1. Evaluation: Adding a new form

We would like to introduce a new special form to our evaluator called `same?`. `Same?` always takes three arguments, and returns `#t` if all three arguments are `eq`. `Same?`, however, is smart in that if the first two arguments are different, then the third argument is not evaluated. Here are some examples of using `same?`.

```
(same? 'x 'x 'x) => #t
(same? 'x 'x 'y) => #f
(same? 'x 'y 'y) => #f
(same? 'x 'y (/ 1 0)) => #f
(same? 'x 'x (/ 1 0)) => Divide by Zero Error
```

To add this special form to the evaluator, we need to define some data abstraction.

Define the function `same??` that checks to see if an expression is a `same?` expression.

```
(define (same?? exp) )
```

Define the functions `same?-first` and `same?-second` that select out the first and second sub-expressions (assume someone else defined `same?-third`).

```
(define (same?-first exp) )
(define (same?-second exp) )
```

Next, write the appropriate clause to add to the `cond` clause of `eval`, assuming that we have the function `eval-same?` that will evaluate a `same?` expression.

```
Finally, write the `eval-same?` function that takes a `same?` expression and an environment and implements the special form as described above.

```
(define (eval-same? exp env)

)
```

We added `same?` as a special form in our language because our language had **applicative order** evaluation. If, instead, our language had **normal order** evaluation, then we could simply define `same?` as a function.
Assuming our Scheme has the normal order evaluation, define \texttt{same?} as a function.

\begin{verbatim}
(define (same? a b c)
)
\end{verbatim}

Now implement \texttt{eval-same?} by doing a syntactic transformation to an \texttt{if} expression (that also uses \texttt{eq?} Assume \texttt{eq?} is implemented in the evaluator).

\begin{verbatim}
(((same?? exp) (eval (same?->if exp)))
(define (same?->if exp)
)
\end{verbatim}

\section*{2. Streams}

Create a stream of Fibonacci numbers using add-streams.

\begin{verbatim}
(1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987 1597 2584 ...)
\end{verbatim}

Create a stream of powers of \(x\). (Hint: write a procedure called scale-stream).

Eg for \(x = 5\):

\begin{verbatim}
(1 5 25 125 625 3125 ...)
\end{verbatim}

Create a stream of factorials. (Hint: write a procedure called mult-streams that multiplies 2 streams).

Write an approximation to \(e^x\) by creating a series in which every successive element is a better approximation to \(e^x\). Remember that \(e^x = \sum_{0}^{\infty} \frac{x^n}{n!}\) (Hint: write stream-accum and div-streams). To get \(\sum_{0}^{20} \frac{x^n}{n!}\) for example, you would do (stream-ref result 20).