This question explores how exceptions might be added to SLANG and the JARGON virtual machine. We will raise an exception with

\texttt{raise e}

where \( e \) is an expression. We will “trap” an exception with the following expression.

\texttt{try e with f end}

If \( e \) evaluates to a value \( v \), then \( v \) is the result of the \texttt{try}-expression. Otherwise, the evaluation of \( e \) raises an exception \( E \) and the \texttt{try}-expression continues by evaluating the function application \( f(E) \). To simplify things we will assume that each \( f \) is an identifier. Uncaught exceptions at the top-level will result in a runtime error.

\((a)\) Do we need to define a fixed type for exceptions? Justify your answer. \[3 \text{ marks}\]

\((b)\) What typing rule or rules would you implement for the expression \texttt{raise e}? Justify your answer. \[3 \text{ marks}\]

\((c)\) A compiler may rewrite expressions in order to optimise generated programs. For example, here are two rewrite rules to simplify conditional expressions:

\begin{align*}
\text{code} & \quad \text{replacement} \\
1 & \quad \text{if true then e1 else e2} \quad e1 \\
2 & \quad \text{if false then e1 else e2} \quad e2
\end{align*}

For each of the rules below, argue that it is, or is not, a valid optimisation rule.

\begin{align*}
\text{code} & \quad \text{replacement} \\
1 & \quad \text{raise (raise e)} \quad \text{raise e} \\
2 & \quad e1 + (\text{raise e2}) \quad \text{raise e2} \\
3 & \quad \text{try (raise e) with f end} \quad f(e) \\
4 & \quad \text{try e with (fun x -> raise x) end} \quad e
\end{align*}

\[6 \text{ marks}\]

\((d)\) Carefully describe the stack-oriented code you would generate for both the \texttt{raise}- and \texttt{try}-expressions. \[8 \text{ marks}\]