

Introduction to Functional Programming in *OCaml*

Roberto Di Cosmo, Yann Régis-Gianas, Ralf Treinen

Week 5 - Sequence 1: Getting and handling your Exceptions



Exceptions and the *exn* type

OCaml provides *exceptions* for signalling and handling exceptional conditions.

- ▶ exceptions are *constructors* of a special sum type *exn*
- ▶ these constructors *can have arguments*, like all other constructors
- ▶ new exceptions can be defined *at any time*
- ▶ this makes the *exn* sum type special:
unlike the usual sum types, it can be *extended*
- ▶ *exceptions cannot be polymorphic*

Declaring exceptions

Exceptions are *declared* using the `exception` keyword

```
# exception E;;
```

```
exception E
```

They are just constructors:

```
# E;;
```

```
— : exn = E
```

Raising exceptions

Exceptions are *signalled* using the `raise` keyword

```
# raise E;;
```

exception: E.

When an exception is raised, the computation is immediately stopped.

```
# let _ = raise E in [1;2];;
```

exception: E.

Let's see a more realistic example.

Taking the head of an empty list I

```
exception Empty_list;;  
# exception Empty_list
```

```
(* define a head function that uses the exception *)  
let head = function  
  a::r -> a  
  | []   -> raise Empty_list;;  
# val head : 'a list -> 'a = <fun>
```

```
(* let's test *)  
head ['a';'b'];;  
# - : char = 'a'  
head [];;  
# Exception: Empty_list.
```

Handling exceptions

Exception can be *captured*, using the `try with` construct.

```
try
  e
with
  p1 -> e1
  | p2 -> e2
  | ...
```

- ▶ `e` is evaluated
- ▶ if `E` is raised, match it with the patterns in the `with` clause
- ▶ *you can use any pattern of type `exn`*
- ▶ if `E` matches pattern `pi`, evaluate expression `ei`
- ▶ *all the `ei` must have the same type as `e`*

Handling examples I

```
(* multiplying all values of an integer list *)  
(* think of a 1 million element list with a 0 at the end *)
```

```
let rec multl = function  
  [] -> 1  
  | a::rest -> if a = 0 then 0 else a * (multl rest)  
;;  
# val multl : int list -> int = <fun>
```

Handling examples II

(* use exceptions to return as soon as we see a zero *)

```
exception Zero;;
```

```
# exception Zero
```

```
let multlexc l =
```

```
  let rec aux = function
```

```
    [] -> 1
```

```
    | a::rest -> if a = 0 then raise Zero else a * (aux rest)
```

```
  in
```

```
  try aux l with Zero -> 0;;
```

```
# val multlexc : int list -> int = <fun>
```


When things go wrong

Run-time errors

OCaml catches type errors at compile time, but other errors may occur *at runtime*

- ▶ division by zero
- ▶ incomplete pattern matching
- ▶ out-of-bound access to indexed data structures like arrays
- ▶ ...

Capturing errors as exceptions

In *OCaml*, these errors do not *crash* the program:
they *raise* an *exception*, which you can handle!

Let's see some examples.

Meet the exceptions I

```
(* division by zero *)
```

```
3/0;;
```

```
# Exception: Division_by_zero.
```

```
(* out of bound access to mutable data structures *)
```

```
let v = [|1;2;3|];;
```

```
# val v : int array = [|1; 2; 3|]
```

```
v.(0);;
```

```
# - : int = 1
```

```
v.(3);;
```

```
# Exception: Invalid_argument "index_out_of_bounds".
```

Meet the exceptions II

(* incomplete pattern matching *)

```
let drop = function  
  | a::rest -> rest;;
```

```
# Characters 47-75:
```

```
.....function  
  | a::rest -> rest..
```

Warning 8: this pattern-matching is **not** exhaustive.

Here is an example **of** a **value** that is **not** matched:

```
[]
```

```
val drop : 'a list -> 'a list = <fun>
```

```
drop [1;2;3;4;5];;
```

```
# - : int list = [2; 3; 4; 5]
```

Meet the exceptions III

```
drop [];;
```

```
# Exception: Match_failure ("//toplevel//", 8, 11).
```

Summary

Exceptions

- ▶ Constructors of a special *exn* sum type.
- ▶ Declared and raised using `exception` and `raise`.
- ▶ Handled using the `try ... with ...` construct.
- ▶ Useful for signalling and handling exceptional conditions, and for altering the flow of control.
- ▶ Good to know: *raising and handling exceptions is very fast.*