Artificial Intelligence and the Game of Go. This course introduces fundamental artificial intelligence algorithms on a theoretical, mathematical level. Combined with practical programming knowledge, students can develop their own AI engines.

MAT240 Mathematics Behind The Technological Society. This is a discrete mathematics course for explaining how computers work from first principles. Today, programming is done on a higher level, so it is not impossible without this knowledge. However, a more comprehensive and deeper understanding computational processes on all levels. It is also a good way to learn about more complex algorithms.

MAT250 Calculus. Seemingly, a traditional advanced Mathematics course has nothing to do with programming. However, symbolic differentiation uses the same general techniques as functional programming: functions are composed of simpler functions.

CCS125 Programming Principles. This course introduces an industry mainstream object-oriented programming language and its tooling for real-world software development. Taking both programming courses will give a wide spectrum of programming techniques. However, since the two languages are quite different, taking them at the same time might be challenging (though absolutely possible as several students did that combination in the past successfully).

Personal development (AILA)

The course is designed to be challenging, therefore it is expected that students sooner or later will 'hit a wall' in their progress. This is not contradicting the statement that everyone can do programming. Overcoming difficulties is a defining characteristic of computer programming activity, therefore it is unavoidable if one wants to have a real experience of coding. Taking challenges builds up mental stamina, which is beneficial in other areas of study.

Further Information

The course was presented at tech conferences, receiving positive feedback. These are the recorded videos.

Clojure conj 2017

The poetry of programming

https://youtu.be/XRjPnuPv6xo

The philosophy of (functional) programming

https://youtu.be/-yGHsXSgYdg