1. Programming languages
   a. Describe the difference between OCaml modules and Java classes.
   b. Describe the difference between strong and weak typing.
   c. Explain how call-by-name simplifies implementing lazy evaluation.
   d. Describe the difference between an L-value and an R-value.
   e. What is an activation record, and why is it usually allocated on a stack?
   f. Describe short circuiting.

2. Function arguments
   For each code, explain whether g is an upward or downward funarg.
   a. let f x = let g y = x + y in let app a b = a b in app g 1 ;;
   b. let f x = let g y = x + y in g ;;

3. Static vs. Dynamic Scoping
   Consider the following OCaml code.
   let a = 1 ;;
   let f = fun ( ) -> a ;;
   let a = 2 ;;
   f ( );;
   a. What value is returned by the invocation of f( ) with static scoping? Explain.
   b. What value is returned by the invocation of f( ) with dynamic scoping? Explain.

   Consider the following OCaml code.
   let app f w = let x = 1 in f w ;;
   let add x y = let incr z = z+x in app incr y;;
   (add 2 3) ;;
   c. What is the order of invocation for the functions app, add, and incr when evaluating the expression (add 2 3)?
   d. What value is returned by (add 2 3) with static scoping? Explain.
   e. What value is returned by by (add 2 3) with dynamic scoping? Explain.
4. Parameter passing
Consider the following C code.

```c
int i = 2;
void foo(int f, int g) {
    f = f-i;
    g = f;
}
int main( ) {
    int a[] = {2, 0, 1};
    foo(i, a[i]);
    printf("%d %d %d %d\n", i, a[0], a[1], a[2]);
}
```

a. Give the output if C uses call-by-value
b. Give the output if C uses call-by-reference
c. Give the output if C uses call-by-name

5. Lazy evaluation
Given the following OCaml code.

```ocaml
let doIf p x = if p then x else 0 ;;
let rec loop n = loop n ;;
doIf false (loop 0) ;;
```

a. What is the result of evaluating the doIf expression if OCaml uses call-by-value?
b. What is the result of evaluating the doIf expression if OCaml uses call-by-name?
c. Rewrite the code (using thunks) so that the result of evaluating the doIf expression is the same as if OCaml used call-by-name, even though OCaml uses call-by-value.

6. Tail recursion
For each OCaml function, explain why it is or is not tail recursive
a. let rec foo x = 1 + (foo x)
b. let rec sum l = match l with
   [] -> 0
   l(x::xs) -> x + (sum xs)
c. let rec last = function
   [x] -> x
   l(_:xs) -> last xs
d. let rec fib x =
   if (x = 0) then 0
   else if (x = 1) then 1
   else (fib (x-1) + fib (x-2))