# Intermediate OCaml: Options, Monads, Modules

## CAS CS 320: Principles of Programming Languages

Thursday, February 22, 2024

# Administrivia

- Homework 4 is due on Friday, Feb 23, by 11:59 pm.
- No homework this week.
- Midterm on Tuesday, Feb 27, during lecture time. There will be two locations (more details on Piazza).

# Administrivia

- Homework 4 is due on Friday, Feb 23, by 11:59 pm.
- No homework this week.
- Midterm on Tuesday, Feb 27, during lecture time. There will be two locations (more details on Piazza).

# Reading Assignment

- OCP, Section 3.7: **Options**
- OCP, Sections 5.1, 5.2: **Modules**, **Module Systems**
- OCP, Section 8.7: **Monads**

# options (OCP 3.7)

options have already been used this semester, in lectures and scripts, so we know what they are.

# options (OCP 3.7)

options have already been used this semester, in lectures and scripts, so we know what they are.

here is another example (from the book), the function **list_max** with options, because the input list may be the empty list [], in which case there are *no* maximum value:

```
let rec list_max = function
  | [] -> None
  | h :: t -> begin
      match list_max t with
        | None -> Some h
        | Some m -> Some (max h m)
      end
```

# options (OCP 3.7)

options have already been used this semester, in lectures and scripts, so we know what they are.

here is another example (from the book), the function **list_max** with options, because the input list may be the empty list [], in which case there are *no* maximum value:

```
let rec list_max = function
  | [] -> None
  | h :: t -> begin
      match list_max t with
        | None -> Some h
        | Some m -> Some (max h m)
      end
```

the type of list_max? `val list_max : 'a list -> 'a option = <fun>`

# **modules and module systems** (OCP 5.1, 5.2)

a **module system** is the collection of features which support and facilitate **modular programming**.

# **modules and module systems** (OCP 5.1, 5.2)

a **module system** is the collection of features which support and facilitate **modular programming**.

features of modular programming: **abstractions**, **namespaces**, **code reuse**.

# **modules and module systems** (OCP 5.1, 5.2)

a **module system** is the collection of features which support and facilitate **modular programming**.

features of modular programming: **abstractions**, **namespaces**, **code reuse**.

an **abstraction** hides some information while revealing other information, thus enabling **encapsulation**, aka information hiding - example: **function definitions**.

# modules and module systems (OCP 5.1, 5.2)

a **module system** is the collection of features which support and facilitate **modular programming**.

features of modular programming: **abstractions**, **namespaces**, **code reuse**.

an **abstraction** hides some information while revealing other information, thus enabling **encapsulation**, aka information hiding - example: **function definitions**.

a **namespace** is a set of names that are grouped together because they are logically related, differentiating them from other namespaces - example: namespace of module **List**.

# modules and module systems (OCP 5.1, 5.2)

a **module system** is the collection of features which support and facilitate **modular programming**.

features of modular programming: **abstractions**, **namespaces**, **code reuse**.

an **abstraction** hides some information while revealing other information, thus enabling **encapsulation**, aka information hiding – example: **function definitions**.

a **namespace** is a set of names that are grouped together because they are logically related, differentiating them from other namespaces – example: namespace of module **List**.

**code reuse** is supported by features allowing code from one module to be used in another module without having to copy that code – example: **functors** (which produce new modules from older modules – not used this semester).

# modules and module systems (OCP 5.1, 5.2)

example of a (tiny!) user-defined module:

```
module MyModule = struct
  let inc x = x + 1
  type primary_color = Red | Green | Blue
  exception Oops
end
```

this script will compile perfectly well. `MyModule` defines a namespace which includes: `inc`, `Red`, `Green`, `Blue`, and `Oops`.

# monads (OCP 8.7)

What is a monad?

# **monads** (OCP 8.7)

What is a monad?

The term comes from an area of mathematics called ***category theory*** – ***but you do not have to worry about this!!!***

# **monads** (OCP 8.7)

What is a monad?

The term comes from an area of mathematics called ***category theory*** – ***but you do not have to worry about this!!!***

All you need to remember are three facts:

1. In *pure functional programming* there are no *side effects*, an example of a *side effect* is printing on the screen.

# **monads** (OCP 8.7)

What is a monad?

The term comes from an area of mathematics called ***category theory*** – ***but you do not have to worry about this!!!***

All you need to remember are three facts:

1. In *pure functional programming* there are no *side effects*, an example of a *side effect* is printing on the screen.

2. No practical programming is possible without *side effects*. For example, we cannot do without printing. OCaml is not a *pure functional language*.

# **monads** (OCP 8.7)

What is a monad?

The term comes from an area of mathematics called ***category theory*** – ***but you do not have to worry about this!!!***

All you need to remember are three facts:

1. In *pure functional programming* there are no *side effects*, an example of a *side effect* is printing on the screen.

2. No practical programming is possible without *side effects*. For example, we cannot do without printing. OCaml is not a *pure functional language*.

3. A *monad* is a design pattern that controls and simulates *side effects*.

( THIS PAGE INTENTIONALLY LEFT BLANK )