Types

What is the most general type of the function

`tail`

?

Types

What is the most general type of the function

`take`

?

Types

What is the most general type of the expression

```
take 3 [True,True,True,True]
```

?

Types

What is the most general type of the expression

```
take 3
```

?

Types

What is the type of the expression

```
(False, 'a')
```

?

Types

What is the type of the expression

```
[(False, '0'), (True, '1')]
```

?

Types

What is the type of the expression

```
(['a', 'b'], [False, True])
```

?

Types

What is the most general type of the function twice defined by

```
twice f x = f (f x)
```

?

Types

What is the most general type of the function add defined by

```
add x y = x * y
```

?

Types

What is the most general type of

```
evens = filter even
```

?

Evaluation

What does the expression

```
foldr (+) 1 [2,3,4]
```

evaluate to?

Evaluation

What does the expression

```
foldr (*) 1 [2,3,4]
```

evaluate to?

Evaluation

What does the expression

```
foldr (-) 1 [2,3,4]
```

evaluate to?

Evaluation

What does the expression

```
foldl (-) 1 [2,3,4]
```

evaluate to?

Evaluation

What does the expression

```
sum [x | x <- [1..10], even x]
```

evaluate to?

Evaluation

What does the expression

```
zip [1,2] ['a','b','c']
```

evaluate to?

Evaluation

What does the expression

```
takeWhile even [2,4,6,7,8]
```

evaluate to?

Defining functions recursively

Define a recursive function

```
insert :: Int -> [Int] -> [Int]
```

that inserts an integer into the correct position in a sorted list of integers.

Defining functions recursively

Define the following library function using recursion:

```
and :: [Bool] -> Bool
```

Defining functions recursively

Define the following library function using recursion:

```
reverse :: [a] -> [a]
```

Defining functions recursively

Define the following library function using recursion:

```
replicate :: Int -> a -> [a]
```

List comprehensions

Express the sum of the squares of the integers between 1 and 100 using a list comprehension.

List comprehensions

Express the sum of the products of all pairs of integers between 1 and 100 using a list comprehension.

Laws

Given the definitions

```
(f . g) x = f (g x)
map f [ ] = [ ]
map f (x:xs) = f x : map f xs
```

prove the following property of map using induction (for finite lists):

```
map f (map g xs) = map (f . g) xs
```