Part 1 : Theoretical Questions

1. Which of the following typing statement is true/false, explain why.
   a. $\{f:[T1 \to T2], g:[T1 \to T2], a:T1\} \vdash \big(f(g\ a)\big):T2$

      False:

      g accepts T1 and returns T2 but f doesn't accept T2 it accepts also T1.

   b. $\{f:[T1 \times T2 \to T3]\} \vdash \big(lambda\ (x)\ (f\ x\ 100)\big):[T2 \to T3]$

      False:

      f accepts T1 X T2 and gets x and 100. 100 is of type T2 but T2 is not necessarily a Number, also x is of type T1 not T2.

   c. $\{f:[T1 \to T2]\} \vdash \Big(\big(lambda\ (x)\ (f\ x)\big)\Big):[T1 \to T2]$

      True:

      f accepts T1 and returns T2, in the same way lambda accepts T1 which is x and returns the value of f which is T2.

   d. $\{f:[T1 \times T2 \to T3], y:T2\} \vdash \big(lambda\ (x)\ (f\ x\ y)\big):[T1 \to T3]$

      True:

      f accepts T1 X T2 and gets x and y, y is of type T2, lambda accepts T1 which is x, so x is of type T1, the lambda returns the value of f which is T3.

2. Perform type inference manually on the following expressions, using the Type Equations method. List all the steps of the procedure.
   a. $\boldsymbol{\Big(\big(lambda\ (f\ x1)\ \big(if\ x1\ (f\ 1\ x1)\ (f\ 3\ x1)\big)\big) + \#t\Big)}$

Stage I: Rename bound variables.

$\Big(\big(lambda\ (f\ x1)\ \big(if\ x1\ (f\ 1\ x1)\ (f\ 3\ x1)\big)\big) + \#t\Big)$ turn to

$\Big(\big(lambda\ (f\ x)\ \big(if\ x\ (f\ 1\ x)\ (f\ 3\ x)\big)\big) + \#t\Big)$

Stage II: Assign type variables for every sub expression:

Expression	Value
$\left(\left(lambda\ (f\ x)\ \left(if\ x\ (f\ 1\ x)\ (f\ 3\ x)\right)\right) + \#t\right)$	T0
$\left(lambda\ (f\ x)\ \left(if\ x\ (f\ 1\ x)\ (f\ 3\ x)\right)\right)$	T1
$\left(if\ x\ (f\ 1\ x)\ (f\ 3\ x)\right)$	T2
$(f\ 1\ x)$	T3
$(f\ 3\ x)$	T4
f	Tf
x	Tx
1	Tnum1
3	Tnum3
+	T+
#t	T#t

Stage III: Construct type equations.

The equations for the sub-expressions are:

Expression	Equation
$\left(\left(lambda\ (f\ x)\ \left(if\ x\ (f\ 1\ x)\ (f\ 3\ x)\right)\right) + \#t\right)$	T1 = [T+ * T#t -> T0]
$\left(lambda\ (f\ x)\ \left(if\ x\ (f\ 1\ x)\ (f\ 3\ x)\right)\right)$	T1 = [Tf * Tx -> T2]
$\left(if\ x\ (f\ 1\ x)\ (f\ 3\ x)\right)$	T2 = T3 T3 = T4
$(f\ 1\ x)$	Tf = [Tnum1 * Tx -> T3]
$(f\ 3\ x)$	Tf = [Tnum3 * Tx -> T4]
1	Tnum1 = Number
3	Tnum3 = Number
+	T+ = [Number * Number -> Number]
#t	T#t = Boolean

Stage IV: Solve the equations.

Equation	Substitution
1. T1 = [T+ * T#t -> T0]	{}
2. T1 = [Tf * Tx -> T2]	
3. T2 = T3	
4. T3 = T4	
5. Tf = [Tnum1 * Tx -> T3]	
6. Tf = [Tnum3 * Tx -> T4]	
7. T+ = [Number * Number -> Number]	

8. T#t = Boolean	
9. Tnum1 = Number	
10. Tnum3 = Number	

Step 1:

(T1 = [T+ * T#t -> T0]) ◦ Substitution = (T1 = [T+ * T#t -> T0])

Equation	Substitution
2. T1 = [Tf * Tx -> T2]	{T1 := [T+ * T#t -> T0]}
3. T2 = T3	
4. T3 = T4	
5. Tf = [Tnum1 * Tx -> T3]	
6. Tf = [Tnum3 * Tx -> T4]	
7. T+ = [Number * Number -> Number]	
8. T#t = Boolean	
9. Tnum1 = Number	
10. Tnum3 = Number	

Step 2:

(T1 = [Tf * Tx -> T2]) ◦ Substitution = (T1 = [T+ * T#t -> T0] = T1 = [Tf * Tx -> T2])

Equation	Substitution
3. T2 = T3	{T1 := [T+ * T#t -> T0]}
4. T3 = T4	
5. Tf = [Tnum1 * Tx -> T3]	
6. Tf = [Tnum3 * Tx -> T4]	
7. T+ = [Number * Number -> Number]	
8. T#t = Boolean	
9. Tnum1 = Number	
10. Tnum3 = Number	
11. Tf = T+	
12. Tx = T#t	
13. T2 = T0	

Step 3:

(T2 = T3) ◦ Substitution = Substitution◦ (T2 = T3)

Equation	Substitution
4. T3 = T4	{T1 := [T+ * T#t -> T0], T2 = T3}
5. Tf = [Tnum1 * Tx -> T3]	

6. Tf = [Tnum3 * Tx -> T4]	
7. T+ = [Number * Number -> Number]	
8. T#t = Boolean	
9. Tnum1 = Number	
10. Tnum3 = Number	
11. Tf = T+	
12. Tx = T#t	
13. T2 = T0	

Step 4:

(T3 = T4) ○ Substitution = Substitution○ (T3 = T4)

Equation	Substitution
5. Tf = [Tnum1 * Tx -> T3]	{T1 := [T+ * T#t -> T0], T2 = T3, T3 = T4}
6. Tf = [Tnum3 * Tx -> T4]	
7. T+ = [Number * Number -> Number]	
8. T#t = Boolean	
9. Tnum1 = Number	
10. Tnum3 = Number	
11. Tf = T+	
12. Tx = T#t	
13. T2 = T0	

Step 5:

(Tf = [Tnum1 * Tx -> T3]) ○ Substitution = Substitution○ (Tf = [Tnum1 * Tx -> T3])

Equation	Substitution
6. Tf = [Tnum3 * Tx -> T4]	{T1 := [T+ * T#t -> T0], T2 = T3, T3 = T4, Tf := [Tnum1 * Tx -> T3]}
7. T+ = [Number * Number -> Number]	
8. T#t = Boolean	
9. Tnum1 = Number	
10. Tnum3 = Number	
11. Tf = T+	
12. Tx = T#t	
13. T2 = T0	

Step 6:

(Tf = [Tnum3 * Tx -> T3]) ○ Substitution =

(Tf = [Tnum3 * Tx -> T3]=Tf = [Tnum1 * Tx -> T3])

Equation	Substitution
7. T+ = [Number * Number -> Number]	{T1 := [T+ * T#t -> T0], T2 = T3, T3 = T4, Tf := [Tnum1 * Tx -> T3], Tnum1=Tnum3}
8. T#t = Boolean	
9. Tnum1 = Number	
10. Tnum3 = Number	
11. Tf = T+	
12. Tx = T#t	
13. T2 = T0	

Step 7:

(T+ = [Number * Number -> Number]) ○ Substitution =

Substitution○ (T+ = [Number * Number -> Number])

Equation	Substitution
8. T#t = Boolean	{T1 := [[Number * Number -> Number] * T#t -> T0], T2 = T3, T3 = T4, Tf := [Tnum1 * Tx -> T3], Tnum1=Tnum3, T+ = [Number * Number -> Number]}
9. Tnum1 = Number	
10. Tnum3 = Number	
11. Tf = T+	
12. Tx = T#t	
13. T2 = T0	

Step 8:

(T#t = Boolean) ○ Substitution = (Boolean = Boolean)

Equation	Substitution
9. Tnum1 = Number	{T1 := [[Number * Number -> Number] * Boolean -> T0], T2 = T3, T3 = T4, Tf := [Tnum1 * Tx -> T3], Tnum1=Tnum3,

	T+ = [Number * Number -> Number], T#t = Boolean }
10. Tnum3 = Number	
11. Tf = T+	
12. Tx = T#t	
13. T2 = T0	

Step 9:

(Tnum1 = Number) ○ Substitution = Substitution○ (Tnum1 = Number)

(Tnum3 = Number) ○ Substitution = Substitution○ (Tnum3 = Number)

Equation	Substitution
11. Tf = T+	{T1 := [[Number * Number -> Number] * Boolean -> T0], T2 = T3, T3 = T4, Tf := [Tnum1 * Tx -> T3], Tnum1=Boolean, Tnum3=Boolean, T+ = [Number * Number -> Number], T#t = Boolean }
12. Tx = T#t	
13. T2 = T0	

Step 10:

(Tf = T+) ○ Substitution = Substitution○(Tf = T+)

Equation	Substitution
12. Tx = T#t	{T1 := [[Number * Number -> Number] * Boolean -> T0], T2 = T3, T3 = T4, Tf := [Tnum1 * Tx -> T3], Tnum1=Boolean, Tnum3=Boolean, T+ = [Number * Number -> Number], T#t = Boolean }
13. T2 = T0	
14. Tx = Number	
15. T3 = Number	

Step 11:

(Tx = T#t) ○ Substitution = Substitution○(Tx = T#t)

Equation	Substitution
13. T2 = T0	{T1 := [[Number * Number -> Number] * Boolean -> T0],

	T2 = T3, T3 = T4, Tf := [Tnum1 * Tx -> T3], Tnum1=Boolean, Tnum3=Boolean, T+ = [Number * Number -> Number], T#t = Boolean , Tx = T#t }
14. Tx = Number	
15. T3 = Number	

Step 12:

(T2 = T0) ○ Substitution = Substitution○(T2 = T0)

Equation	Substitution
14. Tx = Number	{T1 := [[Number * Number -> Number] * Boolean -> T0], T2 = T3, T3 = T4, Tf := [Tnum1 * Tx -> T3], Tnum1=Boolean, Tnum3=Boolean, T+ = [Number * Number -> Number], T#t = Boolean , Tx = T#t , T2 = T0}
15. T3 = Number	

Step 13:

In the substitution we got that Tx = T#t= Boolean but in this step Tx = Number **so we can say that the expression is not well-typed.**

b. $\boldsymbol{\left(\left(lambda\ (f1\ x1\ y1)\ (f1\ x1\ y1)\right) * 1\ 3\right)}$

Stage I: Rename bound variables.

$\left(\left(lambda\ (f1\ x1\ y1)\ (f1\ x1\ y1)\right) * 1\ 3\right)$ turn to

$\left(\left(lambda\ (f\ x\ y)\ (f\ x\ y)\right) * 1\ 3\right)$

Stage II: Assign type variables for every sub expression:

Expression	Value
$((lambda\ (f\ x\ y)\ (f\ x\ y))*1\ 3)$	T0
$(lambda\ (f\ x\ y)\ (f\ x\ y))$	T1
$(f\ x\ y)$	T2
f	Tf
x	Tx
y	Ty
*	T*
1	Tnum1
3	Tnum3

Stage III: Construct type equations.

The equations for the sub-expressions are:

Expression	Equation
$((lambda\ (f\ x\ y)\ (f\ x\ y))*1\ 3)$	T1 = [T* * Tnum1*Tnum3 -> T0]
$(lambda\ (f\ x\ y)\ (f\ x\ y))$	T1 = [Tf * Tx*Ty -> T2]
$(f\ x\ y)$	Tf = [Tx * Ty -> T2]
1	Tnum1 = Number
3	Tnum3 = Number
*	T* = [Number * Number -> Number]

Stage IV: Solve the equations.

Equation	Substitution
1. T1 = [T* * Tnum1*Tnum3 -> T0]	{}
2. T1 = [Tf * Tx*Ty -> T2]	
3. Tf = [Tx * Ty -> T2]	
4. T* = [Number * Number -> Number]	
5. Tnum1 = Number	
6. Tnum3 = Number	

Step 1:

(T1 = [T* * Tnum1*Tnum3 -> T0]) ∘ Substitution = (T1 = [T* * Tnum1*Tnum3 -> T0])

Equation	Substitution
2. T1 = [Tf * Tx*Ty -> T2]	{ T1: = [T* * Tnum1*Tnum3 -> T0]}
3. Tf = [Tx * Ty -> T2]	
4. T* = [Number * Number -> Number]	
5. Tnum1 = Number	
6. Tnum3 = Number	

Step 2:

(T1 = [Tf * Tx*Ty -> T2]) ∘ Substitution =

(T1: = [T* * Tnum1*Tnum3 -> T0] = T1 = [Tf * Tx*Ty -> T2])

Equation	Substitution
3. Tf = [Tx * Ty -> T2]	{ T1: = [T* * Tnum1*Tnum3 -> T0]}
4. T* = [Number * Number -> Number]	
5. Tnum1 = Number	
6. Tnum3 = Number	
7. Tf = T*	
8. Tx = Tnum1	
9. Ty = Tnum3	
10. T2 = T0	

Step 3:

(Tf = [Tx * Ty -> T2]) ∘ Substitution = Substitution ∘ (Tf = [Tx * Ty -> T2])

Equation	Substitution
4. T* = [Number * Number -> Number]	{ T1: = [T* * Tnum1*Tnum3 -> T0], Tf = [Tx * Ty -> T2]}
5. Tnum1 = Number	
6. Tnum3 = Number	
7. Tf = T*	
8. Tx = Tnum1	
9. Ty = Tnum3	
10. T2 = T0	

Step 4:

(T* = [Number * Number -> Number]) ○ Substitution =

Substitution ○ (T* = [Number * Number -> Number])

Equation	Substitution
5. Tnum1 = Number	{ T1: = [[Number * Number -> Number] * Tnum1*Tnum3 -> T0], Tf = [Tx * Ty -> T2], T* = [Number * Number -> Number]}
6. Tnum3 = Number	
7. Tf = T*	
8. Tx = Tnum1	
9. Ty = Tnum3	
10. T2 = T0	

Step 5:

(Tnum1 = Number) ○ Substitution = Substitution ○ (Tnum1 = Number)

Equation	Substitution
6. Tnum3 = Number	{ T1: = [[Number * Number -> Number] * Number*Tnum3 -> T0], Tf = [Tx * Ty -> T2], T* = [Number * Number -> Number], Tnum1 = Number }
7. Tf = T*	
8. Tx = Tnum1	
9. Ty = Tnum3	
10. T2 = T0	

Step 6:

(Tnum3 = Number) ○ Substitution = Substitution ○ (Tnum3 = Number)

Equation	Substitution
7. Tf = T*	{ T1: = [[Number * Number -> Number] * Number*Tnum3 -> T0], Tf = [Tx * Ty -> T2], T* = [Number * Number -> Number], Tnum1 = Number, Tnum3 = Number }
8. Tx = Tnum1	
9. Ty = Tnum3	
10. T2 = T0	

Step 7:

(Tf = T*) ○ Substitution = Substitution ○ (Tf = T*)

Equation	Substitution
8. Tx = Tnum1	{ T1: = [[Number * Number -> Number] * Number*Number -> T0], Tf = [Tx * Ty -> T2], T* = [Number * Number -> Number], Tnum1 = Number, Tnum3 = Number }
9. Ty = Tnum3	
10. T2 = T0	
11. T2 = Number	
12. Tx = Number	
13. Ty = Number	

Step 8:

(Tx = Tnum1) ○ Substitution = Substitution ○ (Tx = Tnum1)

Equation	Substitution
9. Ty = Tnum3	{ T1: = [[Number * Number -> Number] * Number*Number -> T0], Tf = [Tx * Ty -> T2], T* = [Number * Number -> Number], Tnum1 = Number, Tnum3 = Number, Tx = Tnum1 }
10. T2 = T0	
11. T2 = Number	
12. Tx = Number	
13. Ty = Number	

Step 9:

(Ty = Tnum3) ○ Substitution = Substitution ○ (Ty = Tnum3)

Equation	Substitution
10. T2 = T0	{ T1: = [[Number * Number -> Number] * Number*Number -> T0], Tf = [Tx * Ty -> T2], T* = [Number * Number -> Number], Tnum1 = Number, Tnum3 = Number, Tx = Tnum1, Ty = Tnum3 }

11. T2 = Number	
12. Tx = Number	
13. Ty = Number	

Step 10:

(T2 = T0) ○ Substitution = Substitution ○ (T2 = T0)

Equation	Substitution
11. T2 = Number	{ T1: = [[Number * Number -> Number] * Number*Number -> T0], Tf = [Tx * Ty -> Number], T* = [Number * Number -> Number], Tnum1 = Number, Tnum3 = Number, Tx = Tnum1, Ty = Tnum3 , T2 = T0}
12. Tx = Number	
13. Ty = Number	

Step 11:

(T2 = Number) ○ Substitution = Substitution ○ (T2 = Number)

Equation	Substitution
12. Tx = Number	{ T1: = [[Number * Number -> Number] * Number*Number -> Number], Tf = [Tx * Ty -> Number], T* = [Number * Number -> Number], Tnum1 = Number, Tnum3 = Number, Tx = Tnum1, Ty = Tnum3, T2 = T0, T2 = Number }
13. Ty = Number	

Step 12:

(Tx = Number) ∘ Substitution = Substitution ∘ (Tx = Number)

Equation	Substitution
13. Ty = Number	{ T1: = [[Number * Number -> Number] * Number*Number -> Number], Tf = [Number * Ty -> Number], T* = [Number * Number -> Number], Tnum1 = Number, Tnum3 = Number, Tx = Number, Ty = Tnum3, T2 = T0, T2 = Number }

Step 13:

(Ty = Number) ∘ Substitution = Substitution ∘ (Ty = Number)

Equation	Substitution
	{ T1: = [[Number * Number -> Number] * Number*Number -> Number], Tf = [Number * Number -> Number], T* = [Number * Number -> Number], Tnum1 = Number, Tnum3 = Number, Tx = Number, Ty = Number, T2 = T0, T2 = Number }

The type inference succeeds since we have a type for T0, meaning that the expression is **well typed**. Since there are no free variables, the inferred type of T0 is: **Number.**

Our expression can be written now as

$$\Big(\big(lambda\ ([f : (Number \rightarrow Number)][x : Number][y : Number]) : Number\ (f\ x\ y)\big) * 1\ 3\Big)$$

Question 3.1:

Typing rule define:

For every: type environment _tenv, variable declaration _x, expressions _exp and type expressions _texp:

if tenv o { x : texp} |- exp : texp -> Then tenv |- (define x exp) : void

This typing rule allows for type inference of recursive functions. When analyzing the type of exp, which is typically a lambda expression representing a function, we consider it within a tenv (type environment) where the variable x is bound to type texp. This allows us to recursively analyze the body of the function and ensure that the types are consistent throughout.