ИЗПИТ ПО ФУНКЦИОНАЛНО ПРОГРАМИРАНЕ
КН, 2-ри курс, 1-ви поток (27.01.2021 г.)

Задача 1 (*Racket или Haskell*). Да се дефинира функция `rotate`, която получава цяло число `n` и списък `xs` и „завърта“ `xs` с `n` позиции наляво, т.е. елементите на `xs` се преместват с `n` позиции наляво, като тези, които при преместването излизат извън списъка, се добавят в края му. При подаване на отрицателно число `n`, завъртането е надясно с абсолютната стойност на `n`.
Примери (Haskell):

```
rotate 5    ['a','b','c','d','e','f','g','h'] --> "fghabcde"
rotate 8    ['a','b','c','d','e','f','g','h'] --> "abcdefgh"
rotate 11   ['a','b','c','d','e','f','g','h'] --> "defghabc"
rotate (-2) ['a','b','c','d','e','f','g','h'] --> "ghabcdef"
```

Задача 2 (*Racket*). Да се дефинира функция `rf`, която получава като аргументи едноместните целочислени функции `f` и `g` и връща нова двуаргументна функция с първи аргумент – списък от цели числа `ns` и втори аргумент – едноместна целочислена функция `h`. Върнатата функция трябва да върне списък с всички числа `h(n)`, за които е вярно, че `n` е от `ns` и `f(n) > g(n)`.
Пример:

```
((rf (lambda (x) (* 2 x)) (lambda (x) (+ 2 x)))
 '(1 2 3 4 5) (lambda (x) (* 3 x))) --> '(9 12 15)
```

Задача 3 (*Racket или Haskell*). Ако `A(x1,y1)` и `B(x2,y2)` са две точки в декартовата равнина – такива, че **`x1 ≠ x2`**, то уравнението на правата `AB`, която минава през тези две точки, е `y = f(x)`, където `f(x) = y1 + (x-x1)*(y2-y1)/(x2-x1)`.
а) Нека приемем, че точките в декартовата равнина се представят като точкови двойки (Racket) или като стойности от тип `Point = (Double,Double)` (Haskell). Да се дефинира функция `line`, която по две точки връща функцията, определяща уравнението на минаващата през тях права.
б) Да се дефинира функция от по-висок ред `liesOn`, която за дадена функция `f`, определяща уравнението на права, връща като резултат функция, която по дадена точка `P(x,y)` проверява дали точката `P` лежи на правата `f` (дали `y = f(x)`).
Примери (Haskell):

```
diagonal = line (0,0) (1,1)
onDiag = liesOn diagonal
```

```
diagonal 5.5 --> 5.5
diagonal 0.5 --> 0.5

onDiag (5.5,5.5) --> True
onDiag (0.5,0)   --> False
```

Задача 4 (*Haskell*). Да се дефинира функция `deepestLeavesSum :: BTree -> Int`, която връща сумата на най-дълбоките (най-отдалечените от корена) възли в дадено двоично дърво.

Примери:

```
t1 :: BTree                                          --        1
t1 = Node 1 (Node 2 (Node 4 (Node 7 Empty Empty)     --       / \
                            Empty)                   --      2   3
                    (Node 5 Empty                    --     / \   \
                            Empty))                  --    4   5   6
            (Node 3 Empty                            --   /         \
                    (Node 6 Empty                    -- 7           8
                            (Node 8 Empty Empty)))

                                                     --        1
t2 :: BTree                                          --       / \
t2 = Node 1 (Node 2 (Node 4 Empty Empty)             --      2   3
                      Empty)                         --     /
             (Node 3 Empty Empty)                    --    4

deepestLeavesSum t1 --> 15 (7 + 8)
deepestLeavesSum t2 --> 4
```