Attendance Policy Reminder: All students MUST attend at least 2 of the first 3 classes. If you missed the first week, you will be unenrolled from the class. Please see your graduate advisor immediately to adjust your schedule!

Multiple Choice: What does the following OCaml function do?

```ocaml
let foo x y = (match x with y -> "yes");;
```

(A) returns “yes” only when x=y
(B) compiles with an “inexhaustive match” warning
(C) always returns “yes”
(D) both A and B
Currying

- **Def:** A function is **curried** if none of its arguments has a tuple type.
  - Curried functions have types of the form $\tau_1 \rightarrow \tau_2 \rightarrow \ldots \rightarrow \tau_n$
  - The arrow type operator is right-associative, so whenever we write the above, it means $\tau_1 \rightarrow (\tau_2 \rightarrow (\ldots \rightarrow \tau_n)\ldots)$
  - Function application is left-associative, so $(\text{func } a_1 a_2 \ldots a_n)$ is short for $(((\text{func } a_1) a_2) \ldots a_n)$

- **Def:** To **curry** a function means to convert any tuple arguments into arrow arguments
  - Exercise: Curry the function “let add (x,y) = x+y;;”
  - Solution: let add x y = x+y;;
  - Another solution: let add = fun x -> fun y -> x+y;;
Partial Evaluation

- **Def:** To *partially evaluate* a (curried) function means to apply the function to some of its arguments but not to the rest
  - Example function: let add x y = x+y;;
  - Partially evaluated: (add 3)
  - Fully evaluated: (add 3 4)
- Partially evaluating a function of type $\tau_1 \to \tau_2 \to \ldots \to \tau_n$ always yields a new function of type $\tau_i \to \tau_{i+1} \to \ldots \to \tau_n$ (for some $i\in 2..n$)