Assignment 3:

Question 1: Theoretical Questions

2.1 Typing Statements:

a. Statement: $\{f : [T2 \rightarrow T3], g : [T1 \rightarrow T2], a : Number\} \vdash (f\ (g\ a)) : T3$.
   $f : [T2 \rightarrow T3]$ means f takes $T2$ as an input and returns a value of type $T3$.
   $g : [T1 \rightarrow T2]$ means f takes $T1$ as an input and returns a value of type $T2$.
   $a : Number$ means a is a value of type Number.
   from $g : [T1 \rightarrow T2]$ and $a : Number$, we can infer $g\ a : T2$.
   From $f : [T2 \rightarrow T3]$ and $g\ a : T2$ we can infer $f\ (g\ a) : T3$.
   Therefore, the Statements is **true**.

b. Statement: $\{f : [T1 \rightarrow [T2 \rightarrow Boolean]], x : T1, y : T2\} \vdash (f\ x\ y) : Boolean.$
   $f : [T1 \rightarrow [T2 \rightarrow Boolean]]$, means that f is a function that takes an argument of type $T1$ and returns another function of type $[T2 \rightarrow Boolean]$.
   The expression $(f\ x\ y)$ means that f is being applied to x and y , which means f takes two parameters and not one (instead of $((fx)y)$).
   Therefore, the Statements is **false**.

c. Statement: $\{f : [T1 \times T2 \rightarrow T3],\ y : T2\} \vdash (lambda\ (x)\ (f\ x\ y)) : [T1 \rightarrow T3]$.
   $f : [T1 \times T2 \rightarrow T3]$ means f takes $(x{:}T1, y{:}T2)$ and returns $T3$.
   $y : T2$ is given.
   The lambda takes $x : T1$ and applies f to the pair $(\ x\ y)$, resulting in $T3$.
   Therefore, the Statements is **true**.

d. Statement: $\{f : [T2 \rightarrow T1], x : T1, y : T3\} \vdash (f\ x) : T1$.
   From the context, $f : [T2 \rightarrow T1]$ and $x : T1$, we can infer $f\ x : T1$.
   Therefore, the Statements is **true**.

2.1 Simplifying TExps:

a. Simplified TExp: never.
   Explanation: Intersection of two disjoint sets is the empty set.
b. Simplified TExp: string.
   Explanation Intersection of any with string is string.
c. f Simplified TExp: any.
   Explanation: Union of any and never is any.
d. Simplified TExp: number.
   Explanation: Difference of a set with a set containing all its elements but number.
e. Simplified TExp: never.
   Explanation: Difference of a set with a set containing all its elements.
f. Simplified TExp: boolean.
   Explanation: Intersection of union of sets that including both boolean.

2.2 Completing the Code Snippet:

```
(define (is_boolean? : (any -> is? boolean))
  (lambda ((x : any)) : is? boolean
    (boolean? x)))


 (define (good_in_L52 : ((union number boolean) -> is? boolean))
  (lambda ((z : (union number boolean))) : is? boolean
    (is_boolean? z)


  )) * [a] is 'is? Boolean'.
    [b] is ' is? boolean'.
    [c] is '(is_boolean? z)'.
```

In L52, the is? boolean type allows the function is_boolean? to be used in a way that returns a value behaving as a boolean without needing it to be a boolean type.

In L51 the type system doesn't support the is? type. Therefore 'is? Boolean' is not recognized, and the type checker expects a boolean types.

In the good_in_L52 function, the return type is (is? boolean), which is determined by the type predicate is_boolean?. Such a function would not pass type checking in L51 because L51 does not support is? type predicates. The return type in L51 must be a specific type (like boolean), not a type predicate.

Explanation: In L51,

2.3 Completing the Return Type:

The answer is: (union boolean (union number string))

Explanation: The return type of `f` depends on the type of `x` and must include all possible return values:

- If x is a number and greater than 0, it returns the string "positive" (string) and if x not greater than 0, it returns the string "negative" (string).
- If x is a boolean, it returns x (boolean).
- If x is not a number and not a boolean (although by the type declaration, it should always be one of these), it returns the number 1.

Given these conditions, the return type of f must be a union of all possible return types, which includes string, boolean, and number. Therefore, the return type of f is (union string (union boolean number)).

The order is '(union boolean (union number string))' and not '(union string (union boolean number))' because in our implementation, the values are sorted by lexicographic order since there is no inclusion relation between them.

In type checking, the typeChecker must determine the type of each expression in the code based on its structure and possible outcomes rather than on actual runtime values. This ensures that the code is of the specified type and consists of all possible scenarios and not just the ones that actually happened (so a number included in the union even though it wouldn't actually be the value).