coursera

# Extra Practice Problems

Contributed by Pavel Lepin and Charilaos Skiadas

1. Write a function `palindromic` that takes a list of numbers and evaluates to a list of numbers of the same length, where each element is obtained as follows: the first element should be the sum of the first and the last elements of the original list, the second one should be the sum of the second and second to last elements of the original list, etc. Example: `(palindromic (list 1 2 4 8))` evaluates to `(list 9 6 6 9)`.
2. Define a stream `fibonacci`, the first element of which is 0, the second one is 1, and each successive element is the sum of two immediately preceding elements.
3. Write a function `stream-until` that takes a function `f` and a stream `s`, and applies `f` to the values of `s` in succession until `f` evaluates to `#f`.
4. Write a function `stream-map` that takes a function `f` and a stream `s`, and returns a new stream whose values are the result of applying `f` to the values produced by `s`.
5. Write a function `stream-zip` that takes in two streams `s1` and `s2` and returns a stream that produces the pairs that result from the other two streams (so the first value for the result stream will be the pair of the first value of `s1` and the first value of `s2`).
6. Thought experiment: Why can you *not* write a function `stream-reverse` that is like Racket's `reverse` function for lists but works on streams.
7. Write a function `interleave` that takes a list of streams and produces a new stream that takes one element from each stream in sequence. So it will first produce the first value of the first stream, then the first value of the second stream and so on, and it will go back to the first stream when it reaches the end of the list. Try to do this without ever adding an element to the end of a list.
8. Define a function `pack` that takes an integer `n` and a stream `s`, and returns a stream that produces the same values as `s` but packed in lists of `n` elements. So the first value of the new stream will be the list consisting of the first `n` values of `s`, the second value of the new stream will contain the next $\verb|n|$$ values, and so on.
9. We'll use *Newton's Method* for approximating the square root of a number, but by producing a stream of ever-better approximations so that clients can "decide later" how approximate a result they want: Write a function `sqrt-stream` that takes a number `n`, starts with `n` as an initial guess in the stream, and produces successive guesses applying $f_n(x) = \frac{1}{2}((x + \frac{n}{x})$ to the current guess.
10. Now use `sqrt-stream` from the previous problem to define a function `approx-sqrt` that takes two numbers `n` and `e` and returns a number $x$ such that $x \cdot x$ is within `e` of `n`. Be sure not to create more than one stream nor ask for the same