Question 1

1. Examples in L3:
   - Primitive atomic expression: #t
   - Non-Primitive atomic expression: The variable x
   - Non-Primitive compound expression: (+ 4 3)
   - Primitive atomic value: true
   - Non-Primitive atomic value: 'hello
   - Non-Primitive compound value: list(4 5)
2. A special form is an expression that follows special evaluation rules, its semantics does not follow any existing structure. For example - 'define'.
3. A free variable is a variable that "occurs free" in a function. Meaning it wasn't declared in its scope, but in a different scope(usually before the function).for example:

   ```
   (lambda (x)
        (+ x n))
   ```

   The variable n occurs free.
4. A symbolic expression is a notation for nested list (tree-structured) data. For example:
   (3 4 5)
5. A syntactic abbreviation is a syntax within a programming language that is designed to make things easier to read or to express.
   2 examples are:
   a. `i++` instead of `i = i + 1`
   b. `'(2 . 3)` instead of `(cons 2 3)`

6. L3 is Turing complete - which means that every program can be written in L3, that includes programs that are written in L30 ($L30 \subseteq L3$).
   Besides, the difference between L3 to L30 is the list structure, in L30 we can create a list using pairs. for example:
   L3 - `(list 1 2 3 4)`
   L30 - `'(1 . '(2 . '(3 . '(4 . '() ))))`
   ($L3 \subseteq L30$).

7. The advantage of PrimOp is more clarity - it is easier to understand what are the primitive operations of the language and what are user-defined functions.

   The advantage of Closure is more functionality - the user of the language can define his own primitive operations to replace the default operations of the language. For example, one can change the operator "=" to also print to the screen.
8. Let us use the addition 'R' for the new functions defined in the question.

   We want to prove that for every finite array $a = [a_1, ..., a_n]$,

   $map(a) = mapR(a)$

   mapR performs the same function over the array.

   Let's say that $f(x)$ is the function that map implements on every item in the array. We can see that $map(a) = [f(a_1), ..., f(a_n)]$ and

   $mapR(a) = [f(a_1), ..., f(a_n)]$.

   The same goes for filter: we can observe that

   filter(a) = [$a_i, ...a_j] \, for \ i \leq j \ and \ i,j \leq n$, and also

   filterR(a) = [$a_i, ...a_j] \, for \ i \leq j \ and \ i,j \leq n$

   So the functions are the same.

   For compose and reduce the case is different:

   $f(g(x))$ ≠ g(f(x)) for any functions f,g. Therefore compose and composeR will give different results in some cases.

   Reduce as well will give different results in some cases. For example:

```
(0,2,4).reduce(
     (acc, curr) =>
          {acc = acc + curr;
               if(acc==0){acc++}
                     Return acc;}
,0); //initial value for acc is 0
```

   If applied from left to right the result is 7, but if applied from right to left the result is 6.

Question 2

last-element:

Signature: last-element(list)

Type: <T> (list<T>) => T

Purpose: Given list, return the last element of the list

Pre-conditions: None

Tests: (list 123) => 3

power:

Signature:power(number,number)

Type:(number,number)=>number

Purpose: Given two numbers n1,n2, return $n1^{n2}$

Pre-conditions: isNumber(n1),isNumber(n2)

Tests:(2,4)=>16, (0,3)=>0, (3,0)=>1

sum-lst-power :

Signature: sum-list-power(list<number>,number)

Type: (list<number>,number) => number

Purpose:given a number N, returns the sum of all its elements in the power of N

Pre-conditions: all list elements are numbers

Tests:(sum-lst-power (list 1 4 2) 3) → 1^3+ 4^3 + 2^3 = 73

length:

Signature:length(list)

Type:(list<T>) => number

Purpose: given a list, return it's length

Pre-conditions: None

Tests:(length '(1 2) -> 2

num-from-digits:

Signature:num-from-digits(list<number>)

Type:(list) => number

Purpose: Return number consisted from list digits

Pre-conditions:All list elements are numbers

Tests:(num-from-digits (list 2 4 6)) → 246

Is-narcissistic:

Signature: is-narcissistic(lst<number>)

Type:lst<number> -> boolean]

Purpose: Checks if the number consisted from list digits is narcissistic

Pre-conditions: All list elements are numbers

Tests: (list 1 2 3)=>#f, (list 1 5 3)=>#t