Total: 15 points.

1. (2 points) For the following examples, identify for every variable occurence, the declaration that binds it using lexical scoping as done in lectures. You can draw arrows from the variable to the corresponding let binding in your answer.

```
# Example 1
let a = 5 in
let b = 5*a in
let a = a + b in
a + b end end end

# Example 2
let a = 1 in (
let a = 2 in
let b = 3 in
a + b end end;
a
) end
```

2. (2 points) Explain the behaviour of the following program assuming (a) lexical scoping (b) dynamic scoping.

```
letmut a = 5;

fun f(x) = (
    a ← a + 3;
    return a + x
)

letmut a = 3 in print(f(1)) end

print(a);
```

3. ($2\frac{1}{2}$ points) Write down the token sequence in the following Python/C/Java expressions.

   1. a >= b
   2. a > = b
   3. a −> b
   4. a < −b
   5. a <− b

4. ($2\frac{1}{2}$ points) Draw the AST for the following expressions assuming all operators have the (a) same precedence and left-associativity (b) natural precedence and natural associativity.

   Write down the precedence table and associativity for all operators for part (b). The operator ^ is the exponentiation operator.

   1. 2+3∗5
   2. 2^3^5
   3. 2+(3∗5)
   4. −2∗3^5
   5. −2∗−3^5

5. (3 points) Draw the AST assuming "if" expression has (a) higher precedence than infix operators and (b) lower precedence than infix operators. The grammar for expressions is given by:

```
expr := expr "+" expr
      | expr ">" expr
      | number
      | variable
      | if_expr

if_expr :=
  "if" expr "then" expr "else" expr
```

   1. if b > 0 then 3 else 4 + 5
   2. if b > 0 then 3 else (4 + 5)
   3. if b > 0 then 3 + 4 else 5

6. (2 points) Show that the following grammar is ambiguous by writing code that can generate two different syntax trees in this grammar. Which tree is the correct one? Why?

```
if_expr :=
  "if" expr "then" expr "else" expr
| "if" expr "then" expr

expr := expr "+" expr | number | if_expr
```

7. (1 point) For the following expression, write down the output of lexer, parser, and evaluator.

```
letmut a = 1 in (
  while a ≤ 10 do
    a ← a + 1;
    a ← a + if a % 2 = 0 then a/2 else a
  done;
  a
) end
```