Paradigme de Programare

Ș.l. dr. ing. Andrei Olaru

Departamentul Calculatoare

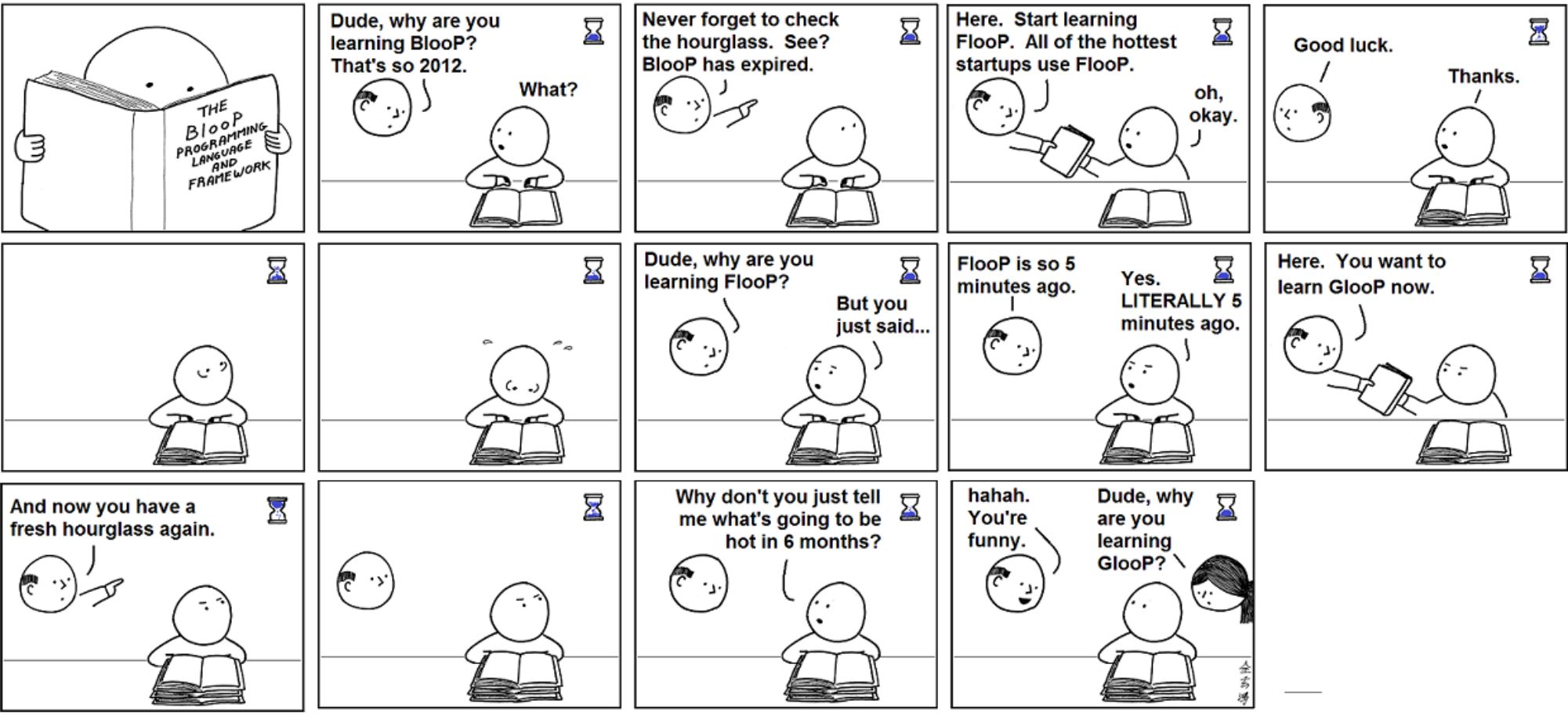
2015 – 2016, semestrul 2

Cursul 1: Introducere

Cursul 1: Introducere

BlooP and FlooP and GlooP

[http://abstrusegoose.com/503]



[(CC) BY-NC abstrusegoose.com]

Ce studiem la PP?

Elemente pe care le vom studia

- Paradigma funcțională și paradigma logică, paradigme de tip declarativ, în contrast cu paradigma imperativă.

Elemente pe care le vom studia

- Paradigma funcțională și paradigma logică, paradigme de tip declarativ, în contrast cu paradigma imperativă.
- Racket: introducere în programare funcțională
- Calculul λ ca bază teoretică a paradigmei funcționale
- Racket: întârzierea evaluării și fluxuri

Elemente pe care le vom studia

- Paradigma funcțională și paradigma logică, paradigme de tip declarativ, în contrast cu paradigma imperativă.
- Racket: introducere în programare funcțională
- Calculul λ ca bază teoretică a paradigmei funcționale
- Racket: întârzierea evaluării și fluxuri
- Haskell: programare funcțională cu o sintaxă avansată
- Haskell: evaluare lenesă și fluxuri
- Haskell: tipuri, sinteză de tip, și clase

Elemente pe care le vom studia

- Paradigma funcțională și paradigma logică, paradigme de tip declarativ, în contrast cu paradigma imperativă.
- Racket: introducere în programare funcțională
- Calculul λ ca bază teoretică a paradigmei funcționale
- Racket: întârzierea evaluării și fluxuri
- Haskell: programare funcțională cu o sintaxă avansată
- Haskell: evaluare lenesă și fluxuri
- Haskell: tipuri, sinteză de tip, și clase
- Prolog: programare logică
- LPOI ca bază pentru programarea logică
- Prolog: strategii pentru controlul execuției

Elemente pe care le vom studia

- Paradigma funcțională și paradigma logică, paradigme de tip declarativ, în contrast cu paradigma imperativă.
- Racket: introducere în programare funcțională
- Calculul λ ca bază teoretică a paradigmei funcționale
- Racket: întârzierea evaluării și fluxuri
- Haskell: programare funcțională cu o sintaxă avansată
- Haskell: evaluare lenesă și fluxuri
- Haskell: tipuri, sinteză de tip, și clase
- Prolog: programare logică
- LPOI ca bază pentru programarea logică
- Prolog: strategii pentru controlul execuției
- Algorimi Markov: calcul bazat pe reguli de transformare

Exemplu

Exemplu

Ex Exemplu

Să se determine dacă un element e se regăsește într-o listă L ($e \in L$)

Modelare funcţională (1)

Racket:

```
1  (define memList (lambda (e L)
2      (if (null? L)
3          #f
4          (if (equal? (car L) e)
5              #t
6              (memList e (cdr L))
7              ))
8      ))
```

Modelare funcțională (2)

λPP

Haskell

```
1  memList x [] = False
2  memList x (y:t) = x == y || memList x t
```

Modelare logică

λPP

Prolog:

```
1  memberA(E, [E|_]) :- !.
2  memberA(E, [_|L]) :- memberA(E, L).
```

De ce studiem această materie?

De ce?

I suppose it is tempting, if the only tool you have is a hammer, to treat everything as if it were a nail.

The law of instrument – Abraham Maslow

De ce?
Mai concret

· până acum ați studiat paradigma imperativă (legată și cu paradigma orientată-obiect)

→ un anumit mod de a privi procesul de rezolvare al unei probleme și de a căuta soluții la probleme de programare.

· paradigmele funcțională și logică (și paradigma declarative în general) oferă o gamă diferită (complementară?) de unelte → alte moduri de a rezolva anumite probleme.

⇒ o pregătire ce permite accesul la poziții de calificare mai înaltă (arhitect, designer, etc.)

De ce?
Sunt aceste paradigme relevante?

- evaluarea lenesă → prezentă în Python (de la v3), .NET (de la v4)
- funcții anonime → prezente în C++ (de la v11), C#/.NET (de la v3.0/v3.5), Dart, Go, Java (de la JDK8), JS/ES, Perl (de la v5), PHP (de la v5.0.1), Python, Ruby, Swift.
- Prolog și programarea logică sunt folosite în software-ul modern de A.I., e.g. Watson.
- În industrie sunt utilizate limbaje puternic funcționale precum Erlang, Scala, F#, Clojure.
- Limbaje multi-paradigmă → adaptarea paradigmei utilizate la necesități.

Modele → paradigme → limbaje

Modele de calculabilitate

C, Pascal → procedural J, C++, Py → orientat-obiect	→ paradigma imperativă	→ Mașina Turing
Racket, Haskell	→ paradigma funcțională	→ Mașina λ
Prolog	→ paradigma logică	→ FOL + Resolution
CLIPS	→ paradigma asociativă	→ Mașina Markov

echivalente !

T | Teza Church-Turing: efectiv calculabil = Turing calculabil

Ce înseamnă paradigma de programare

Ce diferă între paradigme?

- diferă sintaxa

Ce înseamnă paradigma de programare

Ce diferă între paradigme?

- diferă sintaxa ← aceasta este o diferență între limbaje

Ce înseamnă paradigma de programare

Ce diferă între paradigme?

- diferă sintaxa ← aceasta este o diferență între limbaje
- diferă modul de construcție al expresiilor

Ce înseamnă paradigma de programare

Ce diferă între paradigme?

- diferă sintaxa ← aceasta este o diferență între limbaje
- diferă modul de construcție al expresiilor
- diferă structura programului

Ce înseamnă paradigma de programare

Ce diferă între paradigme?

- diferă sintaxa ← aceasta este o diferență între limbaje
- diferă modul de construcție al expresiilor
- diferă structura programului ← omoiconicitate: structura programului este similară cu structura datelor și a expresiilor individuale

Ce înseamnă paradigma de programare

Ce caracterizează o paradigmă?

- modul de tipare al valorilor
- ordinea de evaluare (generare a valorilor)
- modul de legare al variabilelor (managementul valorilor)

· Paradigma de programare este dată de stilul fundamental de construcție al structurii și elementelor unui program.

Ce vom studia?
Conținutul cursului

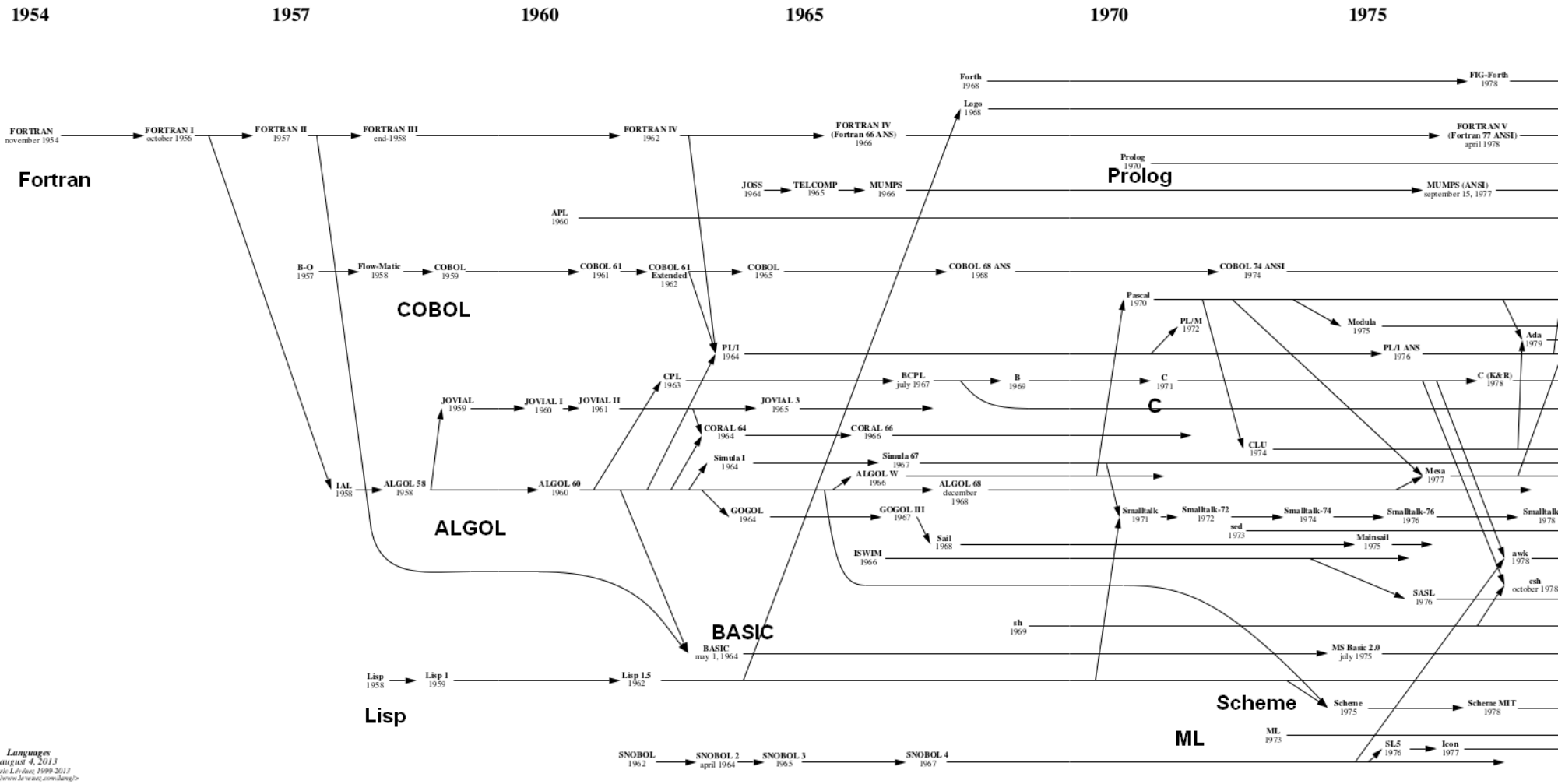
1. Diverse perspective conceptuale asupra noţiunii de calculabilitate efectivă $\rightarrow$ modele de calculabilitate.

2. Influenţa perspectivei alese asupra procesului de modelare şi rezolvare a problemlor $\rightarrow$ paradigme de programare.

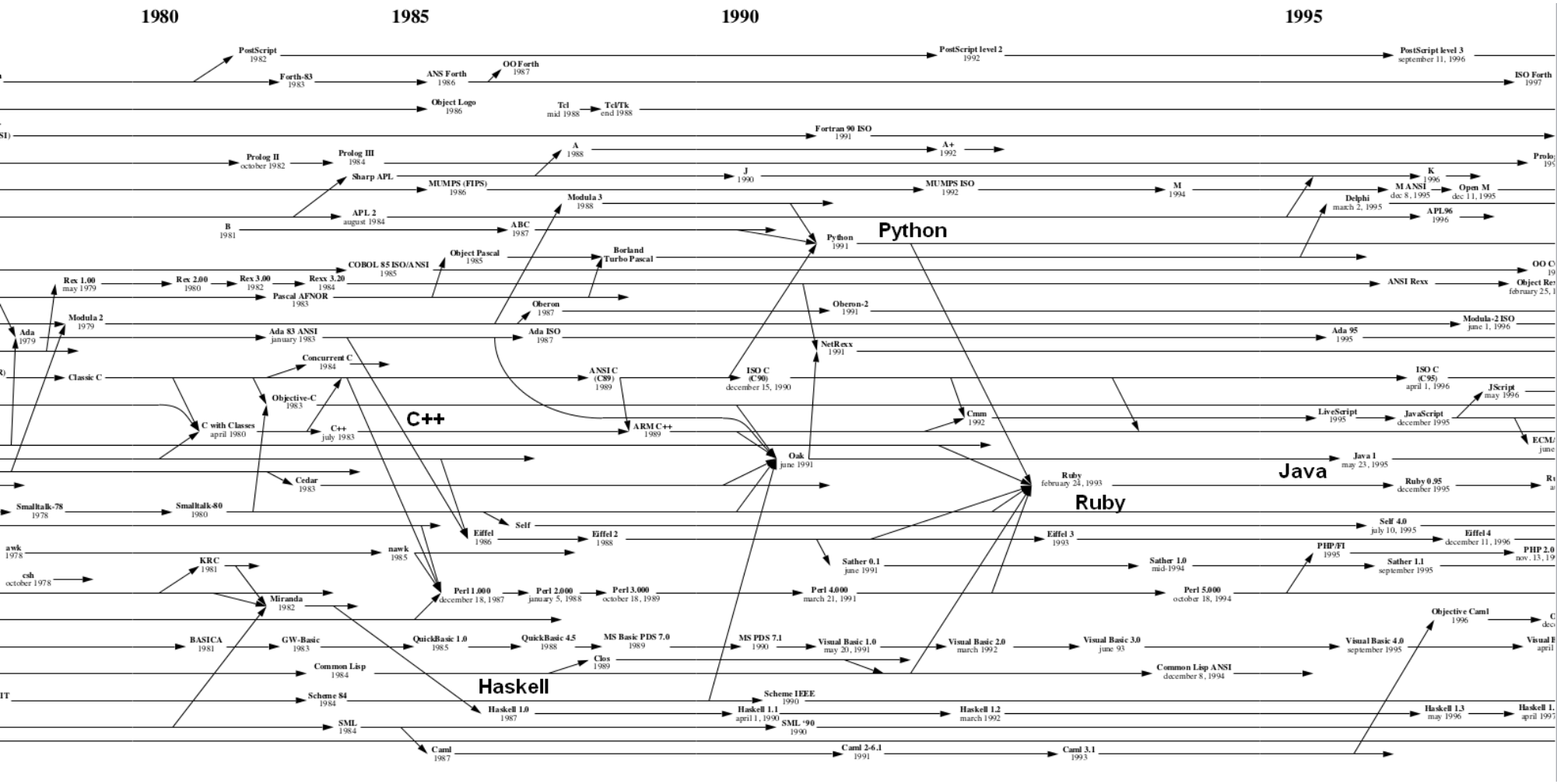
3. Limbaje de programare aferente paradigmelor, cu accent pe aspectul comparativ.

Istoric: Paradigme și limbaje de programare

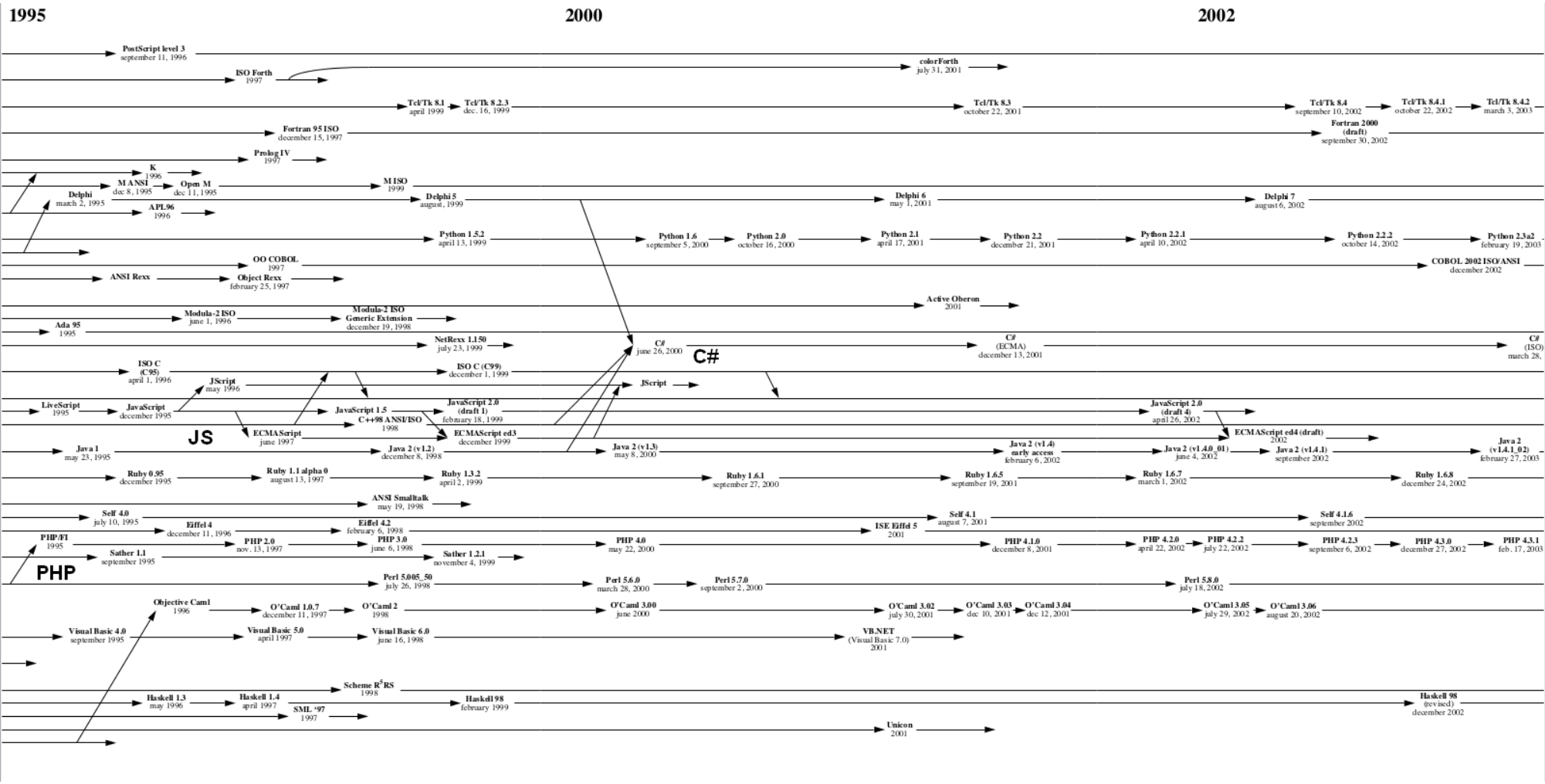
Istorie
1950-1975



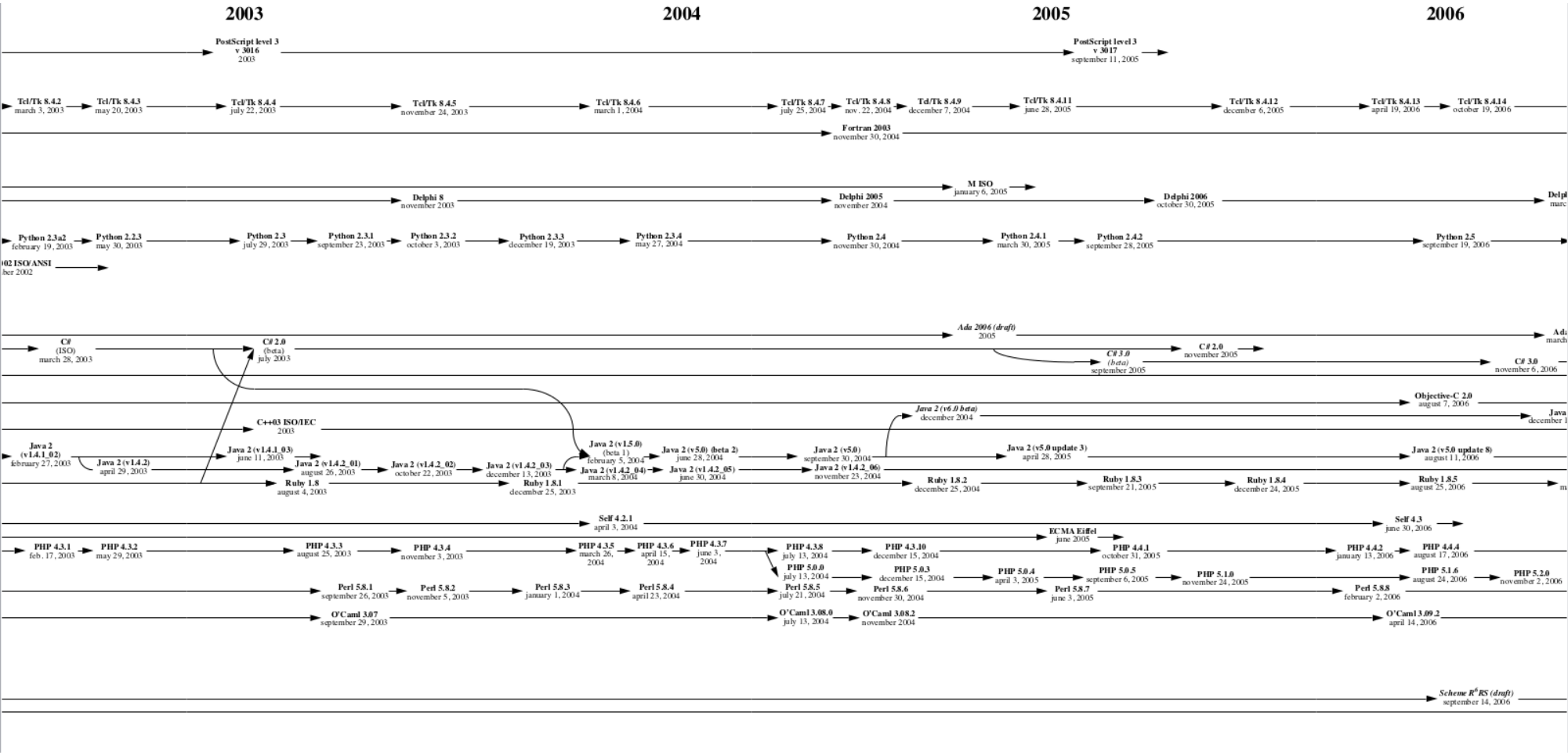
Istorie
1975-1995



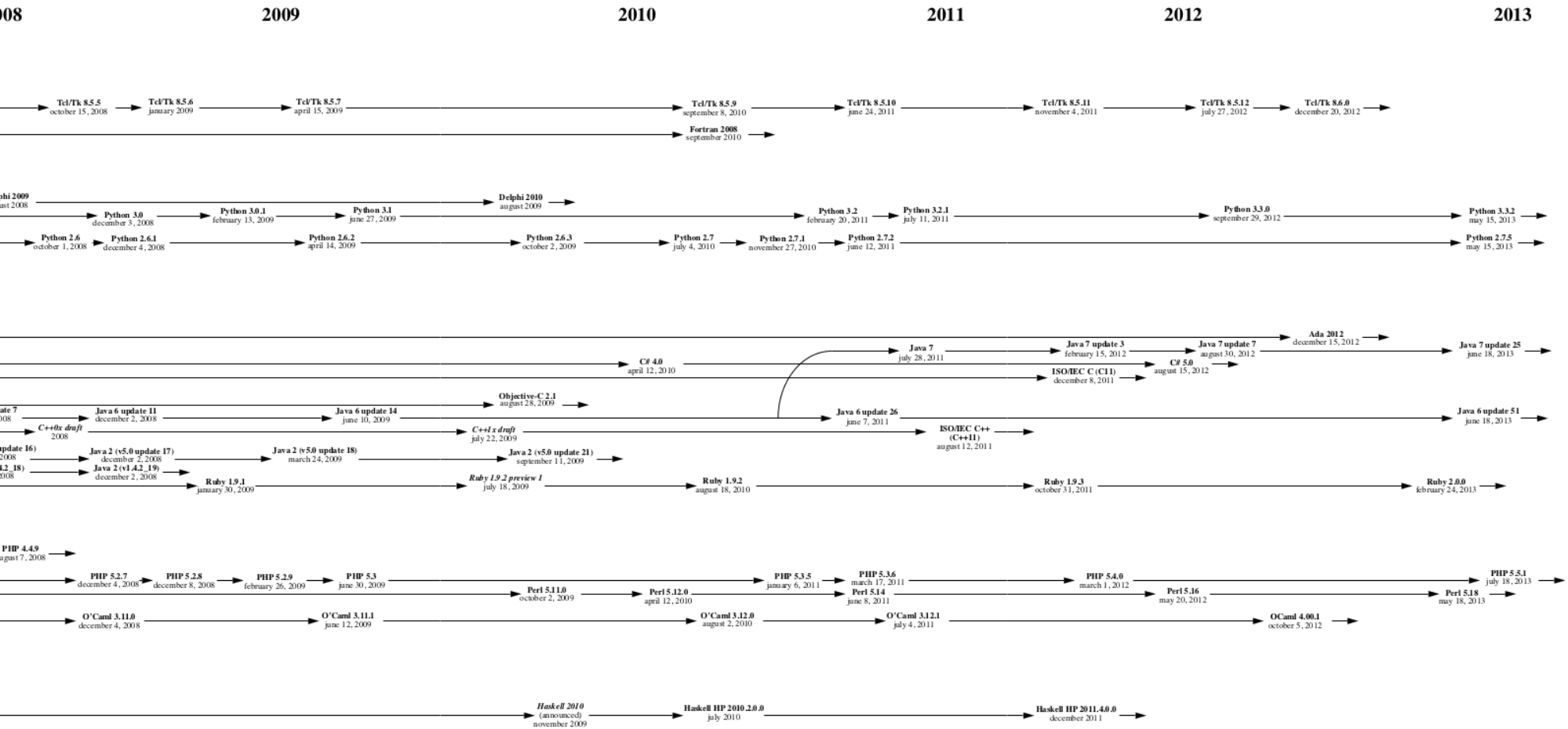
Istorie
1995-2002



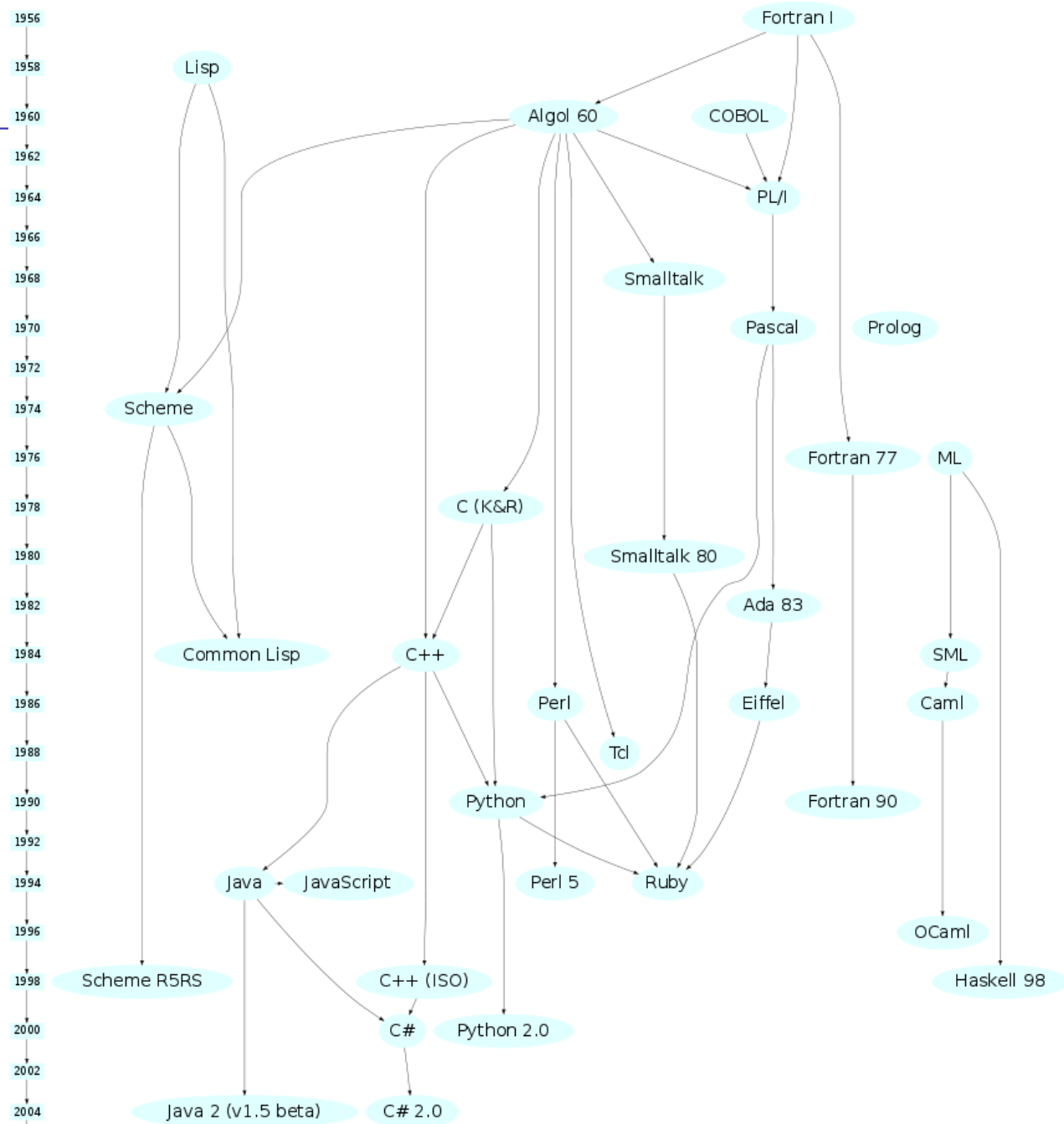
Istorie
2002-2006



Istorie
2006-2013



Istorie
’56-’04 pe scurt



Istorie
Resurse

- imagine navigabilă (slides precedente):

  [http://www.levenez.com/lang/]

- poster (până în 2004):

  [http://oreilly.com/pub/a/oreilly/news/languageposter_0504.html]

- arbore din slide precedent și arbore extins:

  [http://rigaux.org/language-study/diagram.html]

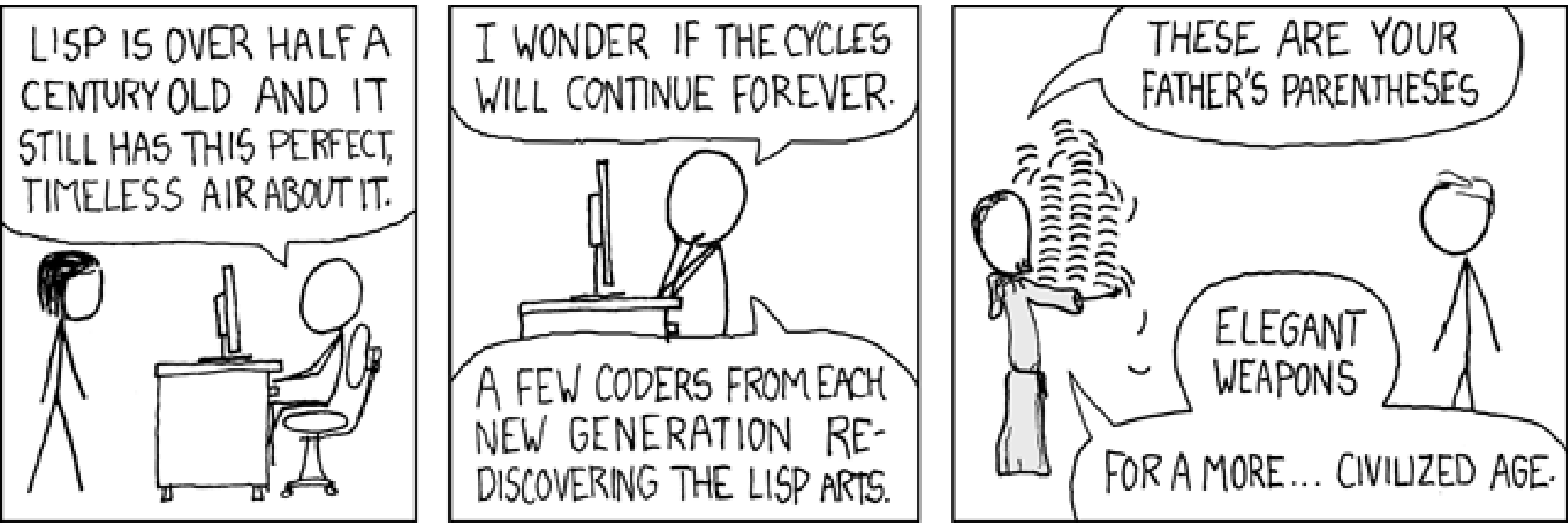
- Wikipedia:

[http://en.wikipedia.org/wiki/Generational_list_of_programming_languages]

Introducere în Racket

Lisp cycles

[http://xkcd.com/297/]



Racket
din 1975

- funcțional
- dialect de Lisp
- totul este văzut ca o funcție
- constante – expresii neevaluate
- perechi / liste pentru structurarea datelor
- apeluri de funcții – liste de apelare, evaluate
- evaluare aplicativă, funcții stricte, cu anumite excepții

Organizare

Unde găsesc informații?
Resurse de bază

λPP

`http://elf.cs.pub.ro/pp/`

Regulament: `http://elf.cs.pub.ro/pp/regulament`

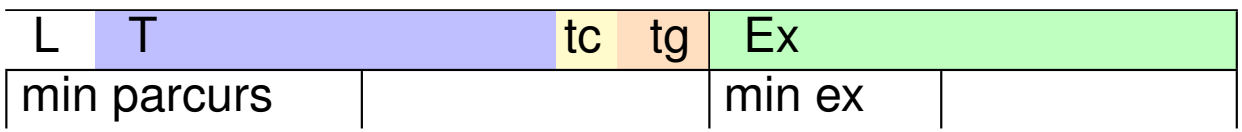
Teme și forumuri: curs.cs → L-2-PP-CA-CC
`http://cs.curs.pub.ro/2015/course/view.php?id=81`

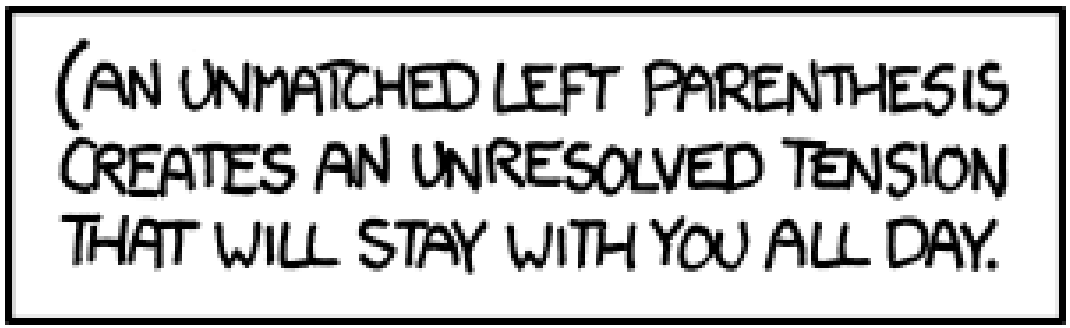
Elementele cursului sunt comune la seriile CA și CC.

Notare

mai multe la `http://elf.cs.pub.ro/pp/regulament`

- Laborator: 1p ← cu bonusuri, dar maxim 1p total (cu extensie până la 1.5 pentru performanță susținută)
- Teme: 4p (3 × 1.33p) ← cu bonusuri, dar în limita a maxim 6p pe parcurs
- Teste la curs: 0.5p ← punctare pe parcurs, la curs, din cursul anterior
- Test din materia de laborator: 0.5p ← test grilă, de cunoaștere a limbajelor
- Examen: 4p ← limbaje + teorie





\+ Dați feedback la acest curs aici:

[http://goo.gl/forms/SjDsW06v5J]