Lab 07: Structural recursion on numbers

Create a separate file for each question. Keep them in your "Labs" folder, with the name **l**`ij`**q**`k` for Lab `ij`, Question `k`.

Download the headers for each function from the file l07-interface`.rkt`.

After you have completed a question (except class exercises), including creating tests for it, you can obtain feedback by submitting it and requesting a public test. Follow the instructions given in the Style Guide.

Question 7 makes use of the following data definitions:

```
;; An association (As) is (list Num Str), where
;; * the first item is the key,
;; * the second item is the associated value.

;; An association list (AL) is one of
;; * empty
;; * (cons As AL)

;; Note: All keys must be distinct.
```

Language level: Beginning Student with List Abbreviations

1. *[Class exercise with lab instructor assistance]*

Create a function *is-prime?* which consumes a natural number *n* and produces `true` if *n* is a prime number. For the purposes of this question, 1 is not considered a prime number.

For example:

```
    (is-prime? 17) => true
    (is-prime? 12) => false
```

2. *[Warm-up question (not to be submitted)]*

Create the function *repeat*, which consumes a natural number, *n*, and string, *s*, and produces a list with *n* occurrences of *s*. You may not use the built-in `make-list` function.

For example:

```
    (repeat 4 "b") => (list "b" "b" "b" "b")
```

3. *[Adapted from HtDP Exercise 11.5.3]* Recall that x^n means multiplying x with itself n times. Create the function *exponent*, which consumes a number, *base,* and a natural number, *expt,* and produces $base^{expt}$ without using the built-in exponentiation function `expt`.

For example:

```
    (exponent 2 4) => 16
```

4. Create the function *largest-prime* that consumes two natural numbers, *bottom* and *top*, and produces either the largest prime number in the range from *bottom* to *top* (inclusive) or `false` is there is no prime in that range. Assume *bottom* is less than or equal to *top*.

   For example:

   ```
   (largest-prime 10 24) => 23
   (largest-prime 510 520) => false
   ```

5. Create the function *total-price-list* that consumes a list *lol*. Each element of *lol* is itself a three element list, where the first element is a string, the second element is a number, and the third element is a natural number. Respectively, these represent the name of an item, the price of an item, and the number of items. *total-price-list* will produce a list of the same length of *lol*, where each element is a list containing the name of the item followed by its total price.

   For example:

   ```
   (total-price-list (list (list "Wind Lace" 2.50 3)
                           (list "Iron Branch" 50 2)))
   => (list (list "Wind Lace" 7.5) (list "Iron Branch" 100))
   ```

6. Create the function *remove-al* that consumes a number, *key*, and an association list, *al*, and produces the association list resulting from removing *key* from *al*.

   For example:

   ```
   (remove-al 5 (list (list 4 "Lina")
                      (list 5 "Dazzle")
                      (list 1 "Sven")))
       => (list (list 4 "Lina") (list 1 "Sven"))
   (remove-al 6 (list (list 2 "Helen"))
       => (list (list 2 "Helen"))
   ```

7. Now, create the function *exponent-without-mult* which consumes natural numbers *base* and *expt* and produces *base* to the *expt*. However, the only built-in arithmetic functions allowed are the functions `add1` and `sub1`. Be prepared for it to be very slow to run!

   For example:

   ```
   (exponent-without-mult 3 3) => 27
   ```

Optional open-ended questions

[Adapted from HtDP Problem 11.1] Create a function that consumes two numbers and produces the result of subtracting the first number from the second. Essentially recreating the built-in function `-`. However, you may not use the built-in function `-`.

Helpful tips

Numbers as a recursive concept

You’re likely used to thinking of natural numbers as a plain concept: they’re just numbers. However, it will be useful when writing these functions to use structural recursion. That is, you must consider the set of natural numbers as they are defined recursively.