OCaml Functional language

START HERE

`let x = 1`
Bind a value

`dune build`
Build the project

`let f x = x + 1`
Define a function

`dune exec ./bin/main.exe`
Run the program

`match v with | ...`
Pattern match

`dune test`
Run tests

`type t = A | B of int`
Declare a variant

`utop`
Interactive toplevel

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

CONCEPTS

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

Types are inferred and checked at compile time; a successful build rules out most runtime type errors.

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

`let add5 = add 5`
Partial application

`f @@ g x`
Apply f to the result of g x

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

`let f ~name ?(n = 1) () = ...`
Labelled and optional arguments

`let (a, b) = pair`
Destructure a tuple

`let f = function | 0 → "zero" | _ → "other"`
Function over one matched argument

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

`if c then a else b`
Conditional expression

`for i = 0 to n - 1 do ... done`
For loop

`print_endline "hi"`
Print a line

`Printf.printf "%d %s\n" n s`
Formatted output

`Printf.sprintf "%.2f" x`
Format to a string

TYPES & PATTERN MATCHING

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

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

`{ p with age = 31 }`
Copy a record with a change

`type 'a tree = Leaf | Node of 'a tree * 'a * 'a tree`
Recursive polymorphic 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

`| n when n > 0 → ...`
Guard in a pattern

`let p = { name = "Ann"; age = 30 }`
Build a record

`p.age`
Read a record field

`type t = { mutable n : int }`
Mutable record field

`exception Oops of string`
Declare an exception

`try f () with Not_found → d`
Catch an exception

`raise (Failure "msg")`
Raise an exception

OPTION & RESULT

`Some x / None`
Optional value

`Option.map f o`
Map over a present value

`Option.value o ~default:d`
Unwrap with a default

`Option.bind o f`
Chain optional steps

`Option.is_some o`
Test for presence

`Ok v / Error e`
Result values

`Result.map f r`
Map over Ok

`Result.bind r f`
Chain fallible steps

`let* x = r in ...`
Result/option binding operator

LISTS & ARRAYS

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

`x :: xs`
Prepend to a list

`a @ b`
Concatenate lists

`List.map f l`
Transform each element

`List.filter p l`
Keep matching elements

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

`List.iter f l`
Run for side effects

`List.length l`
Number of elements

`List.rev l`
Reverse a list

`List.mem x l`
Membership test

`List.find_opt p l`
First match as option

`List.assoc_opt k l`
Look up in association list

`List.sort compare l`
Sort a list

`List.exists p l / List.for_all p l`
Any / all match

`[| 1; 2; 3 |]`
Array literal

`a.(i) / a.(i) ← v`
Read / write array element

`Array.make n 0`
Array of n copies

STRINGS & I/O

`"ab" ^ "cd"`
Concatenate strings

`String.length s`
String length

`String.split_on_char ',' s`
Split on a character

`String.concat ", " l`
Join a list of strings

`string_of_int n / int_of_string s`
Convert int and string

`In_channel.with_open_text f In_channel.input_all`
Read a whole file

MODULES

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

`module type S = sig ... end`
Declare a signature

MODULES (cont.)

`open M`
Bring names into scope

`module IntMap = Map.Make (Int)`
Instantiate Map

`Hashtbl.create 16`
Mutable hash table

`Hashtbl.find_opt h k`
Look up a key

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

DUNE

`dune init project <name>`
Create a project

`dune build`
Build everything

`dune build -w`
Rebuild on file changes

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

`dune test`
Run tests

`dune fmt`
Format with ocamlformat

`dune utop <dir>`
Toplevel with the library loaded

OPAM & TOOLCHAIN

`opam init`
One-time setup

`opam install <pkg>`
Install a package

`opam switch create <name> <version>`
New switch with a compiler

`eval $(opam env)`
Load the switch into the shell

`opam install --deps-only .`
Install project dependencies

`opam install ocaml-lsp-server ocamlformat utop`
Editor and dev tools

TOPLEVEL & COMPILER

`utop`
Interactive toplevel with history

`#require "<lib>";;`
Load a findlib library

RECIPES

New project to first run

```
opam switch create . --deps-only
dune init project hello
cd hello
dune exec hello
```