VINERI
APR 12

LENEA
SUPERPUTEREA SECOLULUI 21

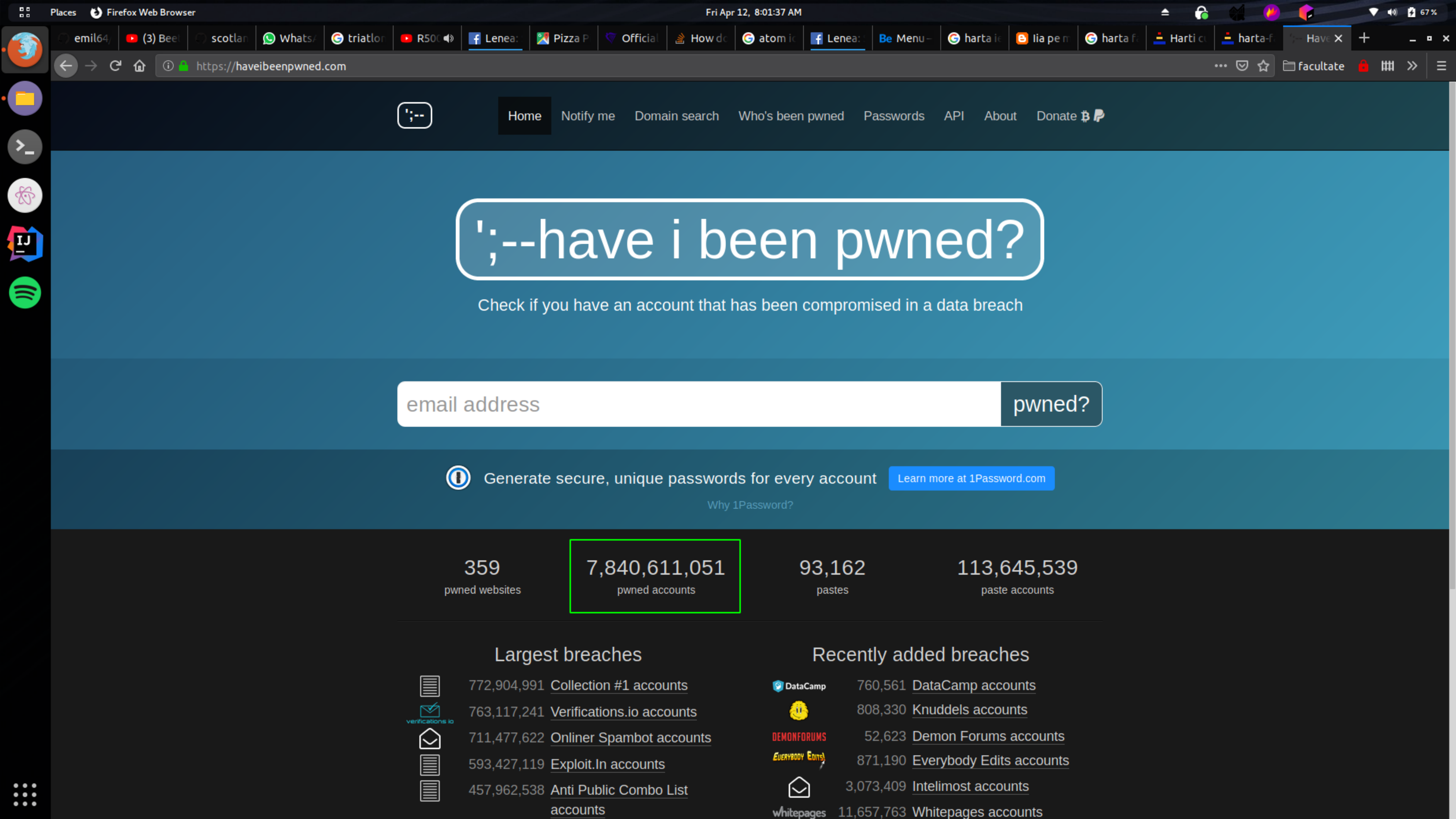
INTRODUCERE IN PROGRAMAREA FUNCTIONALA

Lasa pe poimaine ce poti face azi, poate nu va mai fi nevoie.

Societatea Științifică Brătianu
since 2016

Voluntari implicați:

- Liana Ivașcu
- Alexandra Popa
- Berevoianu Dinu
- Ioniță Ana Maria
- Luiza Dumbravă



Fri Apr 12, 8:01:37 AM

https://haveibeenpwned.com

Home Notify me Domain search Who's been pwned Passwords API About Donate

';--have i been pwned?

Check if you have an account that has been compromised in a data breach

email address pwned?

Generate secure, unique passwords for every account Learn more at 1Password.com

Why 1Password?

359	7,840,611,051	93,162	113,645,539
pwned websites	pwned accounts	pastes	paste accounts

Largest breaches

772,904,991	Collection #1 accounts
763,117,241	Verifications.io accounts
711,477,622	Onliner Spambot accounts
593,427,119	Exploit.In accounts
457,962,538	Anti Public Combo List accounts

Recently added breaches

760,561	DataCamp accounts
808,330	Knuddels accounts
52,623	Demon Forums accounts
871,190	Everybody Edits accounts
3,073,409	Intelimost accounts
11,657,763	Whitepages accounts

Pământul este PLAT!!!

Imperativ

Imperativ

C/C++

JavaScript

Java

php

Hibrid

Imperativ

Hibrid

Functional

C/C++

JavaScript

Java

php

LISP

HASKELL
DEVS
C++
imgflip.com

Haskell

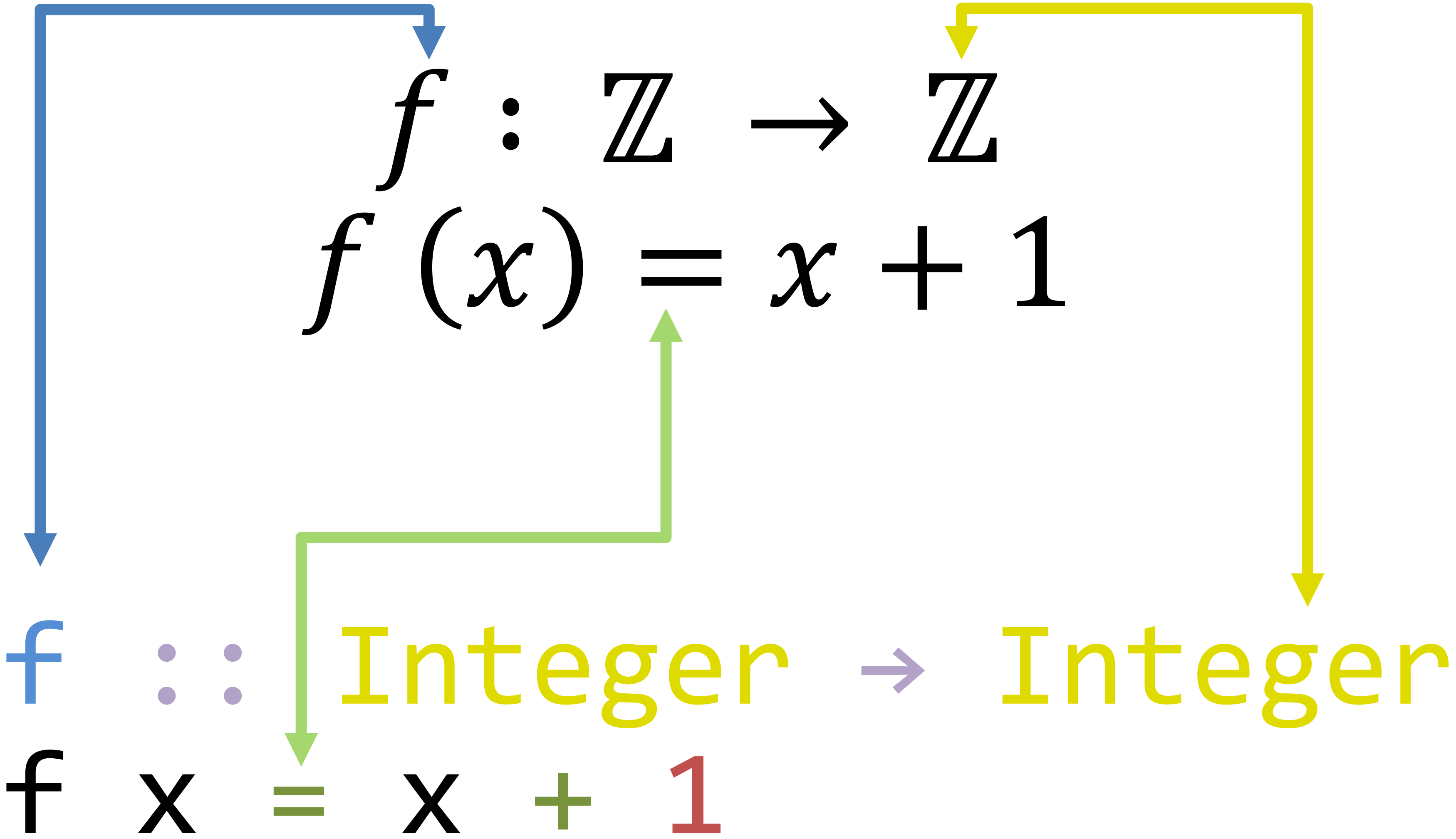
- Funcții pure
- Recursivitate
- Referință transparentă
- Evaluare leneșă
- Funcții tratate ca variablie

$$f : \mathbb{Z} \to \mathbb{Z}$$

$$f(x) = x + 1$$

$$f : \mathbb{Z} \to \mathbb{Z}$$
$$f(x) = x + 1$$

```haskell
f :: Integer → Integer
f x = x + 1
```



$$f : \mathbb{Z} \to \mathbb{Z}$$
$$f(x) = x + 1$$

```
f :: Integer → Integer
f x = x + 1
```

Specialized DEMO 2018

LIVE DEMO

În căutarea puterii...

```
x :: Integer
```

În căutarea puterii...

```
x :: Integer
f :: Integer → Integer
```

În căutarea puterii...

```
x :: Integer
f :: Integer → Integer

(3 + ) ::
```

În căutarea puterii...

```
x :: Integer
f :: Integer → Integer

(3 + ) :: Integer → Integer
```

În căutarea puterii...

```
x :: Integer
f :: Integer → Integer

(3 + ) :: Integer → Integer
3 :: Integer
```

În căutarea puterii...

```
x :: Integer
f :: Integer → Integer

(3 + ) :: Integer → Integer
3 :: Integer
```

În căutarea puterii...

$$\frac{\begin{array}{l} \texttt{x :: Integer} \\ \texttt{f :: Integer} \to \texttt{Integer} \\ \\ \texttt{(3 +) :: Integer} \to \texttt{Integer} \\ \texttt{3 :: Integer} \end{array}}{\texttt{(+) :: Integer} \to \texttt{(Integer} \to \texttt{Integer)}}$$

În căutarea puterii...

```
add :: (Integer, Integer) → Integer
add x y = x + y

plus :: Integer → Integer → Integer
plus x y = x + y
```

În căutarea puterii...

```haskell
add :: (Integer, Integer) → Integer
add x y = x + y

plus :: Integer → Integer → Integer
plus x y = x + y

plus3 :: Integer → Integer
plus3 = plus 3
```

LIVE DEMO

I LOVE
LISTS
memegenerator.net

Liste

```
[1, 2, 3, 4] :: [Integer]
[1] :: [Integer]
```

Liste

```
[1, 2, 3, 4] :: [Integer]
[1] :: [Integer]

[True, False, True] :: [Bool]
[“Ana”, “are”, “mere”] :: [String]
```

Liste

```
[1, 2, 3, 4] :: [Integer]
[1] :: [Integer]

[True, False, True] :: [Bool]
[“Ana”, “are”, “mere”] :: [String]

[] ::
```

Liste

```
[1, 2, 3, 4] :: [Integer]
[1] :: [Integer]

[True, False, True] :: [Bool]
[“Ana”, “are”, “mere”] :: [String]

[] ::                  --[Integer]? [Bool]? …?
```

Liste

```
[1, 2, 3, 4] :: [a]
[1] :: [b]

[True, False, True] :: [c]
[“Ana”, “are”, “mere”] :: [d]

[] ::
```

Liste

```
[1, 2, 3, 4] :: [a]          --a = Integer
[1] :: [b]                   --b = Integer

[True, False, True] :: [c]          --c = Bool
[“Ana”, “are”, “mere”] :: [d]       --d = String

[] :: [a]                    --a = Orice ☺
```

Liste

```
[1, 2, 3, 4] :: a              --a = [Integer]
[1] :: b                       --b = [Integer]

[True, False, True] :: c              --c = [Bool]
[“Ana”, “are”, “mere”] :: d           --d = [String]

[] :: a                        --a = Orice ☺
```

Liste

```
[1, 2, 3, 4] :: [a]
```

=

```
1 : [2, 3, 4] :: [a]
```

Liste

```
[1, 2, 3, 4] :: [a]
```

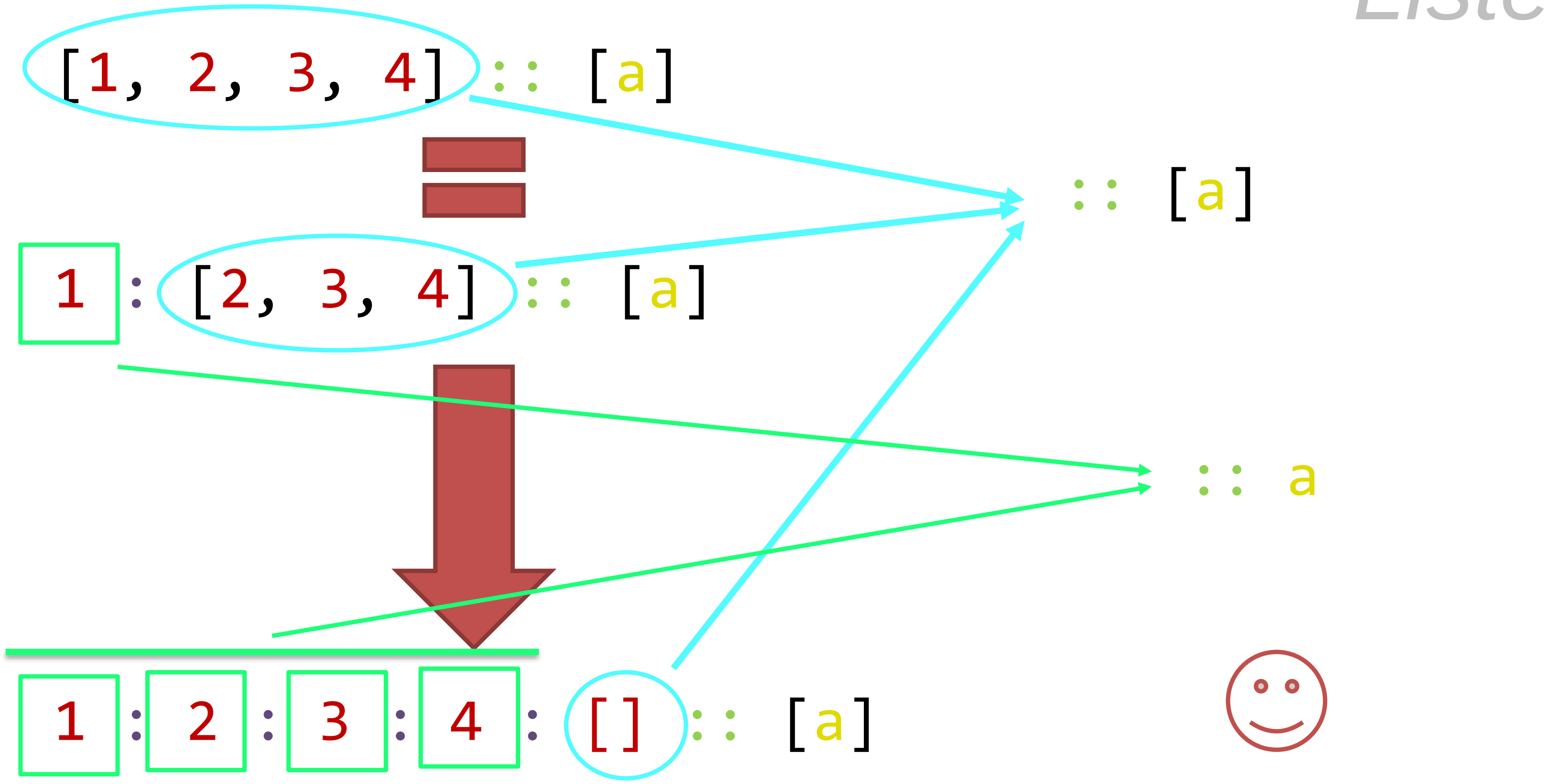
$$=$$

```
1 : [2, 3, 4] :: [a]
```

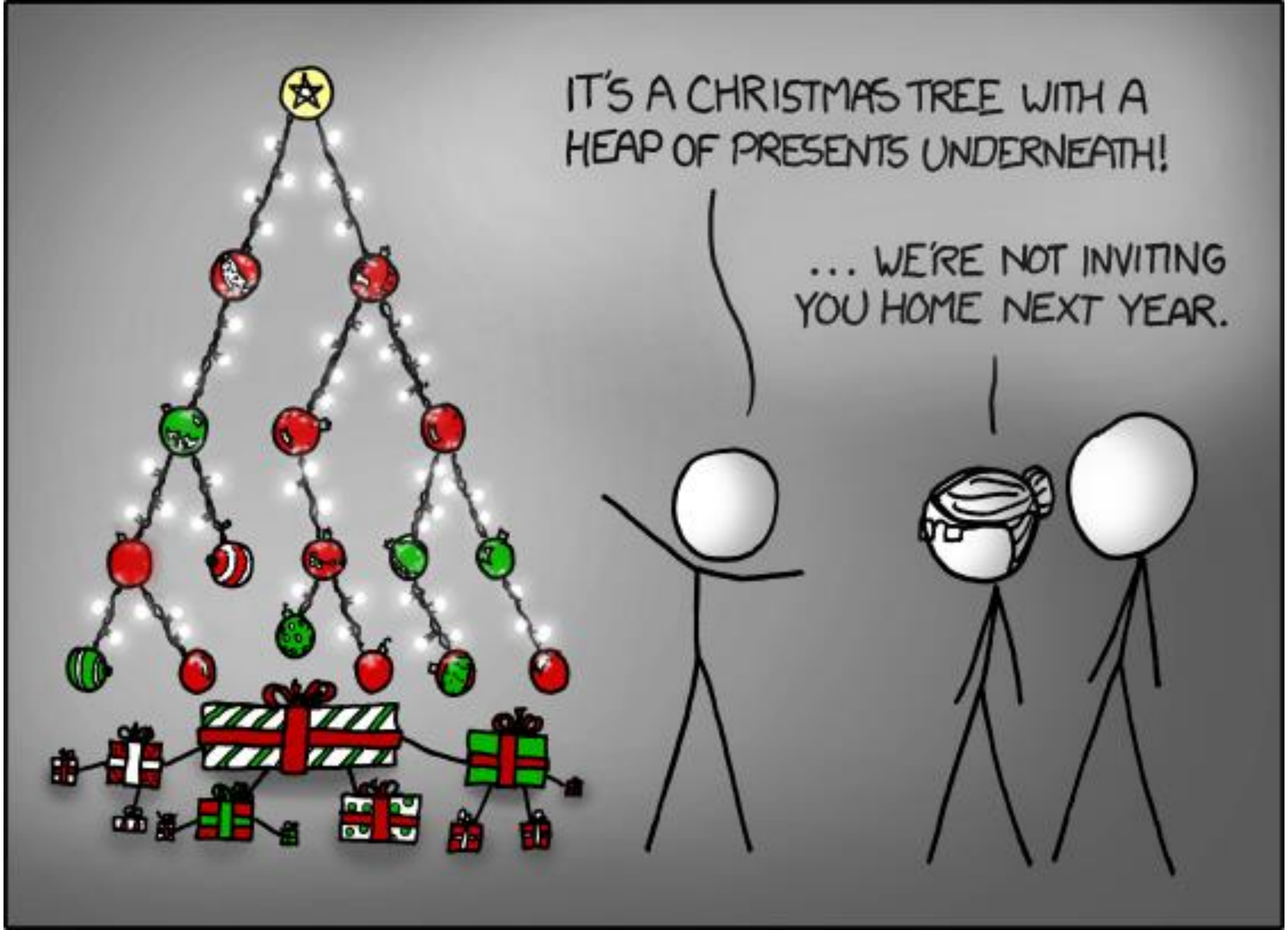
$$\Downarrow$$

```
1 : 2 : 3 : 4 : [] :: [a]
```

Liste



LIVE DEMO


IT'S A CHRISTMAS TREE WITH A HEAP OF PRESENTS UNDERNEATH!
... WE'RE NOT INVITING YOU HOME NEXT YEAR.

```haskell
data Culori = Rosu | Verde | Albastru

data Culori a = Rosu a | Verde a | Albastru a
data Culori = Rosu | Verde | Albastru

data Culori a = Rosu a | Verde a | Albastru a
```

```
data Culori = Rosu | Verde | Albastru

data Culori a = Rosu a | Verde a | Albastru a
```

$$\frac{5}{0} = ???$$

```
data Maybe a = Nothing | Just a
```

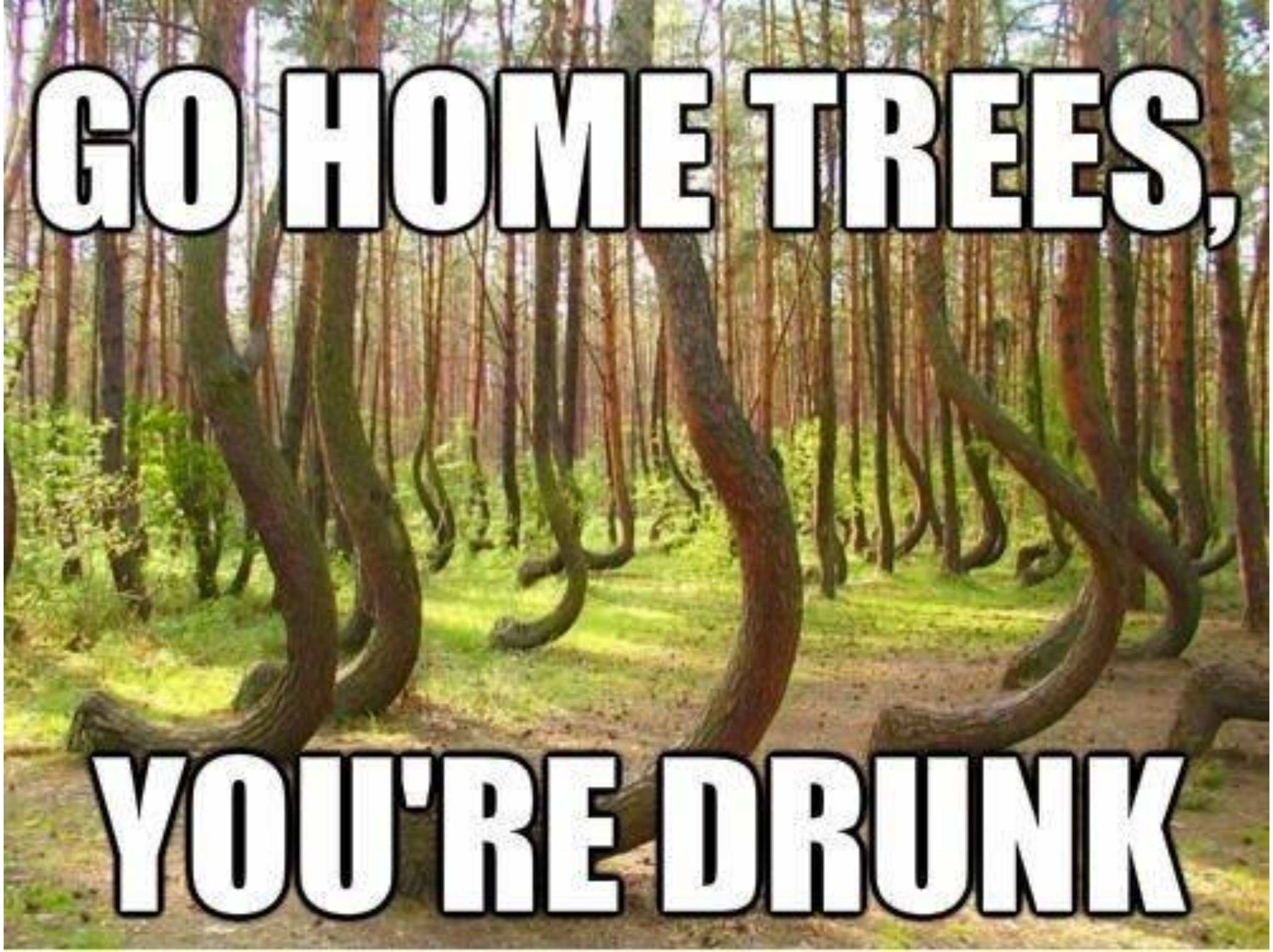
Structuri de date

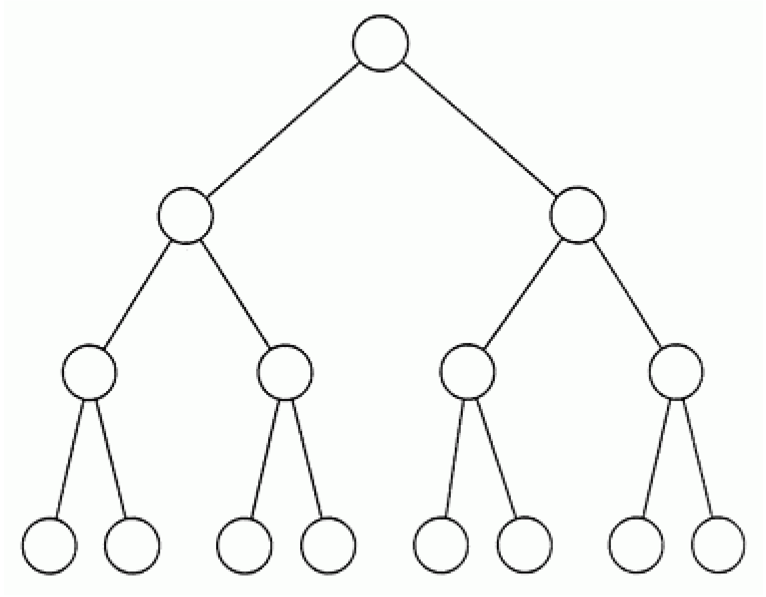
```
data Maybe a = Nothing | Just a

impartire :: Int → Int → Maybe Int
```

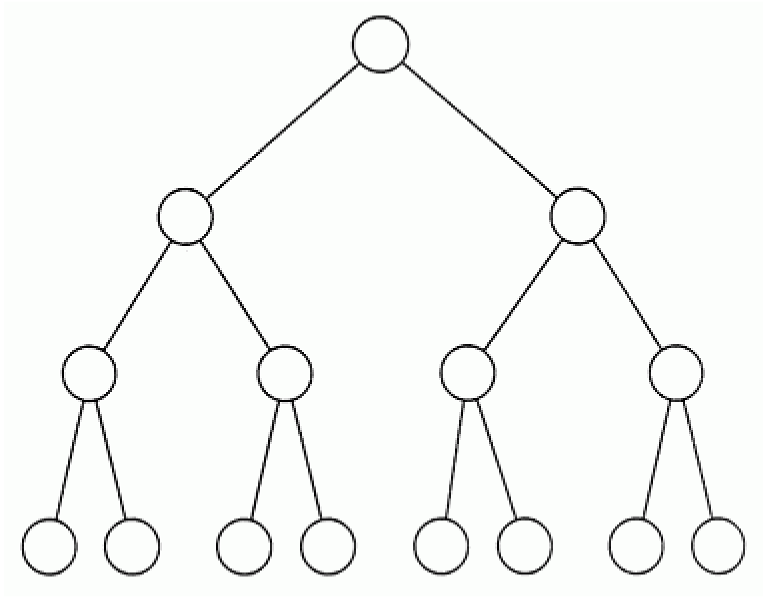
```
data Maybe a = Nothing | Just a

impartire :: Int → Int → Maybe Int
impartire a 0 = Nothing
```

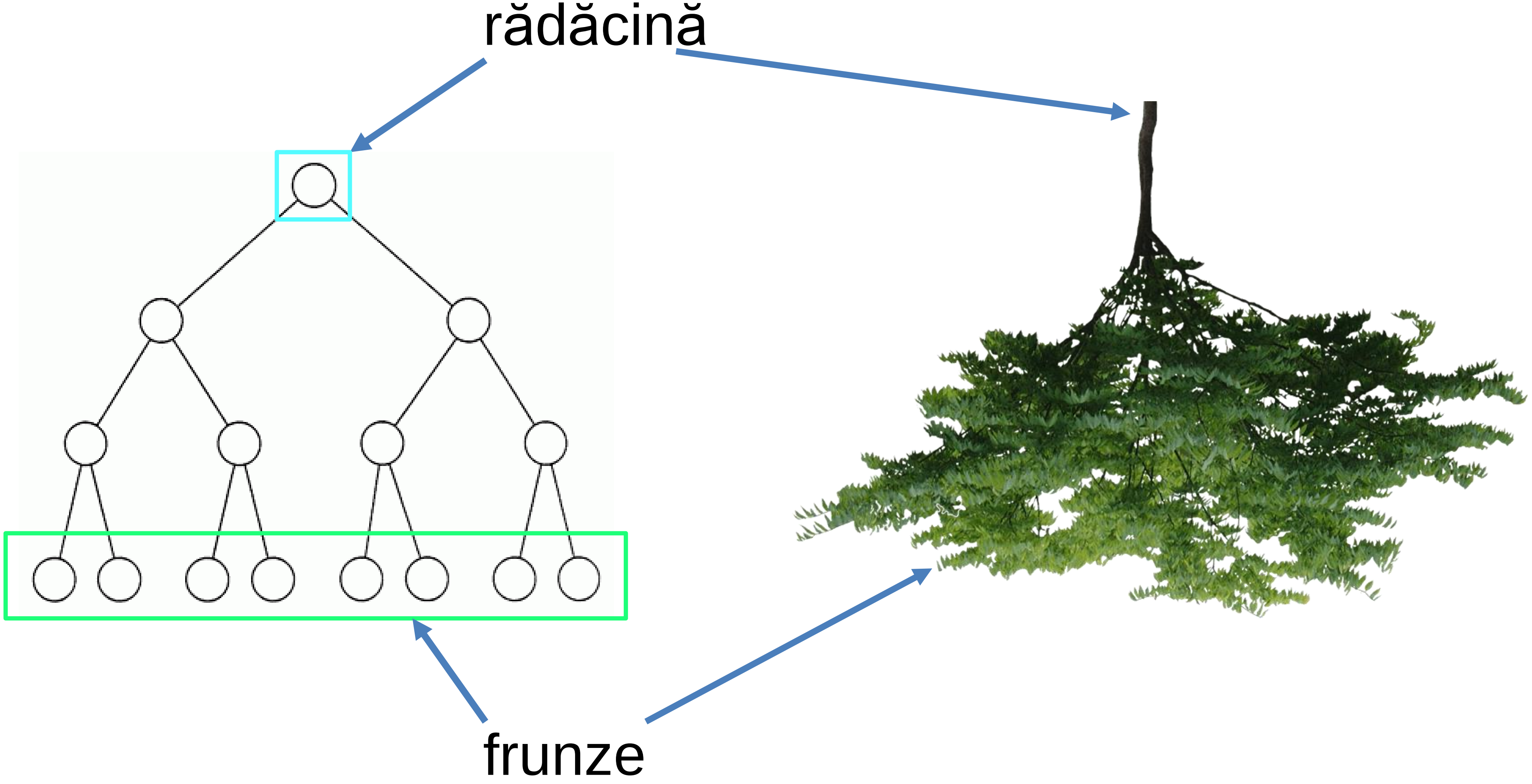
Structuri de date

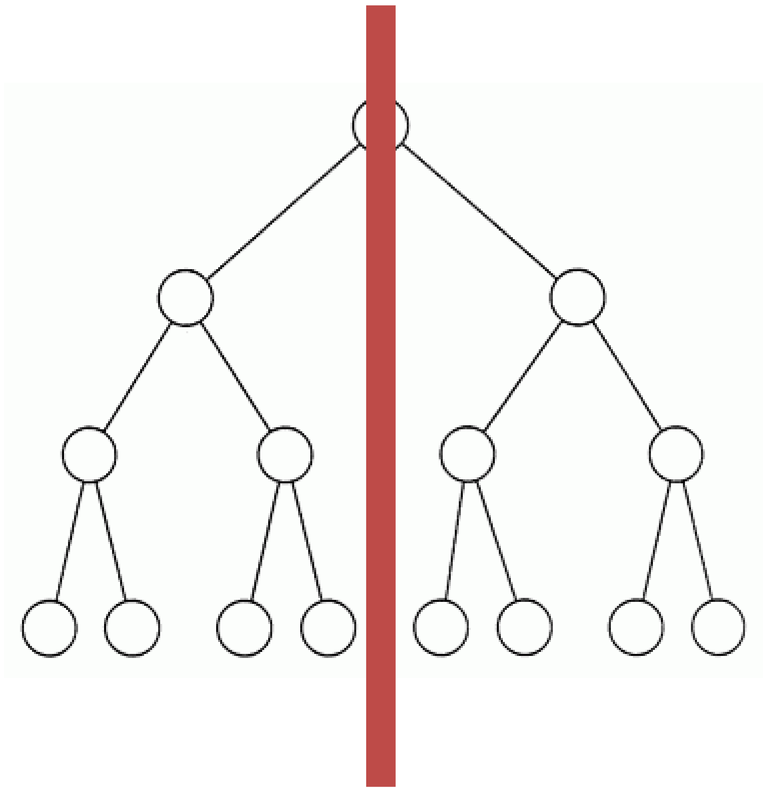
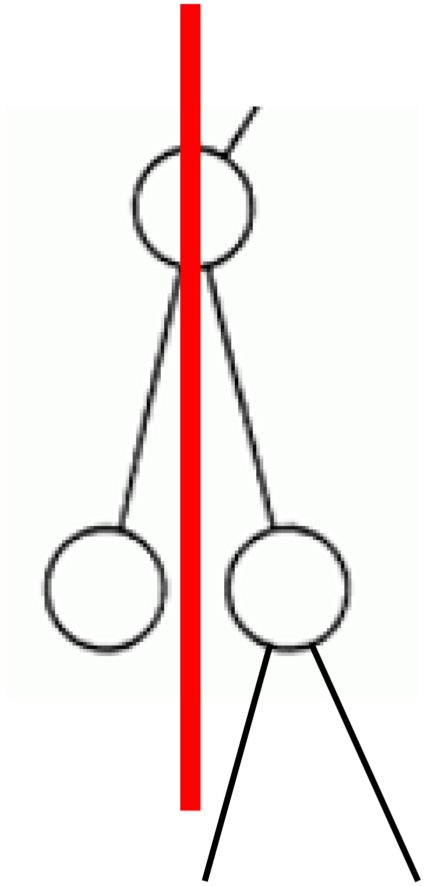
```haskell
data Maybe a = Nothing | Just a

impartire :: Int → Int → Maybe Int
impartire a 0 = Nothing
impartire a b = Just (a `div` b)
```


GO HOME TREES,
YOU'RE DRUNK






rădăcină
frunze

```haskell
data Tree a = Leaf a
data Tree a = Leaf a
            | Node (Tree a) a (Tree a)
```

```haskell
data Tree a = Leaf a
            | Node (Tree a) a (Tree a)

foldTree :: (a → b) → (b → a → b → b) → Tree a → b
```

```haskell
data Tree a = Leaf a
            | Node (Tree a) a (Tree a)

foldTree :: (a → b) → (b → a → b → b) → Tree a → b
foldTree leaf node (Leaf x) = leaf x
```

```haskell
data Tree a = Leaf a
            | Node (Tree a) a (Tree a)

foldTree :: (a → b) → (b → a → b → b) → Tree a → b
foldTree leaf node (Leaf x) = leaf x
foldTree leaf node (Node l x r) = node left x right
     where
          left =
          right =
data Tree a = Leaf a
            | Node (Tree a) a (Tree a)

foldTree :: (a → b) → (b → a → b → b) → Tree a → b
foldTree leaf node (Leaf x) = leaf x
foldTree leaf node (Node l x r) = node left x right
     where
          left = foldTree leaf node l
          right = foldTree leaf node r
```

LIVE DEMO

Mi-e lene!

```
int infinity(){
	return infinity() + 1;
}

int doi(int n){
	return 2;
}
```

Mi-e lene!

```
int infinity(){
      return infinity() + 1;
}

int doi(int n){
      return 2;
}
```

```
doi(infinity()) =
```

Mi-e lene!

```
int infinity(){
	return infinity() + 1;
}

int doi(int n){
	return 2;
}
```

```
doi(infinity()) =
```

Mi-e lene!

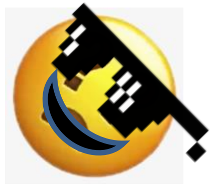
```haskell
infinity :: Integer
infinity = infinity + 1

doi :: Integer → Integer
doi x = 2
```

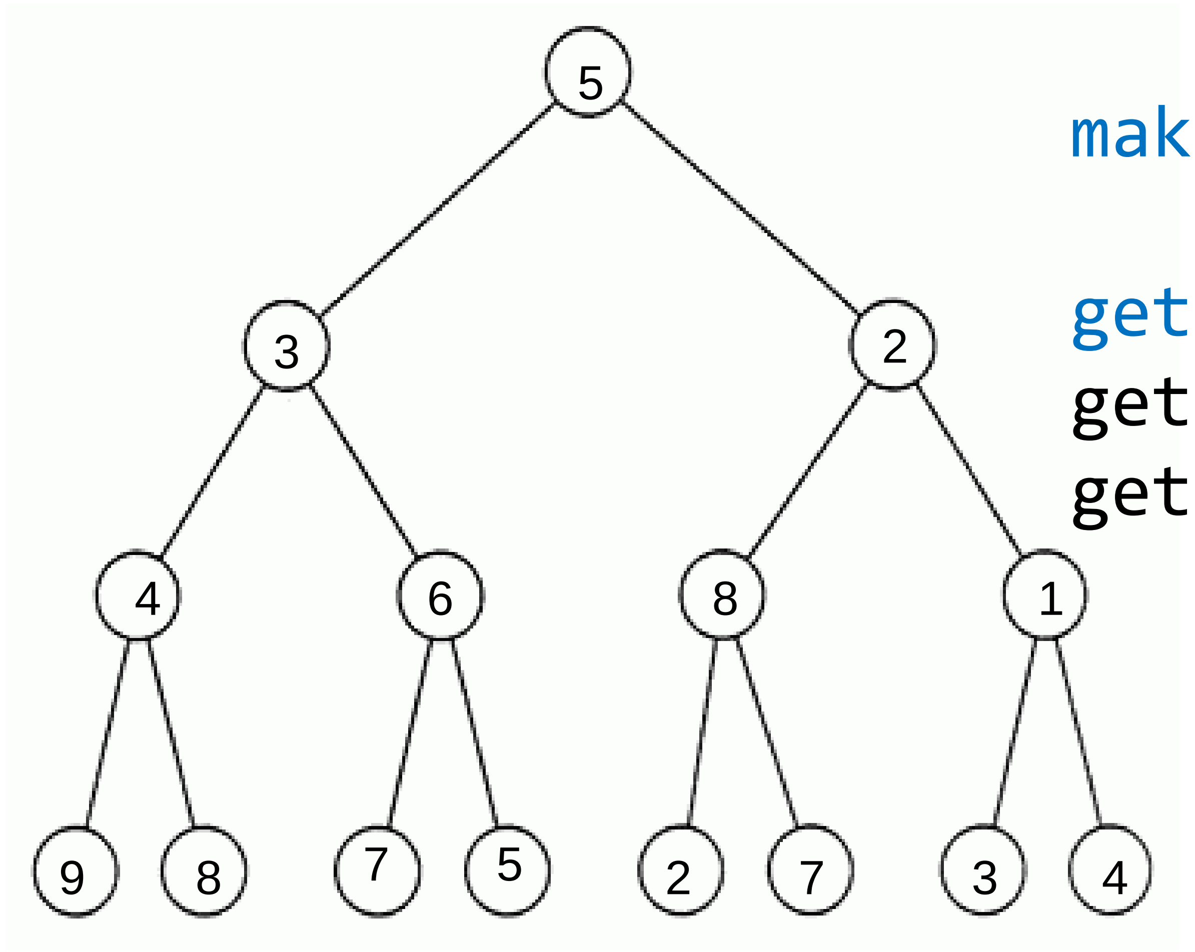
```
doi infinity =
```

Mi-e lene!

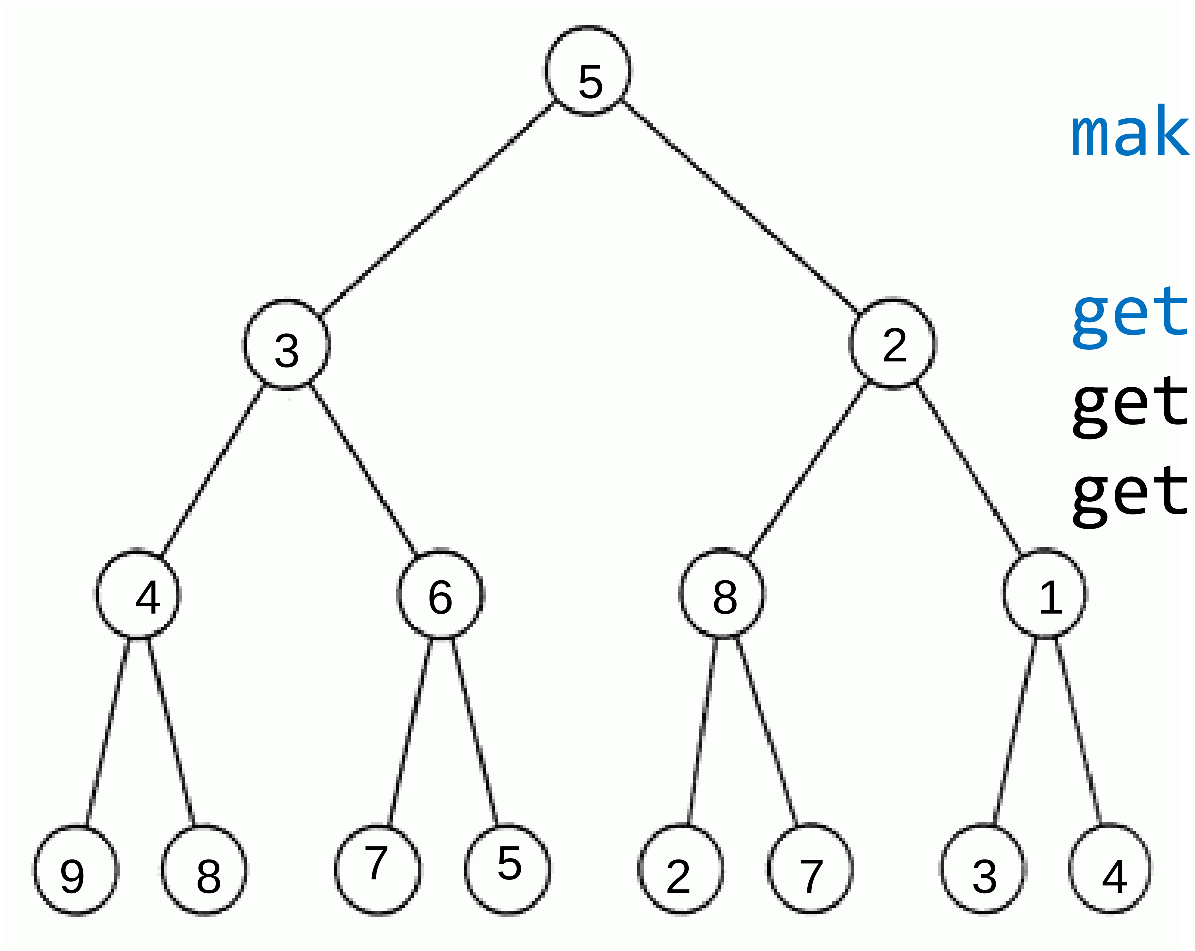
```haskell
infinity :: Integer
infinity = infinity + 1

doi :: Integer → Integer
doi x = 2
```

```haskell
doi infinity = 2
```





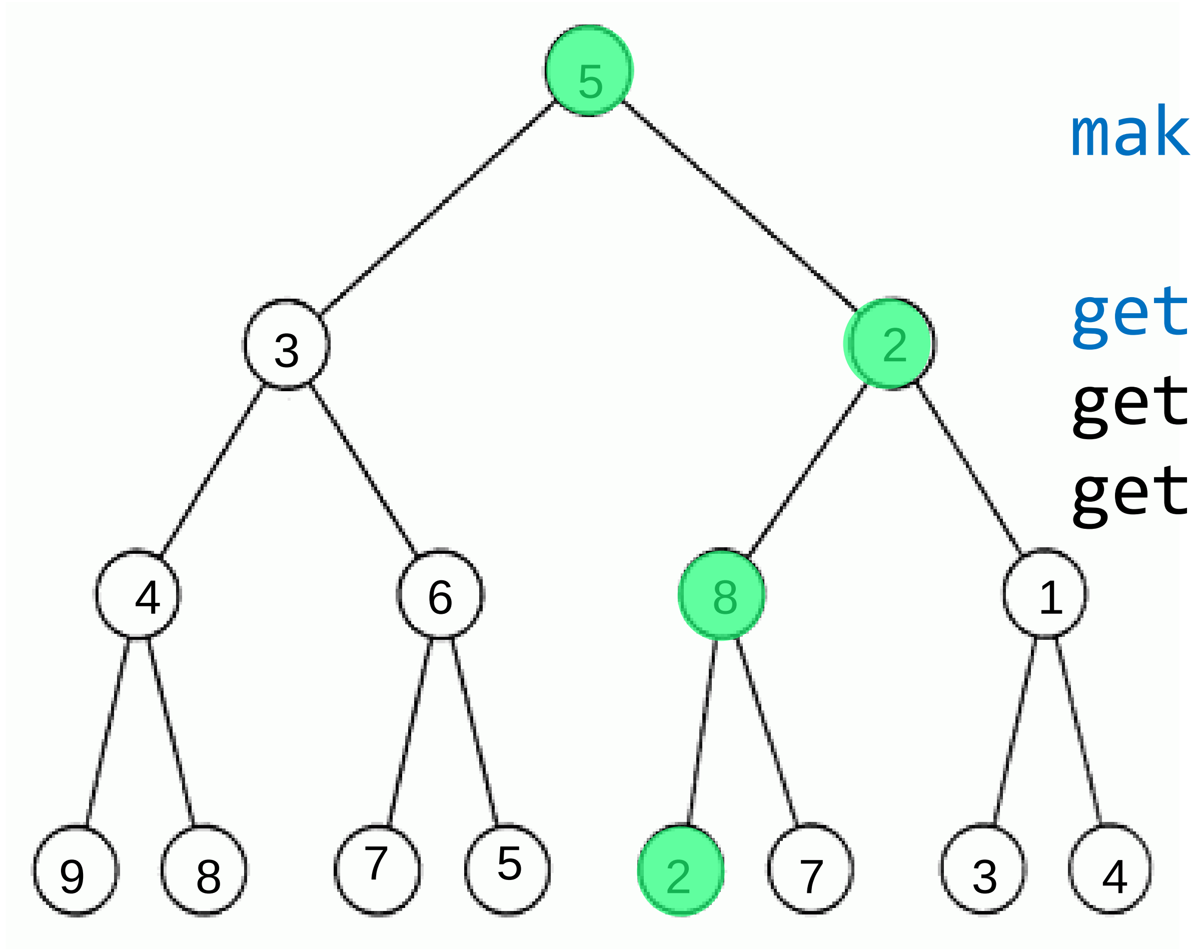
```
makeTree :: [a] → Tree a

get :: Tree a → a
get (Leaf x) = x
get (Node l x r)
    | even x = get l
    | otherwise = get r
```



```
makeTree :: [a] → Tree a

get :: Tree a → a
get (Leaf x) = x
get (Node l x r)
      | even x = get l
      | otherwise = get r
```

```
get makeTree [5, 3, 4, 9, 8, 6, 7, 5, 2, 8, 2, 7, 1, 3, 4]= ?
```

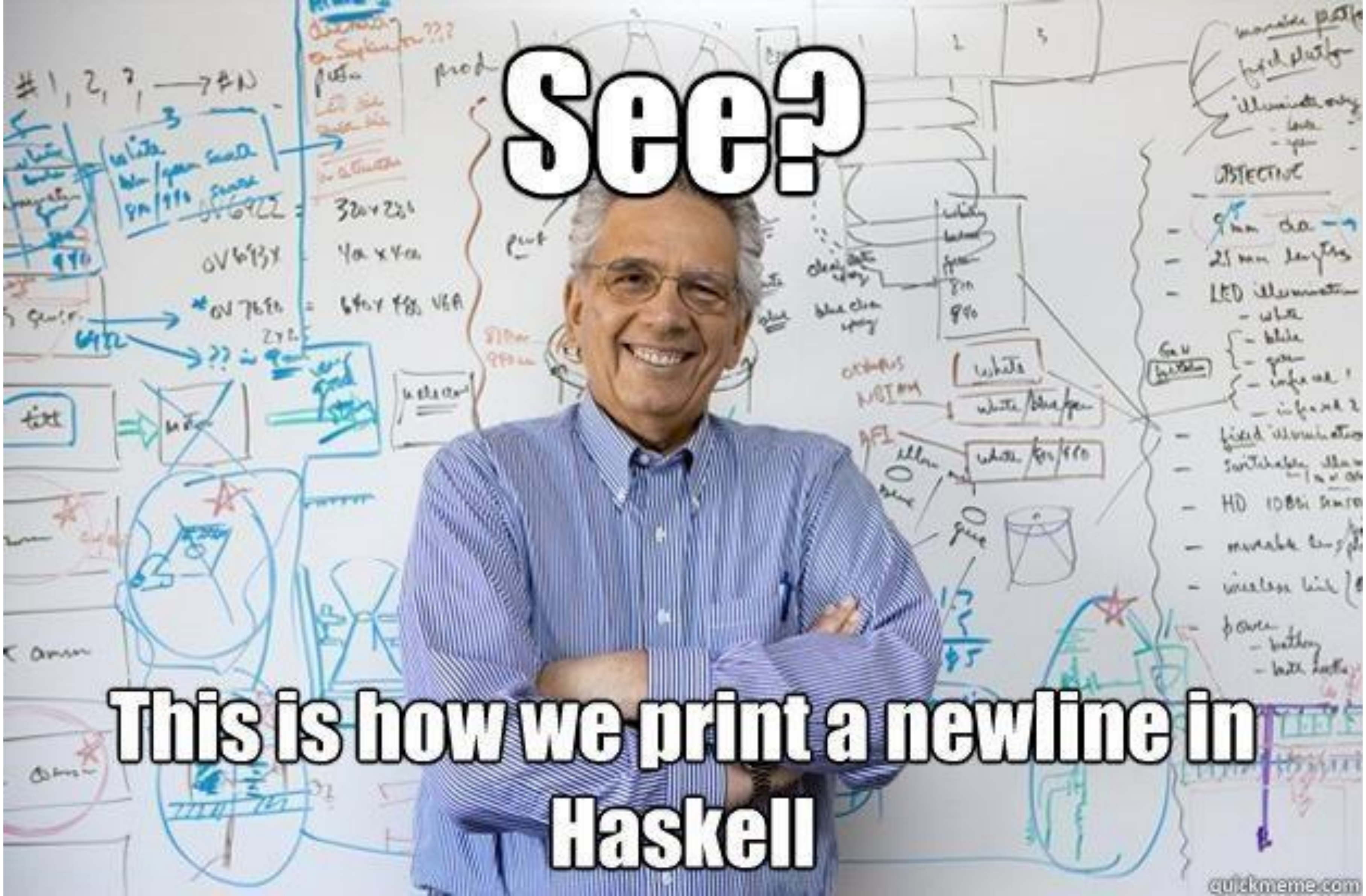
Mi-e lene!



```
makeTree :: [a] → Tree a

get :: Tree a → a
get (Leaf x) = x
get (Node l x r)
    | even x = get l
    | otherwise = get r
```

```
get makeTree [5, 3, 4, 9, 8, 6, 7, 5, 2, 8, 2, 7, 1, 3, 4] = ?
```

*O(Log n)**


See?
This is how we print a newline in Haskell
quickmeme.com

```
main :: IO()
main = do putStrLn “Cum te cheama?”
          nume ← readLn
          putStrLn “Salut, ” ++ nume
```

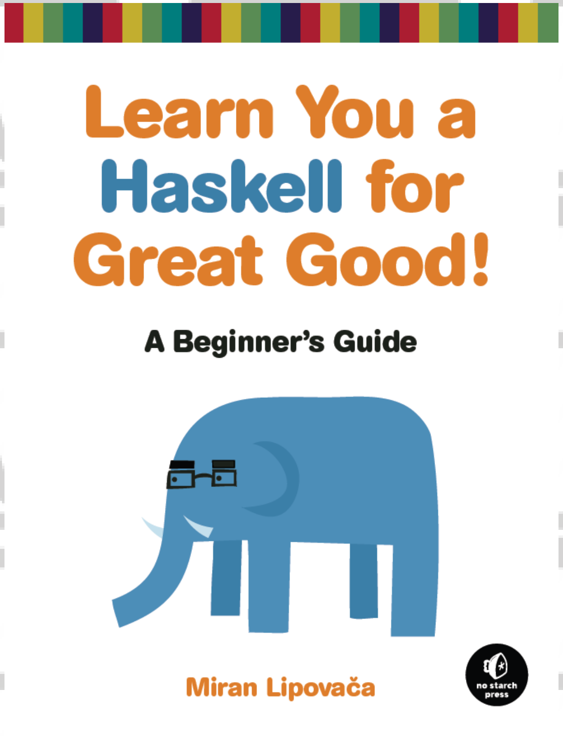
LIVE DEMO

?

Productivitate

- Funcții mult mai ușor de citit
- Debug aproape inexistent
- Cod scurt și clar
- Recursivitate eficientă
- Concurență

stackoverflow



Learn You a Haskell for Great Good!

A Beginner's Guide

Miran Lipovača

no starch press

Hoogλe

https://github.com/emil64/lene

Mulțumesc pentru atenție!

TREASURE HUNT
16 Aprilie
13:30-16:00
Locatie: Parculet Bratianu
Facebook: Societatea Stiintifica Bratianu
Instagram: ssbratianu
L.O.S.T.