This tutorial will cover the following main ideas:

- Creating list values and tracing cons using substitution rules.
- Using primitive list functions such as first, rest, and empty? to solve problems.
Question 1: Creating list values and tracing cons using substitution rules.

Example 1:
Convert the following box-and-pointer diagram into a list in Racket using cons.

```
  2   →   "hello"   →   'up
```
Question 1: Creating list values and tracing cons using substitution rules.

Example 2:
The previous substitution rules still apply:
1. Functions must be applied on values (i.e., all arguments must be values, not expressions)
2. Given a choice, evaluate expressions from left-to-right (or top-to-bottom)
3. When applying a user-defined function, all substitutions of argument values happen in one step.

However, we add new rules for handling cons (from M06, 14/57):
4. \((\text{first } (\text{cons } a \ b)) \Rightarrow a\), where \(a\) and \(b\) are values.
5. \((\text{rest } (\text{cons } a \ b)) \Rightarrow b\), where \(a\) and \(b\) are values.
6. \((\text{empty? } \text{empty}) \Rightarrow \text{true}\).
7. \((\text{empty? } a) \Rightarrow \text{false}\), where \(a\) is any Racket value other than empty.
8. \((\text{cons? } (\text{cons } a \ b)) \Rightarrow \text{true}\), where \(a\) and \(b\) are values.
9. \((\text{cons? } a) \Rightarrow \text{false}\), where \(a\) is any Racket value not created using cons.
Question 1: Creating list values and tracing \texttt{cons} using substitution rules.

\begin{verbatim}
(define (frr lst) (first (rest (rest lst))))

(frr (cons (> 5 3)
    (cons (list? empty)
        (cons (cons? empty)
            (cons false empty))))))
\end{verbatim}
Question 2: Exactly Three

Write a function (exactly-three? lst) to determine if a list has exactly three elements.
Question 3: Three of a Kind

Write a function (three-of-a-kind? lst) to determine if a list of symbols contains exactly three identical symbols (and no other elements in the list).