1. Programming languages
   a. Describe how functional programming concepts may be used to implement objects.
   b. Describe the difference between OCaml modules and Java classes.
   c. Describe the difference between strong and weak typing.
   d. Explain how call-by-name simplifies implementing lazy evaluation.
   e. Describe the difference between an L-value and an R-value.

2. Static vs. Dynamic Scoping
   Consider the following OCaml code.
   ```ocaml
   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.
   ```ocaml
   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 (add 2 3) with dynamic scoping? Explain.
3. 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

4. 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.