CSC324 - Backtracking and choice

There are quite a few subtleties related to continuations and backtracking that we didn’t get a chance to go over during lecture. Answer the following questions to make sure you have the best understanding possible of these complex topics.

1. Warm-up: what is the continuation of \((-< \ 1 \ 2 \ 3)\) in \((f \ (-< \ 1 \ 2 \ 3))\), assuming \(f\) is a function?

2. It is very convenient to think of choice expressions as “partially expanded”: \((\text{lambda} () \ (-< \ 2 \ 3))\). However, while this helps our understanding of how the choice operators work, it’s not actually what happens in the interpreter. Show the full macro expansion of the following choice expression. (We know it’s long. But do this carefully to appreciate how macro expansion really works.)

\([+ \ 10 \ (-< \ 1 \ 2 \ 3)]\)
3. Suppose we don't want to put parentheses around `next`, and instead define it as follows:

```scheme
(define next
  (if (empty? choices)
      "false."
      ((get-choice!))))
```

What is the problem with this implementation?

4. Consider the definition of the helper function `all-helper` in the `all` macro. Here are two incorrect implementations; below each one, explain why it is wrong.

```scheme
; (a)
(lambda (e)
  (next)
  (set! results (cons e results))
  (reverse results))
```

```scheme
; (b)
(lambda (e)
  (set! results (cons e results))
  (reverse results)
  (next))
```
5. What is the continuation of (\(< 1 2\)) in the expression (+ (\(< 1 2\) (\(< 10 20\) (\(< 5 4\))))?

6. Assuming that \texttt{choices} is implemented as a stack (which is provided on the code), what is the value of the following expression (looking for the correct order here):

\[
\text{(all (+ (\(< 1 2\) (\(< 10 20\) (\(< 5 4\)))) \text{)}
\]

7. Repeat the previous question, but assume that \texttt{choices} is implemented as a queue.
Now that we’ve put in so much work into understanding and implementing these operators, the payoff is here! This week, we’re going to see how to use these operators to write some very expressive yet concise code. Let’s start small with some basic computations on numbers.

- Define a function that takes a number, and chooses to either add 1 or 2 to it.

- Define a function that takes a number, and chooses to either add 2 to it or multiply it by 10.

- Define a function that takes a number \( n \), and chooses a number between 0 and \( n \), inclusive. (Hint: this can be defined recursively.)

- Define a function that takes two numbers \( n \) and \( m \), and chooses a number between \( n \) and \( m \), inclusive. (This and the previous question essentially implement Python’s \texttt{range} function, which is the simplest type of \texttt{generator} in Python.)
• Define a function that takes a number $n$, and returns a list of numbers of length $m$, where each item is chosen from the range 1-5, and $m$ is chosen from the range 0 to $n - 1$.

• Define a function which takes a number $n$, and chooses a binary string of length $\leq n$ and returns it.

• Something interesting to think about: can you define a function/macro which abstracts the structure of the previous two questions?