

# Common Functions



```
1 {-# LANGUAGE FlexibleContexts #-}
2 {-# LANGUAGE MultiParamTypeClasses #-}
3 {-# LANGUAGE OverloadedStrings #-}
4 {-# LANGUAGE QuasiQuotes #-}
5 module Handler.Home where
6
7 import Control.Arrow
8 import Control.Monad
9 import Data.Default
10 import Data.Function
11 import Data.List
12 import Data.Ord
13 import Data.Time
14 import Data.Time.Calendar.WeekDate
15 import Import
16 import Network.HTTP.Types
17 import Yesod.Auth
18
19 getHomeR :: Handler Html
20 getHomeR = undefined
21
22 postHomeR :: Handler Html
23 postHomeR = do
24   ((result, formWidget), formEnctype, formPost)
25   <- case result of
26     FormSuccess userRule -> do
27       maybeUserId <- maybeAuthId
28       case maybeUserId of
29         Nothing -> do
30           setErrorMessage [shamlet|You have to be logged in to do that!|]
31           redirect % AuthR.LoginR
32         Just userId -> do
```

# head, last



- Signature:

```
head :: [a] -> a
last  :: [a] -> a
```

- Description:

- `head xs` is the first element of the list `xs`.
- `last xs` is the last element of the list `xs`.

Error if `xs` is empty.

- Examples:

```
λ> head [1..6]
↵ 1
λ> last [1..6]
↵ 6
```

# tail, init



- Signature:

```
tail :: [a] -> [a]
init :: [a] -> [a]
```

- Description

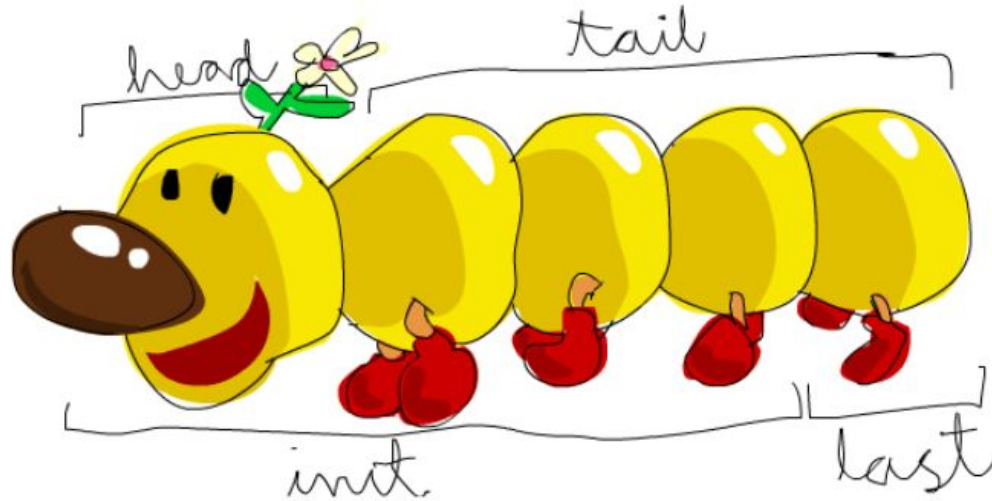
- `tail xs` is the list `xs` without its first element.
- `init xs` is the list `xs` without its last element.

Error if `xs` is empty.

- Examples:

```
λ> tail [1..4]
↳ [2, 3, 4]
λ> init [1..4]
↳ [1, 2, 3]
```

# head, last, init, tail



# reverse



- Signature:

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

- Description:

`reverse xs` is the list `xs` backwards.

- Examples:

```
λ> reverse [1..4]  
↵ [4, 3, 2, 1]
```

# length



- Signature:

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

- Description:

`length xs` is the number of elements in the list `xs`.

- Examples:

```
λ> length []  
↵ 0  
λ> length [1..5]  
↵ 5  
λ> length "Angela"  
↵ 6
```

# null



- Signature:

```
null :: [a] -> Bool
```

- Description:

`null xs` indicates if the list `xs` is empty.

- Examples:

```
λ> null []  
↵ True  
λ> null [1..8]  
↵ False
```

# elem



- Signature:

```
elem :: Eq a => a -> [a] -> Bool
```

- Description:

`elem x xs` indicates if `x` is in the list `xs`.

- Examples:

```
λ> elem 6 [1..10]
☞ True
λ> 6 `elem` [1..10]
☞ True
λ> 'k' `elem` "Ethan"
☞ False
```



# Indexing (!!)



- Signature:

```
(!!) :: [a] -> Int -> a
```

- Description:

`xs !! i` is the `i`th element of the list `xs` (starting from zero).

- Examples:

```
λ> [1..10] !! 3  
4  
λ> [1..10] !! 11  
✗ Exception: index too large
```

# Concatenation of two lists



- Signature:

```
(++) :: [a] -> [a] -> [a]
```

- Description:

`xs ++ ys` is the resulting list of putting `ys` after `xs`.

- Examples:

```
λ> "JIM" ++ "MY"  
↵ "JIMMY"  
λ> [1..5] ++ [1..3]  
↵ [1,2,3,4,5,1,2,3]
```

# maximum, minimum



- Signature:

```
maximum :: Ord a => [a] -> a  
minimum :: Ord a => [a] -> a
```

- Description:

- `maximum xs` is the biggest element of the list (non empty!) `xs`.
- `minimum xs` is the smallest element of the list (non empty!) `xs`.

- Examples:

```
λ> maximum [1..10]  
↵ 10  
λ> minimum [1..10]  
↵ 1  
λ> minimum []  
✗ Exception: empty list
```

# sum, product



- Signature:

```
sum      :: Num a => [a] -> a
product  :: Num a => [a] -> a
```

- Description:

- `sum xs` is the sum of the list `xs`.
- `prod xs` is the product of the list `xs`.

- Examples:

```
λ> sum [1..5]
↵ 15

factorial n = product [1 .. n]

λ> factorial 5
↵ 120
```

# take, drop



- Signature:

```
take :: Int -> [a] -> [a]
drop :: Int -> [a] -> [a]
```

- Description:

- `take n xs` is the prefix of length `n` of the list `xs`.
- `drop n xs` is the suffix of the list `xs` when the first `n` elements are removed.

- Examples:

```
λ> take 3 [1 .. 7]
☞ [1, 2, 3]
λ> drop 3 [1 .. 7]
☞ [4, 5, 6, 7]
```

# zip



- Signature:

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

- Description:

`zip xs ys` is the list that combines, in order, each pair of elements of `xs` and `ys`. If they are missing, they are lost.

- Examples:

```
λ> zip [1, 2, 3] ['a', 'b', 'c']  
↳ [(1, 'a'), (2, 'b'), (3, 'c')]  
λ> zip [1 .. 10] [1 .. 3]  
↳ [(1, 1), (2, 2), (3, 3)]
```

# repeat



- Signature:

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

- Description:

`repeat x` is the infinite list where all elements are `x`.

- Examples:

```
λ> repeat 3
☞ [3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, ...]
λ> take 4 (repeat 3)
☞ [3, 3, 3, 3]
```

# concat



- Signature:

```
concat :: [[a]] -> [a]
```

- Description:

`concat xs` is the list that concatenates all the lists of `xs`.

- Examples:

```
λ> concat [[1, 2, 3], [], [3], [1, 2]]  
↵ [1, 2, 3, 3, 1, 2]
```



# Instructor Youtube Channel: Lucas Science

