Problem 1: Write a procedure called `abs` that takes in a number, and computes the absolute value of the number. It should do this by finding the square root of the square of the argument. (Note: you should use the `Math/sqrt` procedure built in to Clojure, which returns the square root of a number.)

Answer 1:

```
(defn abs [x] (Math/sqrt(* x x)))
```

Problem 2: In both of the following procedure definitions, there are one or more errors of some kind. Explain what's wrong and why, and fix it:

```
(defn take-square
  (* x x))

(defn sum-of-squares [(take-square x) (take-square y)]
  (+ (take-square x) (take-square y)))
```

Answer 2:

```
(defn take-square [x] (* x x))
```

The problem with the code given in the problem is that there no argument/parameter call. `Clojure` does not know what is referred by `x`.

```
(defn sum-of-squares [x y] (+ (take-square x) (take-square y)))
```

The problem with the code given in the problem is that there is a function call where the argument parameter should be instantiated. `Clojure` needs to know what `x` and `y` are.

Problem 3: The expression `(+ 11 2)` has the value `13`. Write four other different Clojure expressions whose values are also the number `13`. Using `def` name these expressions `exp-13-1`, `exp-13-2`, `exp-13-3`, and `exp-13-4`.

Answer 3:

```
(def exp-13-1 (+ 6 7))
(def exp-13-2 (+ (* 2 6) 1))
(def exp-13-3 (/ (* 13 13) 13))
(def exp-13-4 (/ 65 5))
```

Problem 4: Write a procedure, called `third`, that selects the third element of a list. For example, given the list `'(4 5 6)`, third should return the number `6`.

Answer 4:

```
(defn third [lst] (first (rest (rest lst))))
```

Problem 5: Write a procedure, called `compose`, that takes two one-place functions `f` and `g` as arguments. It should return a new function, the composition of its input functions, which computes `f(g(x))` when passed the argument `x`. For example, the function `Math/sqrt` (built in to Clojure from Java) takes the square root of a number, and the function `Math/abs` (also built in to Clojure) takes the absolute value of a number. If we make these functions Clojure native functions using `fn`, then `((compose Math/sqrt Math/abs) -36)` should return `6`, because the square root of the absolute value of `-36` equals `6`.

```
(defn sqrt [x] (Math/sqrt x))
(defn abs [x] (Math/abs x))
((compose sqrt abs) -36)
```

Answer 5:

```
(defn compose [f g]
  (fn [& arg]
    (f (apply g arg))))
```

Problem 6: Write a procedure `first-two` that takes a list as its argument, returning a two element list containing the first two elements of the argument. For example, given the list `'(4 5 6)`, `first-two` should return `'(4 5)`.

Answer 6:

```
(defn first-two [lst]
  (list (first lst) (first (rest lst))))
```

Problem 7: Write a procedure `remove-second` that takes a list, and returns the same list with the second value removed. For example, given `(list 3 1 4)`, `remove-second` should return `(list 3 4)`

```
(defn remove-second [lst]
  (cons (first lst) (rest (rest lst))))
```

Problem 8: Write a procedure `add-to-end` that takes in two arguments: a list `l` and a value `x`. It should return a new list which is the same as `l`, except that it has `x` as its final element. For example, `(add-to-end (list 5 6 4) 0)` should return `(list 5 6 4 0)`.

Answer 8:

```
(defn add-to-end [lst e]
  (if (empty? lst)
    (list e)
    (cons (first lst) (add-to-end (rest lst) e))))
```

Problem 9: Write a procedure, called `reverse`, that takes in a list, and returns the reverse of the list. For example, if it takes in `'(a b c)`, it will output `'(c b a)`.

Answer 9:

```
(defn reverse [lst]
  (if (empty? lst)
    (list)
    (add-to-end (reverse (rest lst)) (first lst))))
```

Problem 10: Write a procedure, called `count-to-1`, that takes a positive integer `n`, and returns a list of the integers counting down from `n` to `1`. For example, given input `3`, it will return `(list 3 2 1)`.

Answer 10:

```
(defn count-to-1 [n]
  (if (zero? n)
    (list)
    (cons n (count-to-1 (- n 1)))))
```

Problem 11: Write a procedure, called `count-to-n`, that takes a positive integer `n`, and returns a list of the integers from `1` to `n`. For example, given input `3`, it will return `(list 1 2 3)`. Hint: Use the procedures `reverse` and `count-to-1` that you wrote in the previous problems.

Answer 11:

```
(defn count-to-n [n] (reverse (count-to-1 n)))
```

Problem 12: Write a procedure, called `get-max`, that takes a list of numbers, and returns the maximum value.

Answer 12:

```
(defn max-int [lst e]
  (if (empty? lst)
    e
    (if (> (first lst) e)
      (max-int (rest lst) (first lst))
      (max-int (rest lst) e))))
(defn get-max [lst]
  (max-int lst (first lst)))
```

Problem 13: Write a procedure, called `greater-than-five?`, that takes a list of numbers, and replaces each number with `true` if the number is greater than `5`, and `false` otherwise. For example, given input `(list 5 4 7)`, it will return `(list false false true)`. Hint: Use the function `map` that we discussed in class.

Answer 13:

```
(defn greater-than-five? [lst] (map (fn [num] (> num 5)) lst))
```

Problem 14: Write a procedure, called `concat-three`, that takes three sequences (represented as lists), `x`, `y`, and `z`, and returns the concatenation of the three sequences. For example, given the sequences `(list 'a 'b)`, `(list 'b 'c)`, and `(list 'd 'e)`, the procedure should return `(list 'a 'b 'b 'c 'd 'e)`.

Answer 14:

```
(defn concat-two [x y]
  (if (empty? x)
    y
    (cons (first x) (concat-two (rest x) y))))
(defn concat-three [x y z]
  (concat-two (concat-two x y) z))
```

Problem 15: Write a procedure, called `sequence-to-power`, that takes a sequence (represented as a list) `x`, and a positive integer `n`, and returns the sequence $\mathtt{x}^{\mathtt{n}}$. For example, given the sequence `(list 'a 'b)` and the number `3`, the procedure should return `(list 'a 'b 'a 'b 'a 'b)`.

Answer 15:

```
(defn sequence-to-power [lst n]
  (if (zero? n)
    (list)
    (concat-two lst (sequence-to-power lst (- n 1)))))
```

Problem 16: Define L as a language containing a single sequence, $L = a$. Write a procedure `in-L?` that takes a sequence (represented as a list), and decides if it is a member of the language L^*. That is, given a sequence x, the procedure should return `true` if and only if x is a member of L^*, and `false` otherwise.

Answer 16:

```
(defn in-L? [x]
  (if (empty? x)
    true
    (if (= (quote a) (first x))
      (in-L? (rest x))
      false)))
```

Problem 17: Let A and B be languages. We'll use $\text{CONCAT}(A, B)$ to denote the concatenation of A and B, in that order. Find an example of languages A and B such that $\text{CONCAT}(A, B) = \text{CONCAT}(B, A)$.

Answer 17:

$$A = \{a, b\}$$
$$B = \varnothing$$
$$\textsc{concat}(A, B) = \varnothing$$
$$\textsc{concat}(B, A) = \varnothing$$

Problem 18: Let A and B be languages. Find an example of languages A and B such that $\textsc{concat}(A, B)$ does not equal $\textsc{concat}(B, A)$

Answer 18:

$$A = \{a, ab\}$$
$$B = \{bb, b\}$$
$$\textsc{concat}(A, B) = \{abb, ab, abbb\}$$
$$\textsc{concat}(B, A) = \{bba, bbab, ba, bab\}$$

Problem 19: Find an example of a language L such that $L = L^2$, i.e. $L = \textsc{concat}(L, L)$.

Answer 19:

$$L = \varnothing$$
$$\textsc{concat}(L, L) = \varnothing$$

Problem 20: Argue that the intersection of two languages L and L' is always contained in L.

Answer 20: $L \cap L' = \{x \mid x \in L \cap x \in L'\}$. If something is in both L and L', it is definitely in L

Problem 21: Let L_1, L_2, L_3, and L_4 be languages. Argue that the union of Cartesian products $(L_1 \times L_3) \cup (L_2 \times L_4)$ is always contained in the Cartesian product of unions $(L_1 \cup L_2) \times (L_3 \cup L_4)$.

Answer 21:

$$(x, y) \in (L_1 \cup L_2) \times (L_3 \cup L_4)$$
$$(x \in L_1 \vee x \in L_2) \wedge (y \in L_3 \vee y \in L_4)$$
$$((x \in L_1 \vee x \in L_2) \wedge y \in L_3) \vee ((x \in L_1 \vee x \in L_2) \wedge y \in L_4)$$
$$(x \in L_1 \wedge y \in L_3) \vee (x \in L_2 \wedge y \in L_3) \vee (x \in L_1 \wedge y \in L_4) \vee (x \in L_2 \wedge y \in L_4)$$
$$(x, y) \in (L_1 \times L_3) \cup (L_2 \times L_4) \cup (L_1 \times L_4) \cup (L_2 \times L_3)$$
$$(L_1 \times L_3) \cup (L_2 \times L_4) \in (L_1 \times L_3) \cup (L_2 \times L_4) \cup (L_1 \times L_4) \cup (L_2 \times L_3)$$

Problem 22: Let L and L' be finite languages. Show that the number of elements in the Cartesian product $L \times L'$ is always equal to the number of elements in $L' \times L$.

Answer 22:

$$L \times L' = \{(x, y) \mid x \in L \text{ and } y \in L'\}$$
$$L' \times L = \{(x, y) \mid x \in L' \text{ and } y \in L\}$$
$$|L \times L'| = |L| \cdot |L'|$$
$$|L' \times L| = |L'| \cdot |L|$$
$$\therefore |L \times L'| = |L' \times L|$$

$|L \times L'| = |L| \cdot |L'|$ because for every element in L, there is an element in L' that can be associated with it. Likewise for $|L' \times L| = |L'| \cdot |L|$ for every element in L', there is an element in L that can be associated with it. When multiplying two numbers x and y, $x \cdot y = y \cdot x$. Therefore, $|L \times L'| = |L' \times L|$.

Problem 23: Suppose that the concatenation of a language L is equal to itself: $\texttt{concat}(L, L) = L$. Show that L is either the empty set or an infinite language. (Remember that the empty set contains the null string.)

Answer 23:

$$L^* = \bigcup_{i=0}^{\infty} L^i \qquad \text{(Kleene Star)}$$
$$L^0 = \{\epsilon\}$$
$$\textsc{concat}(L^0, L^0) = \{\epsilon\} = L^0$$
$$\textsc{concat}(L^{\aleph}, L^{\aleph}) = L^{\aleph}$$

Given a language L, the Kleene Star of L, L^* is given above. The empty set is $L^0 = \{\epsilon\}$. If one concatenates L^0 with L^0, or $\{\epsilon\}$ with $\{\epsilon\}$ he or she will still get $\{\epsilon\}$. If one concatenates the countable infinity, namely $L^{\aleph}$ with $L^{\aleph}$, he or she will still obtain $L^{\aleph}$. If i is finite, this would not work because there would be a problem in the order of concatenation.