Question 1:

Statement a

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

True: this statement is true because T1 is a generic type, and we can replace it with Number and will work correctly

Statement b

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

True: First, f is applied to x, resulting in a function of type [T2→Boolean]

Next, this resulting function is applied to y, resulting in a value of type Boolean

Therefore, the statement **True**.

Statement c

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

True: f has the type [**T1 × T2 → T3],** meaning f takes a pair consisting of a value of type T1 and a value of type T2 and returns a value of type T3 which is True

Statement d

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

False: The function f takes an argument of type T2 and returns a value of type T1. However**,** x is of type T1**,** not T2. Which means that the statement is True if and only if T1=T2

2.1

a: never

b: string

c: any

d: number

e: never

f: Boolean

2.2

a) Boolean

b) Boolean

c) if (is Boolean z) (z) (#f)

2.3

The return type for “f “ in L52 should be (union string (union Boolean number))

Explanation

The function processes x which can be a number or a Boolean. The return types based on x are:

"positive" or "negative" if x is a number.

The Boolean value x itself if x is a Boolean.

The number 1 in a fallback scenario.

Thus, f returns values of type string, Boolean, or number. Therefore, the complete return type for f is (union string (union Boolean number))