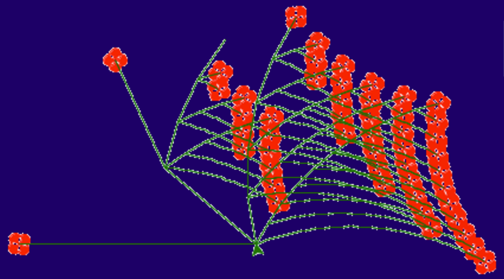
SICP

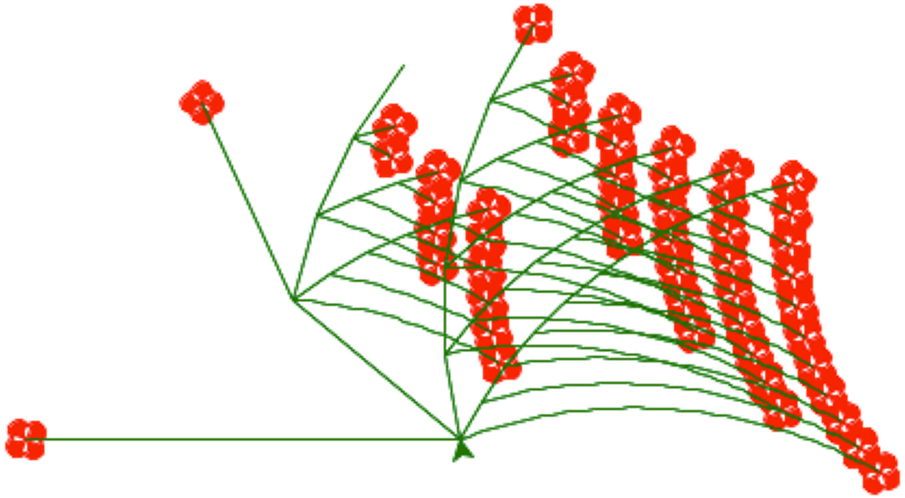
God’s Programming Book



Project-04 Scheme

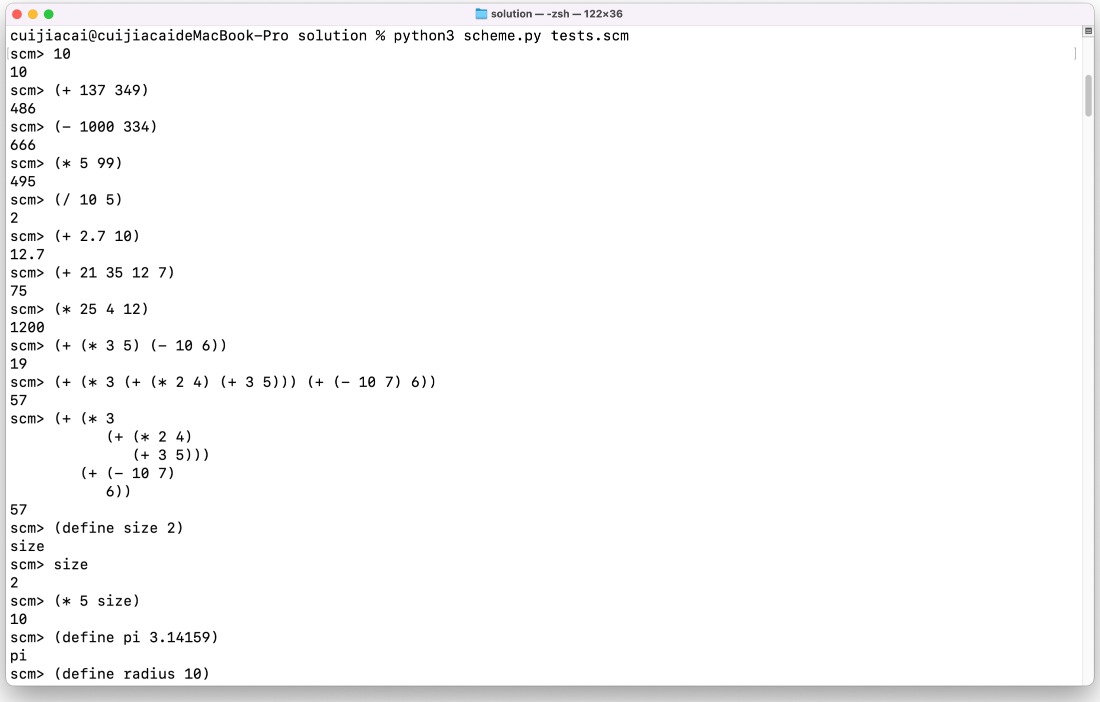
Scheme Interpreter

Project Adapted from cs61a of UC Berkeley

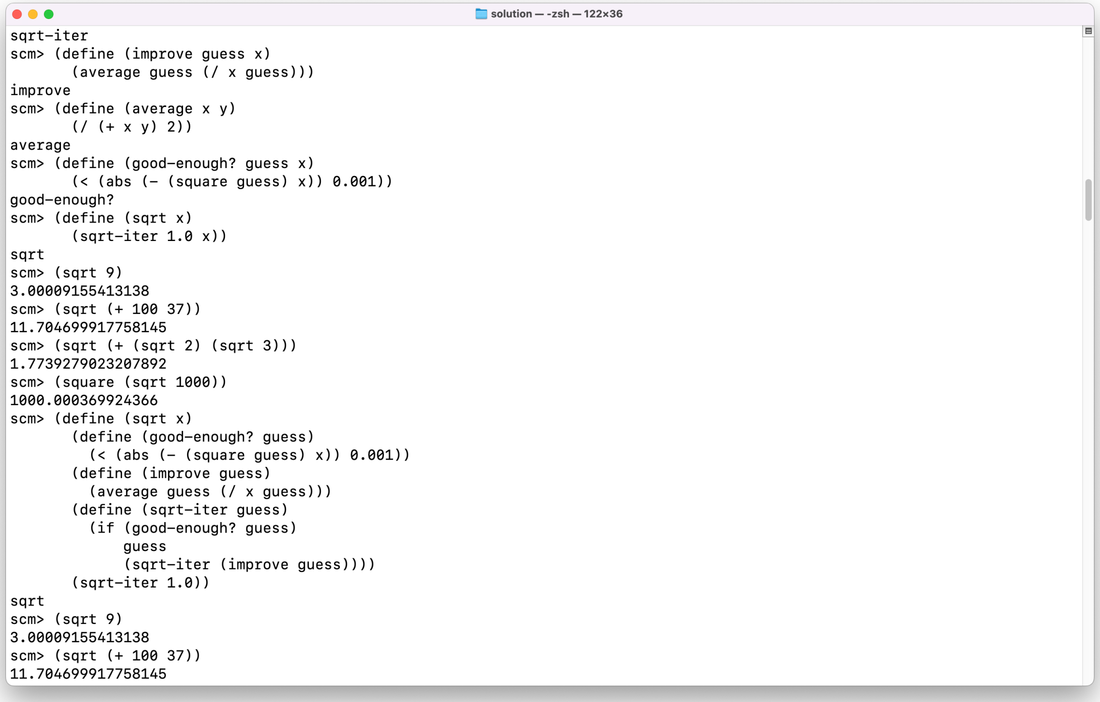


Goal

What will you have after the project?



```
cuijiacai@cuijiacaideMacBook-Pro solution % python3 scheme.py tests.scm
scm> 10
10
scm> (+ 137 349)
486
scm> (- 1000 334)
666
scm> (* 5 99)
495
scm> (/ 10 5)
2
scm> (+ 2.7 10)
12.7
scm> (+ 21 35 12 7)
75
scm> (* 25 4 12)
1200
scm> (+ (* 3 5) (- 10 6))
19
scm> (+ (* 3 (+ (* 2 4) (+ 3 5))) (+ (- 10 7) 6))
57
scm> (+ (* 3
           (+ (* 2 4)
              (+ 3 5)))
        (+ (- 10 7)
           6))
57
scm> (define size 2)
size
scm> size
2
scm> (* 5 size)
10
scm> (define pi 3.14159)
pi
scm> (define radius 10)
```



```
sqrt-iter
scm> (define (improve guess x)
       (average guess (/ x guess)))
improve
scm> (define (average x y)
       (/ (+ x y) 2))
average
scm> (define (good-enough? guess x)
       (< (abs (- (square guess) x)) 0.001))
good-enough?
scm> (define (sqrt x)
       (sqrt-iter 1.0 x))
sqrt
scm> (sqrt 9)
3.00009155413138
scm> (sqrt (+ 100 37))
11.704699917758145
scm> (sqrt (+ (sqrt 2) (sqrt 3)))
1.7739279023207892
scm> (square (sqrt 1000))
1000.000369924366
scm> (define (sqrt x)
       (define (good-enough? guess)
         (< (abs (- (square guess) x)) 0.001))
       (define (improve guess)
         (average guess (/ x guess)))
       (define (sqrt-iter guess)
         (if (good-enough? guess)
             guess
             (sqrt-iter (improve guess))))
       (sqrt-iter 1.0))
sqrt
scm> (sqrt 9)
3.00009155413138
scm> (sqrt (+ 100 37))
11.704699917758145
```

A Fully Implemented Scheme Interpreter

Materials

What have you got before the project?

- Skeleton code of the project and an autograder ok
- A detailed handout covering everything about the project
- My version of solution

https://github.com/JacyCui/sicp-proj04.git

Implementation Overview

Read

This step parses user input (a string of Scheme code) into our interpreter's internal Python representation of Scheme expressions (e.g. Pairs).

- ***Lexical analysis*** has already been implemented for you in the tokenize_lines function in scheme_tokens.py . This function returns a Buffer (from buffer.py) of tokens.
- ***Syntactic analysis*** happens in scheme_reader.py , in the scheme_read and read_tail functions. Together, these mutually recursive functions parse Scheme tokens into our interpreter's internal Python representation of Scheme expressions. You will complete both functions.

Eval

This step evaluates Scheme expressions (represented in Python) to obtain values. Code for this step is in the main scheme.py file.

- ***Eval*** happens in the scheme_eval function.
    - If the expression is a call expression, it gets evaluated according to the rules for evaluating call expressions (you will implement this).
    - If the expression being evaluated is a special form, the corresponding do_?_form function is called. You will complete several of the do_?_form functions.

Eval

This step evaluates Scheme expressions (represented in Python) to obtain values. Code for this step is in the main scheme.py file.

- ***Apply*** happens in the scheme_apply function.
    - If the function is a built-in procedure, scheme_apply calls the apply method of that BuiltInProcedure instance.
    - If the procedure is a user-defined procedure, scheme_apply creates a new call frame and calls eval_all on the body of the procedure, resulting in a mutually recursive eval-apply loop.

Print & Loop

- **Print**: This step prints the __str__ representation of the obtained value.
- **Loop**: The logic for the loop is handled by the read_eval_print_loop function in scheme.py .

Exceptions

- As you develop your Scheme interpreter, you may find that Python raises various uncaught exceptions when evaluating Scheme expressions. As a result, your Scheme interpreter will halt.
- Some of these may be the results of bugs in your program, but some might just be errors in user programs.
  - The former should be fixed by debugging your interpreter and the latter should be handled by your code, usually by raising a SchemeError .
  - All SchemeError exceptions are handled and printed as error messages by the read_eval_print_loop function in scheme.py .
- Ideally, there should never be unhandled Python exceptions for any input to your interpreter.

Requirements

What do you need to finished this project?

- Representation
- Composition
- Efficiency
- Scheme
- Exceptions
- Calculator
- Interpreters

Tips

What you should keep in mind?

- Always figure out what you need to do before writing codes.
- Keep it simple and elegant.
  - Normally no more than 20 lines for each problem.
- Take a challenge to conquer all the optional problems.

Thanks for Listening