Exercise 1:
Consider the following PCF expression \( u \)
\[
\text{letrec } f (x) = 3 \text{ in } \\
\text{letrec } g (x) = g (x) \text{ in } \\
f (g 0)
\]
1. This is not a valid expression because the type annotations are missing. Add them.
2. Calculate the denotational semantics of \( u \).

Exercise 2:
For each OCaml expression below, give the type of the expression, if it exists. Justify.
1. \( \text{let } f x = x \text{ in } (f 3, f \text{ "trois"}) \)
2. \( (\text{fun } f -> (f 3, f \text{ "trois"})) \) \( (\text{fun } x -> x) \)
3. \( \text{let } f x = x \text{ in let } g = \text{ref } f \text{ in } (!g 3, !g \text{ "trois"}) \)

Exercise 3:
We consider the following language
\[
M ::= x | \lambda x : \tau.M | MN | \text{let } x : \tau = M \text{ in } N | \text{ff} | \text{tt} | \text{if } M \text{ then } N \text{ else } P
\]
1. Propose an adapted typing system.
2. Give a derivation of \( \vdash (\lambda x. \text{if } x \text{ then ff else } x)\text{tt} : \text{bool} \)
3. Which element of the programming language syntax is crucial to guarantee typing determinism? Explain with an example.
4. Show that the \texttt{let} is encoded using the other constructs in a well-typed way.
5. Propose small-step semantics for this language.
6. Show that there is a theorem of \textit{subject reduction}, that is, small-step semantics preserves typing.
7. We add to the syntax the following two constructions
\[
\text{try } M \text{ with } N | \text{abort}
\]
Propose an extension of the typing system.
8. Propose an extension of the small step semantics.

Exercise 4:
We add exceptional constructors that we denote as \( C_1, \ldots, C_n \). These are for example exceptions like \texttt{KeyboardInterrupt}. For each \( C_i \), we consider a type \( \tau_i \) of fixed argument and we add the rules of deductions
\[
C_i : \tau_i \rightarrow \text{exn}
\]
1. Adapt the syntax. What are the values? What are the contexts?
2. Adapt the small-step semantics.
3. Use it to reduce the next term assuming that $M \to^* V$.

\[
\text{try } (\lambda x. \lambda y. y) (\text{abort } M) \textbf{ with } C_i(x) \mapsto x
\]

4. OCaml language prohibits building exceptions possessing a polymorphic type. Explain.