Higher Order Functions

CS 151: Introduction to Computer Science I

Binary Trees

```
(define-struct (Tree A)
  ([value :  A]
   [left-child :  (U 'none (Tree A))]
   [right-child :  (U 'none (Tree A))]))
```

Could be a tree of numbers, or a family tree, or an expression tree...

```
(:  number-tree :  (Tree Integer))
(define number-tree
  (Tree 5
        'none
        (Tree 7 'none (Tree -7 'none 'none))))
```

Searching

Write a function that looks for a node of the tree with a given requirement. If there isn't a node, return `'none`

Inputs: a binary tree and a requirement function
For family trees,

```
(: my-family : (Tree String))
(define my-family a-very-complicated-expression)

(: named-barney? : String -> Boolean)
(define (named-barney? name)
  (string=? name "Barney"))

(search my-family named-barney?)
```

Searching

```
(:  search :  (All (A) (Tree A) (A -> Boolean) ->
    (U 'none (Tree A))))
(define (search tree req?)
  (cond
    [(req?  (Tree-value tree)) tree]
    [(and
      (Tree?  (Tree-left-child tree))
      (Tree?  (search (Tree-left-child tree)
req?)))
     (search (Tree-left-child tree) req?)]
    [(and
      (Tree?  (Tree-right-child tree))
      (Tree?  (search (Tree-right-child tree)
req?)))
     (search (Tree-right-child tree) req?)]
    [else 'none]))
```

What to know

- Binary trees and searching
- foldr and foldl
- local
- Using functions as variables