[Home](http://nbviewer.ipython.org/)

[FAQ](http://nbviewer.ipython.org/faq)

[IPython](http://www.ipython.org)

[Bookmarklet](http://penandpants.com/2012/10/16/open-in-nbviewer-bookmarklet/)

[Download Notebook](https://gist.github.com/raw/5523724/27a94532584cd51a5ae7bb93a251a6326b44f377/gistfile1.txt)

Lambda calculus pythonis Karl-Aksel Puulmann, kevad 2013

Sissejuhatus

In [42]:

def add(a, b):

return a + b

add = lambda x, y: x + y

add(5, 6)

Out[42]:

11

In [46]:

def add(a):

def curried(b):

# haskell curry!

return a + b

return curried

add = lambda x: lambda y: x + y

add(5)(6)

Out[46]:

11

In [180]:

add5 = add(5) # karritatud funktsioon

add5(6)

Out[180]:

11

In [181]:

(lambda x: lambda y: x + y)(5)(6) # anonüümne!

Out[181]:

11

Kaotame ära pythonist kõik peale funktsioonide defineerimise ja nende välja kutsumise ja üritame selle peale üles ehitada keelt, mis oleks sama võimas kui python ise.

Reeglid:

Kasutame ainult karritatud funktsioone ja nende välja kutsumist. Kõik muu pythoni keelest unustame ära!

Me võime küll anda funktsioonidele nimed, aga hiljem peab olema iga nimi asendatav oma väärtusega.

In [186]:

# abiülesanne, mida üritame oma uues keeles väljendada

# kuidas vähim võimalik arv münte kulutada arve (täpseks) maksmiseks?

def min\_length(a, b):

if not b or (len(a) < len(b) and a):

return a

else:

return b

def coin\_change(amount, coins, taken = []):

if amount == sum(taken):

return taken

if coins == []:

return []

return min\_length(coin\_change(amount, coins[1:], [coins[0]]+taken),

coin\_change(amount, coins[1:], taken))

In [187]:

coin\_change(15, [1,2,3,4,5,6,7,8])

Out[187]:

[8, 7]

In [188]:

coin\_change(13, [4,4,4,3])

Out[188]:

[]

Numbrid

In [5]:

# church numerals

ZERO = lambda f: lambda x: x

ONE = lambda f: lambda x: f(x)

TWO = lambda f: lambda x: f(f(x))

THREE = lambda f: lambda x: f(f(f(x)))

FOUR = lambda f: lambda x: f(f(f(f(x))))

FIVE = lambda f: lambda x: f(f(f(f(f(x)))))

TEN = lambda f: lambda x: f(f(f(f(f(f(f(f(f(f(x))))))))))

In [165]:

FIVE

Out[165]:

<function \_\_main\_\_.<lambda>>

In [7]:

def to\_int(num):

" Convering the numerals to a form we can read "

return num(lambda x: x+1)(0)

In [166]:

to\_int(ZERO)

Out[166]:

0

In [9]:

to\_int(FIVE)

Out[9]:

5

In [10]:

# http://en.wikipedia.org/wiki/Lambda\_calculus#Arithmetic\_in\_lambda\_calculus

INCREMENT = lambda num: lambda f: lambda x: f(num(f)(x))

DECREMENT = lambda num: lambda f: lambda x: \

num(lambda g: lambda h: h(g(f)))(lambda y: x)(lambda y: y)

SIX = INCREMENT(FIVE)

assert to\_int(SIX) == 6

assert to\_int(DECREMENT(ONE)) == 0

assert to\_int(DECREMENT(FIVE)) == 4

In [11]:

PLUS = lambda a: lambda b: a(INCREMENT)(b)

MINUS = lambda a: lambda b: b(DECREMENT)(a)

MULTIPLY = lambda a: lambda b: a(PLUS(b))(ZERO) # a+a+a+...+a b times

POWER = lambda a: lambda b: b(a)

NINE = MULTIPLY(THREE)(THREE)

FIFTEEN = MULTIPLY(FIVE)(THREE)

THIRTY = MULTIPLY(SIX)(FIVE)

assert to\_int(PLUS(THREE)(FIVE)) == 8

assert to\_int(MINUS(FIVE)(THREE)) == 2

assert to\_int(MINUS(THREE)(FIVE)) == 0 # !!!

assert to\_int(THIRTY) == 30

assert to\_int(POWER(THREE)(FIVE)) == 243

NB: Kuna me lubasime, et kõik meie poolt defineeritud nimed peab olema võimalik tema väärtusega asendada, siis on järgnevad samaväärsed:

In [177]:

PLUS

lambda a: lambda b: a(INCREMENT)(b)

lambda a: lambda b: a(lambda num: lambda f: lambda x: f(num(f)(x)))(b)

Out[177]:

<function \_\_main\_\_.<lambda>>

Sama ka näiteks liitmisel

In [178]:

to\_int(PLUS(THREE)(TWO))

Out[178]:

5

In [179]:

to\_int((lambda a: lambda b: a((lambda num: lambda f: lambda x: f(num(f)(x))))(b))((lambda f: lambda x: f(f(f(x)))))((lambda f: lambda x: f(f(x)))))

Out[179]:

5

Loogika

In [12]:

TRUE = lambda a: lambda b: a

FALSE = lambda a: lambda b: b

def to\_bool(f):

return f(True)(False)

In [13]:

to\_bool(TRUE)

Out[13]:

True

In [14]:

IF = lambda boolean: lambda left: lambda right: boolean(left)(right)

In [167]:

IF(TRUE)("Cake")("Death")

Out[167]:

'Cake'

Vaatame, kas saame IF-i lihtsamaks teha. Vaatleme analoogset funktsiooni:

In [185]:

# näiteks

f1 = lambda x: lambda y: x(y)

f2 = (lambda x: x)

Kui vaadata, mida f1 teeb, siis ta esimese argumendi andmisel funktsiooni, mis võtab teise argumendi ja rakendab esimest argumendiga teine.

Kui aga x on funktsioon, on see täpselt sama, mida teeb f2 talle teise argumendi andmisel - need kaks definitsiooni om samaväärsed!

Kui nüüd vaadata meie IF-i, siis siin on sama muster lausa kaks korda. Seega saab lihtsustada:

In [16]:

# simplify it further!

IF = lambda boolean: boolean

In [17]:

AND = lambda a: lambda b: IF(a)(b)(FALSE)

OR = lambda a: lambda b: IF(a)(TRUE)(b)

NOT = lambda a: IF(a)(FALSE)(TRUE)

In [18]:

assert AND(TRUE)(FALSE) == FALSE

assert AND(TRUE)(TRUE) == TRUE

assert OR(TRUE)(FALSE) == TRUE

assert NOT(TRUE) == FALSE

In [19]:

IS\_ZERO = lambda num: num(lambda x: FALSE)(TRUE) # only ZERO will call the second argument

LESS\_THAN\_EQUAL = lambda a: lambda b: IS\_ZERO(MINUS(a)(b)) # 3-5 is 0 with our numbers

EQUAL = lambda a: lambda b: AND(LESS\_THAN\_EQUAL(a)(b))(LESS\_THAN\_EQUAL(b)(a))

In [20]:

assert IS\_ZERO(ZERO) == TRUE

assert IS\_ZERO(FIVE) == FALSE

assert LESS\_THAN\_EQUAL(THREE)(FIVE) == TRUE

assert LESS\_THAN\_EQUAL(FIVE)(THREE) == FALSE

In [91]:

IF(LESS\_THAN\_EQUAL(THREE)(FIVE))("3 <= 5")("3 > 5")

Out[91]:

'3 <= 5'

Listid

Ehitame üles linked-listid.

In [22]:

PAIR = lambda head: lambda tail: lambda f: f(head)(tail)

LEFT = lambda pair: pair(TRUE) # TRUE = lambda x: lambda y: x

RIGHT = lambda pair: pair(FALSE)

In [23]:

NIL = PAIR(TRUE)(TRUE)

HEAD = lambda lst: LEFT(RIGHT(lst))

TAIL = lambda lst: RIGHT(RIGHT(lst))

PREPEND = lambda el: lambda lst: PAIR(FALSE)(PAIR(el)(lst))

IS\_EMPTY = LEFT # lambda lst: LEFT(lst)

In [24]:

lst = PREPEND(THREE)(PREPEND(ONE)(NIL)) # [3, 1]

assert to\_int(HEAD(lst)) == 3

assert to\_int(HEAD(TAIL(lst))) == 1

assert IS\_EMPTY(NIL) == TRUE

assert IS\_EMPTY(lst) == FALSE

assert IS\_EMPTY(TAIL(TAIL(lst))) == TRUE

In [25]:

def to\_list(f):

" Converts the lambda-calculus list to a normal one "

result = []

while not to\_bool(IS\_EMPTY(f)):

result.append(HEAD(f))

f = TAIL(f)

return result

Eelnevaga võime alguses antud algoritmi muutma hakata.

In [26]:

def min\_length(a, b):

return a if (len(a) < len(b) and a) or (not b) else b

def coin\_change(amount, coins, taken = []):

if amount == sum(taken):

return taken

if coins == []:

return []

return min\_length(coin\_change(amount, coins[1:], [coins[0]]+taken),

coin\_change(amount, coins[1:], taken))

In [27]:

MIN\_LENGTH = lambda a: lambda b: \

IF(OR

(IS\_EMPTY(b))

(AND

(NOT(IS\_EMPTY(a)))

(len(a) < len(b)))

)(a)(b)

COIN\_CHANGE = lambda amount: lambda coins: lambda taken: \

IF(EQUAL(amount)(sum(taken)))(

taken

)(IF(IS\_EMPTY(coins))

(NIL)

(MIN\_LENGTH(

COIN\_CHANGE(amount)(TAIL(coins))(PREPEND(HEAD(coins))(taken)))(

COIN\_CHANGE(amount)(TAIL(coins))(taken))))

Rekursioon

In [189]:

# the useful list functions - map, filter and reduce

def map(f, lst):

return [f(x) for x in lst]

def filter(predicate, lst):

return [x for x in lst if predicate(x)]

def reduce(f, initial, lst):

if lst == []:

return initial

return f(lst[0], reduce(f, initial, lst[1:]))

length = lambda lst: reduce(lambda el, rest: rest + 1, 0, lst)

In [29]:

length([1,2,3,4])

Out[29]:

4

In [30]:

REDUCE = lambda f: lambda init: lambda lst: \

IF(IS\_EMPTY(lst))(

init)(

f(HEAD(lst))(REDUCE(f)(init)(TAIL(lst))))

#LENGTH = lambda lst: REDUCE(PLUS(ONE))(ZERO)(lst)

LENGTH = REDUCE(lambda \_: PLUS(ONE))(ZERO)

# LENGTH(NIL)chrome

# RuntimeError: maximum recursion depth exceeded??

Rekursioon ei peatunud. Probleem selles, et rekursiivne väljakutse REDUCE pihta tehakse agaralt - kohe kui pythoni kood näeb seda väljakutset mingi avaldise sees, asub ta kohe väljakutse väärtust leidma. See tähendab, et ka tühja listi puhul kutsub enne init tagastamist reduce välja f(HEAD(lst))(REDUCE(f)(init)(TAIL(lst))))

Lahenduseks kasutame sama võtet mis enne IF-i lihtsamaks tehes. Kui lambda x: blah(x) on sama kui blah, siis on ka vastupidine tõene. Funktsiooni ei kutsuta enne välja kui argumendid antud. Seega paneme rekursiivse väljakutse funktsioonidefinitsiooni sisse!

In [182]:

REDUCE = lambda f: lambda init: lambda lst: \

IF(IS\_EMPTY(lst))(

init)(

lambda x: ( f(HEAD(lst))(REDUCE(f)(init)(TAIL(lst))) )(x))

#LENGTH = lambda lst: REDUCE(PLUS(ONE))(ZERO)(lst)

LENGTH = REDUCE(lambda \_: PLUS(ONE))(ZERO)

to\_int(LENGTH(NIL))

Out[182]:

0

Me aga eksime siin oma reeglite vastu - igale funktsioonile nime andmisel olime lubanud, et selle nime saab alati asendada väärtusega. Aga kui võrduse vasakul ja paremal pool on REDUCE, siis see ei ole võimalik.

Õnneks on võimalik defineerida anonüümseid (nime mittevajavaid) rekursiivseid funktsioone niinimetatud. Y-kombinaatori abil.

In [31]:

# Y-combinator to the rescue

# λf.(λx.f (λv.((x x) v))) (λx.f (λv.((x x) v)))

Y = lambda f: (lambda x: f(lambda v: x(x)(v)))(

lambda x: f(lambda v: x(x)(v)))

REDUCE = Y(lambda rec: lambda f: lambda init: lambda lst: \

IF(IS\_EMPTY(lst))(

init)(

lambda x: f(HEAD(lst))(rec(f)(init)(TAIL(lst)))(x)))

LENGTH = REDUCE(lambda \_: PLUS(ONE))(ZERO)

In [32]:

to\_int(LENGTH(NIL))

Out[32]:

0

In [33]:

onetwothree = PREPEND(ONE)(PREPEND(TWO)(PREPEND(THREE)(NIL)))

to\_int(LENGTH(onetwothree))

Out[33]:

3

In [34]:

SUM = REDUCE(PLUS)(ZERO)

to\_int(SUM(onetwothree))

Out[34]:

6

Ühendame uued tööriistad

In [35]:

# helpers

def from\_int(x):

return ZERO if x == 0 else INCREMENT(from\_int(x-1))

def from\_list(list):

return NIL if list == [] else PREPEND(list[0])(from\_list(list[1:]))

RANGE = Y(lambda rec: lambda start: lambda end: \

IF(LESS\_THAN\_EQUAL(end)(start))(

NIL)(

lambda x: (

PREPEND(start)(rec(INCREMENT(start))(end))

)(x)))

In [36]:

map(to\_int, to\_list(RANGE(ZERO)(FIVE)))

Out[36]:

[0, 1, 2, 3, 4]

In [37]:

MIN\_LENGTH = lambda a: lambda b: \

IF(OR

(IS\_EMPTY(b))

(AND

(NOT(IS\_EMPTY(a)))

(LESS\_THAN\_EQUAL(LENGTH(a))(LENGTH(b))))

)(a)(b)

COIN\_CHANGE = Y(lambda rec: lambda amount: lambda coins: lambda taken: \

IF(EQUAL(amount)(SUM(taken)))

(taken)

(IF(IS\_EMPTY(coins))

(NIL)

(lambda x: MIN\_LENGTH

(rec(amount)(TAIL(coins))(PREPEND(HEAD(coins))(taken)))

(rec(amount)(TAIL(coins))(taken)) (x))))

In [38]:

assert len(to\_list(MIN\_LENGTH(NIL)(NIL))) == 0

assert len(to\_list(MIN\_LENGTH(NIL)(RANGE(ONE)(THREE)))) == 2

assert len(to\_list(MIN\_LENGTH(RANGE(ONE)(FIVE))(RANGE(ONE)(NINE)))) == 4

In [39]:

result = COIN\_CHANGE(TEN) (RANGE(ONE)(SIX)) (NIL)

map(to\_int, to\_list(result))

Out[39]:

[5, 4, 1]

Tundub, et töötab. Aga nagu me alguses lubasime, siis annab kõik funktsioonidele antud nimed (COIN\_CHANGE, TEN jms) asendada neile antud väärtusega. Teeme seda!

In [176]:

# sama mis a = COIN\_CHANGE(TEN) (RANGE(ONE)(SIX)) (NIL)

a = ((lambda f:(lambda x: f(lambda v: x(x)(v)))(lambda x: f(lambda v: x(x)(v))))(lambda rec: lambda amount: lambda coins: lambda taken:((lambda boolean: boolean)((lambda a: lambda b:((lambda a: lambda b:((lambda boolean: boolean)(a)(b)((lambda a: lambda b: b))))((lambda a: lambda b:((lambda num: num(lambda x:((lambda a: lambda b: b)))((lambda a: lambda b: a)))((lambda a: lambda b: b((lambda num: lambda f: lambda x: num(lambda g: lambda h: h(g(f)))(lambda y: x)(lambda y: y)))(a))(a)(b))))(a)(b))((lambda a: lambda b:((lambda num: num(lambda x:((lambda a: lambda b: b)))((lambda a: lambda b: a)))((lambda a: lambda b: b((lambda num: lambda f: lambda x: num(lambda g: lambda h: h(g(f)))(lambda y: x)(lambda y: y)))(a))(a)(b))))(b)(a))))(amount)((((lambda f:(lambda x: f(lambda v: x(x)(v)))(lambda x: f(lambda v: x(x)(v))))(lambda rec: lambda f: lambda init: lambda lst:((lambda boolean: boolean)(((lambda pair: pair((lambda a: lambda b: a))))(lst))(init)(lambda x: f((lambda lst:((lambda pair: pair((lambda a: lambda b: a)))((lambda pair: pair((lambda a: lambda b: b)))(lst))))(lst))(rec(f)(init)((lambda lst:((lambda pair: pair((lambda a: lambda b: b)))((lambda pair: pair((lambda a: lambda b: b)))(lst))))(lst)))(x)))))((lambda a: lambda b: a((lambda num: lambda f: lambda x: f(num(f)(x))))(b)))((lambda f: lambda x: x)))(taken)))(taken)((lambda boolean: boolean)(((lambda pair: pair((lambda a: lambda b: a))))(coins))(((lambda head: lambda tail: lambda f: f(head)(tail))((lambda a: lambda b: a))((lambda a: lambda b: a))))(lambda x:((lambda a: lambda b:((lambda boolean: boolean)((lambda a: lambda b:((lambda boolean: boolean)(a)((lambda a: lambda b: a))(b)))(((lambda pair: pair((lambda a: lambda b: a))))(b))((lambda a: lambda b:((lambda boolean: boolean)(a)(b)((lambda a: lambda b: b))))((lambda a:((lambda boolean: boolean)(a)((lambda a: lambda b: b))((lambda a: lambda b: a))))(((lambda pair: pair((lambda a: lambda b: a))))(a)))((lambda a: lambda b:((lambda num: num(lambda x:((lambda a: lambda b: b)))((lambda a: lambda b: a)))((lambda a: lambda b: b((lambda num: lambda f: lambda x: num(lambda g: lambda h: h(g(f)))(lambda y: x)(lambda y: y)))(a))(a)(b))))((((lambda f:(lambda x: f(lambda v: x(x)(v)))(lambda x: f(lambda v: x(x)(v))))(lambda rec: lambda f: lambda init: lambda lst:((lambda boolean: boolean)(((lambda pair: pair((lambda a: lambda b: a))))(lst))(init)(lambda x: f((lambda lst:((lambda pair: pair((lambda a: lambda b: a)))((lambda pair: pair((lambda a: lambda b: b)))(lst))))(lst))(rec(f)(init)((lambda lst:((lambda pair: pair((lambda a: lambda b: b)))((lambda pair: pair((lambda a: lambda b: b)))(lst))))(lst)))(x)))))(lambda \_:((lambda a: lambda b: a((lambda num: lambda f: lambda x: f(num(f)(x))))(b))((lambda f: lambda x: f(x)))))((lambda f: lambda x: x)))(a))((((lambda f:(lambda x: f(lambda v: x(x)(v)))(lambda x: f(lambda v: x(x)(v))))(lambda rec: lambda f: lambda init: lambda lst:((lambda boolean: boolean)(((lambda pair: pair((lambda a: lambda b: a))))(lst))(init)(lambda x: f((lambda lst:((lambda pair: pair((lambda a: lambda b: a)))((lambda pair: pair((lambda a: lambda b: b)))(lst))))(lst))(rec(f)(init)((lambda lst:((lambda pair: pair((lambda a: lambda b: b)))((lambda pair: pair((lambda a: lambda b: b)))(lst))))(lst)))(x)))))(lambda \_:((lambda a: lambda b: a((lambda num: lambda f: lambda x: f(num(f)(x))))(b))((lambda f: lambda x: f(x)))))((lambda f: lambda x: x)))(b)))))(a)(b)))(rec(amount)((lambda lst:((lambda pair: pair((lambda a: lambda b: b)))((lambda pair: pair((lambda a: lambda b: b)))(lst))))(coins))((lambda el: lambda lst:((lambda head: lambda tail: lambda f: f(head)(tail))((lambda a: lambda b: b))((lambda head: lambda tail: lambda f: f(head)(tail))(el)(lst))))((lambda lst:((lambda pair: pair((lambda a: lambda b: a)))((lambda pair: pair((lambda a: lambda b: b)))(lst))))(coins))(taken)))(rec(amount)((lambda lst:((lambda pair: pair((lambda a: lambda b: b)))((lambda pair: pair((lambda a: lambda b: b)))(lst))))(coins))(taken)))(x))))))((lambda f: lambda x: f(f(f(f(f(f(f(f(f(f(x))))))))))))(((lambda f:(lambda x: f(lambda v: x(x)(v)))(lambda x: f(lambda v: x(x)(v))))(lambda rec: lambda start: lambda end: IF((lambda a: lambda b:((lambda num: num(lambda x:((lambda a: lambda b: b)))((lambda a: lambda b: a)))((lambda a: lambda b: b((lambda num: lambda f: lambda x: num(lambda g: lambda h: h(g(f)))(lambda y: x)(lambda y: y)))(a))(a)(b))))(end)(start))(((lambda head: lambda tail: lambda f: f(head)(tail))((lambda a: lambda b: a))((lambda a: lambda b: a))))(lambda x:((lambda el: lambda lst:((lambda head: lambda tail: lambda f: f(head)(tail))((lambda a: lambda b: b))((lambda head: lambda tail: lambda f: f(head)(tail))(el)(lst))))(start)(rec((lambda num: lambda f: lambda x: f(num(f)(x)))(start))(end)))(x))))((lambda f: lambda x: f(x)))(((lambda num: lambda f: lambda x: f(num(f)(x)))((lambda f: lambda x: f(f(f(f(f(x))))))))))(((lambda head: lambda tail: lambda f: f(head)(tail))((lambda a: lambda b: a))((lambda a: lambda b: a))))

map(to\_int, to\_list(a))

Out[176]:

[5, 4, 1]

Lisaks - MAP ja FILTER, algarvud ning fizzbuzz

In [62]:

MAP = lambda f: lambda lst: \

REDUCE(lambda el: PREPEND(f(el)))(NIL)(lst)

MAP = lambda f: REDUCE(lambda el: PREPEND(f(el)))(NIL)

In [61]:

SQUARE = lambda num: MULTIPLY(num)(num)

map(to\_int, to\_list(MAP(SQUARE)(RANGE(ONE)(SIX))))

Out[61]:

[1, 4, 9, 16, 25]

In [80]:

FILTER = lambda pred: REDUCE(lambda el: lambda rest: \

IF(pred(el))

(PREPEND(el)(rest))

(rest))(NIL)

In [88]:

map(to\_int, to\_list(

FILTER(lambda x: LESS\_THAN\_EQUAL(x)(FIVE))

(RANGE(ONE)(NINE))))

Out[88]:

[1, 2, 3, 4, 5]

In [96]:

MOD = Y(lambda rec: lambda a: lambda b: \

IF(LESS\_THAN\_EQUAL(b)(a))

(lambda x: (rec(MINUS(a)(b))(b) )(x))

(a))

to\_int(MOD(TEN)(FOUR))

Out[96]:

2

In [112]:

IS\_PRIME = lambda num: \

IS\_EMPTY(

FILTER(lambda x: IS\_ZERO(MOD(num)(x)))

(RANGE(TWO)(num)))

In [113]:

to\_bool(IS\_PRIME(NINE))

Out[113]:

False

In [115]:

SEVEN = PLUS(TWO)(FIVE)

to\_bool(IS\_PRIME(SEVEN))

Out[115]:

True

In [168]:

sieve = lambda upperbound: FILTER(IS\_PRIME)(RANGE(TWO)(upperbound))

map(to\_int, to\_list(sieve(POWER(TWO)(SIX))))

Out[168]:

[2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61]

In [191]:

F = TEN

I = ELEVEN = INCREMENT(TEN)

Z = TWELVE = INCREMENT(ELEVEN)

B = THIRTEEN = INCREMENT(TWELVE)

U = FOURTEEN = INCREMENT(THIRTEEN)

FIFTEEN = MULTIPLY(THREE)(FIVE)

HUNDRED = POWER(TEN)(TWO)

ZZ = PREPEND(Z)(PREPEND(Z)(NIL))

FIZZ = PREPEND(F)(PREPEND(I)(ZZ))

BUZZ = PREPEND(B)(PREPEND(U)(ZZ))

FIZZBUZZ = PREPEND(F)(PREPEND(I)(PREPEND(Z)(PREPEND(Z)(BUZZ))))

In [192]:

APPEND = lambda el: REDUCE(PREPEND)(PREPEND(el)(NIL))

DIV = Y(lambda rec: lambda a: lambda b:

IF(LESS\_THAN\_EQUAL(b)(a))

(lambda x: INCREMENT(rec(MINUS(a)(b))(b))(x))

(ZERO))

TO\_DIGITS = Y(lambda rec: lambda num:

IF(LESS\_THAN\_EQUAL(num)(NINE))

(PREPEND(num)(NIL))

(lambda x: APPEND(MOD(num)(TEN)) (rec(DIV(num)(TEN)))(x)))

In [193]:

def to\_string(lst):

chartable = "0123456789FIZBU"

codes = map(to\_int, to\_list(lst))

return "".join(chartable[x] for x in codes)

In [194]:

FB = lambda num: (

IF(IS\_ZERO(MOD(num)(FIFTEEN)))

(FIZZBUZZ)

(IF(IS\_ZERO(MOD(num)(FIVE)))

(BUZZ)

(IF(IS\_ZERO(MOD(num)(THREE)))

(FIZZ)

(TO\_DIGITS(num)))))

\_101 = INCREMENT(HUNDRED)

a = MAP(FB)(RANGE(ONE)(\_101))

map(to\_string, to\_list(a))

Out[194]:

['1',

'2',

'FIZZ',

'4',

'BUZZ',

'FIZZ',

'7',

'8',

'FIZZ',

'BUZZ',

'11',

'FIZZ',

'13',

'14',

'FIZZBUZZ',

'16',

'17',

'FIZZ',

'19',

'BUZZ',

'FIZZ',

'22',

'23',

'FIZZ',

'BUZZ',

'26',

'FIZZ',

'28',

'29',

'FIZZBUZZ',

'31',

'32',

'FIZZ',

'34',

'BUZZ',

'FIZZ',

'37',

'38',

'FIZZ',

'BUZZ',

'41',

'FIZZ',

'43',

'44',

'FIZZBUZZ',

'46',

'47',

'FIZZ',

'49',

'BUZZ',

'FIZZ',

'52',

'53',

'FIZZ',

'BUZZ',

'56',

'FIZZ',

'58',

'59',

'FIZZBUZZ',

'61',

'62',

'FIZZ',

'64',

'BUZZ',

'FIZZ',

'67',

'68',

'FIZZ',

'BUZZ',

'71',

'FIZZ',

'73',

'74',

'FIZZBUZZ',

'76',

'77',

'FIZZ',

'79',

'BUZZ',

'FIZZ',

'82',

'83',

'FIZZ',

'BUZZ',

'86',

'FIZZ',

'88',

'89',

'FIZZBUZZ',

'91',

'92',

'FIZZ',

'94',

'BUZZ',

'FIZZ',

'97',

'98',

'FIZZ',

'BUZZ']

In [173]:

a = (lambda f: (((lambda f: (lambda x: f(lambda v: x(x)(v)))(lambda x: f(lambda v: x(x)(v))))(lambda rec: lambda f: lambda init: lambda lst: ((lambda boolean: boolean)(((lambda pair: pair((lambda a: lambda b: a))))(lst))(init)(lambda x: f((lambda lst: ((lambda pair: pair((lambda a: lambda b: a)))((lambda pair: pair((lambda a: lambda b: b)))(lst))))(lst))(rec(f)(init)((lambda lst: ((lambda pair: pair((lambda a: lambda b: b)))((lambda pair: pair((lambda a: lambda b: b)))(lst))))(lst)))(x)))))(lambda el: ((lambda el: lambda lst: ((lambda head: lambda tail: lambda f: f(head)(tail))((lambda a: lambda b: b))((lambda head: lambda tail: lambda f: f(head)(tail))(el)(lst))))(f(el))))(((lambda head: lambda tail: lambda f: f(head)(tail))((lambda a: lambda b: a))((lambda a: lambda b: a))))))((lambda num: ((lambda boolean: boolean)((lambda num: num(lambda x: ((lambda a: lambda b: b)))((lambda a: lambda b: a)))(((lambda f: (lambda x: f(lambda v: x(x)(v)))(lambda x: f(lambda v: x(x)(v))))(lambda rec: lambda a: lambda b: ((lambda boolean: boolean)((lambda a: lambda b: ((lambda num: num(lambda x: ((lambda a: lambda b: b)))((lambda a: lambda b: a)))((lambda a: lambda b: b((lambda num: lambda f: lambda x: num(lambda g: lambda h: h(g(f)))(lambda y: x)(lambda y: y)))(a))(a)(b))))(b)(a)) (lambda x: (rec((lambda a: lambda b: b((lambda num: lambda f: lambda x: num(lambda g: lambda h: h(g(f)))(lambda y: x)(lambda y: y)))(a))(a)(b))(b) )(x)) (a))))(num)(((lambda a: lambda b: a((lambda a: lambda b: a((lambda num: lambda f: lambda x: f(num(f)(x))))(b))(b))((lambda f: lambda x: x)))((lambda f: lambda x: f(f(f(x)))))((lambda f: lambda x: f(f(f(f(f(x))))))))))) (((lambda el: lambda lst: ((lambda head: lambda tail: lambda f: f(head)(tail))((lambda a: lambda b: b))((lambda head: lambda tail: lambda f: f(head)(tail))(el)(lst))))(((lambda f: lambda x: f(f(f(f(f(f(f(f(f(f(x)))))))))))))((lambda el: lambda lst: ((lambda head: lambda tail: lambda f: f(head)(tail))((lambda a: lambda b: b))((lambda head: lambda tail: lambda f: f(head)(tail))(el)(lst))))((((lambda num: lambda f: lambda x: f(num(f)(x)))((lambda f: lambda x: f(f(f(f(f(f(f(f(f(f(x)))))))))))))))((lambda el: lambda lst: ((lambda head: lambda tail: lambda f: f(head)(tail))((lambda a: lambda b: b))((lambda head: lambda tail: lambda f: f(head)(tail))(el)(lst))))((((lambda num: lambda f: lambda x: f(num(f)(x)))(((lambda num: lambda f: lambda x: f(num(f)(x)))((lambda f: lambda x: f(f(f(f(f(f(f(f(f(f(x)))))))))))))))))((lambda el: lambda lst: ((lambda head: lambda tail: lambda f: f(head)(tail))((lambda a: lambda b: b))((lambda head: lambda tail: lambda f: f(head)(tail))(el)(lst))))((((lambda num: lambda f: lambda x: f(num(f)(x)))(((lambda num: lambda f: lambda x: f(num(f)(x)))((lambda f: lambda x: f(f(f(f(f(f(f(f(f(f(x)))))))))))))))))(((lambda el: lambda lst: ((lambda head: lambda tail: lambda f: f(head)(tail))((lambda a: lambda b: b))((lambda head: lambda tail: lambda f: f(head)(tail))(el)(lst))))((((lambda num: lambda f: lambda x: f(num(f)(x)))(((lambda num: lambda f: lambda x: f(num(f)(x)))(((lambda num: lambda f: lambda x: f(num(f)(x)))((lambda f: lambda x: f(f(f(f(f(f(f(f(f(f(x)))))))))))))))))))((lambda el: lambda lst: ((lambda head: lambda tail: lambda f: f(head)(tail))((lambda a: lambda b: b))((lambda head: lambda tail: lambda f: f(head)(tail))(el)(lst))))((((lambda num: lambda f: lambda x: f(num(f)(x)))(((lambda num: lambda f: lambda x: f(num(f)(x)))(((lambda num: lambda f: lambda x: f(num(f)(x)))(((lambda num: lambda f: lambda x: f(num(f)(x)))((lambda f: lambda x: f(f(f(f(f(f(f(f(f(f(x)))))))))))))))))))))(((lambda el: lambda lst: ((lambda head: lambda tail: lambda f: f(head)(tail))((lambda a: lambda b: b))((lambda head: lambda tail: lambda f: f(head)(tail))(el)(lst))))((((lambda num: lambda f: lambda x: f(num(f)(x)))(((lambda num: lambda f: lambda x: f(num(f)(x)))((lambda f: lambda x: f(f(f(f(f(f(f(f(f(f(x)))))))))))))))))((lambda el: lambda lst: ((lambda head: lambda tail: lambda f: f(head)(tail))((lambda a: lambda b: b))((lambda head: lambda tail: lambda f: f(head)(tail))(el)(lst))))((((lambda num: lambda f: lambda x: f(num(f)(x)))(((lambda num: lambda f: lambda x: f(num(f)(x)))((lambda f: lambda x: f(f(f(f(f(f(f(f(f(f(x)))))))))))))))))(((lambda head: lambda tail: lambda f: f(head)(tail))((lambda a: lambda b: a))((lambda a: lambda b: a))))))))))))))) ((lambda boolean: boolean)((lambda num: num(lambda x: ((lambda a: lambda b: b)))((lambda a: lambda b: a)))(((lambda f: (lambda x: f(lambda v: x(x)(v)))(lambda x: f(lambda v: x(x)(v))))(lambda rec: lambda a: lambda b: ((lambda boolean: boolean)((lambda a: lambda b: ((lambda num: num(lambda x: ((lambda a: lambda b: b)))((lambda a: lambda b: a)))((lambda a: lambda b: b((lambda num: lambda f: lambda x: num(lambda g: lambda h: h(g(f)))(lambda y: x)(lambda y: y)))(a))(a)(b))))(b)(a)) (lambda x: (rec((lambda a: lambda b: b((lambda num: lambda f: lambda x: num(lambda g: lambda h: h(g(f)))(lambda y: x)(lambda y: y)))(a))(a)(b))(b) )(x)) (a))))(num)((lambda f: lambda x: f(f(f(f(f(x))))))))) (((lambda el: lambda lst: ((lambda head: lambda tail: lambda f: f(head)(tail))((lambda a: lambda b: b))((lambda head: lambda tail: lambda f: f(head)(tail))(el)(lst))))((((lambda num: lambda f: lambda x: f(num(f)(x)))(((lambda num: lambda f: lambda x: f(num(f)(x)))(((lambda num: lambda f: lambda x: f(num(f)(x)))((lambda f: lambda x: f(f(f(f(f(f(f(f(f(f(x)))))))))))))))))))((lambda el: lambda lst: ((lambda head: lambda tail: lambda f: f(head)(tail))((lambda a: lambda b: b))((lambda head: lambda tail: lambda f: f(head)(tail))(el)(lst))))((((lambda num: lambda f: lambda x: f(num(f)(x)))(((lambda num: lambda f: lambda x: f(num(f)(x)))(((lambda num: lambda f: lambda x: f(num(f)(x)))(((lambda num: lambda f: lambda x: f(num(f)(x)))((lambda f: lambda x: f(f(f(f(f(f(f(f(f(f(x)))))))))))))))))))))(((lambda el: lambda lst: ((lambda head: lambda tail: lambda f: f(head)(tail))((lambda a: lambda b: b))((lambda head: lambda tail: lambda f: f(head)(tail))(el)(lst))))((((lambda num: lambda f: lambda x: f(num(f)(x)))(((lambda num: lambda f: lambda x: f(num(f)(x)))((lambda f: lambda x: f(f(f(f(f(f(f(f(f(f(x)))))))))))))))))((lambda el: lambda lst: ((lambda head: lambda tail: lambda f: f(head)(tail))((lambda a: lambda b: b))((lambda head: lambda tail: lambda f: f(head)(tail))(el)(lst))))((((lambda num: lambda f: lambda x: f(num(f)(x)))(((lambda num: lambda f: lambda x: f(num(f)(x)))((lambda f: lambda x: f(f(f(f(f(f(f(f(f(f(x)))))))))))))))))(((lambda head: lambda tail: lambda f: f(head)(tail))((lambda a: lambda b: a))((lambda a: lambda b: a)))))))))) ((lambda boolean: boolean)((lambda num: num(lambda x: ((lambda a: lambda b: b)))((lambda a: lambda b: a)))(((lambda f: (lambda x: f(lambda v: x(x)(v)))(lambda x: f(lambda v: x(x)(v))))(lambda rec: lambda a: lambda b: ((lambda boolean: boolean)((lambda a: lambda b: ((lambda num: num(lambda x: ((lambda a: lambda b: b)))((lambda a: lambda b: a)))((lambda a: lambda b: b((lambda num: lambda f: lambda x: num(lambda g: lambda h: h(g(f)))(lambda y: x)(lambda y: y)))(a))(a)(b))))(b)(a)) (lambda x: (rec((lambda a: lambda b: b((lambda num: lambda f: lambda x: num(lambda g: lambda h: h(g(f)))(lambda y: x)(lambda y: y)))(a))(a)(b))(b) )(x)) (a))))(num)((lambda f: lambda x: f(f(f(x))))))) (((lambda el: lambda lst: ((lambda head: lambda tail: lambda f: f(head)(tail))((lambda a: lambda b: b))((lambda head: lambda tail: lambda f: f(head)(tail))(el)(lst))))(((lambda f: lambda x: f(f(f(f(f(f(f(f(f(f(x)))))))))))))((lambda el: lambda lst: ((lambda head: lambda tail: lambda f: f(head)(tail))((lambda a: lambda b: b))((lambda head: lambda tail: lambda f: f(head)(tail))(el)(lst))))((((lambda num: lambda f: lambda x: f(num(f)(x)))((lambda f: lambda x: f(f(f(f(f(f(f(f(f(f(x)))))))))))))))(((lambda el: lambda lst: ((lambda head: lambda tail: lambda f: f(head)(tail))((lambda a: lambda b: b))((lambda head: lambda tail: lambda f: f(head)(tail))(el)(lst))))((((lambda num: lambda f: lambda x: f(num(f)(x)))(((lambda num: lambda f: lambda x: f(num(f)(x)))((lambda f: lambda x: f(f(f(f(f(f(f(f(f(f(x)))))))))))))))))((lambda el: lambda lst: ((lambda head: lambda tail: lambda f: f(head)(tail))((lambda a: lambda b: b))((lambda head: lambda tail: lambda f: f(head)(tail))(el)(lst))))((((lambda num: lambda f: lambda x: f(num(f)(x)))(((lambda num: lambda f: lambda x: f(num(f)(x)))((lambda f: lambda x: f(f(f(f(f(f(f(f(f(f(x)))))))))))))))))(((lambda head: lambda tail: lambda f: f(head)(tail))((lambda a: lambda b: a))((lambda a: lambda b: a)))))))))) (((lambda f: (lambda x: f(lambda v: x(x)(v)))(lambda x: f(lambda v: x(x)(v))))(lambda rec: lambda num: ((lambda boolean: boolean)((lambda a: lambda b: ((lambda num: num(lambda x: ((lambda a: lambda b: b)))((lambda a: lambda b: a)))((lambda a: lambda b: b((lambda num: lambda f: lambda x: num(lambda g: lambda h: h(g(f)))(lambda y: x)(lambda y: y)))(a))(a)(b))))(num)(((lambda a: lambda b: a((lambda a: lambda b: a((lambda num: lambda f: lambda x: f(num(f)(x))))(b))(b))((lambda f: lambda x: x)))((lambda f: lambda x: f(f(f(x)))))((lambda f: lambda x: f(f(f(x)))))))) ((lambda el: lambda lst: ((lambda head: lambda tail: lambda f: f(head)(tail))((lambda a: lambda b: b))((lambda head: lambda tail: lambda f: f(head)(tail))(el)(lst))))(num)(((lambda head: lambda tail: lambda f: f(head)(tail))((lambda a: lambda b: a))((lambda a: lambda b: a))))) (lambda x: ((lambda el: (((lambda f: (lambda x: f(lambda v: x(x)(v)))(lambda x: f(lambda v: x(x)(v))))(lambda rec: lambda f: lambda init: lambda lst: ((lambda boolean: boolean)(((lambda pair: pair((lambda a: lambda b: a))))(lst))(init)(lambda x: f((lambda lst: ((lambda pair: pair((lambda a: lambda b: a)))((lambda pair: pair((lambda a: lambda b: b)))(lst))))(lst))(rec(f)(init)((lambda lst: ((lambda pair: pair((lambda a: lambda b: b)))((lambda pair: pair((lambda a: lambda b: b)))(lst))))(lst)))(x)))))((lambda el: lambda lst: ((lambda head: lambda tail: lambda f: f(head)(tail))((lambda a: lambda b: b))((lambda head: lambda tail: lambda f: f(head)(tail))(el)(lst)))))((lambda el: lambda lst: ((lambda head: lambda tail: lambda f: f(head)(tail))((lambda a: lambda b: b))((lambda head: lambda tail: lambda f: f(head)(tail))(el)(lst))))(el)(((lambda head: lambda tail: lambda f: f(head)(tail))((lambda a: lambda b: a))((lambda a: lambda b: a)))))))(((lambda f: (lambda x: f(lambda v: x(x)(v)))(lambda x: f(lambda v: x(x)(v))))(lambda rec: lambda a: lambda b: ((lambda boolean: boolean)((lambda a: lambda b: ((lambda num: num(lambda x: ((lambda a: lambda b: b)))((lambda a: lambda b: a)))((lambda a: lambda b: b((lambda num: lambda f: lambda x: num(lambda g: lambda h: h(g(f)))(lambda y: x)(lambda y: y)))(a))(a)(b))))(b)(a)) (lambda x: (rec((lambda a: lambda b: b((lambda num: lambda f: lambda x: num(lambda g: lambda h: h(g(f)))(lambda y: x)(lambda y: y)))(a))(a)(b))(b) )(x)) (a))))(num)((lambda f: lambda x: f(f(f(f(f(f(f(f(f(f(x))))))))))))) (rec(((lambda f: (lambda x: f(lambda v: x(x)(v)))(lambda x: f(lambda v: x(x)(v))))(lambda rec: lambda a: lambda b: ((lambda boolean: boolean)((lambda a: lambda b: ((lambda num: num(lambda x: ((lambda a: lambda b: b)))((lambda a: lambda b: a)))((lambda a: lambda b: b((lambda num: lambda f: lambda x: num(lambda g: lambda h: h(g(f)))(lambda y: x)(lambda y: y)))(a))(a)(b))))(b)(a)) (lambda x: ((lambda num: lambda f: lambda x: f(num(f)(x)))(rec((lambda a: lambda b: b((lambda num: lambda f: lambda x: num(lambda g: lambda h: h(g(f)))(lambda y: x)(lambda y: y)))(a))(a)(b))(b))(x))) ((lambda f: lambda x: x)))))(num)((lambda f: lambda x: f(f(f(f(f(f(f(f(f(f(x))))))))))))))(x))))))(num)))))))(((lambda f: (lambda x: f(lambda v: x(x)(v)))(lambda x: f(lambda v: x(x)(v))))(lambda rec: lambda start: lambda end: ((lambda boolean: boolean)((lambda a: lambda b: ((lambda num: num(lambda x: ((lambda a: lambda b: b)))((lambda a: lambda b: a)))((lambda a: lambda b: b((lambda num: lambda f: lambda x: num(lambda g: lambda h: h(g(f)))(lambda y: x)(lambda y: y)))(a))(a)(b))))(end)(start))(((lambda head: lambda tail: lambda f: f(head)(tail))((lambda a: lambda b: a))((lambda a: lambda b: a))))(lambda x: ((lambda el: lambda lst: ((lambda head: lambda tail: lambda f: f(head)(tail))((lambda a: lambda b: b))((lambda head: lambda tail: lambda f: f(head)(tail))(el)(lst))))(start)(rec((lambda num: lambda f: lambda x: f(num(f)(x)))(start))(end)) )(x)))))((lambda f: lambda x: f(x)))(((lambda num: lambda f: lambda x: f(num(f)(x)))(((lambda a: lambda b: b(a))((lambda f: lambda x: f(f(f(f(f(f(f(f(f(f(x))))))))))))((lambda f: lambda x: f(f(x)))))))))

# map(to\_string, to\_list(a))

Lisalugemist:

[Parser mis genereeris laiendatud avaldised](https://gist.github.com/macobo/5525312)

[Algne inspiratsioon](http://codon.com/programming-with-nothing)

[Lambda calculus wiki](http://en.wikipedia.org/wiki/Lambda_calculus)

[Back to top](http://nbviewer.ipython.org/5523724)

More info on [IPython website](http://ipython.org)

[The code for this site](https://github.com/ipython/nbviewer) is licensed under the [Apache License v2.0](http://www.apache.org/licenses/LICENSE-2.0). Documentation licensed under [CC BY 3.0](http://creativecommons.org/licenses/by/3.0/).

Some icons from [Glyphicons Free](http://glyphicons.com), built thanks to [Twitter Bootstrap](http://twitter.github.com/bootstrap/), licensed under [CC BY 3.0](http://creativecommons.org/licenses/by/3.0/).

This web site does not host notebooks, it only renders notebooks available on other websites.

Thanks to [Zoltàn Vörös](https://github.com/v923z) for his help