Homework 6: Scheme

> Adapted from cs61a of UC Berkeley.

Readings

You might find the following references useful:

- Scheme Specification
  - In the same folder with this handout.
- Scheme Built-in Procedure Reference
  - In the same folder with this handout.
- Section 3.2

Starter Files

Get your starter file by cloning the repository: https://github.com/JacyCui/sicp-hw06.git

```
git clone https://github.com/JacyCui/sicp-hw06.git
```

`hw06.zip` is the starter file you need, you might need to unzip the file to get the skeleton code.

```
unzip hw06.zip
```

`README.md` is the handout for this homework. `solution` is a probrab solution of the homework. However, I might not give my solution exactly when the homework is posted. You need to finish the task on your own first. If any problems occurs, please make use of the comment section.

Scheme Editor

As you're writing your code, you can debug using the Scheme Editor. In your `scheme` 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 and you should see it.

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 code from an earlier part that you'll have to track down.

Required Questions

Scheme

Q1: 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)
  'YOUR-CODE-HERE
)

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

Use Ok to unlock and test your code:

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

Q2: Sign

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

```
(define (sign num)
  'YOUR-CODE-HERE
)
```

Use Ok to unlock and test your code:

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

Q3: Pow

Implement a procedure `pow` for raising the number `x` to the power of a nonnegative integer `y` for which the number of operations grows logarithmically, rather than linearly (the number of recursive calls should be much smaller than the input `y`). 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. $b^{2k} = (b^k)^2$
> 2. $b^{2k+1} = b(b^k)^2$
>
> 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 x) (* x x))

(define (pow x y)
  'YOUR-CODE-HERE
)
```

Use Ok to unlock and test your code:

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