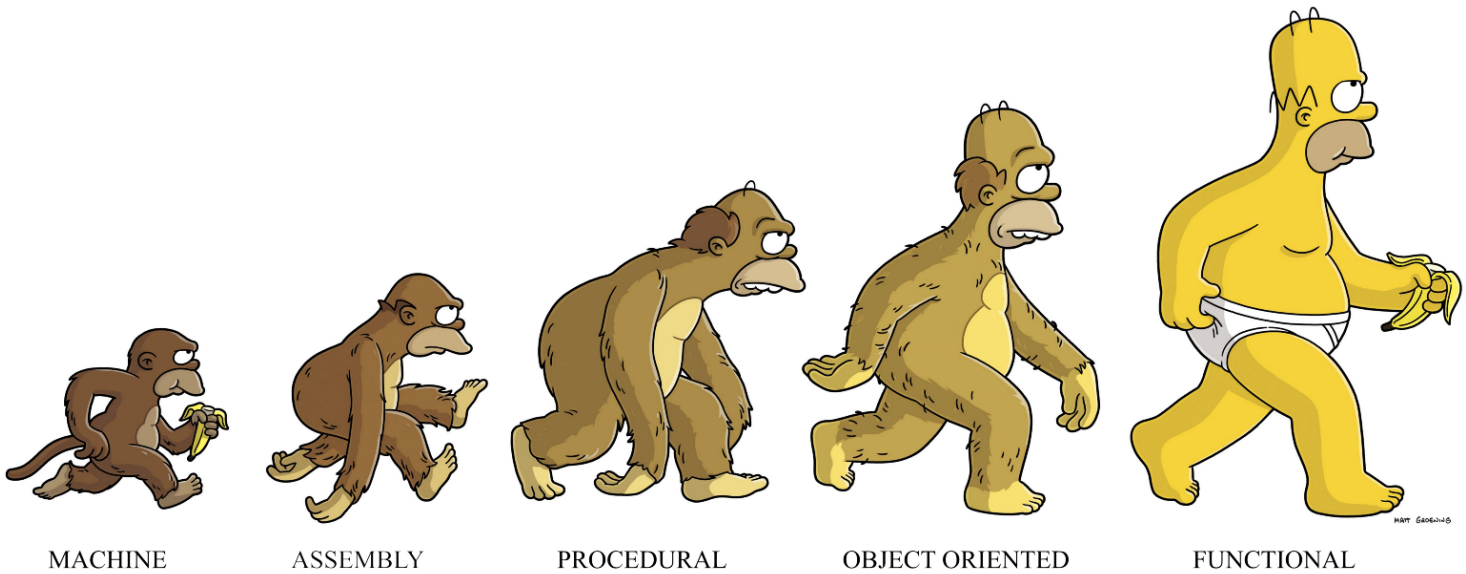
COMP302: Programming Languages and Paradigms

Prof. Brigitte Pientka (Sec 01)
bpientka@cs.mcgill.ca

Francisco Ferreira (Sec 02)
fferre8@cs.mcgill.ca

School of Computer Science
McGill University

Week 2-1, Fall 2017



Let's talk about lists!

Bad Programming Practice: Keeping it Old-Style

```
1  (* head: 'a list -> 'a  *)
2  let head (h::t) = h
3
4  (* tail: 'a list -> 'a list *)
5  let tail l = match l with
6    | [] -> []
7    | h::t -> t
8
9  (* Destructor style *)
10 let rec app (l1, l2) =
11   if l1 = [] then l2
12     else
13       head(l1)::(app (tail(l1), l2))
```

Why do you think this is bad style?

Task: Reverse a list

Task: Reverse a list

Write a function `rev` which given a list `l` of type `'a list` it returns its reverse.

Example: `rev [1, 2, 3, 4] ===> [4, 3, 2, 1]`

What is the type of `rev`?

Task: Reverse a list

Write a function `rev` which given a list `l` of type `'a list` it returns its reverse.

Example: `rev [1, 2, 3, 4] ===> [4, 3, 2, 1]`

```
1 (* rev : 'a list -> 'a list *)
2 let rec rev l = match l with
3   | []    -> []
4   | x::l -> (rev l) @ [x]
```

Task: Reverse a list

Write a function `rev` which given a list `l` of type `'a list` it returns its reverse.

Example: `rev [1, 2, 3, 4] ===> [4, 3, 2, 1]`

```
1 (* rev : 'a list -> 'a list *)
2 let rec rev l = match l with
3   | []    -> []
4   | x::l -> (rev l) @ [x]
```

Is this a good program?

Task: Reverse a list

Write a function `rev` which given a list `l` of type `'a list` it returns its reverse.

Example: `rev [1, 2, 3, 4] ===> [4, 3, 2, 1]`

```
1 (* rev : 'a list -> 'a list *)
2 let rec rev l = match l with
3   | []    -> []
4   | x::l -> (rev l) @ [x]
```

Is this a good program?

Tail-Recursion to the rescue!

Practice Problem 1

Write a function `merge:'a list -> 'a list -> 'a list` which given two **ordered** lists `l1` and `l2`, both of type `'a list`, it returns the sorted combination of both lists.

Practice Problem 2

Write a function `split:'a list -> 'a list * 'a list` which given a list `l` it splits it into two sublists.

Example:

```
split [1, 2, 3, 4]   ⟹   ([1, 3] , [2, 4])
split [1, 2, 3]      ⟹   ([1, 3] , [2])
```

Practice Problem 3

Write a function `zip:'a list * 'a list -> 'a list` which given two lists `l1` and `l2`, zips them together

Example:

```
zip ([1, 3] , [2, 4])   ⟹   [1, 2, 3, 4]
zip ([1, 3] , [2])      ⟹   [1, 2, 3]
```

Practice Problem 3

Write a function `zip:'a list * 'a list -> 'a list` which given two lists `l1` and `l2`, zips them together

Example:

```
zip ([1, 3] , [2, 4])   ⟹   [1, 2, 3, 4]
zip ([1, 3] , [2])      ⟹   [1, 2, 3]
```

Prove: For all `l`, `zip (split l) = l`.

Functional Tidbit: Words of Wisdom

"On theories such as these we cannot rely. Proof we need. Proof!"