OCaml Functional language

CONCEPTS

Everything is an expression and values are immutable by default; reach for ref or mutable fields only when needed.

opam switch + packages
↓ installs
dune dune-project, dune files
↓ compiles
ocamlfind / ocamlopt native or bytecode
↓ produces
Executable _build/default

Operators differ per type: + is int, +. is float, ^ joins strings, @ joins lists.

VALUES & FUNCTIONS

`let x = 42`
Bind a name to a value

`let x = 1 in x + 1`
Local binding in an expression

`let add a b = a + b`
Define a function

`let sq = fun x → x * x`
Anonymous function

`let rec fact n = if n ⩽ 1 then 1 else n * fact (n - 1)`
Recursive function

`x ▷ f ▷ g`
Pipe a value through functions

`let x = ref 0 in x := !x + 1`
Mutable reference

`if c then a else b`
Conditional expression

TYPES & PATTERN MATCHING

`type t = Red | Green | Blue of int`
Variant type

`type p = { name : string; age : int }`
Record type

`match v with | A → 1 | B n → n`
Match a variant

`| x :: rest → ... | [] → ...`
Match list head and tail

`| Some x → x | None → d`
Match an option

OPTION & RESULT

`Some x / None`
Optional value

`Ok v / Error e`
Result values

LISTS & ARRAYS

`[1; 2; 3]`
List literal (semicolons)

`x :: xs`
Prepend to a list

`List.map f l`
Transform each element

`List.filter p l`
Keep matching elements

`List.fold_left f acc l`
Reduce from the left

STRINGS & I/O

`"ab" ^ "cd"`
Concatenate strings

MODULES

`module M = struct ... end`
Define a module

`file.ml / file.mli`
Implementation and interface

DUNE

`dune init project` *<name>*
Create a project

`dune build`
Build everything

`dune exec` *<name>*
Build and run an executable

`dune test`
Run tests

OPAM & TOOLCHAIN

`opam install` *<pkg>*
Install a package

TOPLEVEL & COMPILER

`utop`
Interactive toplevel with history