Fachhochschule Nordwestschweiz

Functional Programming

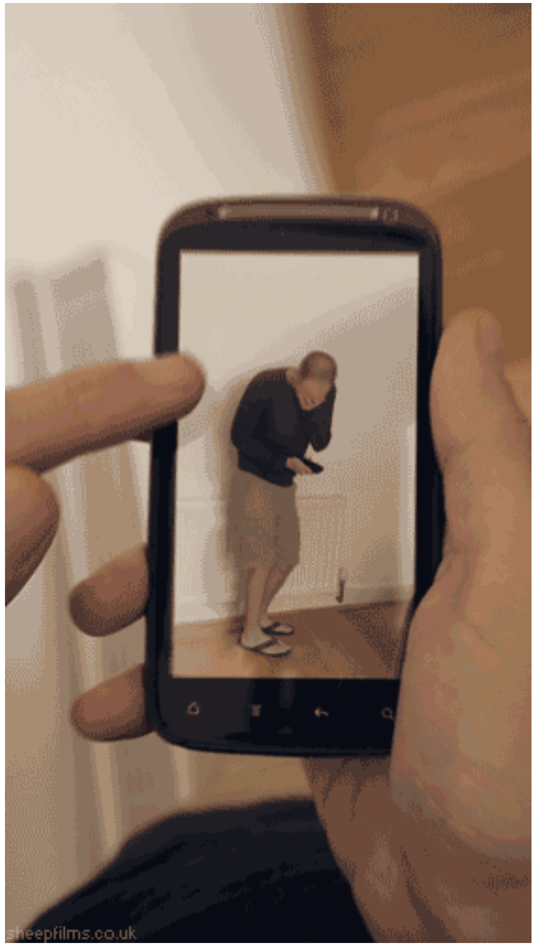
Recursion

Daniel Kröni
University of Applied Sciences Northwestern Switzerland

Learning Targets

You know what recursion is

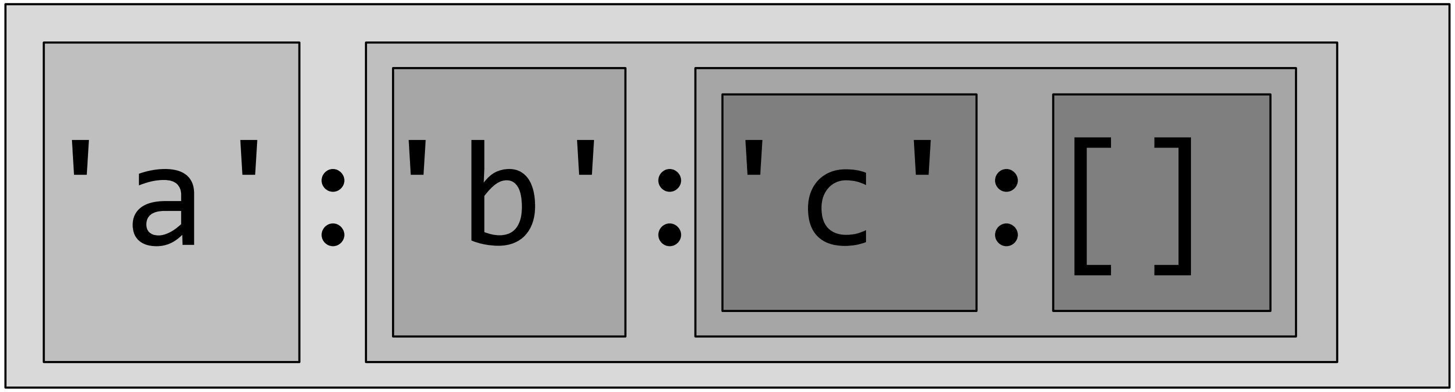
- You know the concept of recursion
- You can rewrite loops using recursion
- You can effectively use pattern matching to program recursive functions



What is recursion?

- **Recursion is the process of repeating items in a self-similar way.**
  E.g. russian dolls

- **The most important datastructure in Haskell is defined in a recursive manner:**



A list consists of a head element that is prepended to a list

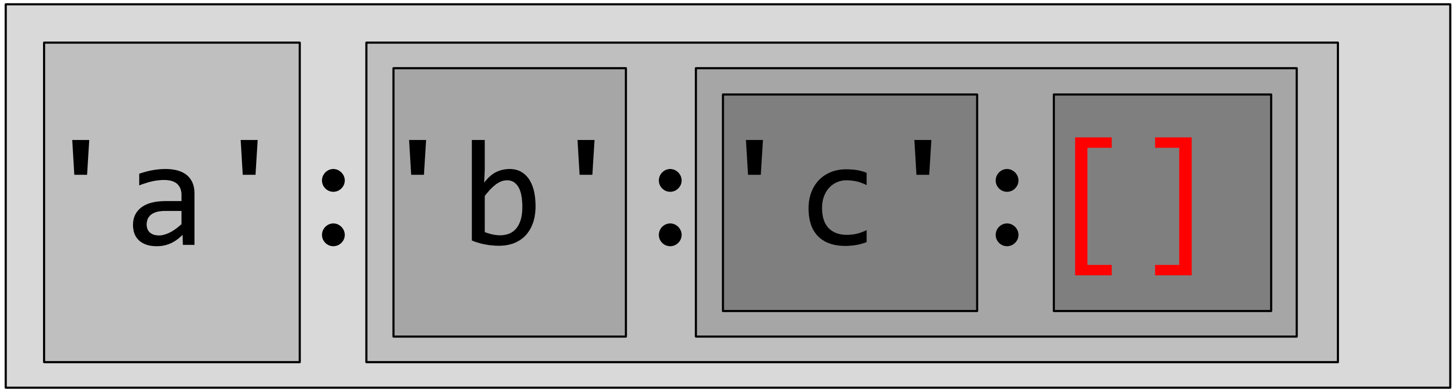
Infinite Recursion?

A list consists of a head element that is prepended to a list.

Wait a moment, what kind of definition is this? This does not stop!

- Correct: this is a definition for a infinite list.

Therefore we need an additional condition in our definition to make lists finite:



A list consists of a head element that is prepended to a list
AND at the end of a finite list is always the empty list [].

[3143, 5797, 6551, 8915]

Fachhochschule Nordwestschweiz

What is recursion

- **We saw that functions can be defined in terms of other functions**

```
flipper :: Picture -> Picture
flipper p = beside (flipH p) (flipV p)
```

- **`flipper` is defined in terms of `beside`, `flipH` and `flipV`**

Recursion occurs when a function is defined in terms of itself!

```
factorial 0 = 1
factorial n = n * factorial (n-1)
```

- **factorial maps 0 to 1, and any other positive integer to the product of itself and the factorial of its predecessor.**

For example:

```
factorial 0 = 1
factorial n = n * factorial (n-1)
```

```
   factorial 3
=
   3 * factorial 2
=
   3 * (2 * factorial 1)
=
   3 * (2 * (1 * factorial 0))
=
   3 * (2 * (1 * 1))
=
   3 * (2 * 1)
=
   3 * 2
=
   6
```

Worksheet Recursion

Controlling Recursion

- **Progress in recursion can be made in many ways.**

```
countFromTo :: Int -> Int -> [Int]
countFromTo from to
  | from < to = from : (countFromTo (from+1) to)
  | from > to = from : (countFromTo (from-1) to)
  | otherwise = [to]
```

- **Oftentimes a function wants to make some preliminary bookkeeping before doing the (recursive) work:**

```
gcd :: (Integral a) => a -> a -> a
gcd 0 0 =  error "gcd 0 0 is undefined"
gcd x y =  gcd' (abs x) (abs y)
  where gcd' a 0  = a
        gcd' a b  = gcd' b (a `rem` b)
```

Then a helper function like gcd' can be used!

Mutual Recursion

- **Two (or even more) functions can also be defined in terms of each other.**
- **Example:**
  - Definition

```
isEven :: Int -> Bool
isEven 0 = True            --(e.1)
isEven n = isOdd (n-1)     --(e.2)
```

```
isOdd :: Int -> Bool
isOdd 0 = False            --(o.1)
isOdd n = isEven (n-1)     --(o.2)
```

  - Evaluation (simplified)

```
isOdd 4
~> isEven 3      --(o.2)
~> isOdd 2       --(e.2)
~> isEven 1      --(o.2)
~> isOdd 0       --(e.2)
~> False         --(o.1)
```

Tail Recursion

- **A function is tail recursive, if the recursive call is the outermost expression**
- **Tail recursion can be optimized by compilers**

Instead of writing:

```
sum :: Num a => [a] -> a  -- not tail recursive
sum []     = 0
sum (i:is) = i + sum is
```

One writes:

```
sum :: Num a => [a] -> a  -- tail recursive
sum l = sum' 0 l
  where sum' acc []     = acc
        sum' acc (i:is) = sum' (i+acc) is  -- tail call
```

Loops and Recursion

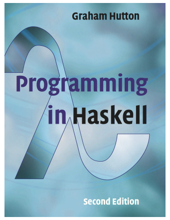
- **Every loop in Java can also be written as a recursive function and vice versa!**
  - Loops are usually controlled by a variable that changes its value from iteration to iteration. This change can be reflected in the recursions progress.
  - The loop's condition is the negation of the base case
  - The loop's body is the recursion step

```java
int sum(int n) {
  int sum = 0;
  while (n > 0) {
    sum += n;
    n--;
  }
  return sum;
}
```

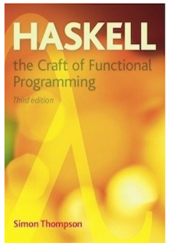
```haskell
sum :: Int -> Int
sum 0 = 0
sum n = n + sum (n-1)
```

Worksheet Recursion 2

Further Reading



Chapter 6



Chapter 7.4, 7.5

Chapter 4: Hello Recursion
http://learnyouahaskell.com/recursion