PPL – Assignment 4

Theoretical Questions:

a) False. If a is of type $T1$, then the application of g to a will return an element of type $T2$, which will then be passed on to f which expects an argument of type $T1$, resulting in an error. If a is not of type $T1$, then the application of g to a will throw an error.
b) True. Since y is of type $T2$, then applying f to it will indeed return an element of type $T1$.

1. (a): False. If a is of type $T1$, then the application of g to a will return an element of type $T2$, which will then be passed on to f which expects an argument of type $T1$, resulting in an error. If a is not of type $T1$, then the application of g to a will throw an error.

   (b) True. Since y is of type $T2$, then applying f to it will indeed return an element of type $T1$.

   (c): The statement is true, the expression is a closure, from the inference method, the 'x' will be inferred to be of type T1, thus the statement is true.

   (d): the statement is false, the input of 'f' is T1*T2, and 100 is of type 'Number', if T2 will not be of the type 'Number' we will get an error.

2. **(a):** Step1 – Renaming:

   ((lambda (x1) (+ x1 1)) 4) ➔ ((lambda (x) (+ x 1)) 4)

   Step2 – Assigning type variables:

((lambda (x) (+ x 1)) 4)	T0
(lambda (x) (+ x 1))	T1
(+ x 1)	T2
+	T+
X	Tx
1	Tnum1
4	Tnum4

Step 3 – Constructing equations:

((lambda (x) (+ x 1)) 4)	T1 = [Tnum4 → T0]
(lambda (x) (+ x 1))	T1 = [Tx → T2]
(+ x 1)	T+ = [Tx*Tnum1 →T2]
+	T+ = [Number*Number → Number]
1	Tnum1 = Number
4	Tnum4 = Number

Step 4 – Solving the equations:

Equation	Substitution
T1 = [Tnum4 → T0]	{}
T1 = [Tx → T2]	
T+ = [Tx*Tnum1 →T2]	
T+ = [Number*Number → Number]	
Tnum1 = Number	
Tnum4 = Number	

(T1 = [Tnum4 → T0]) ○ Substitution = (T1 = [Tnum4 → T0])

Substitution = Substitution ○ ((T1 = [Tnum4 → T0])

Equation	Substitution
T1 = [Tx → T2]	T1 = [Tnum4 → T0]
T+ = [Tx*Tnum1 →T2]	
T+ = [Number*Number → Number]	
Tnum1 = Number	
Tnum4 = Number	

(T1 = [Tx → T2]) ○ Substitution = ([Tx → T2] = [Tnum4 → T0])

Both sides of the equation are composite so we split it into two equations: (Tx=Tnum4), (T2=T0)

Equation	Substitution
T+ = [Tx*Tnum1 →T2]	T1 = [Tnum4 → T0]
T+ = [Number*Number → Number]	
Tnum1 = Number	
Tnum4 = Number	
Tx=Tnum4	
T2=T0	

(T+ = [Tx*Tnum1 →T2]) ○ Substitution = (T+ = [Tx*Tnum1 →T2])

Substitution = Substitutions ○ (T+ = [Tx*Tnum1 →T2])

Equation	Substitution
T+ = [Number*Number → Number]	T1 = [Tnum4 → T0]
Tnum1 = Number	T+ = [Tx*Tnum1 →T2]
Tnum4 = Number	
Tx=Tnum4	
T2=T0	

(T+ = [Number*Number → Number]) ○ Substitution =
([Number*Number →Number] = [Tx*Tnum1 →T2])

Composite type - We add the equation: (T2 = Number)

Equation	Substitution
Tnum1 = Number	T1 = [Tnum4 → T0]
Tnum4 = Number	T+ = [Tx*Tnum1 →T2]
Tx=Tnum4	
T2=T0	
T2=Number	

Substituting in the substitution column (Tnum1 = Number), (Tnum4 = Number) and adding those equations to the column:

Equation	Substitution
Tx=Tnum4	T1 = [**Number** → T0]
T2=T0	T+ = [Tx***Number** →T2]
T2=Number	Tnum4 = Number
	Tnum1 = Number

(Tx = Tnum4) ○ Substitution = (Tx = Number)

Substitution = Substitution ○ (Tx = Number)

Equation	Substitution
T2=T0	T1 = [Number → T0]
T2=Number	T+ = [**Number** *Number →T2]
	Tnum4 = Number
	Tnum1 = Number
	Tx = Number

(T2=T0) ○ Substitution = (T2=T0)

Substitution = Substitution ○ (T2=T0)

Equation	Substitution
T2=Number	T1 = [Number → T0]
	T+ = [Number *Number →**T0**]
	Tnum4 = Number
	Tnum1 = Number
	Tx = Number
	T2 = T0

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

Substitution = Substitution ○ (T2= Number)

Equation	Substitution
	T1 = [Number → **Number**]
	T+ = [Number *Number →**Number**]
	Tnum4 = Number
	Tnum1 = Number
	Tx = Number
	T2 = **Number**
	T0 = Number

The type inference succeeds since we have a type for T0. T0 is of type **Number.**

(b) ((lambda (f1 x1) (f1 x1 1)) 4 +)

Stage 1: renaming

$$\left(\left(\text{lambda (f1 x1)(f1 x1 1)}\right)4\ +\right) \Rightarrow \left(\left(\text{lambda (f x)(f x 1)}\right)4\ +\right)$$

Stage 2: Assigning variables to each sub-expression

Expression	Variable
((lambda (f x) (f x 1)) 4 +)	T0
(lambda (f x) (f x 1))	T1
(f x 1)	T2
f	Tf
x	Tx
4	Tnum4
+	T+

Stage 3: Construct type equations

The equations for the type expressions are

Expression	Equation
((lambda (f x) (f x 1)) 4 +)	T1 = [Tnum4 * T+ -> T0]
(lambda (f x) (f x 1))	T1 = [Tf * Tx -> T2]
(f x 1)	Tf = [Tx -> T2]

The equations for the primitives are

Expression	Equation
4	Tnum4 = Number
+	T+ = [Number * Number -> Number]

Stage 4: Solve the equations.

Equation	Substitution
1. T1 = [Tnum4 * T+ -> T0]	{}
2. T1 = [Tf * Tx -> T2]	
3. Tf = [Tx -> T2]	
4. Tnum4 = Number	
5. T+ = [Number * Number -> Number]	

Equation	Substitution
2. T1 = [Tf * Tx -> T2]	{T1 := [Tnum4 * T+ -> T0]}
3. Tf = [Tx -> T2]	
4. Tnum4 = Number	
5. T+ = [Number * Number -> Number]	

T1 = [Tf * Tx -> T2] ○ Substitution = ([Tnum4 * T+ -> T0] = [Tf * Tx -> T2]) There is not type-sub since both sides of the equation are composite, we split it into three equations (6,7,8) and remove equation 2.

Equation	Substitution
3. Tf = [Tx -> T2]	{T1 := [Tnum4 * T+ -> T0]}
4. Tnum4 = Number	
5. T+ = [Number * Number -> Number]	
6. Tf = Tnum4	
7. Tx = T+	
8. T2 = T0	

Equation	Substitution
4. Tnum4 = Number	{T1 := [Tnum4 * T+ -> T0], Tf = [Tx -> T2]}
5. T+ = [Number * Number -> Number]	
6. Tf = Tnum4	
7. Tx = T+	
8. T2 = T0	

Equation	Substitution
5. T+ = [Number * Number -> Number]	{T1 := [Number * T+ -> T0], Tf = [Tx -> T2], Tnum4 = Number }
6. Tf = Tnum4	
7. Tx = T+	
8. T2 = T0	

Equation	Substitution
5. Tf = Tnum4	{T1 := [Number * [Number * Number -> Number] -> T0], Tf = [Tx -> T2], Tnum4 = Number, T+ = [Number * Number -> Number] }
6. Tx = T+	
7. T2 = T0	

(Tf = Tnum4) ○ Substitution = ([Tx -> T2]=Number). We get incompatible types, cannot continue.

Question 2.2 (b)

The function returns a Promise<R> because the process of wrapping the function includes getting/setting a key/value in the map from the previous question (2.1) – an async process which might not occur directly after executing the code, thus we need to use promises.

Question 3.1

Typing rule for define:

For every: type environment _Tenv,
variable _x1 expressions
_e1 and type expressions _S1, _U1:

If _Tenv ○ **{_x1 : _S1}** |- **_e1 : _S1**

Then _Tenv |- (define _x1 _e1) : **void**

Typing rule for set!:

For every: type environment _Tenv,

Variable _x1, expression _e1

If _Tenv |- _e1 : Tenv(_x1)

Then _Tenv |- (set! _x1 _e1): void