X

(https://swayam.gov.in)

(https://swayam.gov.in/nc_details/NPTEL)

vivekdubey74rr@gmail.com ⌄

NPTEL (https://swayam.gov.in/explorer?ncCode=NPTEL) » Introduction To Haskell Programming (course)

☰

If already registered, click to check your payment status

Week 4: Programming Assignment

Due on 2023-08-24, 23:59 IST

Course outline

How does an NPTEL online course work? ()

Week 1: Introduction ()

Week 2: Lists, Strings, Tuples ()

Week 3: Rewriting, Polymorphism, Higher Order Functions on Lists ()

Week 4: Efficiency, Sorting, Infinite lists, Conditional polymorphism, Using ghci ()

○ Measuring efficiency (unit?unit=38&lesson=39)

○ Sorting (unit?unit=38&lesson=40)

○ Using infinite lists (unit?unit=38&lesson=41)

○ Conditional polymorphism (unit?unit=38&lesson=42)

○ Defining functions in ghci (unit?unit=38&lesson=43)

○ Week 4 Feedback Form: Introduction To Haskell Programming (unit?unit=38&lesson=44)

● **Week 4: Programming Assignment (/noc23_cs94/progassignment?name=98)**

Week 5: User-defined datatypes, abstract

datatypes, modules ()

Week 6: recursive data types, search trees ()

Week 7: arrays, IO ()

DOWNLOAD VIDEOS ()

Text Transcripts ()

Books ()

Problem Solving Session - July 2023 ()

1. Define a function f1 :: [Int] -> [Int] which takes a list l of nonnegative numbers as input, and replaces each n in l by 3*n if n is a power of 3, and by 0 if it is not a power of 3.

Examples:

```
  f1 [] = []
  f1 [1] = [3]
  f1 [1, 2, 3] = [3, 0, 9]
  f1 [0, 2, 4, 6] = [0, 0, 0, 0]
```

2. For a list l, define S(l) to be the set of all indices i of l (remember that indices start from 0) such that l!!i > l!!(i+1). Define a function
f2 :: [Int] -> [Int] which takes a nonempty list l of
integers as input and outputs a S(l) in order.

Examples:

```
  f2 [] = []
  f2 [1] = []
  f2 [1, 2, 3, 2, 1] = [2, 3]
  f2 [1, 2, 3, 4, 5, 6] = []
```

3. Define a function f3 :: [Int] -> [Int] that removes adjacent duplicates.
i.e. if the same element occurs n times contiguously, we retain only one copy.

Examples:

```
  f3 [1, 1, 1, 2, 2, 3, 3, 3, 3] = [1, 2, 3]
  f3 [1, 2, 1, 2, 3, 1, 1, 2, 2] = [1, 2, 1, 2, 3, 1, 2]
```

4. Define a function f4 :: [Int] -> [[Int]] that partitions the list into all its upruns. An uprun is a maximal non-decreasing segment of the given list.

Examples:

```
f4 [] = []
f4 [5] = [[5]]
```

f4 [1, 2, 3, 4, 5] = [[1,2,3,4,5]]
f4 [1, 2, 3, 4, 5, 6, 5, 4, 3, 2, 1] = [[1,2,3,4,5,6],[5],[4],[3],[2],[1]]

Private Test cases used for evaluation	Input	Expected Output	Actual Output	Status
Test Case 1	`f1 [1, 2, 3]`	`[3,0,9]`	`[3,0,9]\n`	Passed
Test Case 2	`f1 [0, 2, 4, 6]`	`[0,0,0,0]`	`[0,0,0,0]\n`	Passed
Test Case 3	`f1 [21, 27, 22, 9]`	`[0,81,0,27]`	`[0,81,0,27]\n`	Passed
Test Case 4	`f1 [3, 3, 3, 3]`	`[9,9,9,9]`	`[9,9,9,9]\n`	Passed
Test Case 5	`f1 [3, 6, 9, 12]`	`[9,0,27,0]`	`[9,0,27,0]\n`	Passed
Test Case 6	`f1 [1, 2, 2, 3, 4, 5, 8, 8, 9, 9]`	`[3,0,0,9,0,0,0,0,27,27]`	`[3,0,0,9,0,0,0,0,27,27]\n`	Passed
Test Case 7	`f1 [1, 3, 9, 27, 81]`	`[3,9,27,81,243]`	`[3,9,27,81,243]\n`	Passed
Test Case 8	`f2 []`	`[]`	`[]\n`	Passed
Test Case 9	`f2 [1]`	`[]`	`[]\n`	Passed
Test Case 10	`f2 [1, 2, 3, 2, 1]`	`[2,3]`	`[2,3]\n`	Passed
Test Case 11	`f2 [1, 2, 3, 4, 5, 6]`	`[]`	`[]\n`	Passed
Test Case 12	`f2 [6, 5, 4, 3, 2, 1]`	`[0,1,2,3,4]`	`[0,1,2,3,4]\n`	Passed

Test Case 13	`f2 [19, 29, 28, 38, 45]`	`[1]`	`[1]\n`	Passed
Test Case 14	`f2 [1, 1, 1, 2, 2, 3, 3, 3, 4, 4, 5, 5, 5]`	`[]`	`[]\n`	Passed
Test Case 15	`f2 [2, 1, 3, 1, 4, 1, 5, 1, 6, 1]`	`[0,2,4,6,8]`	`[0,2,4,6,8]\n`	Passed
Test Case 16	`f2 [5, 4, 1, 2, 3, 4, 3, 4, 1, 2, 0]`	`[0,1,5,7,9]`	`[0,1,5,7,9]\n`	Passed
Test Case 17	`f3 [1, 1, 1, 2, 2, 3, 3, 3, 3]`	`[1,2,3]`	`[1,2,3]\n`	Passed
Test Case 18	`f3 [1, 2, 1, 2, 3, 1, 1, 2, 2]`	`[1,2,1,2,3,1,2]`	`[1,2,1,2,3,1,2]\n`	Passed
Test Case 19	`f3 [1, 2, 2, 1, 3, 3, 4, 1, 2, 2]`	`[1,2,1,3,4,1,2]`	`[1,2,1,3,4,1,2]\n`	Passed
Test Case 20	`f3 [1,2,3,4,5,6,7,8,9,10]`	`[1,2,3,4,5,6,7,8,9,10]`	`[1,2,3,4,5,6,7,8,9,10]\n`	Passed
Test Case 21	`f3 [1, 1, 1, 2, 2, 3, 3, 3, 3, 4, 4, 5, 5, 5, 5, 5]`	`[1,2,3,4,5]`	`[1,2,3,4,5]\n`	Passed
Test Case 22	`f3 [5, 4, 1, 2, 3, 4, 3, 4, 1, 2, 0]`	`[5,4,1,2,3,4,3,4,1,2,0]`	`[5,4,1,2,3,4,3,4,1,2,0]\n`	Passed
Test Case 23	`f3 [1, 2, 3, 2, 1]`	`[1,2,3,2,1]`	`[1,2,3,2,1]\n`	sed

Test Case 24	`f4 []`	`[]`	`[]\n`	Passed
Test Case 25	`f4 [5]`	`[[5]]`	`[[5]]\n`	Passed
Test Case 26	`f4 [1,2,3,4,5]`	`[[1,2,3,4,5]]`	`[[1,2,3,4,5]]\n`	Passed
Test Case 27	`f4 [5,4,3,2,1]`	`[[5],[4],[3],[2],[1]]`	`[[5],[4],[3],[2],[1]]\n`	Passed
Test Case 28	`f4 [5, 4, 1, 2, 3, 4, 3, 4, 1, 2, 0]`	`[[5],[4],[1,2,3,4],[3,4],[1,2],[0]]`	`[[5],[4],[1,2,3,4,3,4,1,2],[0]]\n`	Wrong Answer
Test Case 29	`f4 [1, 1, 1, 2, 2, 3, 3, 3, 3, 4, 4, 5, 5, 5, 5, 5]`	`[[1,1,1,2,2,3,3,3,3,4,4,5,5,5,5,5]]`	`[[1,1,1,2,2,3,3,3,3,4,4,5,5,5,5,5]]\n`	Passed
Test Case 30	`f4 [1, 2, 3, 4, 5, 6, 5, 4, 3, 2, 1]`	`[[1,2,3,4,5,6],[5],[4],[3],[2],[1]]`	`[[1,2,3,4,5,6,5,4,3,2,1]]\n`	Wrong Answer

The due date for submitting this assignment has passed.

28 out of 30 tests passed.

You scored 93.33333333333333/100.

Assignment submitted on 2023-08-23, 22:06 IST

Your last recorded submission was :

```
-- 1. Replace powers of 3 with 3*n and non-powers of 3 with 0
f1 :: [Int] -> [Int]
f1 [] = []
f1 (x:xs)
    | isPowerOfThree x = 3 * x : f1 xs
    | otherwise = 0 : f1 xs
  where
    isPowerOfThree n = n > 0 && (3 ^ round (logBase 3 (fromIntegral n))) == n

-- 2. Find indices where l!!i > l!!(i+1)
f2 :: [Int] -> [Int]
f2 [] = []
f2 l = [i | (i, x) <- zip [0..] l, i < length l - 1, x > l !! (i + 1)]
```

```haskell

-- 3. Remove adjacent duplicates
f3 :: [Int] -> [Int]
f3 [] = []
f3 [x] = [x]
f3 (x:y:xs)
  | x == y = f3 (y:xs)
  | otherwise = x : f3 (y:xs)

-- 4. Partition into upruns
f4 :: [Int] -> [[Int]]
f4 [] = []
f4 [x] = [[x]]
f4 (x:xs) = (x : takeWhile (>= x) xs) : f4 (dropWhile (>= x) xs)

main = do
    line <- getLine;
    let (func, rest) = break (== ' ') line in
        case func of
            "f1" -> let args = read rest :: [Int] in
                putStrLn . show $ f1 args
            "f2" -> let args = read rest :: [Int] in
                putStrLn . show $ f2 args
            "f3" -> let args = read rest :: [Int] in
                putStrLn . show $ f3 args
            "f4" -> let args = read rest :: [Int] in
                putStrLn . show $ f4 args
```