Assignment 3

Question 1:

a. {f : [T2 → T3], g : [T1 → T2], a : Number} ⊢ (f (g a)) : T3

→True, a Number should by type of T1, g should accept only T1 type and the function g send argument a from a specific type Number which already identified as T1 ,g return T2 which is the type that f accepts.

b. {f : [T1 → [T2 → Boolean]], x : T1, y : T2} ⊢ (f x y) : Boolean

→False, f accepts two arguments and according to the environment it must take one arguments, there are no compatibility of the number of arguments I marginal to the function to what it expects.
c. {f : [T1 × T2 → T3], y : T2} ⊢ (lambda (x) (f x y)): [T1 → T3]

→True ,lambda accepts two arguments x & y ,y is defined as T2 in the environment and x is not defined in the environment so its automatically take a Tvar same Its defined in the function structure so x:T1,so we conclude that the function takes T1 and return T3.

d. {f : [T2 → T1], x : T1, y : T3} ⊢ (f x) : T1

→True , there is no explicit contradiction in the given types that prevents T 1 T1 from being equal to T 2 T2. If we assume that T1≠T2,T1=T2, then the typing statement can be true. Let's re-evaluate statement d with this assumption in mind so we got true.

Question 2.1:

a. (inter number boolean) →never
b. (inter any string) →String
c. (union any never) →any
d. (diff (union number string) string) →Number
e. (diff string (union number string)) →never
f. (inter (union boolean number) (union boolean (diff string never))) →boolean

Question 2.2:

```
;; L52 return type is? boolean
;; L51 return type boolean
(define (isBoolean : (any-> is? boolean))
  (lambda ((x : any)) : is? boolean
    (boolean? x)))
;; Function to complete
(define (good_in_L52 : ((union number boolean)-> is?boolean))//הגדרה
    (lambda ((z : (union number boolean))) : is?boolean//
      (isBoolean(z))
    ))
```

- → in L52 this implementation is valid because it explicitly checks if z is of type boolean, which is allowed by the is? type predicate. But in L51 this implementation would not pass type checking because L5 does not support type predicates like is? or explicit handling of union types.

__Question 2.3:__

```
(define (is_number? : (any -> is? number))
   (lambda ((x : any)) : is? number
      (number? x)))
(define (is_boolean? : (any -> is? boolean))
   (lambda ((x : any)) : is? boolean
      (boolean? x)))
(define f
   (lambda ((x : (union number boolean))) : (union string boolean)
      (if (is_number? x)
         (if (> x 0)
            "positive"
            "negative")
         (if (is_boolean? x)
            x
            1))))
```

- ➔ bool according to the f structure body we have 4 possible returned ("positive" or "negative" or x or 1) so the completion of f with return type (union string boolean) in L52 demonstrates the use of union types and type predicates to handle values that can be of different types (number or boolean) and return corresponding values (string or boolean).