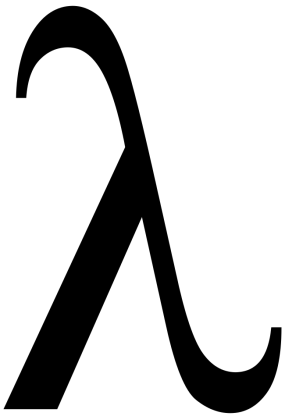
Conceptes Avançats de Programació



Jordi Delgado, Gerard Escudero,

Departament de Ciències de la Computació (UPC)

Conceptes Avançats de Programació

Llegiu la guia docent

Guió: Programació funcional (en Clojure)

- Tema 1: **Clojure: Introducció**
- Tema 2: **Funcions *first-class***
- Tema 3: ***Closures* - Model d'entorns**
- Tema 4: **Tècniques de programació amb funcions d'ordre superior**
- Tema 5: **Immutabilitat d'Estructures de Dades**
- Tema 6: **Seqüències *Lazy***
- Tema 7: **Macros**

A banda dels exercicis de les sessions de laboratori i els que hi ha a les transparències de teoria, aquí teniu una **llista d'exercicis** addicional.

Finalment: **Consells generals per a la programació funcional en Clojure**

Conceptes Avançats de Programació

Teoria:

Classes magistrals amb recolzament de transparències (pensades com a guió de la sessió, no com a apunts), basades en:

- **The Joy of Clojure - 2nd edition**, Fogus, Michael; Houser, Chris, Manning, 2014. ISBN: 9781617291418
- **Clojure Programming: Practical Lisp for the Java World**, Emerick, Chas; Carper, Brian; Grand, Christophe, O'Reilly Media, 2012. ISBN: 9781449394707
- **Mastering Clojure Macros: Write Cleaner, Faster, Smarter Code**, Jones, Colin, The Pragmatic Programmers, 2014. ISBN: 9781941222225. Capítols 1-4.

Aquests llibres *haurien* d'estar a la biblioteca.

Avaluació teoria:

Examens parcial i final. **Nota de teoria**: NT = MAX(final, (parcial + final)/2)

Conceptes Avançats de Programació

Laboratori:

Jutge - https://jutge.org

- Curs **Conceptes Avançats de Programació (Clojure)**

Sessions Laboratori - gebakx.github.io/cap

Avaluació laboratori:

Proposarem un exercici de dificultat mitjana per cada tema (excepte el primer), i caldrà que lliureu la solució de tots ells en grups de dos persones al Racó. Mirarem de fer públics els enunciats en finalitzar cada tema.

La mitjana de les notes dels projectes serà la nota del laboratori (serà una nota individual, malgrat les solucions dels projectes es facin en grup).

Conceptes Avançats de Programació

Avaluació total:

Sigui NT la nota de teoria, i NL la nota de laboratori

Nota CAP: 0.8 $\times$ NT + 0.2 $\times$ NL

Conceptes Avançats de Programació

CAP NO és un curs de Clojure

Clojure és un llenguatge de programació molt gran, adequat al *món real*.

Nosaltres triarem el **subconjunt que ens convingui**.

Aspectes de Clojure que no veurem, o només per sobre:

- Namespaces
- Java interop
- Specs
- Concurrència/Multi-Threading: refs & STM, Agents
- Protocols i Datatypes: Interfaces, protocols, datatypes, records.
- Multimètodes
- Metadata
- ... i més coses

Conceptes Avançats de Programació

Treballar amb Clojure

- the Clojure CLI: És l'utilitat de línia de comandes per gestionar projectes en Clojure que farem servir a CAP. És una de les eines més utilitzades i *està instal·lada als laboratoris de la FIB*.

- També es fa servir molt una altra eina anomenada leiningen. Quan busqueu informació *on-line* sobre Clojure us la trobareu. La podeu fer servir també, si voleu.

Editors

Qualsevol editor una mica actual té algún *plug-in* o similar per poder editar Clojure amb comoditat. Per exemple:

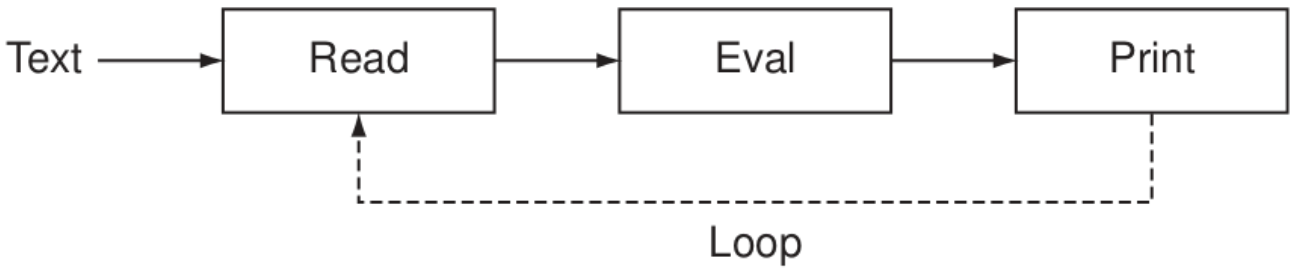
- Calva per a *Visual Studio Code*.

- Cider per a *Emacs*.

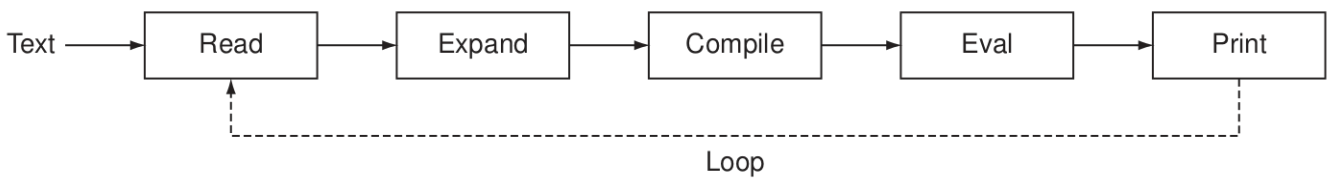
Conceptes Avançats de Programació

Treballar amb Clojure

És habitual fer servir el **REPL** (*Read, Eval, Print Loop*) en treballar amb Clojure. Provem les funcions que definim fent-ne prototipus i les testem. Accedirem al **REPL** via terminal o via editor/IDE. Sigui com sigui, nosaltres el farem servir molt.



encara que en realitat el que fa és:



Font: *The Joy of Clojure*, p. 15

Conceptes Avançats de Programació

Consells generals per a la programació funcional en Clojure:

1. *Avoid direct recursion. The JVM can't optimize recursive calls, and Clojure programs that recurse will blow their stack.*

2. *Use recur when you're producing scalar values or small, fixed sequences. Clojure will optimize calls that use an explicit recur.*

3. *When producing large or variable-sized sequences, always be lazy. (Do not recur.) Then, your callers can consume just the part of the sequence they actually need.*

4. *Be careful not to realize more of a lazy sequence than you need.*

5. *Know the sequence library. You can often write code without using recur or the lazy APIs at all.*

6. *Subdivide. Divide even simple-seeming problems into smaller pieces, and you'll often find solutions in the sequence library that lead to more general, reusable code.*

Font: *Programming Clojure, 3rd ed.*, Alex Miller with Stuart Halloway and Aaron Bedra, Pragmatic 2018, p. 85