1. OCaml and Functional Programming
   a. Define functional programming
   b. Define imperative programming
   c. Define iterative programming.
   d. Define higher-order functions
   e. Describe the relationship between type inference and static types
   f. Describe the properties of OCaml lists
   g. Describe the properties of OCaml tuples
   h. Define pattern variables in OCaml
   i. Describe the usage of “_” in OCaml
   j. Describe polymorphism
   k. Write a polymorphic OCaml function
   l. Describe variable binding
   m. Describe scope
   n. Describe lexical scoping
   o. Describe dynamic scoping
   p. Describe environment
   q. Describe closure
   r. Describe currying

2. OCaml Types & Type Inference
   Give the type of the following OCaml expressions:
   a. []
   b. 1::[]
   c. 1::2::[]
   d. [1;2;3]
   e. [[1];[1]]
   f. (1)
   g. (1,”bar”)
   h. ([1,2], [“foo”,”bar”])
   i. [(1,2,”foo”);(3,4,”bar”)]
   j. let f x = 1
   k. let f (x) = x *. 3.14
   l. let f (x,y) = x
   m. let f (x,y) = x+y
   n. let f (x,y) = (x,y)
   o. let f (x,y) = [x,y]
   p. let f x y = 1
   q. let f x y = x*y
   r. let f x y = x::y
   s. let f x = match x with [] -> 1
   t. let f x = match x with (y,z) -> y+z
   u. let f (x::_) = x
   v. let f (_,:y) = y
w. let f (x::y::_) = x+y
x. let f = fun x -> x + 1
y. let rec x = fun y -> x y
z. let rec f x = if (x = 0) then 1 else 1+f (x-1)
aa. let f x y z = x+y+z in f 1 2 3
bb. let f x y z = x+y+z in f 1 2
c. let f x y z = x+y+z in f
dd. let rec f x = match x with
    [] -> 0
    | (_,::t) -> 1 + f t
e. let rec f x = match x with
    [] -> 0
    | (h::t) -> h + f t
ff. let rec f = function
    [] -> 0
    | (h::t) -> h + (2*(f t))
gg. let rec func (f, l1, l2) = match l1 with
    [] -> []
    | (h1::t1) -> match l2 with
        [] -> [f h1]
        | (h2::t2) -> [f h1; f h2]

3. OCaml Types & Type Inference

Write an OCaml expression with the following types:
a. int list
b. int * int
c. int -> int
d. int * int -> int
e. int -> int -> int
f. int -> int list -> int list
g. int list list -> int list
h. 'a -> 'a
i. 'a * 'b -> 'a
j. 'a -> 'b -> 'a
k. 'a -> 'b -> 'b
l. 'a list * 'b list -> ('a * 'b) list
m. int -> (int -> int)
n. (int -> int) -> int
o. (int -> int) -> (int -> int) -> int
p. ('a -> 'b) * ('c * 'c -> 'a) * 'c -> 'b
4. OCaml Programs

What is the value of the following OCaml expressions? If an error exists, describe the error.

a. 2 ; 3
b. 2 ; 3 + 4
c. (2 ; 3) + 4
d. if 1<2 then 3 else 4
e. let x = 1 in 2
f. let x = 1 in x+1
g. let x = 1 in x ; x+1
h. let x = (1, 2) in x ; x+1
i. (let x = (1, 2) in x) ; x+1
j. let x = 1 in let y = x in y
k. let x = 1 let y = 2 in x+y
l. let x = 1 in let x = x+1 in let x = x+1 in x
m. let x = x in let x = x+1 in let x = x+1 in x
n. let rec x y = x in 1
o. let rec x y = y in 1
p. let rec x y = y in x 1
q. let x y = fun z -> z+1 in x
r. let x y = fun z -> z+1 in x 1
s. let x y = fun z -> z+1 in x 1 1
t. let x y = fun z -> x+1 in x 1
u. let rec x y = fun z -> x+1 in x 1
v. let rec x y = fun z -> x+y in x 1
w. let rec x y = fun z -> x y in x 1
x. let rec x y = fun z -> x z in x 1
y. let x y = y 1 in 1
z. let x y = y 1 in x
aa. let x y = y 1 in x 1
bb. let x y = y 1 in x fun z -> z + 1
cc. let x y = y 1 in x (fun z -> z + 1)
dd. let a = 1 in let f x y z = x+y+z+a in f 1 2 3
e. let a = 1 in let f x y z = x+y+z+a in f 1 2 -3
5. OCaml Programming
   a. Write an OCaml function named \texttt{fib} that takes an int \( x \), and returns the Fibonacci number for \( x \). Recall that \( \text{fib}(0) = 0 \), \( \text{fib}(1) = 1 \), \( \text{fib}(2) = 1 \), \( \text{fib}(3) = 2 \).
   b. Write a function \texttt{find_suffixes} which applied to a list \( \text{lst} \) returns a list of all the suffixes of \( \text{lst} \). For instance, suffixes \([1;2;5]\) = \([ [1;2;5] ; [2;5] ; [5] ]\)
   c. Write an OCaml function named \texttt{map_odd} which takes a function \( f \) and a list \( \text{lst} \), applies the function to every other element of the list, starting with the first element, and returns the result in a new list.
   d. Use \texttt{map_odd} and \texttt{fib} applied to the list \([1;2;3;4;5;6;7]\) to calculate the Fibonacci numbers for 1, 3, 5, and 7.
   e. Using \texttt{map}, write a function \texttt{triple} which applied to a list of ints \( \text{lst} \) returns a list with all elements of \( \text{lst} \) tripled in value.
   f. Using \texttt{fold}, write a function \texttt{all_true} which applied to a list of booleans \( \text{lst} \) returns true only if all elements of \( \text{lst} \) are true.
   g. Using \texttt{fold} and anonymous helper functions, write a function \texttt{product} which applied to a list of ints \( \text{lst} \) returns the product of all the elements in \( \text{lst} \).
   h. Using \texttt{fold} and anonymous helper functions, write a function \texttt{find_min} which applied to a list of ints \( \text{lst} \) returns the smallest element in \( \text{lst} \).
   i. Using the \texttt{fold} function and anonymous helper functions, write a function \texttt{count_vote} which applied to a list of booleans \( \text{lst} \) returns a tuple \((x,y)\) where \( x \) is the number of true elements and \( y \) is the number of false elements.
   j. Using the function \texttt{count_vote}, write a function \texttt{majority} which applied to a list of booleans \( \text{lst} \) returns true if 1/2 or more elements of \( \text{lst} \) are true.