Theoretical Questions:

1.

a. {f : [T2 → T3], g : [T1 → T2], a : Number} ⊢ (f (g a)) : T3 → **true**
Explanation – we apply f that accepts T2 and return T3, the operand of f is the return of the apply of g with a – g gets T1 and return T2, we assume a(which is a number) is T1, thus g return T2 and f return T3, to conclude the application return T3 as required.

b. {f : [T1 → [T2 → Boolean]], x : T1, y : T2} ⊢ (f x y) : Boolean → **false**
Explanation – Notice! There is one parenthesis missing, we only apply f and not the lamda that returns from f(T2 → Boolean) thus, instead of getting Boolean at the end, we returns a closure.

c. {f : [T1 × T2 → T3], y : T2} ⊢ (lambda (x) (f x y)): [T1 → T3] → **true**
Explanation – the lambda expression type is the variables – x which is free at the time, and the last expression in the body of the lambda – which is the appliance of f – T3. In f we send x as T1 so in conclusion the type is T1 → T3.

d. {f : [T2 → T1], x : T1, y : T3} ⊢ (f x) : T1 → **true**
Explanation – we operate f with x – T1 and f expects T2, so we can infare that T1 equals to T2, , then f return T1 as written.

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**

2. 2. [a] → is? Boolean

[b] → is? Boolean

[c] → (isBoolean z)

3. (union string union(Boolean number))