The goal of this project is to implement a Clojure interpreter for a stack-based language. An example of the API is on the next page.

Explanation of functionality:

- Variables are symbols prefixed with "!", e.g. !v
- A stack function is defined with "defstackfn". The first argument is the input declaration which also provides variable names to the arguments. The stack always starts empty.
- The implementation of a stackfn is a sequence of stack operations.
- Using a constant as a stack operation pushes that value onto the stack
- Using a variable as a stack operation pushes the value for that variable onto the stack.
- A variable is assigned the top value of the stack by appending "+" to the variable name, e.g. !v+
- A function is invoked with "invoke>". "invoke>" takes as input the operation and the arity to use.
- <pop> is a special operation which removes the top value of the stack.
- if> tests if the top value of the stack is truthy to determine which branch to follow. The branches are separated with "else>"

Notes about your implementation:

- Should provide an informative error if there's an invalid stack operation or a variable is referenced that doesn't exist.
- The example below contains all functionality you need to implement.
- It should be possible to shadow vars (naming a new local the name of an existing variable).

Code example with stack values in comments:

```
(defstackfn f
  [!a !b !c] ; example uses input: 1 2 4. Stack starts empty.
  !a ; 1
  !b ; 1 2
  (invoke> + 2) ; 3
  !v1+ ; 3
  !c ; 3 4
  !c ; 3 4 4
  <pop> ; 3 4
  2 ; 3 4 2
  (invoke> * 2) ; 3 8
  !v2+ ; 3 8
  (invoke> = 2) ; false
  (if> ; stack empty
    !v1
    !v2
    (invoke> - 2)
   else>
    "false!!" ; "false!!"
    (invoke> println 1) ; nil
    <pop>; stack empty
    !v1 ; 3
    !v2 ; 3 8
    (invoke> * 2) ; 24
    )
  )
;; (f 1 2 4) prints "false!!" and returns 24
```