The Invention of Scheme, 1762 (colorized)

Hi! Welcome to 61A Discussion :)

We will begin at **5:10**!

Attendance form and skeleton notes:

cs61a.bencuan.me

Secret word:

Announcements

- Scheme part 2 and 3 due next Tues
- Scheme part 4 due next next Tues

Agenda

- Attendance
- Programs as Data

What is “programs as data”?

- Code is basically just a bunch of text
- So, programs are basically just text
- What if we ran the output of code (i.e. text) as if it were *also* code?
- We can then make code that writes more code

How do we do this in practice?

- Everything is a list in Scheme
- Scheme `(eval)` `(apply)` procedures
- Quotes and Quasiquotes
- Macros

Eval and Apply

- Eval takes in a list of literals and puts it into the interpreter
  - (eval ‘(+ 1 2)) becomes 3
  - (eval ‘(if (= 1 1) 9 -2)) becomes 9
- Apply takes in an operator and a list of operators, and applies the operator
  - (apply + ‘(1 2)) becomes 3

Q2

Quasiquotes

- If we want to make a list containing both quoted and unquoted expressions:
  - (define world 10)
  - (list ‘hello world) => (hello 10)
  - ` (hello ,world) => (hello 10)

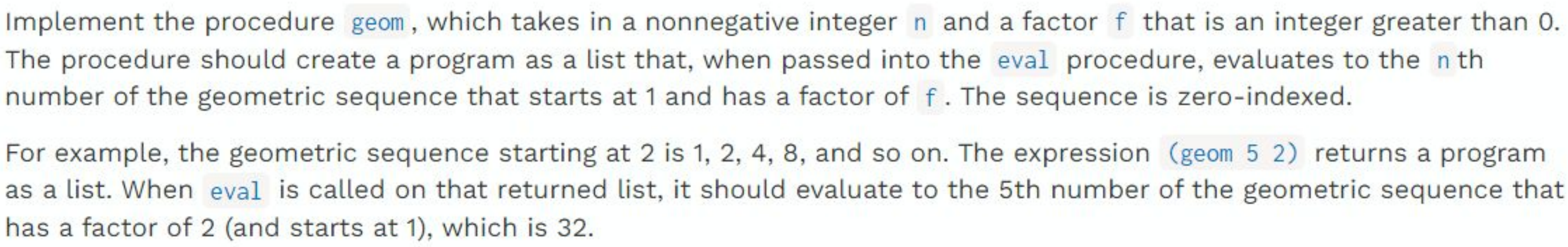
Quasiquotes

- Quote: ‘
- Quasiquote: `
- Unquote: ,

Everything in a quasiquoted expression is quoted by default!

Q1

Q3: Geometric Sequence



Implement the procedure `geom`, which takes in a nonnegative integer `n` and a factor `f` that is an integer greater than 0. The procedure should create a program as a list that, when passed into the `eval` procedure, evaluates to the `n`th number of the geometric sequence that starts at 1 and has a factor of `f`. The sequence is zero-indexed.

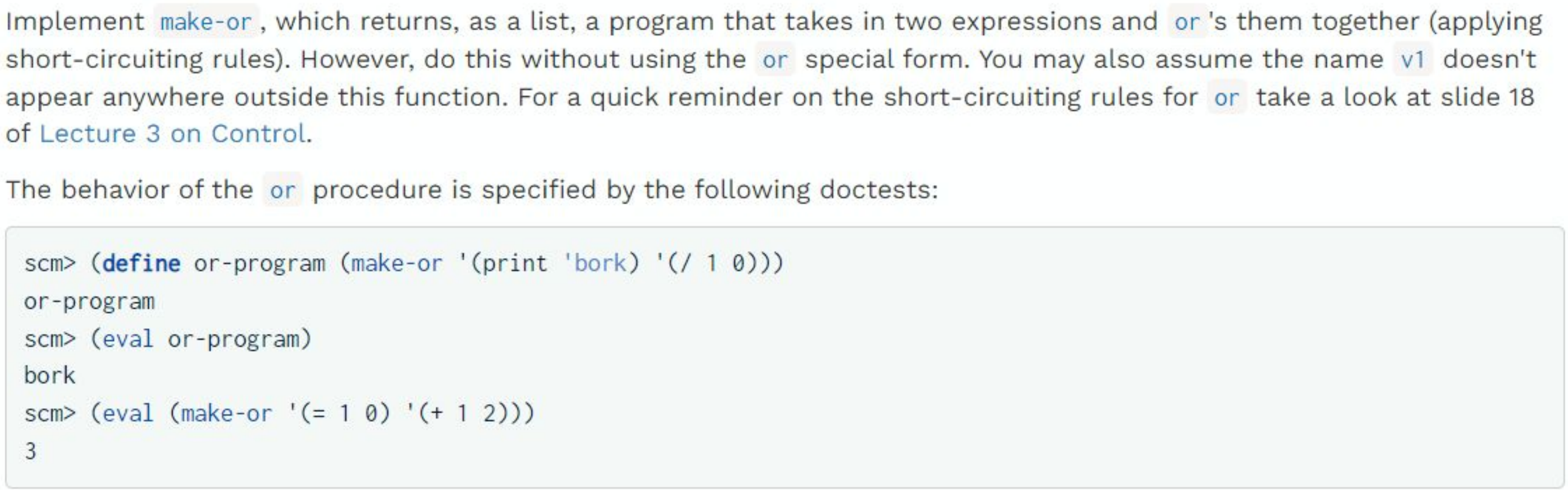
For example, the geometric sequence starting at 2 is 1, 2, 4, 8, and so on. The expression `(geom 5 2)` returns a program as a list. When `eval` is called on that returned list, it should evaluate to the 5th number of the geometric sequence that has a factor of 2 (and starts at 1), which is 32.

Step 1: write the code that does the multiplication

Step 2: write a Scheme list representing that code

Step 3: write a function that returns that Scheme list

Q4: Make Or



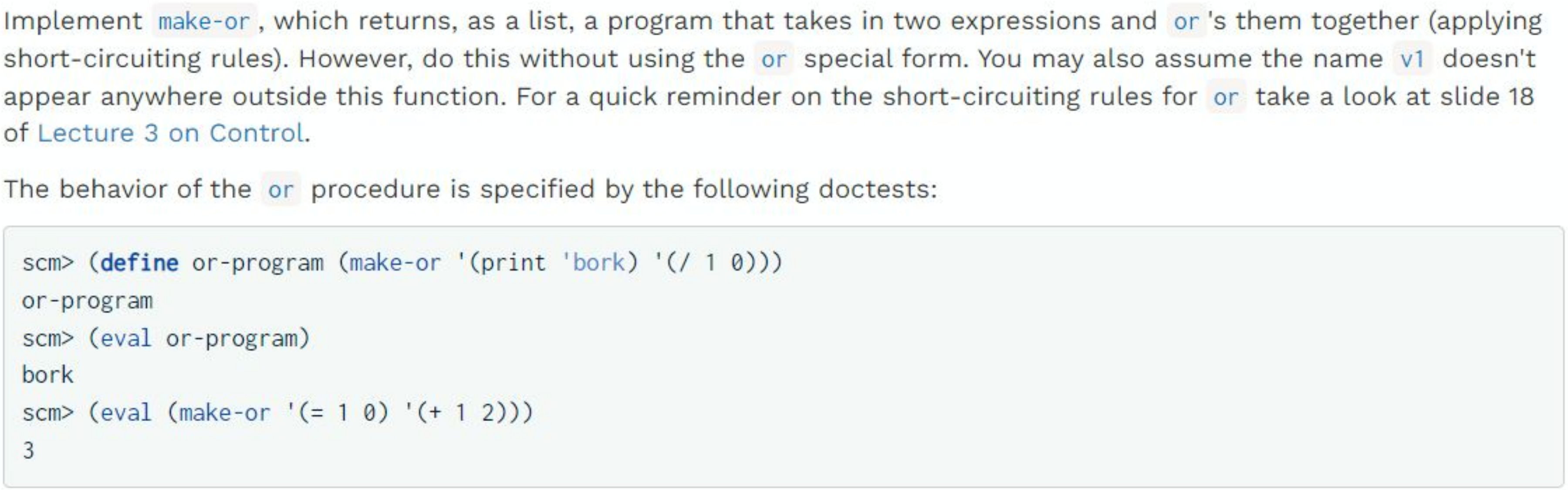
Implement `make-or`, which returns, as a list, a program that takes in two expressions and `or`'s them together (applying short-circuiting rules). However, do this without using the `or` special form. You may also assume the name `v1` doesn't appear anywhere outside this function. For a quick reminder on the short-circuiting rules for `or` take a look at slide 18 of Lecture 3 on Control.

The behavior of the `or` procedure is specified by the following doctests:

```
scm> (define or-program (make-or '(print 'bork) '(/ 1 0)))
or-program
scm> (eval or-program)
bork
scm> (eval (make-or '(= 1 0) '(+ 1 2)))
3
```

Or logic: if expr1 is true, what do you return? what if expr1 is false?

Q5: Make Make Or



Implement `make-or`, which returns, as a list, a program that takes in two expressions and `or`'s them together (applying short-circuiting rules). However, do this without using the `or` special form. You may also assume the name `v1` doesn't appear anywhere outside this function. For a quick reminder on the short-circuiting rules for `or` take a look at slide 18 of Lecture 3 on Control.

The behavior of the `or` procedure is specified by the following doctests:

```
scm> (define or-program (make-or '(print 'bork) '(/ 1 0)))
or-program
scm> (eval or-program)
bork
scm> (eval (make-or '(= 1 0) '(+ 1 2)))
3
```

Hint: write the answer to Q4 as a list!