

# Functional Programming



## Lists

<http://www.digitalsart.com/stockphoto/d15792>

# Learning Targets

You

- understand the structure of lists in Haskell
- know how to create a list and how to access its elements
- can use the available functions which are defined in the prelude
- know how to use hoogle to search for functions

# Content

- **Definition**
- **Lists as Models**
- **Accessing List Elements**
- **The Structure of Lists**
- **Pattern Matching on Lists**
- **Important List Functions**
- **Higher Order List Functions Preview**



# Lists

- A list is a **sequence of elements of the same type**
- **List types**
  - If **T** is a Type then **[ T ]** is the type of a list with elements of type **T**
  - The number of elements contained in a list is not represented by the type
- **List values**
  - Syntax: Enclosed in square brackets and separated by commas

```
[1, 2, 3, 4]           :: [Int]
[True, True, False, True] :: [Bool]
["Milk", "Bread", "Flakes"] :: [String]
[sum, product]         :: [ [Int] -> Int ]
```

## Lists

- **sequence** of **elements** of the **same** type
- Type:  $[T]$  one type for all elements
- Length: The **number of elements** in a
- Lists of the **same type** can have **different lengths**

## Tuples

- **finite sequence** of components of **possibly different** type
- Type:  $(T_1, T_2, \dots, T_n)$  defining the type at each position in the tuple
- Arity: The **number of components** in a tuple
- All tuples of the **same type** have the **same arity**

# Worksheet: Lists as Models

Worksheet: WS\_Lists\_as\_Models.lhs

## Key learnings

- Lists can be utilized to model many different kind of real world data
- When the number of elements is not statically determined, you need a list

## Accessing List Elements

- A list of type `[ T ]` consists of a head element of type `T` and a tail of type `[ T ]`

`[ 'a', 'b', 'c' ] :: [Char]`

**head**      **tail**

- Access the first element of the list

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

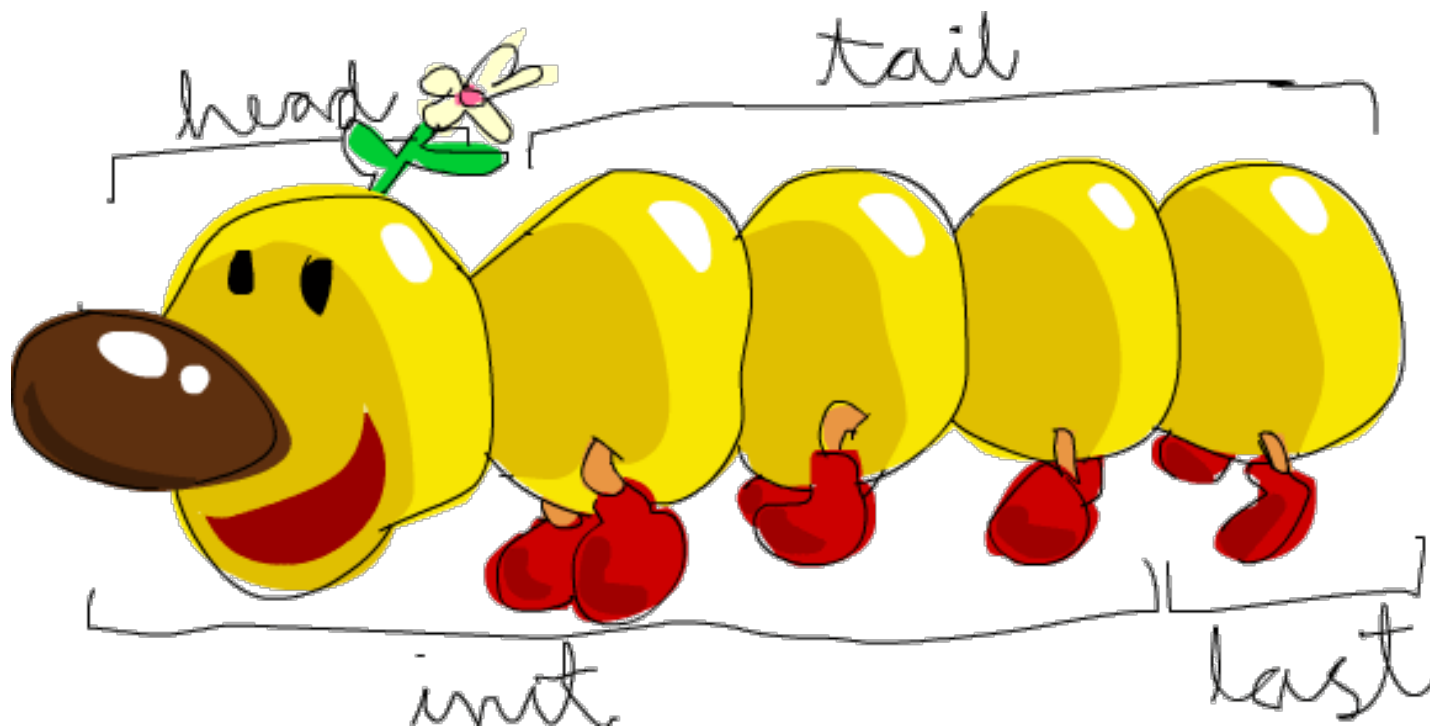
```
head ['a', 'b', 'c'] ~> 'a'
```

- Access the rest of the list

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

```
tail ['a', 'b', 'c'] ~> ['b', 'c']
```

## Structure of a list I

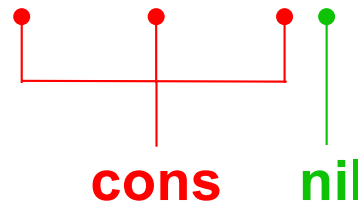


<http://learnyouahaskell.com/starting-out#an-intro-to-lists>



## Constructing Lists

- `['a', 'b', 'c']` translates to `'a' :: ('b' :: ('c' :: []))`



- The empty list is called **nil** and is written as `[]`

`[] :: [a]`

- **cons** `(:)` takes an element and a list and returns a new list with the element as its new head

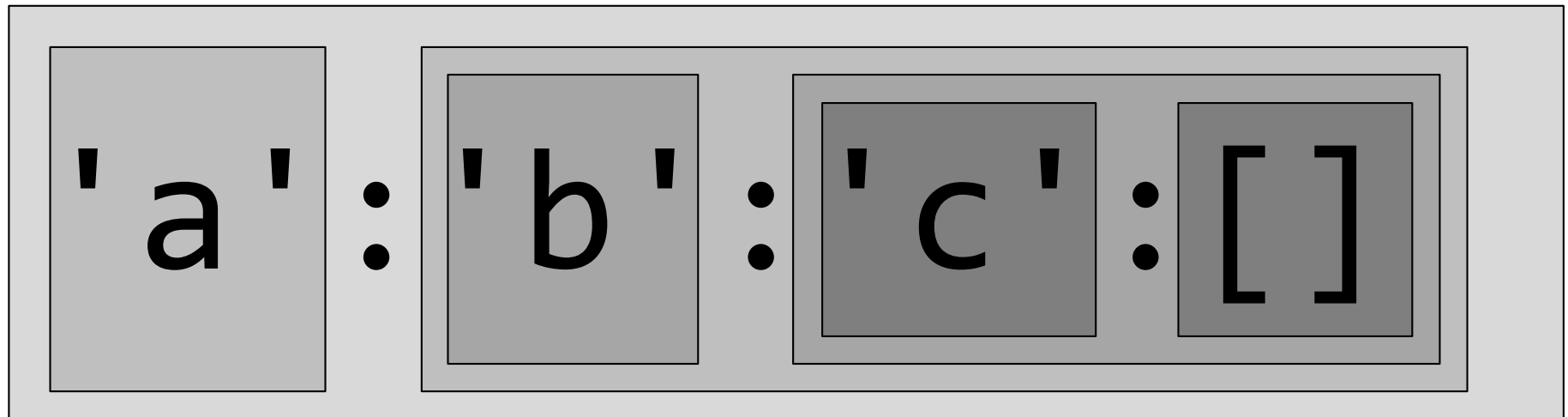
`(:) :: a -> [a] -> [a]`

`'a' : ['b', 'c'] ~> ['a', 'b', 'c']`

- **cons** associates to the right

`'a' : 'b' : 'c' : []` is interpreted as `('a' : ('b' : ('c' : [])))`

## Structure of a list II



- At the end of a finite list is always the empty list [].

## Head in Detail

Input: List of elements of type a

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

Output: First element of input which is of type a

- **head on non-empty list**

```
head [1,2,3] ~> 1
```

- **head on the empty list []**

```
head [] ~> *** Exception: Prelude.head: empty list
```

## Tail in Detail

Input: List of elements of type a

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

Output: All except the first element

- **tail on non-empty list**

```
tail [1,2,3] ~> [2,3]
```

- **tail on the empty list []**

```
tail [] ~> *** Exception: Prelude.tail: empty list
```

## Length of a List

- Check whether the list is empty or not

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

```
null ['a', 'b', 'c'] ~> False  
null []             ~> True
```

- Get the number of elements in the list

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

```
length ['a', 'b', 'c'] ~> 3  
length []             ~> 0
```

- Before accessing elements of list always check that it is
  - not null
  - length > 0**to avoid runtime errors**

# String is a List of Characters

- In Haskell the type `String` is simply an alias for `[Char]`

```
type String = [Char]
```

- Which means that the following expressions are all equal

```
"abc" == ['a', 'b', 'c'] == 'a':'b':'c':[]
```

- As a consequence: Strings can be processed like any other list

```
head "hello"    ~> 'h'  
tail "hello"    ~> "ello"  
length "hello" ~> 5
```

# Pattern Matching Lists

- **Matching type constructors**
  - One case for the empty list
  - One case for a list with a head element

```
stdMatch :: Show a => [a] -> String
stdMatch []      = "Matched empty list"
stdMatch (x:xs) = "Matched list with head " ++ show x
```

- **The following two functions are equivalent**

```
m1 :: Show a => [a] -> String
m1 [x]          = "Matched list with one element" ++ show x
m1 [x,y]        = "Matched List with two elements"
```

```
m1 :: Show a => [a] -> String
m1 (x:[])       = "Matched list with one element" ++ show x
m1 (x:y:[])     = "Matched List with two elements"
```

# List Functions WS

The screenshot shows a web browser window with the URL `www.haskell.org/hoogle/?hoo...`. The page title is "Hoogle" and it includes a search bar with the text `[e] -> [e] -> [e]`. Below the search bar, the results for the type signature `[e] -> [e] -> [e]` are displayed. The results include the type signature `(++) :: [a] -> [a] -> [a]` from `base Prelude, base Data.List`, and the function `deleteFirstsBy :: (a -> a -> Bool) -> [a] -> [a] -> [a]` from `base Data.List`. The `deleteFirstsBy` function is described as taking a predicate and two lists and returning the first list with the elements that satisfy the predicate removed.



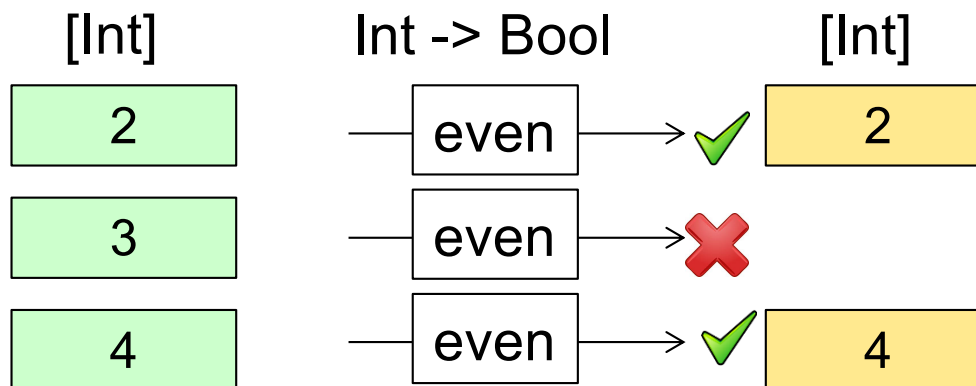
## filter – Remove List Elements

- Removes elements if they do not fulfill a condition

```
filter :: (a -> Bool) -> [a] -> [a]
```

- Illustration

```
filter even [2, 3, 4] ~> [2,4]
```



- Properties
  - The type of the list does not change
  - The length of the list may change

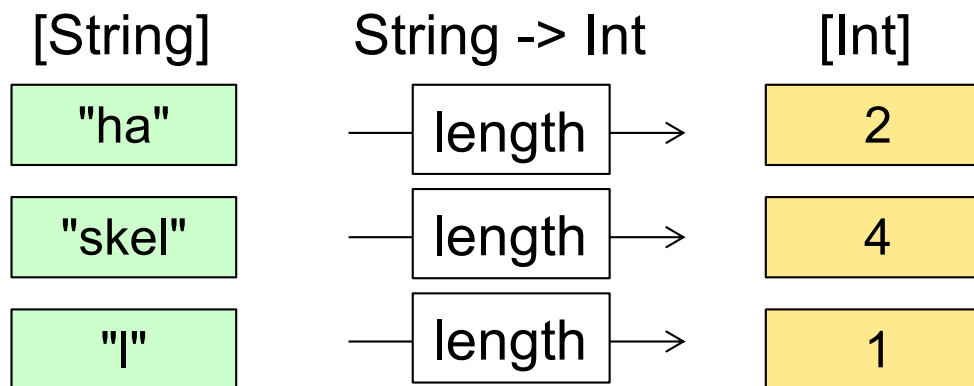
## map – Transform List Elements

- Transforms each element with the given function

```
map :: (a -> b) -> [a] -> [b]
```

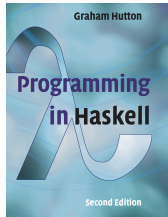
- Illustration

```
map length ["ha", "skel", "l" ] ~> [2,4,1]
```

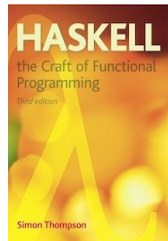


- Properties
  - The type of the list may change
  - The length of the list does not change

## Further Reading



Chapter 2  
Page 20



Chapter 5, 6  
Pages 97 – 99, 109 – 111, 123 – 128



Chapter 1  
Pages 7 – 12

[http://www.haskell.org/haskellwiki/How to work on lists](http://www.haskell.org/haskellwiki/How_to_work_on_lists)