Lab 6: Lazy Lists

In last week's lecture, we introduced the notion of *streams*, an abstract model of a sequence of values that are made available over time (in contrast to all values being available simultaneously). In this lab, you'll look at *lazy lists*, an implementation of streams which use functions to generate new elements of the list dynamically at runtime, when they are accessed. (The name "lazy" here is analogous to lazy function evaluation: list elements are only computed when accessed, and not all at once.)

Starter code

- `lab6.rkt`

Task 0: The lazy list implementation

Open the `lab6.rkt` starter code file, and read through the stream implementation carefully. Note that this implementation is exactly parallel to the standard `list` API you've already been using in Racket.

Then, to make sure you understand the basic idea, complete the first set of tests by using the provided stream functions. (The provided starter code is *very* suggestive, and so if you spend the time to really understand this implementation, you should find this task pretty straight-forward.)

Task 1: Lazy list functions

In the same file, complete the stream functions in the order provided. If you're stuck, try writing the equivalent *Racket list* function first; given this, you should be able to translate that definition immediately to a stream version.

Warning: because Racket provides its own stream implementation, we're shadowing a few built-in names here. This shouldn't matter to you, but if you see some strange errors referring to functions you haven't implemented, that's why.

Task 2: Infinite streams

One of the consequences of a stream-based approach to thinking about sequences is that it enables a very elegant representation of *infinite* sequences. It may seem strange at first to even consider a data structure representing an infinite number of values, but remember here that the stream model does not require that we store all the values simultaneously in memory. Complete the functions in the starter code listed under Task 2, which get you to define a variety of functions that return *infinite streams*. We'll build on this pretty cool idea in this week's lecture and exercise!