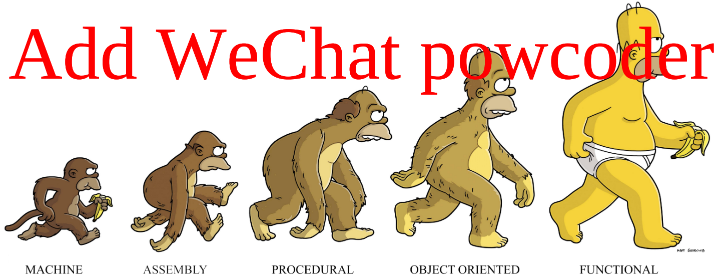
COMP302: Programming Languages and Paradigms

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

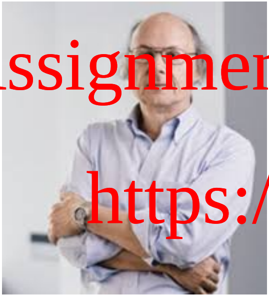
Francisco Ferreira (Sec 02)
fferre8@cs.mcgill.ca

School of Computer Science
McGill University

Week 6-1, Fall 2017



Functional Tidbit: Imperative vs Functional Programming!



"I find languages that support just one programming paradigm constraining."

- Bjarne Stroustrup

Warm-Up: Computation and Effects

So far:

Expressions in OCaml have characteristics:

- An expression has a type
- An expression evaluates to a value (or diverges).

Today:

Expressions in OCaml may also have an *effect*.

Warm-Up: Type, Values, and Effect

Given the following expression write down its type, its value (i.e. what the expression evaluates to), and its effect, if it has any.

- 3 + 2

Warm-Up: Type, Values, and Effect

Given the following expression write down its type, its value (i.e. what the expression evaluates to), and its effect, if it has any.

- 3 + 2
- 55

Warm-Up: Type, Values, and Effect

Given the following expression write down its type, its value (i.e. what the expression evaluates to), and its effect, if it has any.

- `3 + 2`
- `55`
- `fun x -> x + 3 * 2`

Warm-Up: Type, Values, and Effect

Given the following expression write down its type, its value (i.e. what the expression evaluates to), and its effect, if it has any.

- `3 + 2`
- `55`
- `fun x -> x + 3 * 2`
- `((fun x -> match x with [] -> true | y::ys -> false) , 3.2 *. 2.0)`

Warm-Up: Type, Values, and Effect

Given the following expression write down its type, its value (i.e. what the expression evaluates to), and its effect, if it has any.

- `3 + 2`
- `55`
- `fun x -> x + 3 * 2`
- `((fun x -> match x with [] -> true | y::ys -> false) , 3.2 *. 2.0)`
- `let x = ref 3 in x := !x + 2`

Warm-Up: Type, Values, and Effect

Given the following expression write down its type, its value (i.e. what the expression evaluates to), and its effect, if it has any.

- `3 + 2`
- `55`
- `fun x -> x + 3 * 2`
- `((fun x -> match x with [] -> true | y::ys -> false) , 3.2 *. 2.0)`
- `let x = ref 3 in x := !x + 2`
- `fun x -> x := 3`

Warm-Up: Type, Values, and Effect

Given the following expression write down its type, its value (i.e. what the expression evaluates to), and its effect, if it has any.

- `3 + 2`
- `55`
- `fun x -> x + 3 * 2`
- `((fun x -> match x with [] -> true | y::ys -> false) , 3.2 *. 2.0)`
- `let x = ref 3 in x := !x + 2`
- `fun x -> x := 3`
- `fun x -> (x := 3; x)`

Warm-Up: Type, Values, and Effect

Given the following expression write down its type, its value (i.e. what the expression evaluates to), and its effect, if it has any.

- 3 + 2
- 55
- `fun x -> x + 3 * 2`
- `((fun x -> match x with [] -> true | y::ys -> false) , 3.2 *. 2.0)`
- `let x = ref 3 in x := !x + 2`
- `fun x -> x := 3`
- `fun x -> (x := 3; x)`
- `fun x -> (x := 3; !x)`

Today: Programming with State

- Mutable Data-Structures
- Closures and Objects

– Demo –