Рекурсия

```
1) Целочисленный остаток от деления: rem'
2) Значение целочисленного деления: quot'

let sign a
  | a > 0 = 1
  | a == 0 = 0
  | otherwise = -1


 Наибольший общий делитель: gcd
```

Рекурсия

```
sign' a
  | a > 0 = 1
  | a < 0 = (-1)
  | a == 0 = 0

rem' a b =
  let rem'' a b
      | a < b = a
      | otherwise = rem'' (a-b) b
  in
  sign' a * sign' b * rem'' (abs a) (abs b)
```

Рекурсия

```
sign' a
  | a > 0 = 1
  | a < 0 = (-1)
  | a == 0 = 0

rem' a b =
  let rem'' a b
        | a < b = a
        | otherwise = rem'' (a-b) b
      c = a
      d = b
  in
    sign' c * sign' d * rem'' (abs c) (abs d)
```

Рекурсия

```
sign' a
  | a > 0 = 1
  | a < 0 = (-1)
  | a == 0 = 0

rem' a b =
  let
    rem'' a b
      | a < b = a
      | otherwise = rem'' (a-b) b
    c = a
    d = b
  in
    sign' c * sign' d * rem'' (abs c) (abs d)
```

Рекурсия

```
sign' a
  | a > 0 = 1
  | a < 0 = (-1)
  | a == 0 = 0

rem' a b =
  let
    rem'' a b | a < b = a | otherwise = rem'' (a-b) b; c = a;
d = b
  in
    sign' c * sign' d * rem'' (abs c) (abs d)
```

Рекурсия

```
quot' a b
  | a < b = 0
  | otherwise = 1 + quot' (a-b) b

Ограничения по вкусу
```

Списки

```
isEmpty :: [a] -> Boolean
isEmpty [] = True
isEmpty _ = False

tell :: (Show a) => [a] -> String
tell [] = "Список пуст"
tell (x:[]) = "В списке только один элемент: " ++ show x
tell (x:y:[]) = "Два элемента: " ++ show x ++ " and " ++
show y
tell (x:y:_) = "Много. Первые 2: " ++ show x ++ " and " ++
show y
```

Списки

```
-- scala
-- draw
sum' :: (Num a) => [a] -> a
sum' [] = 0
sum' (x:xs) = x + sum' xs

max' :: (Ord a) => [a] -> a
max' [] = error "У пустого списка нет максимального эл-та!"
max' [x] = x
max' (x:xs) = max x (max' xs)

Типы можно не писать, но это дурной тон

Самостоятельно length, с указанием типа
```

Рекурсия, построение списков

- `replicate' :: (Num i, Ord i) => i -> a -> [a]`
- `take' :: (Num i, Ord i) => i -> [a] -> [a]`
- `reverse' :: [a] -> [a]`
- `repeat' :: a -> [a]`
- `elem' :: (Eq a) => a -> [a] -> Bool`
- `zip' :: [a] -> [b] -> [(a,b)]` — зачем он нужен?
- `append' :: [a] -> [a] -> [a]`

Рекурсия, построение списков

```
replicate' :: (Num i, Ord i) => i -> a -> [a]

replicate' i _ | i < 1 = []

replicate' i a = a : (replicate' (i-1) a)
```

Рекурсия, построение списков

```
take' :: (Num i, Ord i) => i -> [a] -> [a]

take' i list | i < 1 = []

take' i (x:xs) = x : (take' (i-1) xs)
```

Рекурсия, построение списков

```
repeat' :: a -> [a]

repeat' a = a : (repeat a)
```

Рекурсия, построение списков

```
elem' :: (Eq a) => a -> [a] -> Bool

elem' _ [] = False

elem' a (x:xs)
  | a == x = True
  | otherwise = elem' a xs
```

Рекурсия, построение списков

```
zip' :: [a] -> [b] -> [(a,b)]

zip' [] _ = []

zip' _ [] = []

zip' (x:xs) (y:ys) = (x,y) : (zip' xs ys)
```

Рекурсия, построение списков

```
append' :: [a] -> [a] -> [a]

append' [] list = list

append' (x:xs) list = x : (append' xs list)
```

Рекурсия

```
> zip [1..] ["Трус", "Балбес", "Бывалый"]
[(1, "Трус"), (2, "Балбес"), (3, "Бывалый")]

> zip [1..] ['a'..'z']
[(1,'a'),(2,'b'),(3,'c'),(4,'d'),(5,'e'),(6,'f'),(7,'g'),
(8,'h'),(9,'i'),(10,'j'),(11,'k'),(12,'l'),(13,'m'),
(14,'n'),(15,'o'),(16,'p'),(17,'q'),(18,'r'),(19,'s'),
(20,'t'),(21,'u'),(22,'v'),(23,'w'),(24,'x'),(25,'y'),
(26,'z')]

> ones = 1 : ones
```

Рекурсия (КР)

Функция delete :: Char -> String -> String, кото-
рая принимает на вход строку и символ и возвращает
строку, в которой удалены все вхождения символа. При-
мер: delete ’l’ "Hello world!" должно возвращать "Heo
word!".

Функция substitute :: Char -> Char -> String -> String,
которая заменяет в строке указанный символ на заданный.
При-
мер: substitute ’e’ ’i’ "eigenvalue" возвращает
"iiginvalui"

Святой факториал

```
fac 0 = 1
fac n = n * (fac (n-1))
```

Stack overflow

Хвостовая рекурсия

```
fac' a 0 = a
fac' a n = fac' (a*n) (n-1)
fac = fac' 0
```

same as

```
fac n = fac' 0 n
```

Сами

reverse n
fib n

Сами

```
fib' a b 0 = a
fib' a b n = fib' b (a+b) (n-1)
fib n = fib' 1 1 n
```

Как вариант

```
reverse' list =
    let
        reverse'' [] acc = acc
        reverse'' (x:xs) acc = reverse'' xs (x:acc)
    in
        reverse'' list []
```

Задание

Write a recursive function which verifies the balancing of parentheses in a string, which we represent as a List[Char] not a String. For example, the function should return true for the following strings:

(if (zero? x) max (/ 1 x))

I told him (that it’s not (yet) done). (But he wasn’t listening)

The function should return false for the following strings:

:-)

())(

The last example shows that it’s not enough to verify that a string contains the same number of opening and closing parentheses.

balance :: String → Bool

Задание

```
balance' "" 0 = True
balance' "" _ = False
balance' (x:xs) n
  | n < 0 = False
  | x == '(' = balance' xs (n+1)
  | x == ')' = balance' xs (n-1)
  | otherwise = balance' xs n
balance str = balance' str 0
```

}{очу эстетики

```
balance' "" a = a == 0
balance _ n | n < 0 = False
balance' (x:xs) n = balance' xs m
  where m = case x of '(' -> n+1
                      ')' -> n-1
                      _ -> n
balance str = balance' str 0
```

Пред КР

- reverseAll — функция, получающая на вход списочную структуру и обращающая все её элементы, а также её саму.
- firstElem — функция, возвращающая номер первого вхождения заданного атома в список.

КР!

- set — функция, возвращающая список из всех атомов, содержащихся в заданном списке. Каждый атом должен присутствовать в результирующем списке в единственном числе.
- freq — функция, возвращающая список пар (символ, частота). Каждая пара определяет атом из заданного списка и частоту его вхождения в этот список.