


Practice Problems for Section 5

[Subscribe for email updates.](#)

 **PINNED**

 No tags yet. [+ Add Tag](#)

Sort replies by: [Oldest first](#) [Newest first](#) [Most popular](#)



Pavel Lepin COMMUNITY TA · 14 days ago 

Practice Problems for Section 5

NOTE: You do not need to handle corner cases, particularly receiving arguments of the wrong type, unless explicitly asked to do that in the problem statement.

Warm Up

Knights Who Say 'Ni'

I'm sure you saw this coming. We want... another factorial!

Write a function `factorial` that takes a single non-negative integer number n as an argument and evaluates to $n!$.

EXAMPLE: `(factorial 4)` evaluates to `24`

Another Shrubbery!

Now do it again. But your solution must be tail-recursive!

Name your function `tr-factorial`.

EXAMPLE: `(tr-factorial 4)` evaluates to `24`

Palindromic Addition

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)`

Streams

We're going to use the same definition of streams here as in the lectures and in the homework assignment. There are no examples in this sub-section, as providing useful examples would require defining some extra stream functions -- which you should do on your own for practice! If you really want to see some examples, take a peek at provided tests, but beware of **SPOILERS**!

HINT: You may want to scour the reading notes for useful stream functions. Yes, using stuff found in the class materials is fair play.

Undecided

Define a stream `undecided` that starts with a `#t` and then alternates between `#f` and `#t` infinitely.

Fibonacci

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.

Interleave (*)

Write a function `interleave` that takes two streams `xs` and `ys` and evaluates to another stream, such that the first element of it is the first element of `xs`, the second element is the first element of `ys`, the third element is the second element of `xs` etc. For example, `(interleave fibonacci undecided)` would evaluate to a stream the elements of which would be, in order: `0`, `#t`, `1`, `#f`, `1`, `#t`, `2`, `#f` ...

Not These Guys Again

Nil!!! We want... another factorial. Recall **A Novel Approach** problem from Section 3. The unfortunate issue with that implementation of factorial function was that it used up extra $O(n)$ space due to explicit building of a list of all numbers from 1 to n .

Write a function `stream-factorial` that will compute the factorial by multiplying the numbers from the infinite stream of all natural numbers, in $O(1)$ space.

HINT: You may want to borrow an implementation of an infinite stream of natural numbers from class materials. Or you could write your own from scratch for extra practice.

EXAMPLE: `(stream-factorial 4)` evaluates to `24`

More Bananas

Write a function `repeats` that takes a string and evaluates to a stream the first element of which is the original string, the second is the string repeated twice etc. This is similar to **BananaBanana -- Continued** problems in Standard ML sections, but instead of a finite list we're producing an infinite

stream of repeats. (Note that we could do the same in Standard ML had we defined an appropriate stream datatype. You may try to do that as a challenge.)

Newton's Back For More

We'll revisit the **Newton's Method** problem from Section 3, but instead of returning a single result we will produce an infinite stream of approximations.

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. Guesses sufficiently far into the stream should be very close to \sqrt{n} .

Macros

Perl Style

Write a macro `perform` that has the following two forms:

```
(perform e1 if e2)
(perform e1 unless e2)
```

`e1` should be evaluated (once) depending on the result of evaluating `e2` -- only if `e2` evaluates to `#f` in the latter case, and only if it doesn't in the former case. If `e1` is never evaluated, the entire expression should evaluate to `e2`. Neither `e1` nor `e2` should be evaluated more than once in any case.

HINT: You may want to take a look at special forms for conjunction and/or disjunction in the standard lib.

EXAMPLE: `(perform (print "can't happen!") unless null)` evaluates to `null`, producing no side effects

(*) Problems contributed by Charilaos Skiadas.

↑ 4 ↓ · flag



Pavel Lepin · COMMUNITY TA · 14 days ago 🔗

Added another problem -- Not These Guys Again.

↑ 0 ↓ · flag

[+ Comment](#)



Marco Fabbri · Signature Track · 14 days ago 🔗

Thanks Pavel! I tried myself a (supposedly fun) variation on the Interleave problem: interleaving an arbitrary number of streams (had to use `map`, `range` and define an helper function `(list-replace lst pos el)`). The streams can be passed as a list argument or one can take advantage of the nice variadic function definition `(define (interleave . streams) ...)`.

↑ 0 ↓ · flag



Pavel Lepin COMMUNITY TA · 14 days ago 🔗

That's an interesting (and useful) generalization, indeed.

↑ 0 ↓ · flag



Marco Fabbri Signature Track · 14 days ago 🔗

[SPOILER] Posted a solution to the variadic interleave on <https://gist.github.com/mrfabbri/7b3dbc506c7ba47cbea7>.

↑ 0 ↓ · flag

Pierre Barbier de Reuille · 13 days ago 🔗

Sorry, but I think your solution doesn't work! Instead of using constant streams, try using some generated from lists or the natural numbers.

↑ 1 ↓ · flag



Pavel Lepin COMMUNITY TA · 13 days ago 🔗

Good catch. But it shouldn't be hard to fix.

↑ 0 ↓ · flag

Pierre Barbier de Reuille · 13 days ago 🔗

[SPOILER] I added my solution (well, two actually). The first one is purely functional, but the second one uses mutable pairs organized as a cyclic list: <https://gist.github.com/PierreBdR/b37e5f17073f72370fc4>

Note that most of the second solution is on creating the cyclic list (in term of line of code). There may be a simpler way to do it ... I haven't spent much time on this.

↑ 1 ↓ · flag



Marco Fabbri Signature Track · 13 days ago 🔗

Thanks Pierre! Fixed (funny thing is when I was writing it I actually thought about passing the

next stream iteration rather than the current one but it slipped nonetheless somehow).

PS: Pierre, I'm implementing shadow-stream right now.

↑ 0 ↓ · flag

[+ Comment](#)

New post

To ensure a positive and productive discussion, please read our [forum posting policies](#) before posting.

B	<i>I</i>			Link	<code>	Pic	Math		Edit: Rich ▾	Preview
<div></div>										

☐ Make this post anonymous to other students

☒ Subscribe to this thread at the same time

Add post