Problem 1: Queues

Consider the queue implementation discussed in class in which a queue is represented by three variables, **head**, **tail**, **queue**, all of which are defined at the top-level. **head** and **tail** are integers, and **queue** is (vectorof number). Assume the basic, non-circular vector implementation of queues.

Let's design a function **print-queue** that non-destructively prints all the elements in the queue in order of their position in the queue.

Examples

Create an example queue to print using the procedures **enqueue** and **dequeue**. There should be at least **4** entries in the queue. You must include calls to both enqueue and dequeue.

Give the sequence of **enqueue** and **dequeue** calls.

Function

Create a function called **print-queue** that builds a list of the numbers in the queue.

```
(define (print-queue)
;; print-queue: -> (listof numbers)
```

Test

Show the output of your function on the example queue you constructed in part one.

Alternate implementations

We also considered a circular implementation of queues using vectors in class.

Would the function **print-queue** change under the circular vector implementation of queues? If so, please modify your function to handle the circular queue implementation. If not, explain why.

Abstraction and Concrete Implementation

Explain what would happen if you used **dequeue** to implement **print-queue**.

Problem 2: Streams: Powers

Stream Creation

Recall the data definition of streams:
A stream is:

- (cons number (delay stream))

Create a stream **powers-of-2** composed of the powers of two, i.e. $2^1, 2^2, ..., 2^i, ...$

There are many alternative ways to define a stream of powers-of-2. Feel free to create powers-of-2 in whatever way makes sense to you.
Here's a function to help you test your code.

```
;; take : number stream -> list-of-numbers
;; extracts the first n elements from the stream
(define (take n s)
  (cond
    [(zero? n) '()]
    [else (cons (car s) (take (- n 1) (force (cdr s))))]))
```

Hint: One possibility is to create a function **powers-of-2-from** which takes a starting number **n** and creates a stream consisting of $n * 2^0, n * 2^1, n * 2^2, ..., n * 2^i, ...$

```
(define (powers-of-2-from n)
;; powers-of-2-from: number -> stream-of-numbers
```

Powers: Generalization

Consider a function **power-stream** that will construct a stream of all the powers of *any* x, starting with the first power; that is, a stream given by the sequence $x, x^2, x^3, x^4, ...$ for ANY value of x.

```
(define (power-stream x)
;; power-stream: number -> stream-of-numbers
```

Examples

Give three examples of calls to **power-stream** and the first few elements of the output stream.

Template

Write the template for functions on streams.

Function

Create the function **power-stream** as described above.