Introduction to functional programming with OCAML (part I)

Class 02, 2016-02-02, WWH 312

Patrick Cousot
pcousot@cs.nyu.edu  cs.nyu.edu/~pcousot

Obtaining OCaml

- Home page: ocaml.org
- Install: (Linux, Windows, MacOS X): ocaml.org/docs/install.html
- Online User Manual: caml.inria.fr/pub/docs/manual-ocaml/
- Quick introduction: en.wikipedia.org/wiki/OCaml
- You should now have Ocaml installed on your machine

Objective CAML (OCaml)

Ocaml is a functional language

- One can define functionals that is functions taking functions as arguments and returning functions
- Example: function composition

```ocaml
# let incr x = x + 1;;
val incr : int -> int = <fun>
# let o f g x = (f (g (x)));;
val o : ('a -> 'b) -> ('c -> 'a) -> 'c -> 'b = <fun>
# let incre2 = o incr incr;;
val incre2 : int -> int = <fun>

# incr 7;;
- : int = 9
# (o incr2 incr2) 7;;
- : int = 11
# let ooo = o o o;;
val ooo : (_a -> _b) -> (_c -> _d -> _a) -> _c -> _d -> _b
  = <fun>
```
OCaml is statically, polymorphically and automatically typed

```ocaml
# let incr x = x + 1;;
val incr : int -> int = <fun>
# let o f g x = (f (g (x)));;
val o : ('a -> 'b) -> ('c -> 'a) -> 'c -> 'b = <fun>
# let incre2 = o incr incr;;
val incre2 : int -> int = <fun>
# incre2 7;;
- : int = 9
# (o incre2 incre2) 7;;
- : int = 11
```

OCaml is an imperative language

- **Assignment** to variables (hence **impure, side-effects** in functions are possible)

```ocaml
# let x = ref 0;;
val x : int ref = {contents = 0}
# !x;;
- : int = 0
# x := 1;;
- : unit = ()
# !x;;
- : int = 1
```

Side-effects

```ocaml
# let s = "abcdef";;
val s : string = "abcdef"
# s;;
- : string = "abcdef"
# set s 'X';;
Unbound value set
# String.set s 'X';;
- : unit = ()
# s;;
- : string = "Xbcdef"
```

Module system

- **Modules**: separately compiled program parts
- **Functors**: modules with formal module parameters that can be instantiated with different module actual parameters
- **Separate compilation**: the program parts can be type-checked and compiled without requiring the whole program
Ocaml has many libraries

- The standard library
- The unix library: Unix system calls
- The num library: arbitrary-precision rational arithmetic
- The str library: regular expressions and string processing
- The threads library
- The graphics library
- The dbm library: access to NDBM databases http://en.wikipedia.org/wiki/Ndbm
- The dynlink library: dynamic loading and linking of object files
- The LablTk library: Tcl/Tk GUI interface
- The bigarray library
- ...

Ocaml has many libraries (cont’d)

21. Module Marshal: marshaling of data structures
22. Module MoreLabels: Include modules Hashtbl, Map and Set with labels
23. Module Nativeint: processor-native integers
24. Module Oo: object-oriented extension
25. Module Parsing: the run-time library for parsers generated by ocamlyacc
26. Module Printexp: facilities for printing exceptions
27. Module Printf: formatting printing functions
28. Module Queue: first-in first-out queues
29. Module Random: pseudo-random number generator (PRNG)
30. Module Scanf: formatted input functions
31. Module Set: sets over ordered types
32. Module Sort: sorting and merging lists
33. Module Stack: last-in first-out stacks
34. Module StdLabels: Include modules Array, List and String with labels
35. Module Stream: streams and parsers
36. Module String: string operations
37. Module StringLabels: string operations (with labels)
38. Module Sys: system interface
39. Module Weak: arrays of weak pointers

Ocaml is interfaced with Unix


Ocaml is interfaced with C

- See “Interfacing C with Objective Caml”: http://caml.inria.fr/pub/docs/manual-ocaml/intfc.html
OCaml has many programming tools

- Batch compilation (ocamlc)
  - The toplevel system (ocaml)
  - The runtime system (ocamlr
- Native-code compilation (ocamlopt)
- Lexer and parser generators (ocamllex, ocamlyacc)
- Dependency generator (ocamldep)
- The browser/editor (ocamlbrowser)
- The documentation generator (ocamldoc)
- The debugger (ocamldebug)
- Profiling (ocamlprof)
- Interfacing C with Objective Caml

Ocaml can be interpreted or compiled

- Interpreted with ocaml
- Compiled into
  - bytecode by ocamlc, run by ocamlr
  - native machine code by ocamlopt

OCaml is free

- The tools and libraries are “free”
- The code produced by these tools is “free”

see details of OCaml’s licence

ocaml.org/docs/license.html

Ocaml is also used in the industry

- e.g. Google Ad

Ad related to OCaml

OCaml Hackers Wanted - Hard problems, great pay - janestreet.com
www.janestreet.com
Work with technology you love.
Follow our OCaml blog - Apply for a job at Jane Street

Note: nothing is free, the french taxpayers did pay for OCaml
Launching the OCaml interpreter

```
~ % ocaml
OCaml version 4.01.0

# "hello word";;
- : string = "hello word"

# ^D ctrl+D
~ %
```

Comments

```
# (* This is a comment (* with another
 * comment inside *) so program parts with
 * comments can be commented *)
```

Basic types

OCaml is a typed expression language.
Basic types

- **unit**: singleton ()
- **bool**: Booleans (true, false)
- **int**: integers
- **char**: characters (ASCII)
- **string**: character strings (Unicode)
- **float**: IEEE 754 floating point numbers

unit type

```ocaml
# ();;
- : unit = ()
# (* parameterless function *)
  let f () = 1;;
  val f : unit -> int = <fun>
# ();;
- : int = 1
# (* procedure: resultless function *)
  ignore;;
- : 'a -> unit = <fun>
# ignore 12345;;
- : unit = ()
# (* no operation on (): can be a parameter or returned *)
  let id x = x;;
  val id : 'a -> 'a = <fun>
# id ();;
- : unit = ()
```

bool type

```ocaml
# (* booleans *)
true;;
- : bool = true
# false;;
- : bool = false
# (* equality of booleans *)
  true = true;;
- : bool = true
# true = false;;
- : bool = false
- # (* prefix unary negation *)
  not true;;
- : bool = false
# not false;;
- : bool = true
```

bool type (cont’d)

```ocaml
# (* infix, left associative, lazy boolean operators *)
  failwith "fail";;
Exception: Failure "fail".
# true && (failwith "fail");;
Exception: Failure "fail".
# false && (failwith "fail");;
- : bool = false
# true || (failwith "fail");;
- : bool = true
# min true false;;
- : bool = false
# max true false;;
- : bool = true
```
Infixed and prefixed operators

(* infix operators *)
true && true
- : bool = true

(* prefix operators *)
(&&) true true
- : bool = true
(||) false (failwith "fail")
- : bool = false

Boolean operators

<table>
<thead>
<tr>
<th>Function</th>
<th>Examples</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>not : bool -&gt; bool</td>
<td>not true, not (i = j)</td>
<td>(Prefix) Unary negation.</td>
</tr>
<tr>
<td>&amp;&amp; : bool -&gt; bool -&gt; bool</td>
<td>(i = j) &amp; &amp; (j = k)</td>
<td>(Infix, left associative) Conjunction, with short-circuit evaluation.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>: bool -&gt; bool -&gt; bool</td>
</tr>
</tbody>
</table>

% ocaml
OCaml version 4.01.0

# not
- : bool -> bool = <fun>

# (&&)
- : bool -> bool -> bool = <fun>

# (||)
- : bool -> bool -> bool = <fun>

from http://www.csc.villanova.edu/~dmatusze/resources/ocaml/ocaml.html

Conditionals

# if true then 1 else 0
- : int = 1

# if false then 1 else 0
- : int = 0

# if false then true else 0
Error: This expression has type int but an expression was expected of type
bool

# if true then 0
Error: This expression has type int but an expression was expected of type
unit

Both branches of the conditional must have the same type (restriction due to the typing system)

Relational expressions

# 1 < 2
- : bool = true

# let x = 1
val x : int = 1
# x = 1
- : bool = true
# 1 < 2.0
Error: This expression has type float but an expression was expected of type
int
# 1.0 < 2.0
- : bool = true

The type inference algorithm allows overloaded/generic relational operators such as <, <=, =, ==, !=, >=, >, min, max for bool, char, string, int, and float.
### Comparison operators

<table>
<thead>
<tr>
<th>Function</th>
<th>Examples</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt; : 'a -&gt; 'a -&gt; bool</td>
<td>i &lt; 0</td>
<td>Less than. 'a' can be int, float, char, or string.</td>
</tr>
<tr>
<td>&lt;= : 'a -&gt; 'a -&gt; bool</td>
<td>x &lt;= 0.0</td>
<td>Less than or equal to. 'a' can be int, float, char, or string.</td>
</tr>
<tr>
<td>= : 'a -&gt; 'a -&gt; bool</td>
<td>x = &quot;abc&quot;</td>
<td>Equals. 'a' can be int, char, or string, but not float.</td>
</tr>
<tr>
<td>&lt;&gt; : 'a -&gt; 'a -&gt; bool</td>
<td>ch &lt;&gt; &quot;\n&quot;</td>
<td>Not equal. 'a' can be int, char, or string, but not float.</td>
</tr>
<tr>
<td>&gt; : 'a -&gt; 'a -&gt; bool</td>
<td>i &gt; j</td>
<td>Greater than or equal to. 'a' can be int, float, char, or string.</td>
</tr>
<tr>
<td>&gt;= : 'a -&gt; 'a -&gt; bool</td>
<td>x &gt;= y</td>
<td>Greater than. 'a' can be int, float, char, or string.</td>
</tr>
</tbody>
</table>

From [http://www.csc.villanova.edu/~dmatusze/resources/ocaml/ocaml.html](http://www.csc.villanova.edu/~dmatusze/resources/ocaml/ocaml.html)

### Encoding of integers

- Integers are encoded on 63 bits (one for garbage collection)
- Integers are encoded in 2's complement and use modular arithmetics and so `max_int + 1 = min_int`

### Integer operators

<table>
<thead>
<tr>
<th>Function</th>
<th>Examples</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>- : int -&gt; int</td>
<td>-5, -limit</td>
<td>(Prefix) Unary negation.</td>
</tr>
<tr>
<td>* : int -&gt; int -&gt; int</td>
<td>2 * limit</td>
<td>(Infix, left associative) Multiplication; operands and result are all ints.</td>
</tr>
<tr>
<td>/ : int -&gt; int -&gt; int</td>
<td>7 / 3, score / average</td>
<td>(Infix, left associative) Division; truncates fractional part.</td>
</tr>
<tr>
<td>mod : int -&gt; int -&gt; int</td>
<td>limit mod 2</td>
<td>(Infix, left associative) Modulus; result has sign of first operand.</td>
</tr>
<tr>
<td>+ : int -&gt; int -&gt; int</td>
<td>2 + 2, limit + 1</td>
<td>(Infix, left associative) Addition.</td>
</tr>
<tr>
<td>- : int -&gt; int -&gt; int</td>
<td>2 - 2, limit - 1</td>
<td>(Infix, left associative) Subtraction.</td>
</tr>
<tr>
<td>abs : int -&gt; int</td>
<td>abs (-5)</td>
<td>(Prefix) Absolute value.</td>
</tr>
</tbody>
</table>

From [http://www.csc.villanova.edu/~dmatusze/resources/ocaml/ocaml.html](http://www.csc.villanova.edu/~dmatusze/resources/ocaml/ocaml.html)
Library of arbitrary precision integers

```ocaml
# # load "nums.cma";;
# open Num;;
# let rec fact n =
  if n = Int 0 then Int 1
  else mult_num n (fact (pred_num n));;
val fact : Num.num -> Num.num = <fun>
# string_of_num (fact (Int 100));;
- : string =
"933262154439441526816992388562667004907159682643816
2146859296389521759999322991560894146397615651828625
36979208272237582511852109168640000000000000000000000"
```

Library of arbitrary precision integers

```ocaml
Float operators

<table>
<thead>
<tr>
<th>Function</th>
<th>Examples</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>** : float -&gt; float -&gt; float</td>
<td>15.5 ** 2.0</td>
<td>(Infix, right associative) Exponentiation.</td>
</tr>
<tr>
<td>sqrt : float -&gt; float</td>
<td>sqrt 8.0</td>
<td>(Prefix) Square root.</td>
</tr>
<tr>
<td>-. : float -&gt; float</td>
<td>-1.10</td>
<td>(Prefix) Unary negation.</td>
</tr>
<tr>
<td>* : float -&gt; float -&gt; float</td>
<td>3.1416 * .5 * .5</td>
<td>(Infix, left associative) Multiplication; operands and result are all real numbers.</td>
</tr>
<tr>
<td>/ : float -&gt; float</td>
<td>7.8 / 3.5</td>
<td>(Infix, left associative) Division of real numbers.</td>
</tr>
<tr>
<td>+ : float -&gt; float -&gt; float</td>
<td>score + .10</td>
<td>(Infix, left associative) Addition of real numbers.</td>
</tr>
<tr>
<td>-. : float -&gt; float -&gt; float</td>
<td>score -. .10</td>
<td>(Infix, left associative) Subtraction of real numbers.</td>
</tr>
<tr>
<td>** : float -&gt; float -&gt; float</td>
<td>15.5 ** .2</td>
<td>(Infix, right associative) Exponentiation.</td>
</tr>
<tr>
<td>sqrt : float -&gt; float</td>
<td>sqrt 8.0</td>
<td>Square root.</td>
</tr>
<tr>
<td>ceil : float -&gt; float</td>
<td>ceil 9.5</td>
<td>Round up to nearest integer (but result is still a real number).</td>
</tr>
<tr>
<td>floor : float -&gt; float</td>
<td>floor 9.5</td>
<td>Round down to nearest integer (but result is still a real number).</td>
</tr>
<tr>
<td>exp, log, log10, cce, sin, tan, acos, ... : float -&gt; float</td>
<td>exp 10.0</td>
<td>The usual transcendental functions.</td>
</tr>
</tbody>
</table>
```

Characters and character strings

- **ASCII** characters

  ```ocaml
  # 'a';;
  - : char = 'a'
  # 'é';;
  Error: Syntax error
  ```

- **Unicode character strings**

  ```ocaml
  # "a";;
  - : string = "a"
  # "é";;
  - : string = "é"
  # "ac" ^ "cen" ^ "tué";; (* concatenation *)
  - : string = "accentué"
  ```

Character operators

- To use these functions without typing Char. each time, enter `open Char;;`.

<table>
<thead>
<tr>
<th>Function</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Char.uppercase : char -&gt; char</td>
<td>Given a lowercase letter, returns the corresponding capital letter. Given any other character, returns that same character.</td>
</tr>
<tr>
<td>Char.lowercase : char -&gt; char</td>
<td>Given a capital letter, returns the corresponding lowercase letter. Given any other character, returns that same character.</td>
</tr>
<tr>
<td>Char.escaped : char -&gt; string</td>
<td>Returns a string consisting of the single character. The name refers to the fact that the character may be escaped (quoted).</td>
</tr>
</tbody>
</table>

- The comparison operators `<` `<=` `=` `!=` `>=` `>` can be applied to characters (ASCII/UTF8/UTF16 order, see [http://en.wikipedia.org/wiki/UTF-8](http://en.wikipedia.org/wiki/UTF-8)).

from [http://www.csc.villanova.edu/~dmatusze/resources/ocaml/ocaml.html](http://www.csc.villanova.edu/~dmatusze/resources/ocaml/ocaml.html)

Standard “string” module

```ocaml
# length "0123456789";;
Unbound value length
# String.length "0123456789";;
- : int = 10
# open String;;
# length "0123456789";;
- : int = 10
# make 10 'X';;
- : string = "XXXXXXXXXX"
```

String comparison

- The operators `<` `<=` `=` `!=` `>=` `>` can be applied to strings for alphabetic comparisons.

```ocaml
# "ab" < "abc";;
- : bool =
# "abc" <= "b";;
- : bool = true
```

Handling errors by exceptions

- Invalid calls to module functions raise exceptions.

```ocaml
# open String;;
# let s = make 10 'X';;
val s : string = "XXXXXXXXXX"
# get s 0;; (* indexing starts at 0 *)
- : char = 'X'
# get s 10;; (* indexing ends at 9 *)
Exception: Invalid_argument "index out of bounds".
# try get s (-1) with
   Invalid_argument(m) -> 'e';;
- : char = 'e'
# Invalid_argument("error");
- : exn = Invalid_argument "error"
```

- The exception (with argument) can be left uncaught and will appear at the top level or can be caught and handled by the program (try ... with ...).
Example of module specification: String

http://caml.inria.fr/pub/docs/manual-ocaml/libref/String.html

String

val length : string -> int

Return the length (number of characters) of the given string.

val get : string -> int -> char

String.get s n returns character number n in string s. You can also write s.[n] instead of String.get s n. Raise Invalid_argument if n not a valid character number in s.

val set : string -> int -> char -> unit

String.set s n c modifies string s in place, replacing the character number n by c. You can also write s.[n] <- c instead of String.set s n c. Raise Invalid_argument if n is not a valid character number in s.

Syntactic sugar

- The OCaml interpreter/compiler can translate string expressions into calls to the String module.

<table>
<thead>
<tr>
<th>Expression</th>
<th>Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>s.[i]</td>
<td>Returns the i-th character of string s.</td>
</tr>
<tr>
<td>s.[i] &lt;- c</td>
<td>Sets the i-th character of string s to c.</td>
</tr>
</tbody>
</table>

Operations of module String

<table>
<thead>
<tr>
<th>Function</th>
<th>Examples</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot; : string * string -&gt; string</td>
<td>&quot;Hello, &quot; &quot; name&quot;</td>
<td>Concatenates the strings of the left with the right argument inserted between each pair.</td>
</tr>
<tr>
<td>String.concat : string * string list -&gt; string</td>
<td>String.concat &quot; &quot; [(&quot;ah&quot;; &quot;e&quot;)</td>
<td>Concatenates the strings of the list with the first argument inserted between each pair.</td>
</tr>
<tr>
<td>String.length : string -&gt; int</td>
<td>String.length &quot;Hello&quot;</td>
<td>Returns the number of characters in string.</td>
</tr>
<tr>
<td>String.get : string -&gt; int -&gt; char</td>
<td>String.get &quot;Hello&quot; 0</td>
<td>Returns a character of the string, counting from 0.</td>
</tr>
<tr>
<td>String.set : string -&gt; int -&gt; char -&gt; unit</td>
<td>String.set name s 'a'</td>
<td>Modifies the original string by changing the character at the given location.</td>
</tr>
<tr>
<td>String.index : string -&gt; char -&gt; int</td>
<td>String.index &quot;adar&quot; 'a'</td>
<td>Returns the position of the first occurrence of the char in the string.</td>
</tr>
<tr>
<td>String.rindex : string -&gt; char -&gt; int</td>
<td>String.rindex &quot;radar&quot; 'a'</td>
<td>Returns the position of the last occurrence of the char in the string.</td>
</tr>
<tr>
<td>String.contains : string -&gt; char -&gt; bool</td>
<td>String.contains &quot;adar&quot; 'a'</td>
<td>Tests whether the char occurs in the string.</td>
</tr>
<tr>
<td>String.sub : string -&gt; int -&gt; int -&gt; string</td>
<td>String.sub &quot;ahd&quot;ifg&quot; p n</td>
<td>Returns a substring of length p starting at position p.</td>
</tr>
<tr>
<td>String.make : int -&gt; char -&gt; bool</td>
<td>String.make n c</td>
<td>Returns a string consisting of n copies of character c.</td>
</tr>
<tr>
<td>String.uppercase : string -&gt; string</td>
<td>String.uppercase &quot;OCCam&quot;</td>
<td>Returns a copy of the string with all letters translated to uppercase.</td>
</tr>
<tr>
<td>String.lowercase : string -&gt; string</td>
<td>String.lowercase &quot;OCCam&quot;</td>
<td>Returns a copy of the string with all letters translated to lowercase.</td>
</tr>
<tr>
<td>String.capitalize : string -&gt; string</td>
<td>String.capitalize &quot;OCCam&quot;</td>
<td>Returns a copy of the string with the first letter translated to uppercase.</td>
</tr>
<tr>
<td>String.uncapitalize : string -&gt; string</td>
<td>String.uncapitalize &quot;OCCam&quot;</td>
<td>Returns a copy of the string with the first letter translated to lowercase.</td>
</tr>
</tbody>
</table>

Logical (=) and physical (==) equality

# 1 = 2 - 1;;
- : bool = true
# 1 == 2 - 1;;
- : bool = true
# 1./.2. = 3./.2. -. 1.;;
- : bool = true
# 1./.2. == 3./.2. -. 1.;;
- : bool = false

# let f x = 1;;
val f : 'a -> int = <fun>
# let g x = 1;;
val g : 'a -> int = <fun>
# f = g;;
Exception: Invalid_argument "equal: functional value".
# f = f;;
- : bool = true
# f = g;;
- : bool = false
# 

avoid == which is machine dependent!
Coercion functions

```ocaml
# string_of_int 12345;;
- : string = "12345"
# int_of_string "-12345";;
- : int = -12345
# bool_of_string "true";;
- : bool = true
# float_of_string "1000000000000000000000";;
- : float = 1e+21
# float_of_string "0.000000000000000000000000000000000000000000001";;
- : float = 1e-45
```

Coersions

<table>
<thead>
<tr>
<th>Function</th>
<th>Examples</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>float : int -&gt; float</code></td>
<td><code>float 5, float 5</code></td>
<td>Convert integer to real.</td>
</tr>
<tr>
<td><code>truncate : float -&gt; int</code></td>
<td><code>truncate average</code></td>
<td>Fractional part is discarded.</td>
</tr>
<tr>
<td><code>int_of_char : char -&gt; int</code></td>
<td><code>int_of_char 'a'</code></td>
<td>ASCII value of character.</td>
</tr>
<tr>
<td><code>char_of_int : int -&gt; char</code></td>
<td><code>char_of_int 97</code></td>
<td>Character corresponding to ASCII value; argument must be in range 0..255.</td>
</tr>
<tr>
<td><code>int_of_string : string -&gt; int</code></td>
<td><code>int_of_string &quot;54&quot;</code></td>
<td>Convert string to integer.</td>
</tr>
<tr>
<td><code>string_of_int : int -&gt; string</code></td>
<td><code>string_of_int 54</code></td>
<td>Convert integer to string.</td>
</tr>
<tr>
<td><code>float_of_string : string -&gt; float</code></td>
<td><code>float_of_string &quot;3.78&quot;</code></td>
<td>Convert float to string.</td>
</tr>
<tr>
<td><code>string_of_float ; float -&gt; string</code></td>
<td><code>string_of_float 3.78</code></td>
<td>Convert string to float.</td>
</tr>
<tr>
<td><code>bool_of_string : string -&gt; bool</code></td>
<td><code>bool_of_string &quot;true&quot;</code></td>
<td>Convert string to bool.</td>
</tr>
<tr>
<td><code>string_of_bool : bool -&gt; string</code></td>
<td><code>string_of_bool true</code></td>
<td>Convert bool to string.</td>
</tr>
</tbody>
</table>

from http://www.csc.villanova.edu/~dmatusze/resources/ocaml/ocaml.html

Variant types

```ocaml
# type number = Int of int | Float of float | Error;;
# type sign = Positive | Negative;;
# let sign_int n = if n >= 0 then Positive else Negative;;
# let add_num n1 n2 =
  match (n1, n2) with
  (Int i1, Int i2) ->
    (* Check for overflow of integer addition *)
    if sign_int i1 = sign_int i2 && sign_int(i1 + i2) <> sign_int
      then Float(float i1 +. float i2)
    else Int(i1 + i2)
  | (Int i1, Float f2) -> Float(float i1 +. f2)
  | (Float f1, Int i2) -> Float(f1 +. float i2)
  | (Float f1, Float f2) -> Float(f1 +. f2)
  | (Error, _) -> Error
  | (_, Error) -> Error;;
# add_num (Int 123) (Float 3.14159) ;;
# add_num (Int 123) (Float 3.14159) ;;
```

Basic input/output in OCaml
Writing the standard output

```ocaml
# print_string "abcdef";;
abcdef- : unit = ()
# print_endline "abcdef";;
abcdef- : unit = ()
# print_newline ();;
- : unit = ()
# print_char 'A';;
A- : unit = ()
# print_int 12345;;
12345- : unit = ()
# print_float 1.2345;;
1.2345- : unit = ()
```

Reading the standard input

```ocaml
# read_line ();;
12345 67 8 9
- : string = "12345 67 8 9"
# read_int ();;
-12345
- : int = -12345
# read_int ();;
9999999999999999999999999999999
Exception: Failure "int_of_string".
# read_float ();;
9999999999999999999999999999999
- : float = 1e+31
```

Writing a file

```bash
~ % ocaml
OCaml version 4.01.0
```

```ocaml
# let file = open_out "file.txt";;
val file : out_channel = <abstr>
# output_string file "abc\n";;
- : unit = ()
# output_string file "cde\n";;
- : unit = ()
# ^D
```

```bash
~ % cat file.txt
abc
cde
```

Reading a file

```bash
~ % cat file.txt
abc
cde
```

```ocaml
# let file = open_in "file.txt";;
val file : in_channel = <abstr>
# input_line file;;
- : string = "abc"
# input_line file;;
- : string = "cde"
# input_line file;;
Exception: End_of_file.
# ^D
```

```bash
~ %
```
Advanced input/output

Arg
Parsing of command line arguments.
Filename
Operations on file names.
Format
Pretty printing.
Genlex
A generic lexical analyzer.
Lexing
The run-time library for lexers generated by ocamllex.
Marshal
Marshaling of data structures.
Parsing
The run-time library for parsers generated by ocamlyacc.
Printexc
Facilities for printing exceptions.
Printf
Formatted output functions.
Scanf
Formatted input functions.

Str
Regular expressions and high-level string processing
Stream
Streams and parsers.
String
String operations.
StringLabels
String operations (with labeled and optional arguments)

Global variables

% ocaml
OCaml version 4.01.0

# x;;
Error: Unbound value x
# let x = 555;;
val x : int = 555
# x * x;;
- : int = 308025
# let x = 666;;
val x : int = 666
# x * x;;
- : int = 443556
#

the previous x no longer exists!

Unmutable variables

# x;;
- : int = 666
# let x = 444 in x * x;;
- : int = 197136
# x;;
- : int = 666
#

Local variables

Global variables

% ocaml
OCaml version 4.01.0

# x;;
Error: Unbound value x
# let x = 555;;
val x : int = 555
# x * x;;
- : int = 308025
# let x = 666;;
val x : int = 666
# x * x;;
- : int = 443556
#

the previous x no longer exists!
The End, Thank You