It Will Warp Your Brain
(And We Like Doing That Kind of Stuff)
Features of OCaml

- OCaml is a statically typed language
Features of OCaml

- OCaml is an **IMPLIED** statically typed language
Features of OCaml

• OCaml is an implied statically typed language

• let x, y, z = 2, false, “Hello, World!”
  val x : int = 2
  val y : bool = false
  val z : string = “Hello, World!”

• More static typing goodness, less pain.
More on Static Typing

• Gives free check to make sure that the return types are consistent.

• Derives the type of the argument(s) and genericizes arguments for free.

• Catches typos, API changes, etc.

• Provides the basis for the most advanced safety features of the language.
OCaml is a Functional Language

- The key operation is function application, and functions are the key type.
- First mind-bend in OCaml:
  - let \( f \ x \ y = x \ y \)
    \[ \text{val} \ f : ('a \rightarrow 'b) \rightarrow 'a \rightarrow 'b = <\text{fun}> \]
  - let \( g = f \ \text{int	extunderscore of	extunderscore string} \)
    \[ \text{val} \ g : \text{string} \rightarrow \text{int} = <\text{fun}> \]
  - let \( h = f \ ((+) \ 2) \)
    \[ \text{val} \ h : \text{int} \rightarrow \text{int} = <\text{fun}> \]
  - let \( k = ((+) \ 2) \)
    \[ \text{val} \ k : \text{int} \rightarrow \text{int} = <\text{fun}> \]
Simple Data Types

- string: “Hello, World!”, “”
- int: 2
- float: 0.5
- bool: true, false
- unit: ()
Complex Data Types

Tuples

- let xyz = ( 2, false, "Hello, World!" )
- val xyz : int * bool * string

- Retains type safety
- Fixed size
- Sometimes called "n-ple"
- "()" is really the 0-arity tuple
Complex Data Types

Lists

• let xyz = ["2", "false", "Hello, World!"]
  var xyz : string list

• Single type

• Arbitrary (but not variable) length
Complex Data Types

Records

• `type abc = { x:int; y:bool; z:string }`
• `let xyz = {x=2; y=false; z="Hello, World!"} val xyz : abc`

• Nice field access notation (xyz.x)
• Provides an easy “mutable” hook
Complex Data Types

References

• Succinct mutable data declaration.

• `let x = ref 2`
  
  ```ml
  val x : int ref = { contents = 2 }
  ```

• `x := 3`
  
  ```ml
  - : unit = ()
  ```

• `x`
  
  ```ml
  - : int ref = { contents = 3 }
  ```
Complex Data Types

Variant Types

- Think enums, but awesome.
- type hobbit = Bilbo | Frodo
- let hobb it =
  match it with
  | Bilbo -> “There and Back Again”
  | Frodo -> “Big Book o’ Whining”
- type rational = Int of int | Frac of int * int
Some Fun Variant Types

- type idx = StrIdx of string | NumIdx of int
- type 'a option = Some of 'a | None
- type 'a maybe = Result of 'a | Exception of exn
- type 'a list_t = Empty | Node of 'a * 'a list_t
- type xml =
  | PCDATA of string
  | Element of (string * (string * string) list * xml list)
  (* text or (tag-name, attributes, children) *)
More Fun Variant Types

- type http_status =
  | Ok
  | NotFound
  | NotModified
  | BadRequest of string
  | Unknown of int * string

- type 'a val_or_ref = Val of 'a | Ref of 'a ref

- type 'a thunk = Result of 'a | Factory of (unit -> 'a)
Weird Accents and Training Wheels

- OCaml has imperative loops (for, while)
- You don’t really want to use them
- See Section 1.5 of the manual
- OCaml has objects
  - Structurally typed
  - Functionality is somewhat redundant
Goal: Make it easier to express pipelines of operations.

```ocaml
let (|>) x f = f x;;
val (|>) :  'a -> ('a -> 'b) -> 'b = <fun>
```

2 |> string_of_int |> ((^) "I had a: ")

This example brought to you courtesy of Hamlet D’Arcy.
OCaml Example

Handle Handling

• Goal: Create a global handle with init and cleanup

• let handle =
  global_init ();
  let h = inst_init () in
  at_exit (fun () ->
    inst_clean h; global_clean ());
  h
OCaml Example

Ranges

• Goal: Akin to Ruby’s (5..10) or (10..5)

• let (->>) a b =
  let next, p = if a < b then pred,(<)
  else succ,(>) in
  let rec loop i acc =
    if p i a then acc
    else loop (next i) (i::acc) in
  loop b []
OCaml Example
List of Primes

- Goal: Get a list of first $n$ primes.
- `let prime_list n =`
  if $n < 0$ failwith “Need a nonnegative n” else
  match $n$ with | 0 -> [] | 1-> [1] | 2-> [1;2]|
  | $n$ ->
    let rec builder $x$ $lst$ =
      if List.length $lst$ = $n$ -2 then List.rev $lst$ else
      let $m$ = (mod) $x$ in
      let is_prime = List.for_all (fun $i$ -> $m$ $i$ != 0) $lst$ in
      let nxt = builder ($x$+2) in
      if is_prime then nxt ($x$::$lst$) else nxt $lst$ in
    in
    [1;2]@(builder 3 [])
OCaml Example

Rails’ Date DSL/API

• Goal: Something like 5.days.ago --> date

• Start with thinking in types!

• We’ll end up with:
  let x = date 5 Days Ago
  let y = date 2 Weeks Hence

• type scale = Hours | Days | Weeks
  type direction = Ago | Hence
• let date x y z =
  let seconds_per_hour = 60.0 *. 60.0 in
  let scl = match y with
    | Hours -> seconds_per_hour
    | Days  -> 24.0 *. seconds_per_hour
    | Weeks -> 7.0 *. 24.0 *. seconds_per_hour
  in
  let amt = match z with
    | Ago   -> -1.0 *. scl *. x
    | Hence -> scl *. x
  in
  localtime (gettimeofday() +. amt)
let (@>) filename handler =
  let chan = open_in filename in
  let rec loop acc =
    let line = try Some(input_line chan)
    with End_of_file -> None in
    match line with
    | Some line -> loop ((handler line) :: acc)
    | None -> List.rev acc
  in
  let out = loop [] in
  close_in chan;
  out
Things I Don’t Get Yet

• Monads
  • Inversion of Control on Steroids
    http://enfranchisedmind.com/blog/2007/08/06/a-monad-tutorial-for-ocaml/

• Phantom Types
  • Semantic Metadata That Replaces Runtime Checks with Compiletime Checks
    http://ocaml.janestcapital.com/?q=node/11
Future Directions

- JoCaml, the Parallel GC, and CloudProxy
- XDuce
- F#, rocaml, ocaml-java
- Ocsigen, Laconic/Web, Ex Nunc
OCaml Needs You!

- Ruby’s culture of intuitive tooling and intense documentation
- Ruby’s culture of pragmatic development
- Break OCaml’s performance-obsessed culture and hyperoptimization
- OCaml needs people
Further References

- Godi: OCaml’s “gem” functionality
- OCaml’s manual is a pretty good introduction
- #ocaml on FreeNode (IRC) for questions
- ocaml-beginners on Yahoo!
  [http://groups.yahoo.com/group/ocaml_beginners/](http://groups.yahoo.com/group/ocaml_beginners/)
- Inria’s OCaml mailing list
  [http://yquem.inria.fr/cgi-bin/mailman/listinfo/caml-list](http://yquem.inria.fr/cgi-bin/mailman/listinfo/caml-list)
- Others: OCaml Forge, Caml Hump, Jane St’s Blog
How To Come To Love OCaml:
Program in it long enough to start being annoyed by it, and then go back and program something else, to discover it annoys you more.  (Brian Hurt)

More on our blog
http://enfranchisedmind.com/blog/?s=ocaml