Racket:
Lists (& Recursion)
What kind of data structure is this?

Yes, but what kind?

(Your response)
Recap:
Functions in Racket
Inductive Data Types & Recursive Operations (Putting Together & Taking Apart) or (Defining & Computing)
Defining lists
## Racket lists

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Semantics</th>
</tr>
</thead>
<tbody>
<tr>
<td>what we write</td>
<td>what it means</td>
</tr>
</tbody>
</table>

### Syntax Examples

<table>
<thead>
<tr>
<th>Expression</th>
<th>Diagram</th>
</tr>
</thead>
<tbody>
<tr>
<td>'()</td>
<td><img src="image" alt="empty list diagram" /></td>
</tr>
<tr>
<td>(cons &lt;value&gt; &lt;list&gt;)</td>
<td><img src="image" alt="cons diagram" /></td>
</tr>
<tr>
<td>'(&lt;value1&gt; ... &lt;valueN&gt;) or (list &lt;value1&gt; ... &lt;valueN&gt;)</td>
<td><img src="image" alt="racket list diagram" /></td>
</tr>
<tr>
<td>(cons &lt;value&gt; &lt;non-pair&gt;) or '(&lt;value&gt; &lt;non-pair&gt;)</td>
<td><img src="image" alt="cons non-pair diagram" /></td>
</tr>
</tbody>
</table>
Constructing lists: let’s practice

write down answers as either a drawing or a Racket expression

1. `(cons 3 (cons 2 (cons 1 '())))`

2. `'(1 2 3)``

3. `1 2`

4. `1 2`

5. **Bonus:** `(cons 1 (cons (cons 2 '()) '()))`

6. **Bonus:** `'((()) . 1)`
   - What *other* expression could you write to make Racket respond with this value?
   - How would you draw this value?
Recursive operations
;; len
;;  inputs: a list, L
;;  outputs: the number of elements in the list
(define (len L)

;; tests
(check-expect (len '(())) 0)
(check-expect (len '(1 2 3)) 3)
(check-expect (len '(((1 2 3)))) 1)
;;; len
;;; inputs: a list, L
;;; outputs: the number of elements in the list
(define (len L)
  ; base case
  (if (null? L)
      0
      (+ 1 (len (rest L))))

; tests
(check-expect (len '()) 0)
(check-expect (len '(1 2 3)) 3)
(check-expect (len '(((1 2 3))) 1)
;; len
;; inputs: a list, L
;; outputs: the number of elements in the list
(define (len L)
  (if (empty? L)
      0
      (+ 1 (len (rest L)))))

; tests
(check-expect (len '()) 0)
(check-expect (len '(1 2 3)) 3)
(check-expect (len '((1 2 3))) 1)
## Racket lists

Constructors, accessors, and operations

<table>
<thead>
<tr>
<th>Constructors</th>
<th>Accessors</th>
<th>Operations</th>
</tr>
</thead>
<tbody>
<tr>
<td>'()</td>
<td>(first &lt;list&gt;)</td>
<td>(length &lt;list&gt;)</td>
</tr>
<tr>
<td>(cons &lt;value&gt; &lt;list&gt;)</td>
<td>(rest &lt;list&gt;)</td>
<td>(member &lt;value&gt; &lt;list&gt;)</td>
</tr>
<tr>
<td>'(&lt;value1&gt; ... &lt;valuenn&gt;)</td>
<td>...</td>
<td>(empty? &lt;list&gt;)</td>
</tr>
<tr>
<td>(list &lt;value1&gt; ... &lt;valuenn&gt;)</td>
<td>...</td>
<td>(list? &lt;value&gt;)</td>
</tr>
</tbody>
</table>

... more in the Racket documentation: [http://docs.racket-lang.org/reference/pairs.html](http://docs.racket-lang.org/reference/pairs.html)
Recursion over lists: let’s practice!

See recursion.rkt for solutions