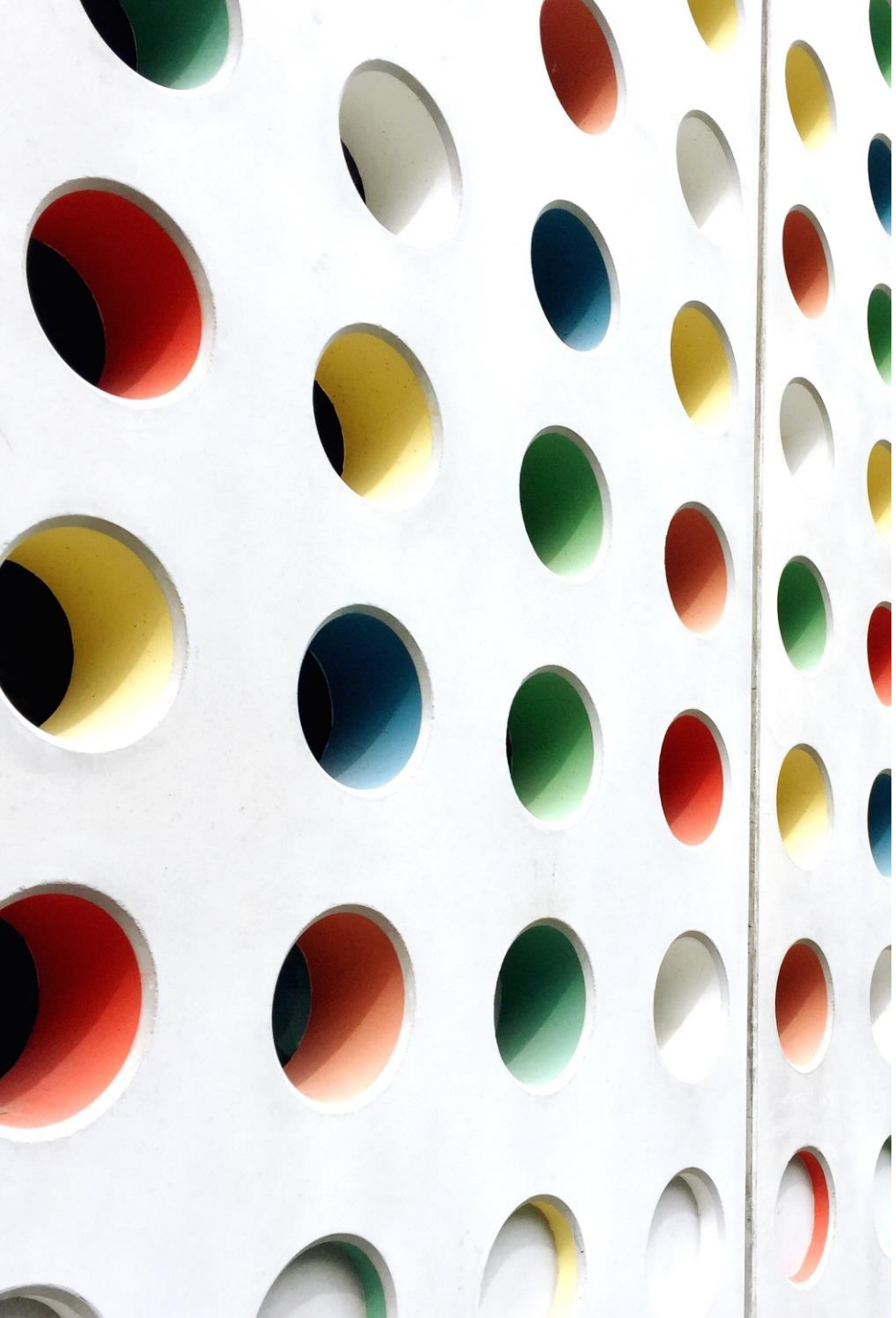


Logičko i funkcijsko programiranje

ŠKOLSKA 2023/24

Organizacija predmeta

Dve velike oblasti:

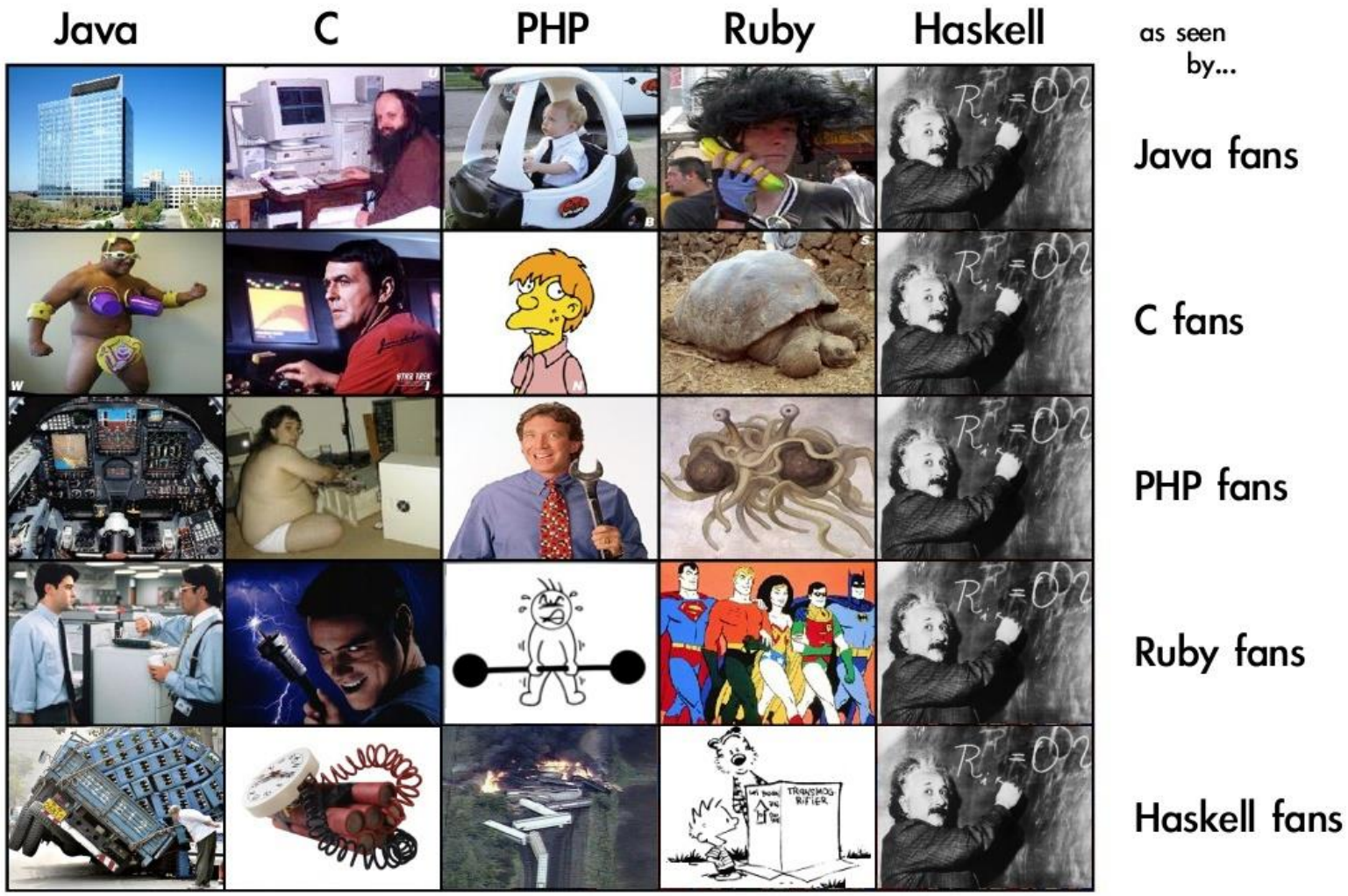
- Funkcijsko programiranje – Haskell (na vežbama i Scala)
- Logičko programiranje – Prolog

Predispitne obaveze:

- Dolasci – 4 poena
- Kolokvijumi – 23 + 23 poena
  - Obavezan bar jedan zadatak u Scala programskm jeziku
  - Minimum po kolokvijumu 10 poena
- Seminarski rad – 20 poena

Završni ispit – 30 poena

- Uslov – 26 poena na kolokvijumima i dolascima


Java
C
PHP
Ruby
Haskell
as seen by...
Java fans
C fans
PHP fans
Ruby fans
Haskell fans

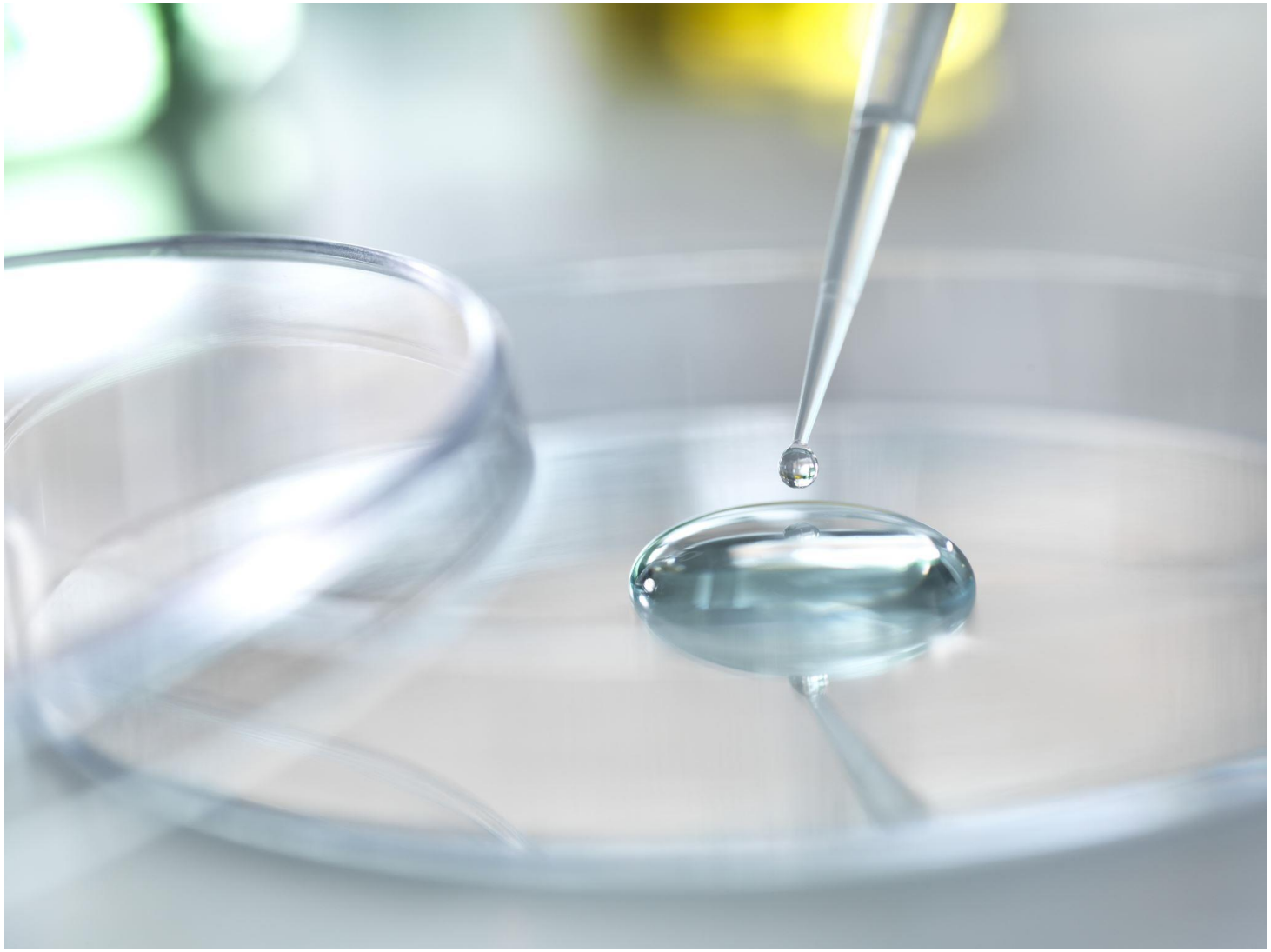
Zašto funkcijsko programiranje?

Omogućava pisanje jasnih, konciznih programa sa abstrakcijom viskog nivoa

Podržave „reciklirane“ softverske komponente

Podstiče korišćenje formalne verifikacije

U ovo se mogu uklopiti i objektno-orijentisani jezici



Zašto funkcijsko programiranje?

„Petrijeva šolja“ za ostale jezike

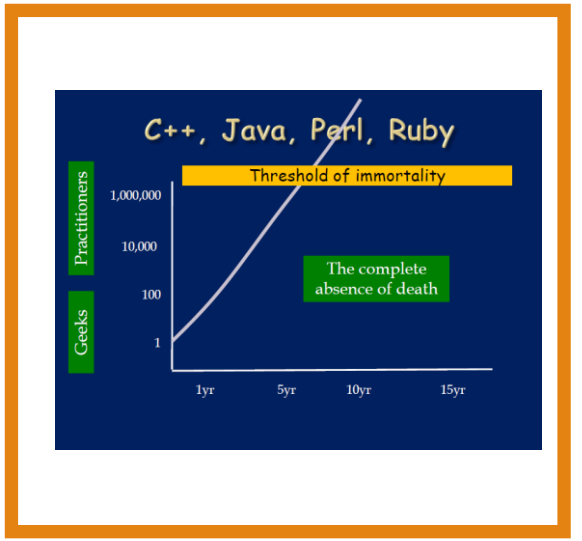
Šta je funkcijsko programiranje?

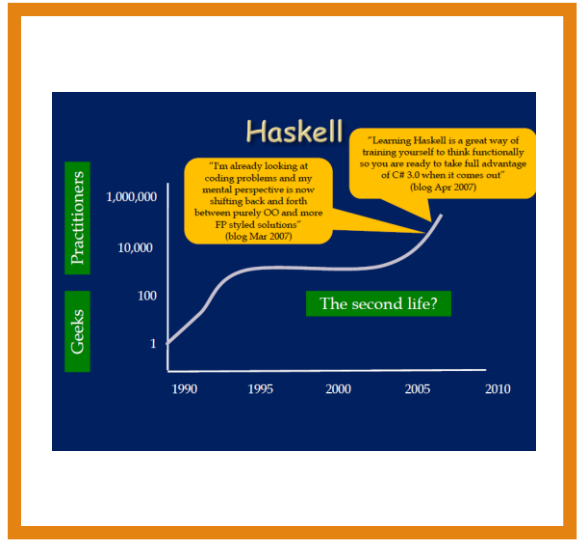
Stil programiranja zasnovan na definisanju (matematičkih) funkcija i primeni funkcija za prosleđene argumente

Funkcijski jezik podržava i ohrabruje funkcijski stil programiranja

Životni vek programskih jezika

FROM SIMON PEYTON JONES' TALK





Razvoj funkcijskih jezika

1930 – Alonzo Church je razvio lambda (λ) račun

- Jednostavna ali moćna matematička teorija funkcija
- Omogućava vrlo precizno definisanje kompleksnih funkcija, ali ne postoje računari na kojim se mogu izvršavati

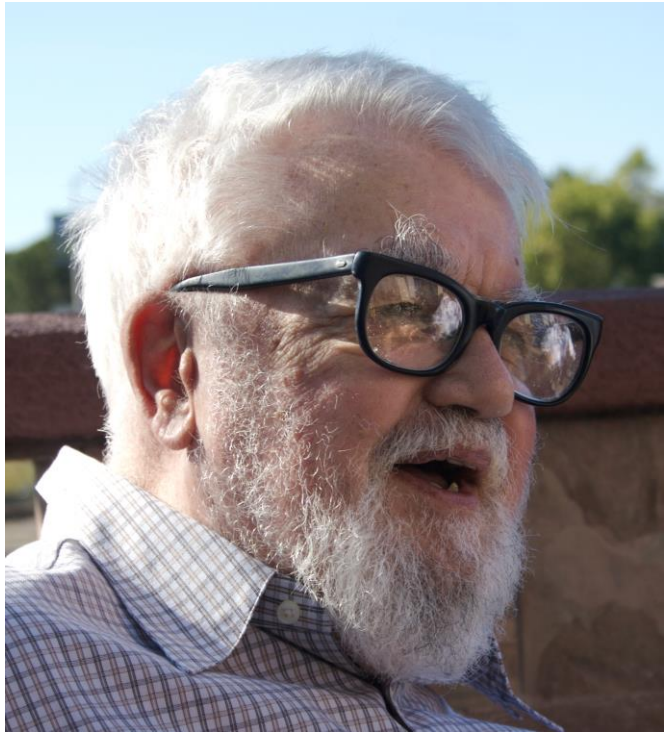
- Church encodings – boolean, integer i drugi tipovi podataka definisani preko funkcija
- Prve verzije Haskell-a implementirane na ovaj način

Razvoj funkcijskih jezika

1950ih - John McCarthy razvio LISP

- LISt Processor
- Prvi funkcijski jezik inspirisan lambda računu, ali uz dodelu vrednosti promenljivim

- AutoLISP – verzija LISP-a na kome je razvijen AutoCAD
- Commnon LISP, Scheme, …



Razvoj funkcijskih jezika

1960ih - Peter Landin je razvio ISWIM

- “If you See What I Mean”
- Prvi čisti funkcijski jezik zasnovan strogo na lambda računu
- Pojedini elementi preuzeti u Python-u i Haskell-u
- Uveo pojam *domain specific languages*

Razvoj funkcijskih jezika

1970ih - John Backus razvio FP (“Functional Programming”) sa idejom funkcija višeg reda
- Variable-free style

1970ih - Robin Milner i drugi razvili ML (“Meta-Language”) gde je prvi put uvedena ideja polimorfizma i nasleđivanja
- Prethodnik F# jezika

1970ih i 1980ih - David Turner razvio brojne *lazy* funkcijske jezike i najpopularniji među njima Miranda (“admirable”).

1987 – Istraživači International committee of programming language inicirali razvoj Haskell-a (nazvanog po logičaru Haskell Curry) kao standardnog *lazy* funkcijskog jezika

2003 - Haskell committee je objavio Haskell 98 izvešaj
- GHC
- Hugs
- …

Funkcijsko programiranje

- Stil programiranja u kome je osnovni metod primena funkcija na argumente
- Proživljava „drugi život“
- Mnoge ideje iz „čistog“ funkcijskog jezika su implementirane u druge jezike – C#, Link
- Programiranje matematičkih funkcija vodi do čistog koda
- Funkcijski ili funkcionalni jezici?

Features

Purely functional

Statically typed

Lazy

Purely functional

Svaki ulaz ima odgovarajući izlaz

- $f(x) = x^2 + 1$

Kompozicija funkcija

- $g(x) = 2x - 1$
- $g\big(f(1)\big) = 3$

„Čiste“ funkcije

- Nema *side effects*
- Funkcija nikada ne menja vrednost globalne promenljive
- Redosled nije bitan!
- „Laka“ konkuretnost

Imperativni vs. funkcijski jezici

Suma celih bojeva

```
int total = 0;
for (count = 1; count <= 5; count++)
    total = total + count;
```

Variable assignment

Drugoj metodi se može predati konačan rezultat izračunavanja (promenljiva)

Objedinjena petlja i sabiranje

Monilitni kod – mora da se posmatra kao celina

vs.

```
sum [1 .. 5]
```

```
= { applying [..] }
sum [1,2,3,4,5]
= { applying sum }
1 + 2 + 3 + 4 + 5
= { applying + }
15
```

Function application

Drugoj metodi se može predati funkcija

Petlja odvojena od sabiranja

Kompozicija funkcija - svaki izraz može da se posmatra odvojeno od ostatka

Imperativni vs. funkcijski jezici

```
void f(int a[], int lo, int hi)
{
  int h, l, p, t;

  if (lo < hi) {
    l = lo;
    h = hi;
    p = a[hi];

    do {
      while ((l < h) && (a[l] <= p))
          l = l+1;
      while ((h > l) && (a[h] >= p))
          h = h-1;
      if (l < h) {
          t = a[l];
          a[l] = a[h];
          a[h] = t;
      }
    } while (l < h);

    a[hi] = a[l];
    a[l] = p;

    f( a, lo, l-1 );
    f( a, l+1, hi );
  }
}
```

```
qsort :: Ord a => [a] -> [a]

qsort []     = []
qsort (p:xs) = (qsort lesser) ++ [p] ++ (qsort greater)
    where
        lesser  = filter (< p) xs
        greater = filter (>= p) xs
```

No variable assignments,

No array indices,

No memory management!

No Side Effects

Haskell code:

```
count :: List -> Int
```

Funkcija prihvata listu i vraća ceo broj

- Ni manje ni više!

C++ code:

```
int count( List l ) { ... }
```

C++ ne obećava integritet

- Možda radi sa IO fajlom ili ažurira globalnu promenljivu

You can't trust the code won't burn down your house.

Statically Typed

```
f x = x*x + 1
```

```
f :: Int → Int
```

Ne postoji konfuzija oko tipova (Bool, Int, Char, etc)

Strog formalizam – Dokaz je kod

If your code compiles, you're 99% done

Tipovi

Svaka funkcija u Haskell-u ima *Type signature*

```
foo :: Int -> String
```

I don't know what foo means, but I know what it does!

Lazy

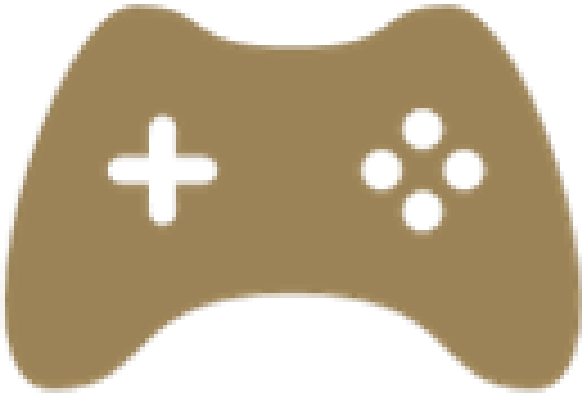
Ništa se ne izračunava ukoliko nije neophodno

```
head (sort ls)
```

- Lista će biti sortirana dovoljno da se odredi minimum

Dozvoljava beskonačne strukture podataka

```
[1..]
```



Lambda račun

Moderni računari ne mogu da izračunaju ništa više nego Tjurninova mašina – teorijski računar sa beskonačnom memorijom

- Brži su i postoje, ali mogućnosti su im iste

Tjuringove mašine nam omogućavaju da odredimo šta možemo da izračunamo, bez razmišljanja o samoj mašini i načinu računanja

Lambda račun je drugi način izražavanja izračunljivosti

Lambda račun $\overset{može\ da\ se\ izrazi\ preko}{\longleftrightarrow}$ Tjuringova mašina

Lamda račun

Jezik za izražavanje primena funkcija

- Definisanje (anonimnih) funkcija
- Primena funkcija

Vrlo formalno zapisivanje, u tesnoj vezi sa mnogim programskim jezicima

Sintaksa

Izazi u lambda računu su jedino:

- $E \rightarrow ID$
- $E \rightarrow \lambda\ ID\ .\ E$
- $E \rightarrow E\ E$
- $E \rightarrow (E)$

Primeri:

x

$\lambda\ x\ .\ x$

$x\ y$

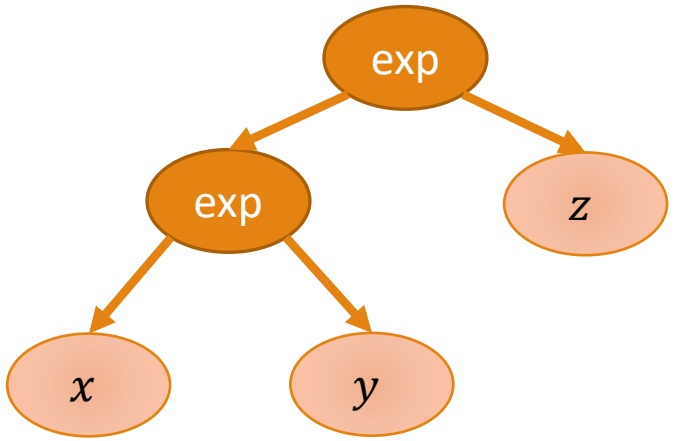
$\lambda\ y . \lambda\ x\ . y$

$\lambda\ x\ . y\ z \quad ? \quad \lambda\ x\ . (y\ z) \qquad (\lambda\ x\ . y)\ z$

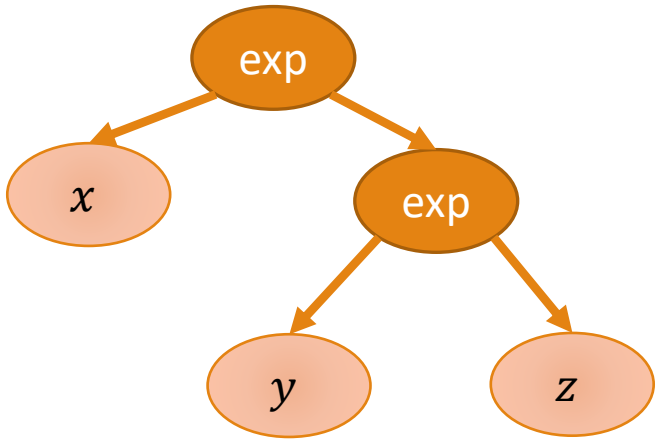
$\text{foo}\ \lambda\ \text{bar}\ .\ \big(\text{foo}\ (\text{bar baz})\big)$

Dvosmislena sintaksa

$x\ y\ z$



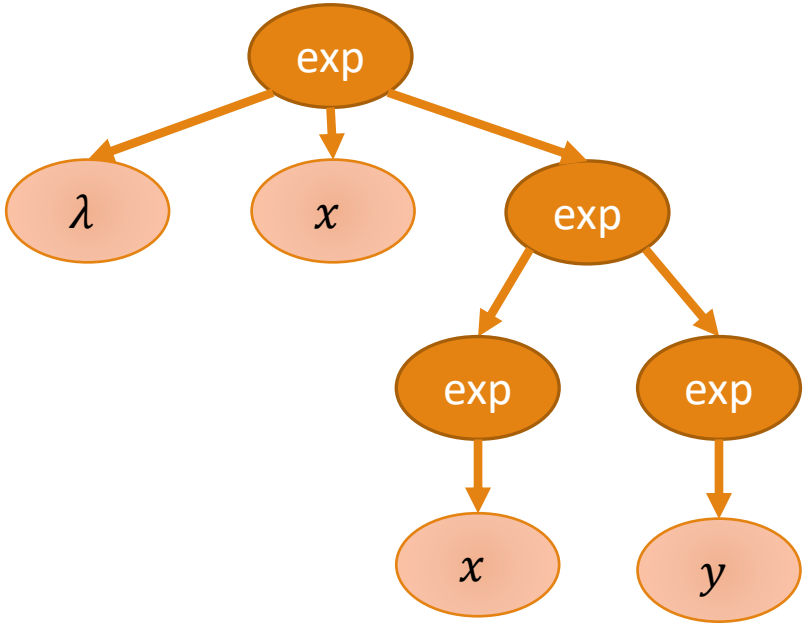
$(x\ y)\ z$



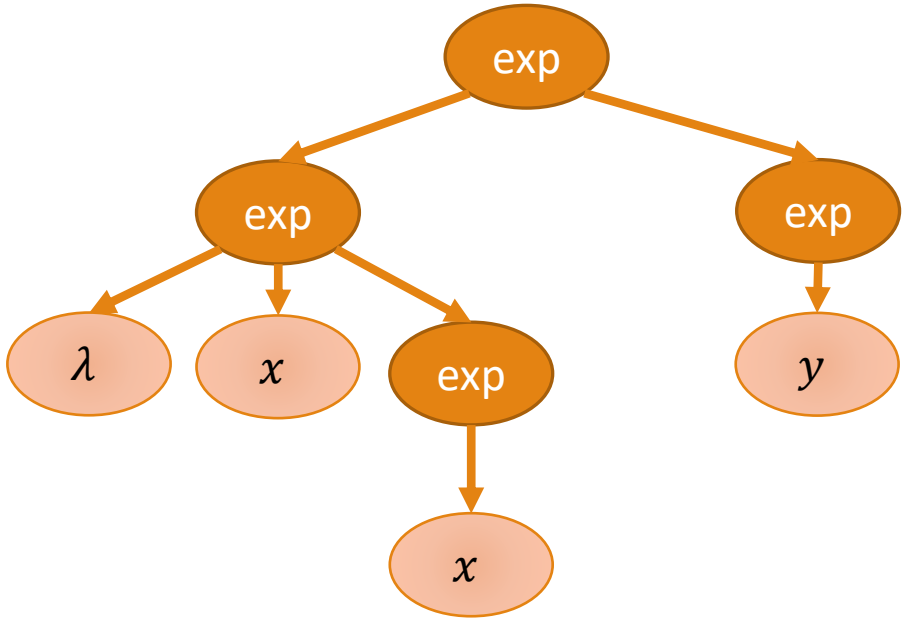
$x\ (y\ z)$

Dvosmislena sintaksa

$\lambda\, x\, .x\, y$



$\lambda\, x.(x\, y)$



$(\lambda\, x.x)\, y$

Uklanjanje dvosmislenosti

$E \to E\ E$ – levo asocijativno

- $x\ y\ z$ je
  $(x\ y)\ z$
- $w\ x\ y\ z$ je
  $((w\ x)\ y)\ z$

$E \to \lambda\ ID\ .\ E$ - proširuje se na desno koliko god je moguće

- $\lambda\ x\ .x\ y$ je
  $\lambda\ x\ .(x\ y)$
- $\lambda\ x\ .\lambda\ x\ .x$ je
  $\lambda\ x\ .(\lambda\ x\ .x)$
- $\lambda\ x\ .\lambda\ y\ .\ \lambda\ z\ .x\ y\ z$ je
  $\lambda\ x\ .\left(\lambda\ y.\left(\lambda\ z\ .\left((x\ y)\ z\right)\right)\right)$

Semantika

Svaki ID je promenljiva (varijabla)

$E \rightarrow \lambda\, ID\, .\, E$ je abstrakcija

- ID je promenljiva abstrakcije (metapromenljiva, atribut)
- E je telo abstrakcije

  ⇒ definisanje nove anonimne funkcije

$E \rightarrow E\ E$ je primena

- $E \rightarrow E_1\ E_2$ je poziv funkcije E_1 pri čemu je formalni parametar E_2

Primer

Neka je definisana funkcija + i konstanta 1

Kako definisati funkciju koja će parametar funkcije uvećati za 1?

$$\lambda\, x\, .+\, x\, 1$$

Šta radi $(\lambda\, x\, .+\, x\, 1)\, 2$? Kako?

- Redukcijom (primenom funkcije) se dobija + 2 1 i rezultat 3

Kako definisati funkciju koja sabira tri broja?

- $\lambda\, a\, .\lambda\, b\, .\lambda\, c\, .(+\, ((+\, a)\, b)\, c)$ - foo
- $foo\ 1\ 2\ 3$ → $\big((foo\ 1)\ 2\big)\ 3\big)$
- $foo\ 1$ → $\big(\lambda\, a\, .\lambda\, b\, .\lambda\, c\, .(+\, ((+\, a)\, b)\, c)\big)\, 1$ → $\lambda\, b\, .\lambda\, c\, .(+\, ((+\, 1)\, b)\, c)$
- $(foo\ 1)\ 2$ → $\big(\lambda\, b\, .\lambda\, c\, .(+\, ((+\, 1)\, b)\, c)\big)\, 2$ → $\lambda\, c\, .(+\, ((+\, 1)\, 2)\, c)$
- $\big((foo\ 1)\ 2\big)\ 3\big)$ → $\big(\lambda\, c\, .(+\, ((+\, 1)\, 2)\, c)\big)\, 3$ → $(+\, ((+\, 1)\, 2)\, 3)$ → $(+\, 3\ 3)$ → 6

Curryjev postupak

1924g, Haskell Brooks Curry – Bilo koja funkcija sa više argumenata može se definisati pomoću funkcije sa jednim argumentom

Slobodne promenljive

$(x\ y\ z)$ vs. $\lambda x\ .x$

$\lambda y\ .(\lambda x\ .x\ y)\ x$

Promenljiva x je slobodna u E ako

- $E = x$
- $E = \lambda y\ .E_1$, gde je y različito od x i pri tome je x slobodna promenljiva u E_1
- $E = E_1\ E_2$, gde je x slobodna u E_1 ili slobodna u E_2

Primer

Da li je x slobodna promenljiva u sledećim izrazima

$x\ \lambda x.x \rightarrow (x)\ (\lambda x.x)$

$(\lambda\ x\ .x\ y)\ x$

$\lambda\ x\ .y\ x$

Kombinatori

Izraz je kombinator ako nema ni jednu slobodnu promenljivu

$\lambda\, x\, .\lambda\, y.x\; y\; x$

$\lambda\, z\, .\lambda\, x\, .x\; y\; z$

Vezane promenljive

Promenljive koje nisu slobodne su vezane

Kojom abstrakcijom je promenljiva vezana?

- $\lambda\, x\, .\lambda\, x\, .\, x$

Ako je x slobodna promenljiva u E, tada je vezana sa $\lambda\, x\, .$ u $\lambda\, x\, .\, E$

Ako je pojava x vezana konkretnim $\lambda\, x\, .$ u E, tada je x istim $\lambda\, x\, .$ vezana u $\lambda\, z\, .\, E$,

- Čak i ako je $z = x$
- Dodavanja novih „spoljnih“ apstrakcija ne menja čime je promenljiva vezana

Ako je pojava x vezana konkretnim $\lambda\, x\, .$ u E_1, tada je x istim $\lambda\, x\, .$ vezana u $E_1\ E_2$ i $E_2\ E_1$

- Dodavanje aplikacija ne menja čime je promenljiva vezana

```
int y;
fun(int y) {
    y = 5;
    …
}
```

Primer

$(\lambda\, x\, .x\, (\lambda\, y\, .x\, y\, z\, y)\, x)\, x\, y$

- $(\lambda\, \boldsymbol{x}\, .\boldsymbol{x}\, (\lambda\, y\, .\boldsymbol{x}\, y\, \underline{z}\, y)\, \boldsymbol{x})\, \underline{x}\, \underline{y}$

$(\lambda\, x\, .\lambda\, y\, .x\, y)\, (\lambda\, z\, .x\, z)$

- $(\lambda\, \boldsymbol{x}\, .\lambda\, y\, .\boldsymbol{x}\, y)\, (\lambda\, z\, .\underline{x}\, z)$

$(\lambda\, x\, .x\, \lambda\, x\, .z\, x)$

- $(\lambda\, \boldsymbol{x}\, .\boldsymbol{x}\, \lambda\, x\, .\underline{z}\, x)$

Ekvivalentnost funkcija

Da li su funkcije $foo(x, y) = x + y$ i $bar(w, z) = w + z$ ekvivalentne?

A $baz(x, y) = y + x$?

$\lambda\, y\, .\, y = \lambda\, x\, .\, x$?

$\lambda\, y\, .\, y\, x = \lambda\, x\, .\, x\, y$?

- $foo(a) = a + x$ i $bar(x) = x + a$
  - $x = 10 \quad a = -10 \quad foo(5)? \quad bar(5)?$
  - Nije poznat kontekst (opseg dostižnosti) slobodnih promenljivih
- $foo(a) = a + x$ i $bar(b) = b + x$
  - Figuriše ista slobodna promenljiva

$\lambda\, x\, .\, x = \lambda\, x\, .\, x$

α-ekvivalencija

Dve funkcije koje se razlikuju samo u imenima vezanih promenljivih su α-ekvivalentne

$E_1 =_\alpha E_2$

Promena imena promenljivih

- Find and replace?
- $\lambda\, x\, .\, x\, \lambda\, y\, .\, x\, y\, z$
  - Može li se x preimenovati u foo?
  - Može li se y preimenovati u bar?
  - Može li se y preimenovati u x?
  - Može li se x preimenovati u z?

Operator preimenovanja

$E\{y/x\}$

- $x\{y/x\} = y$
- $z\{y/x\} = z$, ako je $x \neq z$
- $(E_1\ E_2)\ \{y/x\} = (E_1\{y/x\})\ (E_2\{y/x\})$
- $(\lambda\ x\ .E)\{y/x\} = (\lambda\ y\ .E\ \{y/x\})$
- $(\lambda\ z\ .E)\{y/x\} = (\lambda\ z\ .E\ \{y/x\})$ ako je $x \neq z$

Operator preimenovanja

Primer

$\big((\lambda\, x\, .x\ (\lambda\, y\, .x\ y\ z\ y)\ x)\ x\ y\big)\ \{bar/x\}$

- $(\lambda\, x\, .x\ (\lambda\, y\, .x\ y\ z\ y)\ x)\ \{bar/x\}\ (x)\{bar/x\}\ (y)\ \{bar/x\}$
- $(\lambda\, x\, .x\ (\lambda\, y\, .x\ y\ z\ y)\ x)\ \{bar/x\}\ (x)\{bar/x\}\ y$
- $(\lambda\, x\, .x\ (\lambda\, y\, .x\ y\ z\ y)\ x)\ \{bar/x\}\ bar\ y$
- $(\lambda\, bar\, .(x\ (\lambda\, y\, .x\ y\ z\ y)\ x)\ \{bar/x\})\ bar\ y$
- $(\lambda\, bar\, .(bar\ (\lambda\, y\, .x\ y\ z\ y)\{bar/x\}bar)\ )\ bar\ y$
- $(\lambda\, bar\, .(bar\ (\lambda\, y\, .(x\ y\ z\ y)\{bar/x\})bar)\ )\ bar\ y$
- $\big(\lambda\, bar\, .(bar\ \big(\lambda\, y\, .(bar\ y\ z\ y)\big)bar)\ \big)\ bar\ y$

α-ekvivalencija

Za sve izraze E i promenljive y koje se ne pojavljuju u E

- $\lambda\, x\, .E =_{\alpha} \lambda\, y\, .(E\, \{y/x\})$

Substitucija

Substitucija je zamena **slobodnih** promenljivih lambda izrazom

$E[x \to N]$, gde su E i N lambda izrazi i x je ime

$(+\ x\ 1)\ [x\ \to 2]$?

- $(+\ 2\ 1)$

$(\lambda\ x\ .+\ x\ 1)[x \to 2]$

- $(\lambda\ x\ .+\ x\ 1)$

$(\lambda\ x\ .y\ x)\ [y \to \lambda\ z\ .x\ z]$

- $(\lambda\ x\ .(\lambda\ z\ .x\ z)\ x)$?
- $(\lambda\ w\ .(\lambda\ z\ .x\ z)\ w)$

Operator substitucije

$E[x \to N]$

- $x[x \to N] = N$
- $y[x \to N] = y, x \neq y$
- $(E_1\ E_2)[x \to N] = (E_1[x \to N])\ (E_2[x \to N])$
- $(\lambda\ x\ .E)[x \to N] = (\lambda\ x\ .E)$
- $(\lambda\ y\ .E)[x \to N] = (\lambda\ y\ .E\ [x \to N])$ ako je $x \neq y$ i y nije slobodna varijabla u N
- $(\lambda\ y\ .E)[x \to N] = (\lambda\ y'.E\{y'/y\}\ [x \to N])$ ako je $x \neq y$ i y je slobodna varijabla u N, a y' novo ime promenljive

Primeri

$(\lambda\, x\, .x)\ [x \to foo]$

- $(\lambda\, x\, .x)$

$(+\ 1\ x)[x \to 2]$

- $(+\ [x \to 2]\ 1\ [x \to 2]\ x\ [x \to 2])$
- $(+\ 1\ 2)$

$(\lambda\, x\, .y\ x)\ [y \to \lambda\, z\, .x\ z]$

- $(\lambda\, x\, .y\ x)\ \{w/x\}\ \ [y \to \lambda\, z\, .x\ z]$
- $(\lambda\, w\, .y\ w)\ [y \to \lambda\, z\, .x\ z]$
- $(\lambda\, w\, .(\lambda\, z\, .x\ z)\ w)$

Primer

$\big(x\ (\lambda\ y\ .x\ y)\big)\ [x \to y\ z]$

- $(x\ [x \to y\ z]\ (\lambda\ y\ .x\ y)\ [x \to y\ z])$
- $\big((y\ z)\ (\lambda\ y\ .x\ y)\ [x \to y\ z]\big)$
- $((y\ z)\ \ (\lambda\ y\ .x\ y)\ \ \{q/y\}\ [x \to y\ z])$
- $\big((y\ z)\ (\lambda\ q\ .x\ q)\ [x \to y\ z]\big)$
- $\big((y\ z)\ (\lambda\ q\ .(y\ z)\ q)\ \big)$