Acadia vs. Business

I just wrote the most beautiful code of my life.

They casually handed me an impossible problem. In 48 hours and 200 lines, I solved it.

Academia: My god... This will mean a half-dozen papers, a thesis or two, and a paragraph in every textbook on queuing theory!

Business: You got the program to stop jamming up? Great. While you're fixing stuff, can you get Outlook to sync with our new phones?

https://xkcd.com/664/
Real programmers use Emacs

nano? REAL PROGRAMMERS USE emacs

HEY. REAL PROGRAMMERS USE vim.

WELL, REAL PROGRAMMERS USE ed.

NO, REAL PROGRAMMERS USE cat.

REAL PROGRAMMERS USE A MAGNETIZED NEEDLE AND A STEADY HAND.

EXCUSE ME, BUT REAL PROGRAMMERS USE BUTTERFLIES.

THEY OPEN THEIR HANDS AND LET THE DELICATE WINGS FLAP ONCE.

THE DISTURBANCE RIPPLES OUTWARD, CHANGING THE FLOW OF THE EDDY CURRENTS IN THE UPPER ATMOSPHERE.

WHICH ACT AS LENSES THAT DEFLECT INCOMING COSMIC RAYS, FOCUSING THEM TO STRIKE THE DRIVE PLATTER AND FLIP THE DESIRED BIT.

NICE. 'COUSE, THERE'S AN EMACS COMMAND TO DO THAT. OH YEAH! GOOD OL' C-x M-c M-butterfly...

DAMMIT, EMACS.

https://xkcd.com/378/
Quiz: Write the following expressions with parentheses added to indicate how they will be evaluated by Ocaml. You can assume the identifiers have been bound to appropriate values.


- \( a + f x \)
- \( f x \times a \)
- \( a::f x \)
- \( f x::a \)
Quiz: Suppose x and y are bound to a Boolean value. Without using Ocaml's built-in OR operation, which is infix ||, write an expression using match whose value is the logical OR of the two values (i.e. is true if one or both of x and y is true). Be as brief as you can.

```
match x, y with
  | true, _ -> true
  | _, true -> true
  | _ -> false
```
Without using Ocaml's built-in OR operation, write a procedure `my_or` that takes a pair of Booleans and returns their OR.

Also write the type signature.

(* bool*bool -> bool *)

```ocaml
let my_or = fun x,y ->
  match x,y with
  | true, _ -> true
  | _, true -> true
  | _ -> false
```

We will require as part of design recipe that Ocaml code include a type signature, either in a comment or in the definition itself.

Also write the type signature.

(* bool*bool -> bool *)

```ocaml
let my_or = function
  true, _ -> true
  | _, true -> true
  | _ -> false
```

```
let my_or: bool*bool -> bool = function
  true, _ -> true
  | _, true -> true
  | _ -> false
```
The `my_or` procedure takes two arguments. Now let's do a curried version. It takes one argument and returns a procedure that takes one argument and returns the or.

```
let curried_or = fun x -> fun y -> match x,y with
   true, _   -> true
  | _, true  -> true
  | _       -> false;;
```

Here's how we can use it.

```
# (curried_or true) false;;
- : bool = true
```

Don't need the parens—leftmost wins.

```
# curried_or true false;;
```
Quiz: Write a pattern that matches a triple of two-element lists.

Answer: [a ; b] , [c, d] , [e , f]

Can use any identifiers but beware of shadowing existing variables. Cannot use same identifier twice in same pattern.

Answer: [ _ ; _ ] , [ _ ; _ ] , [ _ ; _ ]
Quiz: Find and correct three syntax errors and one likely programming bug.

type suit_length = Short | Regular | Long | X_Long;;
type color = Red | Blue | Yellow;;
type mensware = Jacket of int * suit_length | Pants of int * int | Necktie of color ;;
type inventory = mensware list;;
let inv : inventory = [Jacket (36, Short) ; Jacket (42, Long); Necktie Blue; Pants (29, 29); Necktie Red];;;
let mine = function
  Jacket (36, short) -> true
  | Pants (29, 29) -> true
  | Necktie _ -> true
  | _ -> false
Parameterized types and polymorphic procedures and type variables

list is not a type; it is analogous to a mathematical function that
takes a type argument and returns a type. Some types are:
• int list, which is the type consisting of lists of ints.
• (int*int) list, which is the type consisting of lists of pairs of
  ints, and
• int list list, which is the type consisting of lists of lists
• of ints.

Recall the length procedure:
let rec length = function
  [] -> 0
| _::rest -> 1 + length rest

This single procedure can be applied to all kinds of lists.
In Scheme, length could be applied to arbitrary lists.
How does it work in Ocaml? length is a polymorphic procedure.