Extra: addressing OCaml pitfalls

(before it drives you crazy …)

Kenjiro Taura

You might have already hate OCaml?

- In my past experiences (and this time), OCaml seems by far the most confusing language for you
- It *partly* stems from its being *functional* (awkward loop/mutable variable syntax), but it largely comes from *syntax* that has nothing to do with it

```
let sum_to n =
  let s = ref 0 in
  for i = 1 to n do
        s := !s + i
  done;
  !s
```

Source of confusions

- function application by juxtaposition
- parentheses are still often necessary in function applications
- *partial applications* turn missing arguments on a function call into confusing type errors
- `f(a, b)` is not an immediate syntax error, but means something else
- different operator for integers and floating point numbers
- delimiters are often `;` not `,`
- delimiting with `,` means something else (tuple), not an immediate syntax error

Function application by juxtaposition

- `f x` instead of `f(x)`
- this alone does not make it that confusing

Parentheses are still often necessary in function applications

- `f x - 1` means $f(x) - 1$, not $f(x - 1)$
- so, you need to write `f (x - 1)` to pass $(x - 1)$ to f
- parentheses are necessary anyway here, but the purpose is different

Partial applications turn missing arguments error into confusing type errors

- OCaml allows multi-parameter functions like

```
let f x y = x + y
```

to be applied to a single argument, like

```
f 3
```

- it means *a function that takes y and returns $3 + y$*
- while being powerful and useful, if you somehow miss an argument to a function, you end up putting a function where you meant to put an ordinary value (like int)

Partial applications turn missing arguments error into confusing type errors

- e.g., if you miss an argument to `g` below,

```
(g 1 2 3 ...) + 5
```

you do not get an error *"missing arguments to g"*

- you instead get a *type error* saying *"a value of function type … appears where `int` is expected"*

$f(a, b)$ is not an immediate syntax error, but means something else

- `f(a, b)` is not a syntax error
- it applies `f` to `(a, b)` and it is a *tuple* consisting of `a` and `b`
- if `f` is defined as a two-parameter function like `let f x y = ...`, then you probably get a *type error* due to passing `(a, b)` to `x` (or the resulting expression being an unintended type because of the missing argument)

Different operator for integers and floating point numbers

- OCaml uses `+` for integers and `+.` for floating point numbers (same for `-`, `*`, `/`, etc.)
- it is partly to make it possible (or simple) to infer types of function parameters from function body
  - e.g., `let f x y = x + y` $\Rightarrow$ `x` and `y` are `int` because `+` applies only for int
- but you often write `x + y` when you in fact need to write `x +. y` etc.

Different operator for integers and floating point numbers

- in languages where the programmer has to declare types of function parameters, like:

```
fn f(x : f64, y : f64) { x + y }
```

this error is caught for the definition of `f`, like *"you apply `+` to floating point numbers"*

- in OCaml, in contrast, the error happens when you *apply* `f` like:

```
f 3.1 4.1
```

Different operator for integers and floating point numbers

and the error message becomes *“you put floating point numbers where integers are expected”*, probably not the error you committed

Delimiters are often semicolon ; not comma ,

- you will later learn sequence data types (lists/arrays) and their syntax is not what you are accustomed to:
  - Python: `[1, 2, 3]`
  - OCaml list: `[1; 2; 3]`
  - OCaml array: `[| 1; 2; 3 |]`
- this alone is not that confusing

Delimiting with , means something else (tuple), not an immediate syntax erro

- in OCaml, `[1, 2, 3]` is not a syntax error
- it is a single-element list whose sole element is `1, 2, 3`, which is a tuple
- remember I talked about `f(x,y)`? `(x, y)` was a tuple
- thus, passing `[1, 2, 3]` to a function that expects a list of integers like `f [1,2,3]` is not a syntax error, but a type error saying *"a value of type* `(int * int * int) list` *appears where* `int list` *is expected"*