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

Due by 11:59pm on Thursday, November 2

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 the assignment by uploading all code files you've edited to Gradescope. 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 Gradescope. See Lab 0 (/lab/lab00#submitting-the-assignment) for more instructions on submitting assignments.

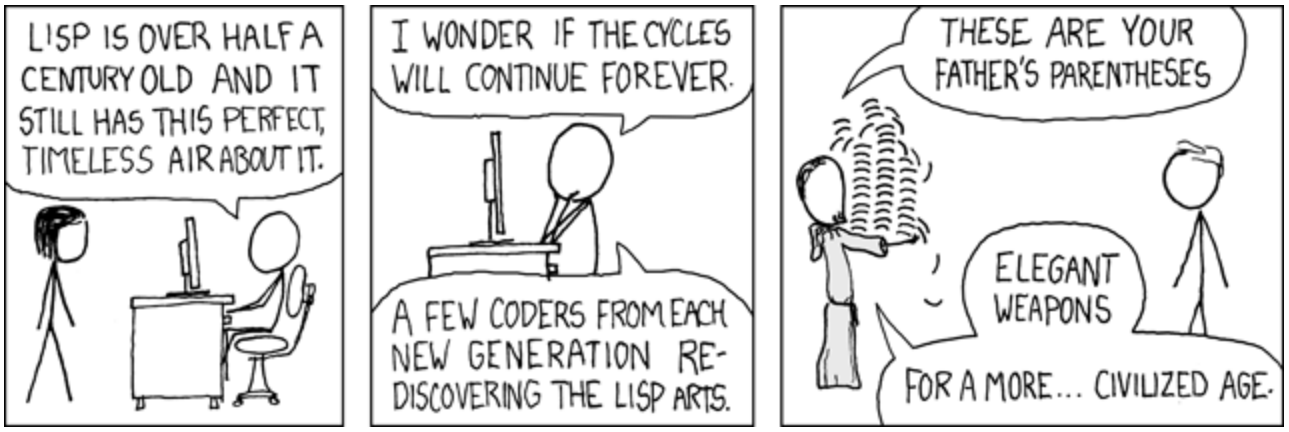
Using Ok: If you have any questions about using Ok, please refer to this guide. (/articles/using-ok)

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 syntax of Scheme is very unique: it involves prefix notation and many nested parentheses (see http://xkcd.com/297/ (http://xkcd.com/297/)). Scheme features first-class functions and optimized tail-recursion, which were fairly new when Scheme was introduced.



Recommended VS Code Extensions

If you use VS Code as your text editor, we have found these extensions to be quite helpful for Scheme :). You only need to install one because both do syntax coloring!

Before:

```
(define foo (lambda (x y z) (if x y z)))

(foo 1 2 (print 'hi))

((lambda (a) (print 'a)) 100)
```

After:

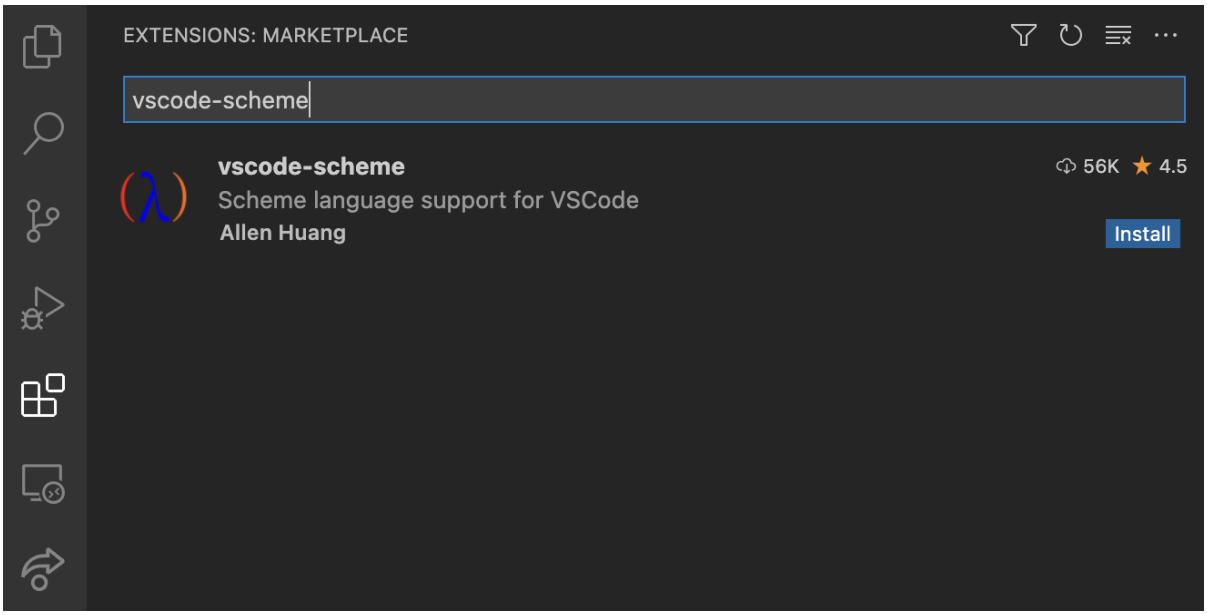
```
(define foo (lambda (x y z) (if x y z)))

(foo 1 2 (print 'hi))

((lambda (a) (print 'a)) 100)
```

Extensions:

vscode-scheme (https://marketplace.visualstudio.com/items?itemName=sjhuangx.vscode-scheme)



61a-bot (/articles/61a-bot)

61A Bot is an experimental extension for VS Code and is powered by GPT-4 through Microsoft's Azure OpenAI service. Microsoft has generously sponsored this project for the Fall 2023 semester, and so using this extension is free to both students and the university.

The extension itself adds a "Get 61A Help" button to the top of the main editor that appears whenever the editor loads a 61A assignment file. Clicking this "Get 61A Help" button sends parts of your currently-open code to Microsoft Azure and OpenAI's GPT-4 model, so make sure you're comfortable doing so before clicking the button.

Installation instructions **here (/articles/61a-bot)**.

There is also a 61A Bot integration with OK that will be automatically run whenever you run OK on homework problems going forward this semester. After running `python3 ok` you will see a prompt to get help on the current problem. You can accept by typing `y` and hitting Enter. You can decline by just hitting Enter. If you have questions about this project, you can ask on Ed (https://edstem.org/us/courses/42930/discussion/3754057)

Scheme Editor

You can write your code by opening the designated `.scm` file in your text editor.

You can also type directly into the 61A Scheme Editor, which provides testing and debugging tools. To open the 61A Scheme Editor, run `python3 editor` inside the `hw07` folder. This should pop up a window in your browser; if it does not, please navigate to localhost:31415 (localhost:31415) while `python3 editor` is running and you should see it.

If you choose to code in the 61A Scheme Editor, don't forget to save your work before running Ok tests and before closing the editor. Make sure to run `python3 ok` in a separate tab or window so that the editor keeps running. To stop running the editor and return to the command line, type `Ctrl-C`.

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!

You may find code.cs61a.org/scheme (https://code.cs61a.org/scheme) useful. This Scheme interpreter can draw environment and box-and-pointer diagrams. It also lets you walk through your code step-by-step, like Python Tutor. But don't forget to submit your code to the appropriate Gradescope assignment!

Required Questions

Getting Started Videos

Q1: Pow

Implement a procedure `pow` that raises a `base` to the power of a nonnegative integer `exp`. The number of recursive `pow` calls should grow logarithmically with respect to `exp`, rather than linearly. For example, `(pow 2 32)` should result in 5 recursive `pow` calls rather than 32 recursive `pow` calls.

Refer to the Scheme Specification (/articles/scheme-spec/), the Scheme Built-in Procedure Reference (/articles/scheme-builtins/), and the Lab 10 Scheme Refresher (/lab/lab10/#scheme) to review Scheme syntax.

> *Hint:* Consider the following observations:
>
> 1. $x^{2y} = (x^y)^2$
> 2. $x^{2y+1} = x(x^y)^2$
>
> For example, $2^{16} = (2^8)^2$ and $2^{17} = 2 * (2^8)^2$.
>
> You may use the built-in predicates `even?` and `odd?`. Also, the `square` procedure is defined for you.
>
> Keep in mind that Scheme doesn't support iteration in the same manner as Python, so you'll need recursion to solve this problem.

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

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

Use Ok to test your code:

```
python3 ok -q pow
```

There is an experimental 61A Bot that will be automatically run whenever you run OK on homework problems going forward this semester. After running `python3 ok` you will see a prompt to get help on the current problem. You can accept by typing `y` and hitting Enter. You

can decline by just hitting Enter.

There is also a VS Code Extension for 61A Bot. Read more under recommended-vs-code-extensions.

Q2: Repeatedly Cube

Implement `repeatedly-cube`, which receives a number `x` and cubes it `n` times.

Here are some examples of how `repeatedly-cube` should behave:

```
scm> (repeatedly-cube 100 1) ; 1 cubed 100 times is still 1
1
scm> (repeatedly-cube 2 2) ; (2^3)^3
512
scm> (repeatedly-cube 3 2) ; ((2^3)^3)^3
134217728
```

> For information on `let`, see the Scheme spec (/articles/scheme-spec/#let).

```
(define (repeatedly-cube n x)
    (if (zero? n)
        x
        (let
            (_________________)
            (* y y y))))
```

Use Ok to test your code:

```
python3 ok -q repeatedly-cube
```

Q3: Thane of Cadr

Note: *Scheme lists will be covered in lecture on Wednesday, November 1. If you are working ahead, read the Scheme Specification (/articles/scheme-spec/) for details on `car` and `cdr`.*

Define the procedure `cadr`, which returns the second element of a list. Also define `caddr`, which returns the third element of a list.

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

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

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

Use Ok to test your code:

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