This tutorial will cover the following main ideas:

- Tracing recursion on natural numbers using substitution rules.
- Recursion on lists and natural numbers.
- Using wrapper functions to simplify problems.
**Question 1**
A summary of some important substitution rules so far:

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.
4. When tracing a conditional expression,
   a. \((\text{cond} \ [\text{false exp}]...) \Rightarrow (\text{cond} ...)\)
   b. \((\text{cond} \ [\text{true exp}]...) \Rightarrow \text{exp}\)
   c. \((\text{cond} \ [\text{else exp}]) \Rightarrow \text{exp}\)
Question 1

(define (parity n)
  (cond [(zero? n) (cons 'even empty)]
        [(even? n) (cons 'even (parity (sub1 n)))]
        [else (cons 'odd (parity (sub1 n)))]))

(parity 2)
**Question 1**

Also, we can show a “condensed trace” as:

```
(parity 2) => (cons 'even (parity 1)) => (cons 'even (cons 'odd (parity 0))) => (cons 'even (cons 'odd (cons 'even empty)))
```
**Question 2: n-th-item**

Complete the following exercise from Module 7:

;;; (n-th-item lst n) Produce the n-th item in lst, where (first lst) is ;; the 0th.

;;; Examples:
(check-expect (n-th-item (cons 3 (cons 7 (cons 31 (cons 63 empty)))) 0) 3) 
(check-expect (n-th-item (cons 3 (cons 7 (cons 31 (cons 63 empty)))) 3) 63)

;;; n-th-item: (listof Any) Nat → Any 
;;; requires: n < (length lst)
(define (n-th-item lst n) ...)
Question 3: Caesar Cipher

Given a string text and a natural number shift, write a function (encrypt text shift) that produces a new text encrypted using the Caesar cipher, which shifts each letter a fixed number of positions down the alphabet. For this problem, all characters in the input and output texts will be capital letters from the alphabet A-Z.

Examples:
(check-expect (encrypt "ABCD" 2) "CDEF")
(check-expect (encrypt "ZAP" 1) "ABQ")

Note: the Caesar cipher is a well-known encryption method, but it is not secure and can be easily hacked. If you would like to learn more, consider taking CS 458: Computer Security and Privacy.
Question 3: Caesar Cipher

Examples:
(check-expect (encrypt "ABCD" 2) "CDEF")
(check-expect (encrypt "ZAP" 1) "ABQ")

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>G</th>
<th>H</th>
<th>I</th>
<th>J</th>
<th>K</th>
<th>L</th>
<th>M</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td>65</td>
<td>66</td>
<td>67</td>
<td>68</td>
<td>69</td>
<td>70</td>
<td>71</td>
<td>72</td>
<td>73</td>
<td>74</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>65</td>
<td>66</td>
<td>67</td>
<td>68</td>
<td>69</td>
<td>70</td>
<td>71</td>
<td>72</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>65</td>
<td>66</td>
<td>67</td>
<td>68</td>
<td>69</td>
<td>70</td>
<td>71</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>65</td>
<td>66</td>
<td>67</td>
<td>68</td>
<td>69</td>
<td>70</td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>65</td>
<td>66</td>
<td>67</td>
<td>68</td>
<td>69</td>
</tr>
</tbody>
</table>

Table 1: ASCII encodings of English capital letters.
(image source: https://mvngu.wordpress.com/tag/shift-cipher/)
Question 3: Caesar Cipher

Design Recipe steps:
1. Purpose
2. Examples
3. Definition Header and Contract
4. Finalize purpose
5. Body
6. Tests
Question 3: Caesar Cipher
Template for processing a list (from Module 7):

;; listof-X-template: (listof X) → Any
(define (listof-X-template lst)
  (cond [(empty? lst) ...]
        [else (... (first lst) ...
                   (listof-X-template (rest lst)) ...)]))