

Bulova logika

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

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

- Nije obavezno da tačno i netačno budu definisani na ovaj način

$$and = (\lambda a . \lambda b . a b F)$$

- $and\ T\ T$
- $(\lambda a . \lambda b . a b (\lambda x . \lambda y . y))\ T\ T$

and T T

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

$(\lambda b. a b (\lambda x. \lambda y. y)) [a \rightarrow (\lambda x. \lambda y. x)] (\lambda x. \lambda y. x)$

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

$((\lambda x. \lambda y. x) b (\lambda x. \lambda y. y)) [b \rightarrow (\lambda x. \lambda y. x)]$

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

$(\lambda y. x)[x \rightarrow (\lambda x. \lambda y. x)] (\lambda x. \lambda y. y)$

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

$(\lambda x. \lambda y. x) [y \rightarrow (\lambda x. \lambda y. y)]$

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

T

and T F

$(\lambda a. \lambda b. a b F) T F$

$(\lambda b. a b F) [a \rightarrow T] F$

$(\lambda b. T b F) F$

$(T b F)[b \rightarrow F]$

$(T F F)$

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

$(\lambda y. x) [x \rightarrow F] F$

$(\lambda y. F) F$

$F [y \rightarrow F]$

F

and F T

$(\lambda a. \lambda b. a b F) F T$

$(F T F)$

F

and F F

$(\lambda a. \lambda b. a\ b\ F)\ F\ F$

$(F\ F\ F)$

F

not

- $\text{not } T = F$
- $\text{not } F = T$

$\text{not} = (\lambda a . a F T)$

$\text{not } T$

- $(\lambda a . a F T) T$
- $T F T$
- F

$\text{not } F$

- $(\lambda a . a F T) F$
- $F F T$
- T

IF grananje

```
if c then  
  a  
else  
  b
```

if c a b

if T a b = a

if F a b = b

if = ($\lambda a . a$)

Primeri

if T a b

- $(\lambda a . a) T a b$
- $T a b$
- a

if F a b

- $(\lambda a . a) F a b$
- $F a b$
- b

Čerčovi numerali

$$0 = \lambda f . \lambda x . x$$

$$1 = \lambda f . \lambda x . f x$$

$$2 = \lambda f . \lambda x . f f x$$

$$3 = \lambda f . \lambda x . f f f x$$

$$4 = \lambda f . \lambda x . f f f f x$$

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

$$4 a b$$

- $a a a a b$

Funkcija naslednika

$succ = \lambda n . \lambda f . \lambda x . f (n f x)$

$0 = \lambda f . \lambda x . x$

$succ\ 0$

- $\lambda n . \lambda f . \lambda x . f (n f x)\ 0$
- $\lambda f . \lambda x . f (0 f x)$
- $\lambda f . \lambda x . f ((\lambda f . \lambda x . x) f x)$
- $\lambda f . \lambda x . f x$
- 1

Funkcija naslednika

$$succ = \lambda n . \lambda f . \lambda x . f (n f x)$$

$$1 = \lambda f . \lambda x . f x$$

$$succ\ 1$$

- $\lambda n . \lambda f . \lambda x . f (n f x)\ 1$
- $\lambda f . \lambda x . f (1 f x)$
- $\lambda f . \lambda x . f ((\lambda f . \lambda x . f x) f x)$
- $\lambda f . \lambda x . f f x$

$$2 = \lambda f . \lambda x . f f x$$

$$succ\ 1 = 2$$

$$succ\ n = n + 1$$

Sabiranje

$add\ 0\ 1 = 1$

$add\ 1\ 2 = 3$

$add = \lambda n . \lambda m . \lambda f . \lambda x . n\ f\ (m\ f\ x)$

$add\ 0\ 1$

- $(\lambda n . \lambda m . \lambda f . \lambda x . n\ f\ (m\ f\ x))\ 0\ 1$
- $(\lambda m . \lambda f . \lambda x . 0\ f\ (m\ f\ x))\ 1$
- $\lambda f . \lambda x . 0\ f\ (1\ f\ x)$
- $\lambda f . \lambda x . 0\ f\ (f\ x)$
- $\lambda f . \lambda x . f\ x$

Množenje

mult 0 1 = 0

mult 1 2 = 2

mult 2 5 = 10

mult = $\lambda n . \lambda m . m \text{ (add } n) 0$

mult 0 1

- $(\lambda n . \lambda m . m \text{ (add } n) 0) 0 1$
- $(\lambda m . m \text{ (add } 0) 0) 1$
- $1 \text{ (add } 0) 0$
- $\text{add } 0 0$
- 0

Množenje

mult 1 2

- $(\lambda n . \lambda m . m (add\ n)\ 0)\ 1\ 2$
- $(\lambda m . m (add\ 1)\ 0)\ 2$
- $2 (add\ 0)\ 0$
- $(add\ 1)\ ((add\ 1)\ 0)$
- $(add\ 1)\ (add\ 1\ 0)$
- $(add\ 1)\ (1)$

Faktorijel

$n!$

- $\text{fact}(0) = 1$
- $\text{fact}(n) = n * \text{fakt}(n-1)$

$\text{fact} = \left(\lambda n . \text{if } (\text{iszero } n) (1) \left(\text{mult } n (\text{fact } (\text{pred } n)) \right) \right)$

- Funkciju ne možemo definisati ovako!

```
int fact(int n)
{
    if (n==0)
        return 1;
    return n * fact(n-1);
}
```

Y kombinator

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

$Y \text{ foo}$

- $(\lambda x . \lambda y . y (x x y)) (\lambda x . \lambda y . y (x x y)) \text{ foo}$
- $\left(\lambda y . y \left((\lambda x . \lambda y . y (x x y)) (\lambda x . \lambda y . y (x x y)) y \right) \right) \text{ foo}$
- $\text{foo} \left((\lambda x . \lambda y . y (x x y)) (\lambda x . \lambda y . y (x x y)) \text{ foo} \right)$
- $\text{foo} (Y \text{ foo})$
- $\text{foo} (\text{foo} (Y \text{ foo}))$
- $\text{foo} (\text{foo} (\text{foo} (Y \text{ foo})))$
- ...

Rekurzija

$$\times fact = (\lambda n . if \ (iszero \ n) \ (1) \ (mult \ n \ (fact \ (pred \ n))))$$

$$fact = Y \left(\lambda f . \lambda n . if \ (iszero \ n) \ (1) \ (mult \ n \ (f \ (pred \ n))) \right)$$

$fact \ 1$

- $Y \left(\lambda f . \lambda n . if \ (iszero \ n) \ (1) \ (mult \ n \ (f \ (pred \ n))) \right) \ 1$
- $\left(\lambda f . \lambda n . if \ (iszero \ n) \ (1) \ (mult \ n \ (f \ (pred \ n))) \right) \left(Y \left(\lambda f . \lambda n . if \ (iszero \ n) \ (1) \ (mult \ n \ (f \ (pred \ n))) \right) \right) \ 1$
- $\left(\lambda n . if \ (iszero \ n) \ (1) \ \left(mult \ n \ \left(\left(Y \left(\lambda f . \lambda n . if \ (iszero \ n) \ (1) \ (mult \ n \ (f \ (pred \ n))) \right) \right) (pred \ n) \right) \right) \right) \ 1$
- $if \ (iszero \ 1) \ (1) \ \left(mult \ 1 \ \left(\left(Y \left(\lambda f . \lambda n . if \ (iszero \ n) \ (1) \ (mult \ n \ (f \ (pred \ n))) \right) \right) (pred \ 1) \right) \right)$
- $if \ F \ (1) \ \left(mult \ 1 \ \left(\left(Y \left(\lambda f . \lambda n . if \ (iszero \ n) \ (1) \ (mult \ n \ (f \ (pred \ n))) \right) \right) (pred \ 1) \right) \right)$
- $mult \ 1 \ \left(\left(Y \left(\lambda f . \lambda n . if \ (iszero \ n) \ (1) \ (mult \ n \ (f \ (pred \ n))) \right) \right) (pred \ 1) \right)$

Rekurzija

fact 1

- $\text{mult } 1 \left(\left(Y \left(\lambda f. \lambda n. \text{if } (\text{iszero } n) (1) \left(\text{mult } n (f (\text{pred } n)) \right) \right) \right) (\text{pred } 1) \right)$
- $\text{mult } 1 \left(\left(\lambda f. \lambda n. \text{if } (\text{iszero } n) (1) \left(\text{mult } n (f (\text{pred } n)) \right) \right) \left(Y \left(\lambda f. \lambda n. \text{if } (\text{iszero } n) (1) \left(\text{mult } n (f (\text{pred } n)) \right) \right) \right) (\text{pred } 1) \right)$
- $\text{mult } 1 \left(\lambda n. \text{if } (\text{iszero } n) (1) \left(\text{mult } n \left(\left(Y \left(\lambda f. \lambda n. \text{if } (\text{iszero } n) (1) \left(\text{mult } n (f (\text{pred } n)) \right) \right) \right) (\text{pred } n) \right) \right) (\text{pred } 1) \right)$
- $\text{mult } 1 \left(\lambda n. \text{if } (\text{iszero } n) (1) \left(\text{mult } n \left(\left(Y \left(\lambda f. \lambda n. \text{if } (\text{iszero } n) (1) \left(\text{mult } n (f (\text{pred } n)) \right) \right) \right) (\text{pred } n) \right) \right) 0 \right)$
- $\text{mult } 1 \left(\text{if } (\text{iszero } 0) (1) \left(\text{mult } 0 \left(\left(Y \left(\lambda f. \lambda n. \text{if } (\text{iszero } n) (1) \left(\text{mult } n (f (\text{pred } n)) \right) \right) \right) (\text{pred } 0) \right) \right) \right)$
- $\text{mult } 1 \left(\text{if } T (1) \left(\text{mult } 0 \left(\left(Y \left(\lambda f. \lambda n. \text{if } (\text{iszero } n) (1) \left(\text{mult } n (f (\text{pred } n)) \right) \right) \right) (\text{pred } 0) \right) \right) \right)$
- $\text{mult } 1 \ 1$
- 1

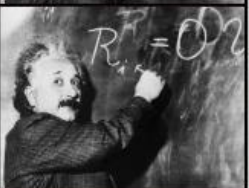
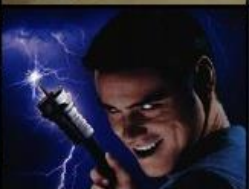
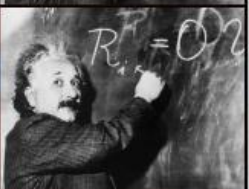
Tjuring-kompletan sistem

Bulova
logika

Aritmetika

Petlje

Funkcijsko programiranje

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

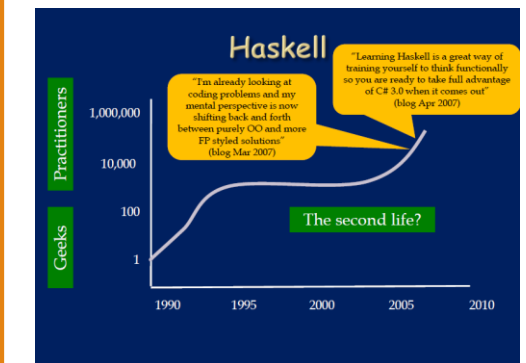
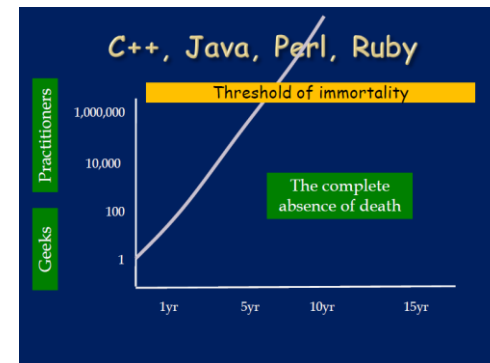
Šta je funkcijsko programiranje?

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

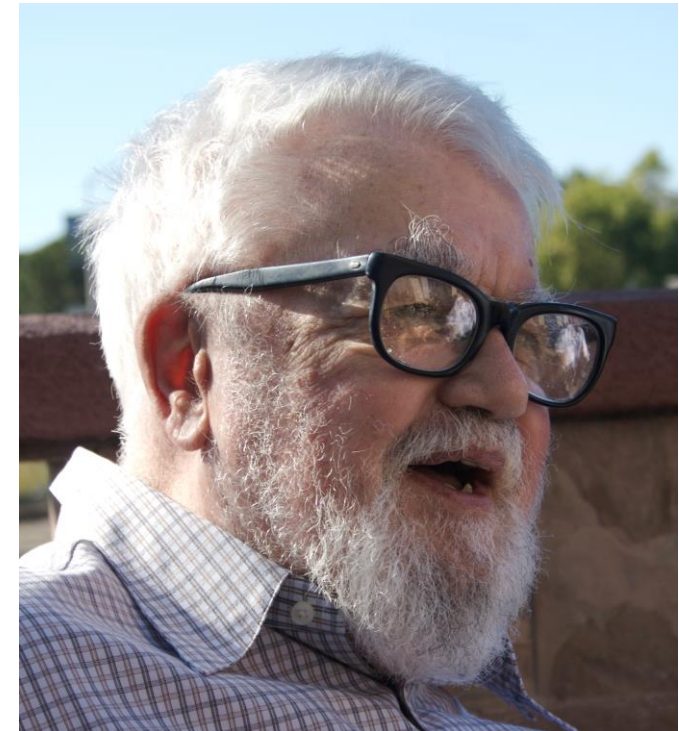
- Jednostavna ali moćna matematička teorija funkcija
- Omogućava vrlo precizno definisanje kompleksnih funkcija, ali ne postoje računari na kojim se mogu izvršavati
- Church encodings – boolean, integer i drugi tipovi podataka definisani preko funkcija
- Prve verzije Haskell-a implementirane na ovaj 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
- Common 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

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

„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

Monolitni kod – mora da se posmatra kao celina

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

Function application

Drugoj metodi se može predati
funkcija

Petlja odvojena od sabiranja

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

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

vs.

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

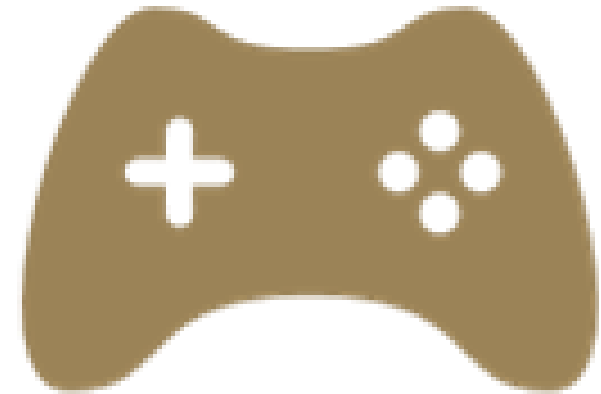
Ništa se ne izračunava ukoliko nije neophodno

head (sort ls)

- Lista će biti sortirana dovoljno da se odredi minimum

Dozvoljava beskonačne strukture podataka

[1..]



Haskell

GRAHAM HUTTON, PROGRAMMING IN HASKELL,
CAMBRIDGE UNIVERSITY PRESS (2007)

Primene Haskell-a

Cardano is a public blockchain platform. It is open-source and decentralized, with consensus achieved using proof of stake.

Xmonadis a window manager for the X Window System, written fully in Haskell.

Facebook implements its anti-spam programs in Haskell, maintaining the underlying data access library a sopen-source software.

AT&T: Haskell is being used in the Network Security division to automate processing of internet abuse complaints.

GitHub implemented **Semantic**, an open-source library for analysis, diffing, and interpretation of untrusted source code, in Haskell.

[Lista sa primerima korišćenja u industriji na zvaničnom sajtu Haskell-a.](#)

Neke od najčešćih primena su : grafika, networking , obrada teksta, sistemsko programiranje i razvoj korisničkih softvera.

Preporuka za okruženje i instalaciju

GHC okruženje (Win i Linux)

- GHC – prevodilac (kompajler)
- GHCi – interaktivni interpreter
- (opciono) Stack – alat za upravljanje projektima i zavisnostim
- (opciono) Cabal – alat sličan Stack-u

Najjednostavniji način je korišćenje GHCup, koji sve to instalira i održava ažurnim.

Ekstenzije za VS Code

- Haskell
- Haskell Syntax Highlighting
- **VS Code ne instalira automatski GHC!**

Dokumentacija

- Za pretragu funkcija i biblioteka po imenu ili potpisu postoji servis koji se zove **Hoogle**

*Main> :?

Commands available from the prompt:

<statement>	evaluate/run <statement>
:	repeat last command
:{\n ..lines.. \n:}\n	multiline command
:add [*]<module> ...	add module(s) to the current target set
:browse[!] [[*]<mod>]	display the names defined by module <mod> (!: more details; *: all top-level names)
:cd <dir>	change directory to <dir>
:cmd <expr>	run the commands returned by <expr>::IO String
:complete <dom> [<rng>] <s>	list completions for partial input string
:ctags[!] [<file>]	create tags file <file> for Vi (default: "tags") (!: use regex instead of line number)
:def[!] <cmd> <expr>	define command :<cmd> (later defined command has precedence, ::<cmd> is always a builtin command) (!: redefine an existing command name)
:doc <name>	display docs for the given name (experimental)
:edit <file>	edit file
:edit	edit last module
:etags [<file>]	create tags file <file> for Emacs (default: "TAGS")
:help, :?	display this list of commands
:info[!] [<name> ...]	display information about the given names (!: do not filter instances)
:instances <type>	display the class instances available for <type>
:issafe [<mod>]	display safe haskell information of module <mod>
:kind[!] <type>	show the kind of <type> (!: also print the normalised type)
:load[!] [*]<module> ...	load module(s) and their dependents (!: defer type errors)

Standardne komande

Najčešće korišćene komande

```
:load <filenames>  load modules from specified files
:load              clear all files except prelude
:also <filenames>  read additional modules
:reload            repeat last load command
:edit <filename>    edit file
:edit              edit last module
:module <module>    set module for evaluating expressions
<expr>             evaluate expression
:type <expr>        print type of expression
:?                display this list of commands
:set <options>      set command line options
:set              help on command line options
:names [pat]        list names currently in scope
:info <names>        describe named objects
:browse <modules>   browse names exported by <modules>
:main <aruments>    run the main function with the given arguments
:find <name>         edit module containing definition of name
:cd dir             change directory
:gc                 force garbage collection
:version            print Hugs version
:quit              exit Hugs interpreter
Hugs>
```

- `:?` – help, prikazuje listu prikazanu sa desne strane
- `:load fajl.hs` – učitava Haskell kod
- `:reload` – ponavlja poslednje učitavanje
- `:type izraz` – prikazuje nam tip datog izraza
- `:edit` – u podrazumevanom tekst editoru otvara modul koji je poslednji učitán
- `:quit` – izlaz iz Hugs-a

Sve komande se mogu koristiti i u skraćenom obliku koji predstavlja njihovo prvo slovo, na primer: `:t 'a'`

Haskell kao kalkulator

Pokrenutu GHCi interpreter

```
Prelude> 2+3*4
14
Prelude> (2+3)*4
20
Prelude> sqrt(3^2 + 4^2)
5.0
Prelude> 2^1000
107150860718626732094842504906000181056140481170553360744375038837035105112493612249319837881569585812759467291755314682
518714528569231404359845775746985748039345677748242309854210746050623711418779541821530464749835819412673987675591655439
46077062914571196477686542167660429831652624386837205668069376
Prelude>
```

```
Prelude> (+) 2 3
5
Prelude> (^) 2 5
32
```

Primeri

```
div 10 2
```

```
div 11.0 2
```

```
6 `div` 2
```

```
mod 5 2
```

```
mod 5 2.0
```

```
[1..10]
```

```
sum [1..10]
```

Standardne funkcije

Prvi element liste

```
Prelude> head [1, 2, 3, 4]  
1
```

Svi elementi sem prvog

```
Prelude> tail [1, 2, 3, 4]  
[2,3,4]
```

N-ti element liste

```
Prelude> [1, 2, 3, 4] !! 2  
3
```

Prvih n elemenata liste

```
Prelude> take 3 [1, 2, 3, 4]  
[1,2,3]
```

Standardne funkcije

Uklanjanja prvih n elemenata liste

```
Prelude> drop 3 [1, 2, 3, 4]  
[4]
```

```
xs = take n xs ++ drop n xs
```

Dužina liste

```
Prelude> length [1, 2, 3, 4]  
4
```

Suma elemenata liste

```
Prelude> sum [1, 2, 3, 4]  
10
```

Standardne funkcije

Proizvod elemenata liste

```
Prelude> product [1, 2, 3, 4]  
24
```

Nadovezivanje lista

```
Prelude> [1, 2, 3] ++ [4, 5]  
[1,2,3,4,5]
```

Inverzna lista

```
Prelude> reverse [1, 2, 3, 4]  
[4,3,2,1]
```