

Supervision 1

Michaelmas Term 2026 · Michael Lee

●●●●● difficulty · ★ optional

WORK DUE A single PDF, 24 hours before the supervision**EXPECTED TIME** 6 hours 5 min**NAMES IN SCOPE**

In this sheet, functions come from one of three places:

1. Defined in-line
2. Defined in the lecture notes (e.g. `member`, `zip`, `unzip`)
3. Defined in the OCaml standard library (e.g. `List.map`)

If your editor complains that a function has been used but has not been defined, please copy in the definition from the notes.

NOTE

Sections 1 and 2 assume you have successfully installed OCaml and Dune, by following the instructions of the installation guide. **If you are unable to install OCaml or Dune, please email me as soon as you can.**

1 Representations

PREREQUISITES **Lecture 1** · TIME 40 min

Good representations make it easy to see why a solution is correct, and easy to maintain when the problem changes. Bad representations, in contrast, tend to result in brittle, bug-prone, and hard-to-maintain solutions.

QUESTION 1

PRACTICAL · 40 MIN

●●●●● BASIC

A hands-on introduction to the OCaml module system.

One solution to the year 2000 bug mentioned in **Lecture 1** involves storing years as two digits, but interpreting them so that 50 means 1950, 0 means 2000 and 49 means 2049.

- (a) Have a look at the `representations/lib` directory, and in particular at the file `two_digit.ml` (see [here](#) for information about code organisation). Execute these commands:

```
cd representations
dune build
```

Shell

You should see an error:

```
File "lib/two_digit.ml", line 1:
Error: The implementation lib/two_digit.ml does not match the interface
lib/.representations.objs/byte/representations__Two_digit.cmi:
The type year is required but not provided
The value make is required but not provided
The value get is required but not provided
The value is_gteq is required but not provided
The value add is required but not provided
```

Shell

Dune is complaining that some types and values are not provided in `two_digit.ml`. We will fix that by answering the following questions.

- (b) The type `year` is required but not provided

Choose a representation for the `year` type that allows years to be stored as two digits, by writing `type year = ?`.

- (c) The value `make` is required but not provided

Write a `make` function. `make` should take an integer and return a `year`, raising an exception if the integer is less than 1950 or greater than 2049. You can declare an exception with `exception Year_out_of_bounds_exception` and raise it with `raise Year_out_of_bounds_exception`.

- (d) The value `get` is required but not provided

Write a `get` function. `get` should take a `year` and return a full four digit integer `get (make 1987) = 1987`.

- (e) The value `is_gteq` is required but not provided

Write an OCaml function `is_gteq` to compare two years, just as `>=` compares integers.

- (f) The value `add` is required but not provided

Write an OCaml function `add : year -> int -> year` to add or subtract a given number of years from another year. Raise an exception if the result falls outside 1950–2049.

- (g) To test your answers, you should write some programs (do not submit these).

Write your answers in `representations/bin/dates.ml`. You can execute `dates.ml` by running `dune exec dates` in the terminal. Alternatively, you can navigate to the `representations/bin` directory, and execute `dune utop` in the terminal.

THE REPL

dune utop opens a REPL (read-eval-print loop), an interactive environment for executing OCaml. Once you've opened the REPL, run `open Representations;;` and `open Two_digit;;`. This imports your code. Remember to end every statement with `;;`. Use `#quit;;` to quit the REPL once you're done.

- (h) Comment on the merits and demerits of the two digit representation. In what situations might the two digit representation be useful?

2 Abstraction barriers

PREREQUISITES **Lecture 1** · TIME 40 min

Concepts change much more slowly than implementation. An abstraction barrier lets you define concepts. Implementers of the abstraction barrier implement these concepts as implementations. Consumers see and use only the concept, not the implementation. This divide allows the implementation to change internally, without impacting consumers that rely only on the concept.

QUESTION 2

PRACTICAL · 10 MIN

● ● ● ● ● BASIC

A hands-on introduction to OCaml signatures.

In OCaml, abstraction barriers are enforced by *signatures*. This question will walk you through the creation of a signature for your `two_digit.ml` file.

- (a) There should be a `representations/lib/two_digit.mli` file, containing:

```
type year
val make : int -> year
val get : year -> int
val is_gteq : year -> year -> bool
val add : year -> int -> year
```

This is known as a *signature*. A signature for an OCaml file tells other files that `two_digit.ml` exports a type `year`, and four functions, `make`, `get`, `is_gteq`, and `add`. This signature is the reason why Dune was complaining earlier, when these functions had been promised, but not actually implemented.

- (b) Extend your earlier tests with a test that tries to manipulate the underlying representation of years. For example, if years are represented as integers, you could try to manipulate the integer by writing

```
1 (* ... *)
2 open Two_digit;;
3 (make 1950) + 1;;
```

OCaml

Does OCaml let you write this expression? What is being abstracted over?

- (c) Replace `type year` in the signature (`two_digit.mli`, not `two_digit.ml`) with

```

1  (*or whatever type you chose to use for the representation*)
2  type year = int

```

Now repeat part (b) and report your results. What is the difference between `type year` and `type year = int`?

- (d) Undo your change to the signature, so that it reads `type year` once more, and check that your tests still compile.

QUESTION 3

PRACTICAL · 20 MIN

● ● ● ● ● BASIC

- (a) Repeat question 1, but with a four digit representation of years. Your code should raise an error for years before 1950 and after 2049.

Create `four_digit.ml` and `four_digit.mli` in `representations/lib`, alongside `two_digit.ml`. `four_digit.mli` should be exactly the same as `two_digit.mli`. `four_digit.ml` should implement the same type and functions using four digits rather than two.

- (b) Modify your test code to use `Four_digit` rather than `Two_digit`. That is, replace `open Two_digit` with `open Four_digit`. Run the tests, as before. Do you expect to have to make any changes? Why, or why not?

QUESTION 4

REFLECTION · 10 MIN

● ● ● ● ● BASIC

Using your experience from questions 2 and 3, what is the main idea behind *abstraction barriers*? Why are they useful?

3 Floating point imprecision

PREREQUISITES Lecture 1 · TIME 10 min

A good starting point for representations is: “the ones covered in A-level maths”. Unfortunately, computers cannot encode all representations, only finite representations. In this section we look at $\mathbb{R}$, which computers *approximate* using floating point numbers.

QUESTION 5

PRACTICAL · 5 MIN

● ● ● ● ● BASIC

Seeing a decimal that binary cannot hold, and watching the error accumulate.

Because computers represent numbers as binary strings, simple decimals such as 0.1 often cannot be represented exactly. Write a function that computes

$$\underbrace{x + x + \cdots + x}_n$$

where x has type `float`. It is essential to use repeated addition rather than multiplication. See what happens when you call it with $n = 1\,000\,000$ and $x = 0.1$.

QUESTION 6

PRACTICAL · 5 MIN

● ● ● ● ● BASIC

Another example of the inaccuracy of floating-point arithmetic takes the golden ratio $\phi = 1.618\dots$ as its starting point:

$$\gamma_0 = \frac{1 + \sqrt{5}}{2} \quad \text{and} \quad \gamma_{n+1} = \frac{1}{\gamma_n - 1}$$

In theory, it is easy to prove that $\gamma_n = \dots = \gamma_1 = \gamma_0$ for all $n > 0$. Write this computation in OCaml and report the value of γ_{50} .

HINT In OCaml, $\sqrt{5}$ is written `sqrt 5.0`.

4 Functional programming basicsPREREQUISITES **Lecture 1** · TIME 30 min

An introduction to functional thinking.

QUESTION 7

CONCEPTUAL · 5 MIN

● ● ● ● ● BASIC

Why is it silly to write an expression of the form `if b then true else false`? What about `if b then false else true`? How about `if b then 5 else 5`?

QUESTION 8

CONCEPTUAL · 10 MIN

● ● ● ● ● BASIC

Briefly discuss the meaning of the terms (pure) *expression*, *value*, *command* and *effect*, using the following examples.

- `true`
 - `57 + 9`
 - `print_string "Hello world!"`
 - `print_float (8.32 *. 3.3)`
-

QUESTION 9

CONCEPTUAL · 10 MIN

● ● ● ● ● BASIC

Which of these is a valid OCaml expression and why? Assume that variable `x` has previously been declared, e.g. with `let x = 1`.

- `if x < 6 then x + 3 else x + 8`
- `if x < 6 then x + 3`
- `if x < 6 then print_string "hello"`

- `if x < 6 then x + 3 else "A"`
- `x + (if x < 6 then 3 else 8)`

HINT Use the OCaml compiler to check your answers

QUESTION 10

CONCEPTUAL · 5 MIN

● ● ● ● ● BASIC

Comment on the similarities and differences between:

- the `if` conditional in OCaml;
- the `if` conditional in Python (to be precise, the `if` statement, not the inline `b if c else d`);
- the `if` conditional in Java, if you have met it;
- the `a ? b : c` ternary operator in Java, if you have met it.

5 Recursion and iteration

PREREQUISITES [Lecture 2](#) · TIME 1 hour 45 min

You may have heard that recursion is elegant and, in contrast, iteration is efficient. These questions are about seeing why that is only half right.

QUESTION 11

PRACTICAL · 5 MIN

● ● ● ● ● BASIC

The triangular numbers are the sequence 0, 1, 3, 6, 10, 15, ... Implement `triangle` in OCaml, so that `triangle n` calculates the n^{th} triangular number, that is:

```
triangle 0 = 0
triangle 1 = 1
triangle 2 = 3
triangle 3 = 6
...
```

QUESTION 12

PRACTICAL · 10 MIN

● ● ● ● ● BASIC

Code an iterative (tail-recursive) version of the efficient power function from [Section 1.6](#).

QUESTION 13

PRACTICAL · 10 MIN

● ● ● ● ● BASIC

Write an iterative version of your answer to the previous question in Python, using a loop. Try to make your answer look similar to your answer to question 12.

QUESTION 14

CONCEPTUAL · 20 MIN

● ● ● ● ● CORE

Using your answer to the previous question, devise a procedure for converting a tail-recursive algorithm into an iterative one. You may assume the base cases and recursive cases of the tail-recursive algorithm can be identified. Your procedure should explain how to treat:

- (a) The arguments to the function (in the previous question, x , n , and acc)
 - (b) The base cases (when $n == 0$. In general, there might be more than one base case)
 - (c) The recursive cases (when $n > 0$. In general, there might be more than one recursive case).
-

QUESTION 15

CONCEPTUAL · 30 MIN

● ● ● ● ● CORE

- (a) Write down a definition for tail recursive functions. Your definition should begin with “a tail recursive function is”
 - (b) Give an example of a non-tail recursive function, and explain why your definition precludes it.
 - (c) Are the following functions tail recursive according to your definition? Explain why, or why not.
 - `let id x = x`
 - `let rec f () = f ()`
 - `let rec f x acc = if x = 0 then 0 else (f (x-1) (acc + x); 0)`
 - `let rec f x acc = if x = 0 then 0 else id (f (x-1) (acc + x))`
 - `let rec f x acc = if x = 0 then 0 else (() ; f (x-1) (acc + x))`
 - (d) If you have previous understanding of how functions are evaluated in a computer (stack frames), can you explain why tail-recursive functions are often more space-efficient than recursive ones?
-

QUESTION 16

OPEN-ENDED · 30 MIN

● ● ● ● ● ADVANCED

Are there recursive functions that cannot be written in a tail recursive fashion? Justify your answer by either demonstrating a function that cannot be written in tail recursive style, or explain why all recursive functions can be written in a tail recursive fashion.

HINT This one is genuinely hard, and I will be very surprised if anyone correctly answers it. For open-ended questions like these, I suggest a strategy: read it, think about it for at least the time indicated above, do something else (finish the sheet, go on a walk, meet friends) while it marinates in the back of your mind, and then come back to it several hours / days later.

6 Efficiency

PREREQUISITES Lecture 2 · TIME 55 min

This section aims to familiarise you with big- O notation and recurrence relations.

QUESTION 17

CONCEPTUAL · 30 MIN

● ● ● ● ● CORE

Consider an arbitrary non-negative integer k . Set n to be 2^k . Use a *recurrence relation* to find a closed form for the recurrence given by $T(1) = 1$ and $T(n) = 2T(n/2) + 1$, and hence a bound on its growth. You should be able to find a tighter bound than $O(n \log n)$. Prove that your closed form is correct for all k using mathematical induction.

QUESTION 18

CONCEPTUAL · 25 MIN

● ● ● ● ● CORE

Let $a_1, \dots, a_k$ be constants with $a_i > 0$, and let $g_1, \dots, g_k$ be functions such that $g_i(n) \geq 0$ for $i = 1, \dots, k$ and all sufficiently large n . Show that if $f(n) = O(a_1 g_1(n) + \dots + a_k g_k(n))$ then $f(n) = O(g_1(n) + \dots + g_k(n))$.

7 Lists

PREREQUISITES Lecture 3 · TIME 1 hour 25 min

Lists are one of the fundamental data structures of functional programming. In this section, you should develop a basic grasp for programming with lists.

QUESTION 19

CONCEPTUAL · 15 MIN

● ● ● ● ● CORE

There are two competing definitions for “a list of type T ”

Def 1. A list of type T is some sequence of elements $a, b, c, \dots$, all of type T .

Def 2. A list of type T is one of the following:

- The empty list, $[]$
- An element of type T , x , and a list of type T , xs , joined together $x :: xs$.

Assume we are given a list of type T .

- According to Def 1, how should we process the elements of the list?
- According to Def 2, how should we process the elements of the list?
- Identify which of the definitions is imperative, appealing to a physical description of how elements might be laid out.
- Identify which of the definitions is functional, appealing to a mathematical description for constructing lists.

QUESTION 20

CONCEPTUAL · 10 MIN

 BASIC

Explain why seeing the following expressions in OCaml code should be a cause for concern:

- `1 :: [2, 3, 4]`
 - `"hello " @ "world"`
 - `xs @ [x]`
 - `[x] @ xs`
 - `List.hd xs + List.length (List.tl xs)`
-

QUESTION 21

PRACTICAL · 10 MIN

 BASIC

Write a recursive and an iterative function to compute the sum of a list's elements. Compare their relative efficiency.

HINT For FoCS, when you see “efficiency”, it is a directive to use big-*O* notation.

QUESTION 22

PRACTICAL · 10 MIN

 BASIC

Write a function to return the last element of a non-empty list. How efficiently can this be done? See if you can come up with two different solutions.

QUESTION 23

PRACTICAL · 10 MIN

 BASIC

Write a function `odds` that keeps the odd-indexed elements of a list, and discards the rest. For example, given `[a; b; c; d]` (whose indices are 0, 1, 2 and 3) it should return `[b; d]`.

HINT Pattern-matching is a very flexible concept.

QUESTION 24

PRACTICAL · 15 MIN

 BASIC

Code a function `tails` to return the list of the tails of its argument. For example, given the input list `[1; 2; 3]` it should return `[[1; 2; 3]; [2; 3]; [3]; []]`.

QUESTION 25

PRACTICAL · 15 MIN

● ● ● ● ● BASIC

- (a) Write a function `sum` that returns the sum of the elements of an integer list
 - (b) Write a function `mul` that returns the product of the elements of an integer list
 - (c) What can you say about your answers to parts a and b? Is there a way of sharing the majority of their implementations? Why might sharing be good?
-