CS181, Winter 2021, Assignment 4
Due: Thursday Feb 17, 11:00PM

You will write 8 Racket functions (not counting helper functions).

Download the starter zip file from the course website. Add to these files to complete your homework.

Provided Code:

The code at the top of `hw4tests.rkt` uses a graphics library to provide a simple, entertaining (?) outlet for your streams. You need not understand this code (though it is not complicated) or even use it, but it may make the homework more fun. This is how you use it:

- `(open-window)` returns a graphics window you can pass as the first argument to `place-repeatedly`.
- `(place-repeatedly window pause stream n)` uses the first `n` values produced by `stream`. Each stream element must be a pair where the first value is an integer between 0 and 5 inclusive and the second value is a string that is the name of an image file (e.g., `.jpg`). (Sample image files that will work well are available on the course website. Put them in the same directory as your code.) Every `pause` seconds (where `pause` is a decimal, i.e., floating-point, number), the next stream value is retrieved, the corresponding image file is opened, and it is placed in the window using the number in the pair to choose its position in a 2x3 grid as follows:

0	1	2
3	4	5

Two of the provided tests demonstrate how to use `place-repeatedly`. The provided tests require you to complete several of the problems, of course. We hope these tests' expected (visual) behavior is not difficult for you to figure out.

Helpful Guide / Warning:

The first three problems are "warm-up" exercises for Racket. Subsequent problems dive into streams (4–8). Some short problems may be difficult. Go slowly and focus on using what you learned about thunks, streams, etc.

Some problems require that you use a few standard-library functions that were not used in lecture. See the Racket documentation at `http://docs.racket-lang.org/`, particularly The Racket Guide, as necessary — looking up library functions even in languages new to you is an important skill. It is fine to discuss with others in the class what library functions are useful and how they work.

Problems:

1. Write a function `sequence` that takes 3 arguments `spacing`, `low`, and `high`, all assumed to be numbers. Further assume `spacing` is positive. `sequence` produces a list of numbers from `low` to `high` (including `low` and possibly `high`) separated by `spacing` and in sorted order. Sample solution: 4 lines. Examples:

Call	Result
`(sequence 2 3 11)`	`'(3 5 7 9 11)`
`(sequence 3 3 8)`	`'(3 6)`
`(sequence 1 3 2)`	`'()`

2. Write a function `string-append-map` that takes a list of strings `xs` and a string `suffix` and returns a list of strings. Each element of the output should be the corresponding element of the input with `suffix` appended to the end (with no extra space between the element and `suffix`). You must use Racket-library functions `map` and `string-append`. Sample solution: 2 lines. Example: the result of `(string-append-map '("hi" "bye") "2")` should be `'("hi2" "bye2")`.

3. Write a function `list-nth-mod` that takes a list `xs` and a number `n`. If the number is negative, terminate the computation with `(error "list-nth-mod: negative number")`. Else if the list is empty, terminate the computation with `(error "list-nth-mod: empty list")`. Otherwise, let i be the remainder when dividing `n` by the length of `xs`. The function should return the i^{th} element of `xs` (and note that we *count from zero*, so the first list element is the 0^{th} element). Library functions `length`, `remainder`, `car`, and `list-tail` are all useful – see the Racket documentation. Sample solution is 6 lines.

4. Write a function `stream-for-k-steps` that takes a stream `s` and a number `k`. It returns a list holding the first `k` values produced by `s` in order. Assume `k` is non-negative. Sample solution: 5 lines. Note: You can test your stream solutions for later problems with this function instead of the graphics code.

5. Write a stream `funny-number-stream` that is like the stream of natural numbers (i.e., 1, 2, 3, ...) except numbers divisble by 6 are negated (i.e., 1, 2, 3, 4, 5, -6, 7, 8, 9, 10, 11, -12, 13, ...). Remember our definition of a stream from lecture: a stream is a thunk that when called produces a pair. In this case, the car of the pair will be a number and the cdr will be another stream.

6. Write a stream `manu-then-dog`, where the elements of the stream alternate between the strings `"manu.jpg"` and `"dog.jpg"` (starting with `"manu.jpg"`). In other words, `manu-then-dog` should be a thunk that when called produces a pair of `"manu.jpg"` and a thunk that when called produces a pair of `"dog.jpg"` and a thunk that when called... etc. Sample solution: 4 lines.

7. Write a function `stream-add-one` that takes a stream `s` and returns another stream. If `s` would produce v for its i^{th} element, then `(stream-add-one s)` would produce the pair `(1 .` v`)` for its i^{th} element. Sample solution: 4 lines. Hint: Use a thunk that when called uses `s` and recursion. Note: One of the provided tests uses `(stream-add-one manu-then-dog)` with `place-repeatedly`.

8. Write a function `cycle-lists` that takes two lists `xs` and `ys` and returns a stream. The lists may or may not be the same length, but assume they are both non-empty. The elements produced by the stream are pairs where the first part is from `xs` and the second part is from `ys`. The stream cycles forever through the lists. For example, if `xs` is `'(1 2 3)` and `ys` is `'("a" "b")`, then the stream would produce, `(1 . "a")`, `(2 . "b")`, `(3 . "a")`, `(1 . "b")`, `(2 . "a")`, `(3 . "b")`, `(1 . "a")`, `(2 . "b")`, etc.

   Sample solution is 6 lines and is more complicated than the previous stream problems. Hints: Use one of the functions you wrote earlier. Use a recursive helper function that takes a number `n` and calls itself with `(+ n 1)` inside a thunk.

9. **Challenge Problem:** Write `cycle-lists-challenge`. It should be equivalent to `cycle-lists`, but its implementation must be more efficient. In particular, for each time the stream produces a new value, the code must perform only two `car` operations and two `cdr` operations, including operations performed by any function calls. So, for example, you cannot use `length` because it uses `cdr` multiple times to compute a list's length.

Test your functions: Put your testing code in a second file. We will not grade it, but you must turn it in.

Assessment: Your solutions should be correct, in good style, and use only features we have used in class. Do not use mutation.

Turn-in Instructions

- Turn in your modified `hw4.rkt` and `hw4tests.rkt`.
- Turn in the assignment on Gradescope.