Homework 7: Scheme **hw07.zip (hw07.zip)**

Due by 11:59pm on Thursday, November 3

Instructions

Download hw07.zip (hw07.zip). Inside the archive, you will find a file called hw07.scm (hw07.scm), along with a copy of the `ok` autograder.

Submission: When you are done, submit with `python3 ok --submit`. You may submit more than once before the deadline; only the final submission will be scored. Check that you have successfully submitted your code on okpy.org (https://okpy.org/). See Lab 0 (/lab/lab00#submitting-the-assignment) for more instructions on submitting assignments.

Using Ok: If you have any questions about using Ok, please refer to this guide. (/articles/using-ok)

Readings: You might find the following references useful:

- Scheme Specification (/articles/scheme-spec/)
- Scheme Built-in Procedure Reference (/articles/scheme-builtins/)

Grading: Homework is graded based on correctness. Each incorrect problem will decrease the total score by one point. There is a homework recovery policy as stated in the syllabus. **This homework is out of 2 points.**

Scheme is a famous functional programming language from the 1970s. It is a dialect of Lisp (which stands for LISt Processing). The first observation most people make is the unique syntax, which uses a prefix notation and (often many) nested parentheses (see http://xkcd.com/297/ (http://xkcd.com/297/)). Scheme features first-class functions and optimized tail-recursion, which were relatively new features at the time.

You may find it useful to try code.cs61a.org/scheme (https://code.cs61a.org/scheme) when working through problems, as it can draw environment and box-and-pointer diagrams and it lets you walk your code step-by-step (similar to Python Tutor). Don't forget to submit your code through Ok though!

Scheme Editor

You can write your code by either opening the designated `.scm` file in your text editor, or by typing directly in the Scheme Editor, which can also be useful for debugging. To run this editor, run `python3 editor`. This should pop up a window in your browser; if it does not, please navigate to localhost:31415 (localhost:31415) while `python3 editor` is still

running and you should see it. If you choose to code directly in the Scheme Editor, don't forget to save your work before running Ok tests and before closing the editor. To stop running the editor and return to the command line, type `Ctrl-C`.

Make sure to run `python3 ok` in a separate tab or window so that the editor keeps running.

If you find that your code works in the online editor but not in your own interpreter, it's possible you have a bug in your code from an earlier part that you'll have to track down. Every once in a while there's a bug that our tests don't catch, and if you find one you should let us know!

Required Questions

Getting Started Videos

Q1: Thane of Cadr

Define the procedures `cadr` and `caddr`, which return the second and third elements of a list, respectively. If you would like a quick refresher on Scheme syntax consider looking at the Scheme Specification (/articles/scheme-spec/) and Scheme Built-in Procedure Reference (/articles/scheme-builtins/) (and the Lab 10 Scheme Refresher (/lab/lab10/#scheme) when it's released).

```
(define (cddr s)
  (cdr (cdr s)))

(define (cadr s)
  'YOUR-CODE-HERE
)

(define (caddr s)
  'YOUR-CODE-HERE
)
```

Use Ok to unlock and test your code:

```
python3 ok -q cadr-caddr -u
python3 ok -q cadr-caddr
```

Q2: Ascending

Implement a procedure called `ascending?`, which takes a list of numbers `asc-lst` and returns `True` if the numbers are in nondescending order, and `False` otherwise. Numbers are considered nondescending if each subsequent number is either larger or equal to the previous, that is:

```
1 2 3 3 4
```

Is nondescending, but:

```
1 2 3 3 2
```

Is not.

> **Hint**: The built-in `null?` function returns whether its argument is `nil`.
>
> **Note**: The question mark in `ascending?` is just part of the function name and has no special meaning in terms of Scheme syntax. It is a common practice in Scheme to name a function with a question mark at the end if the function returns a boolean value indicating whether or not a condition is satisfied. For instance, `ascending?` is a function that essentially asks "Is the argument in ascending order?", `null?` is a function that asks "Is the argument `nil`?", `even?` asks "Is the argument even?", etc.

```
(define (ascending? asc-lst)
  'YOUR-CODE-HERE
)
```

Use Ok to unlock and test your code:

```
python3 ok -q ascending -u
python3 ok -q ascending
```

Q3: Pow

Implement a procedure `pow` for raising the number `base` to the power of a nonnegative integer `exp` for which the number of operations grows logarithmically, rather than linearly (the number of recursive calls should be much smaller than the input `exp`). For example, for `(pow 2 32)` should take 5 recursive calls rather than 32 recursive calls. Similarly, `(pow 2 64)` should take 6 recursive calls.

Hint: Consider the following observations:

1. $x^{2y} = (x^y)^2$
2. $x^{2y+1} = x(x^y)^2$

For example we see that 2^{32} is $(2^{16})^2$, 2^{16} is $(2^8)^2$, etc. You may use the built-in predicates `even?` and `odd?`. Scheme doesn't support iteration in the same manner as Python, so consider another way to solve this problem.

```
(define (square n) (* n n))

(define (pow base exp)
  'YOUR-CODE-HERE
)
```

Use Ok to unlock and test your code:

```
python3 ok -q pow -u
python3 ok -q pow
```