Lab 06: More Lists and 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 `lab-interface06.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.

Language level: Beginning Student

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

Complete the function *canadianize* from lecture. This function consumes a string, *s*, and produces a string where each "o" in *s* is replaced with "ou".

For example:
```
(canadianize "About") => "Abouut"
```

2. *[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
```

3. *[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")
```

4. *[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
   ```

5. Create a function *switch-case* that consumes a string, *str*, and produces a string in which each lower-case letter is converted to an upper-case letter, each upper-case letter is converted to a lower-case letter, and all other characters are preserved.

   For example:

```
(switch-case "AbC-DeF!") => "aBc-dEf!"
```

6. The first item in *alist* is at position 0. Complete the function *list-pos* that consumes a non-empty list of strings, *nelos*, and an element in the list, *elt*, and produces the position of the first occurrence of *elt* in the list.

   For example:
   ```
   (list-pos
     (cons "a" (cons "b" (cons "c" (cons "b" empty)))) "b")
   => 1
   ```

7. Create a function *differences* that consumes a non-empty list of numbers, *nelon*, and produces a list of differences between adjacent pairs. The function produces `empty` for a list of length 1.

   For example:
   ```
   (differences
     (cons 25 (cons 16 (cons 9 (cons 1 (cons 4 empty))))))
   => (cons 9 (cons 7 (cons 8 (cons -3 empty))))
   ```

8. A grade is a number from 0 to 100, inclusive. Create a function *average-filter* that consumes a non-empty list of grades, *log,* and produces a list of the grades from *log* that are above the average of the grades in *log*.

   For example:
   ```
   (average-filter (cons 45 (cons 55 (cons 79 empty)))
   => (cons 79 empty)
   ```

Optional open-ended questions

If you have not completed the open-ended questions from previous labs, the tools you now have available should be very helpful to solve these problems.

Helpful tips

String documentation

The string documentation has helpful information on strings and characters. You may find it under the heading Resources on the course website.

Base cases with non-empty lists

In the past we worked with lists that could possibly be empty. When dealing with the base case, we asked "What is the answer to this function, on the empty list?". For example, when the function should produce the length of the list, that question becomes "What is the length of the empty list?". Now, with non-empty lists we can no longer ask this question. Instead, to answer the base case, we must ask "What is the answer to this function, on list with one element?".