Homework 6 Solutions **hw06.zip (hw06.zip)**

Solution Files

Scheme Editor

How to launch

In your `hw06` folder you will find a new editor. 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) and you should see it.

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

Features

The `hw06.scm` file should already be open. You can edit this file and then run `Run` to run the code and get an interactive terminal or `Test` to run the `ok` tests.

`Environments` will help you diagram your code, and `Debug` works with environments so you can see where you are in it. We encourage you to try out all these features.

`Reformat` is incredibly useful for determining whether you have parenthesis based bugs in your code. You should be able to see after formatting if your code looks weird where the issue is.

By default, the interpreter uses Lisp-style formatting, where the parens are all put on the end of the last line

```
(define (f x)
    (if (> x 0)
        x
        (- x)))
```

However, if you would prefer the close parens to be on their own lines as so

```
(define (f x)
    (if (> x 0)
        x
        (- x)
    )
)
```

you can go to Settings and select the second option.

Warning

The editor will neither back up nor submit on your behalf. You still need to run `python ok -u` to unlock cases from the terminal as well as `python ok --submit`.

Reporting bugs

This is relatively new software! While we have tested it extensively, it probably has bugs. Please let us know on piazza if there is an issue.

Questions

Q1: Survey

Please fill out the survey at this link (https://docs.google.com/forms/d/1SUFnD0HKODdoHqkllocJ3RIGbBU2B0pMd3dDtT4O_kA/viewform) and fill in `hw06.py` with the token. The link might not work if you are logged into some google account other than your Berkeley account, so either log out from all other accounts or open the link in a private/incognito window and sign in to your Berkeley account there.

To check that you got the correct token run

Use Ok to test your code:

```
python3 ok -q survey
```

Scheme

Q2: Thane of Cadr

Define the procedures `cadr` and `caddr`, which return the second and third elements of a list, respectively:

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

(define (cadr s)
  (car (cdr s))
)

(define (caddr s)
  (car (cddr s))
)
```

Following the given example of `cddr`, defining `cadr` and `caddr` should be fairly straightforward.

Just for fun, if this were a Python linked list question instead, the solution might look something like this:

```
cadr = lambda l: l.rest.first
caddr = lambda l: l.rest.rest.first
```

Use Ok to unlock and test your code:

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

Q3: Sign

Using a `cond` expression, define a procedure `sign` that takes in one parameter `x` and returns -1 if `x` is negative, 0 if `x` is zero, and 1 if `x` is positive.

```
(define (sign x)
  (cond ((> x 0) 1)
        ((= x 0) 0)
        ((< x 0) -1))
)
```

The order of the cases doesn't really matter, and we could also use an `else` clause in the place of one of the conditions.

The condition looks something like this in Python:

```
if x > 0:
    return 1
elif x == 0:
    return 0
elif x < 0:
    return -1
```

Opinions differ on which is better, but hopefully you can see that the Scheme `cond` is quite readable and compact once you get used to it.

Use Ok to unlock and test your code:

```
python3 ok -q sign -u
python3 ok -q sign
```

Q4: Pow

Implement a procedure `pow` for raising the number `b` to the power of a nonnegative integer `n` for which the number of operations grows logarithmically (as opposed to linearly).

> *Hint:* Consider the following observations:
>
> 1. $b^{2k} = (b^k)^2$
> 2. $b^{2k+1} = b(b^k)^2$
>
> You may use the built-in predicates `even?` and `odd?`.

```
(define (square x) (* x x))

(define (pow b n)
  (cond ((= n 0) 1)
        ((even? n) (square (pow b (/ n 2))))
        (else (* b (pow b (- n 1)))))
)
```

Use Ok to unlock and test your code:

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

The `else` clause shows the basic recursive version of `pow` that we've seen before in class.

We save time in computation by avoiding an extra n/2 multiplications of the base. Instead, we just square the result of `b^(n/2)`.

Video walkthrough: https://youtu.be/-yt2EOv9ZLU (https://youtu.be/-yt2EOv9ZLU)

Q5: Unique

Implement `unique`, which takes in a list `s` and returns a new list containing the same elements as `s` with duplicates removed.

```
scm> (unique '(1 2 1 3 2 3 1))
(1 2 3)
scm> (unique '(a b c a a b b c))
(a b c)
```

> Hint: you may find it useful to use the built-in `filter` procedure. See the built-in procedure reference (/articles/scheme-builtins.html) for more information.

```
(define (unique s)
  (if (null? s) nil
      (cons (car s)
            (unique (filter (lambda (x) (not (eq? (car s) x))) (cdr s)))))
)
```

Use Ok to test your code:

```
python3 ok -q unique
```

Submit

Make sure to submit this assignment by running:

```
python3 ok --submit
```

CS 61A (/)

Weekly Schedule (/weekly.html)

Office Hours (/office-hours.html)

Staff (/staff.html)

Resources (/resources.html)

Studying Guide (/articles/studying.html)

Debugging Guide (/articles/debugging.html)

Composition Guide (/articles/composition.html)

Policies (/articles/about.html)

Assignments (/articles/about.html#assignments)

Exams (/articles/about.html#exams)

Grading (/articles/about.html#grading)