

Development

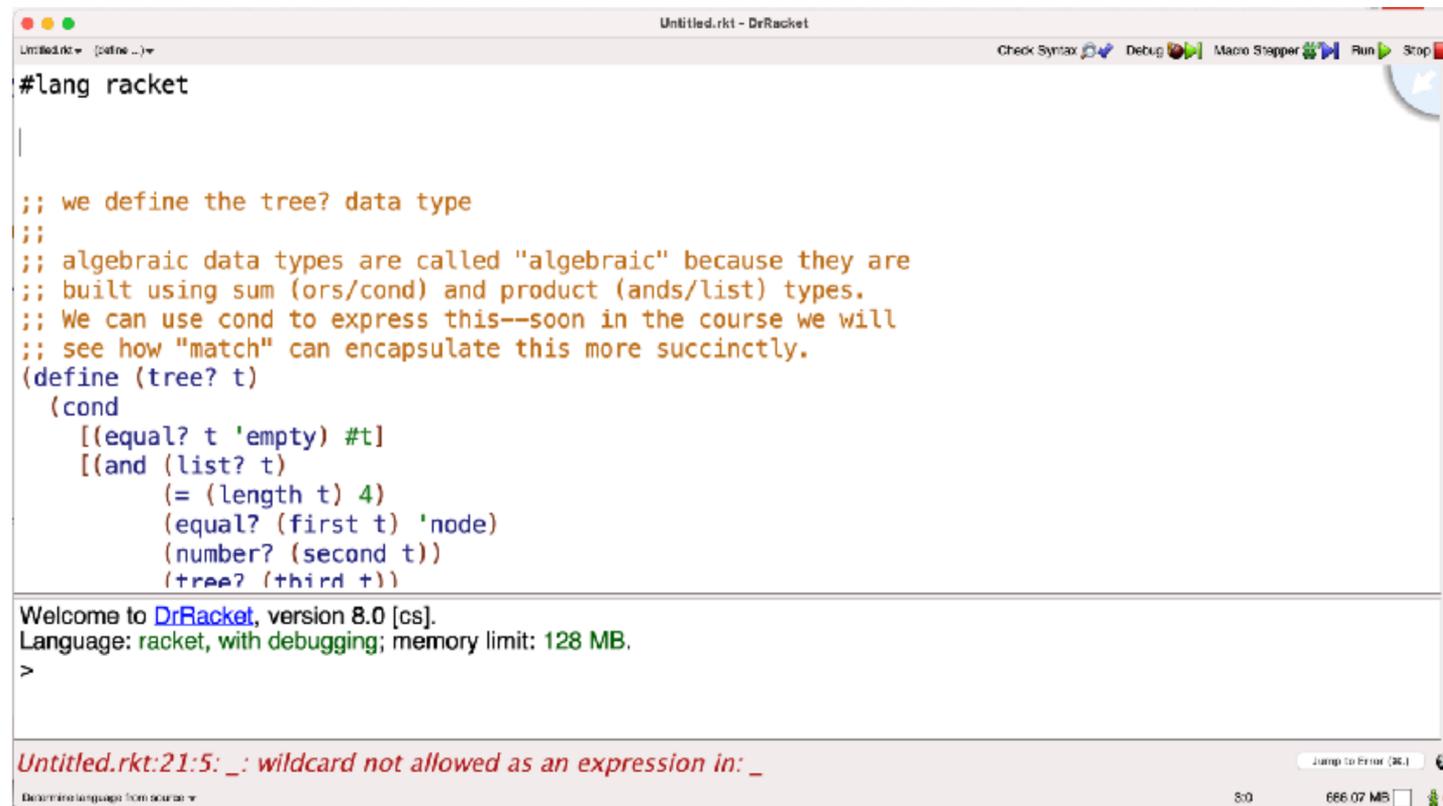
Experience

CIS352 — Fall 2022

Kris Micinski

Development Environment

- I keep two windows open:
 - (a) an editor (Dr. Racket, emacs, VSCode, ...)
 - (b) a command-line application (iTerm2)



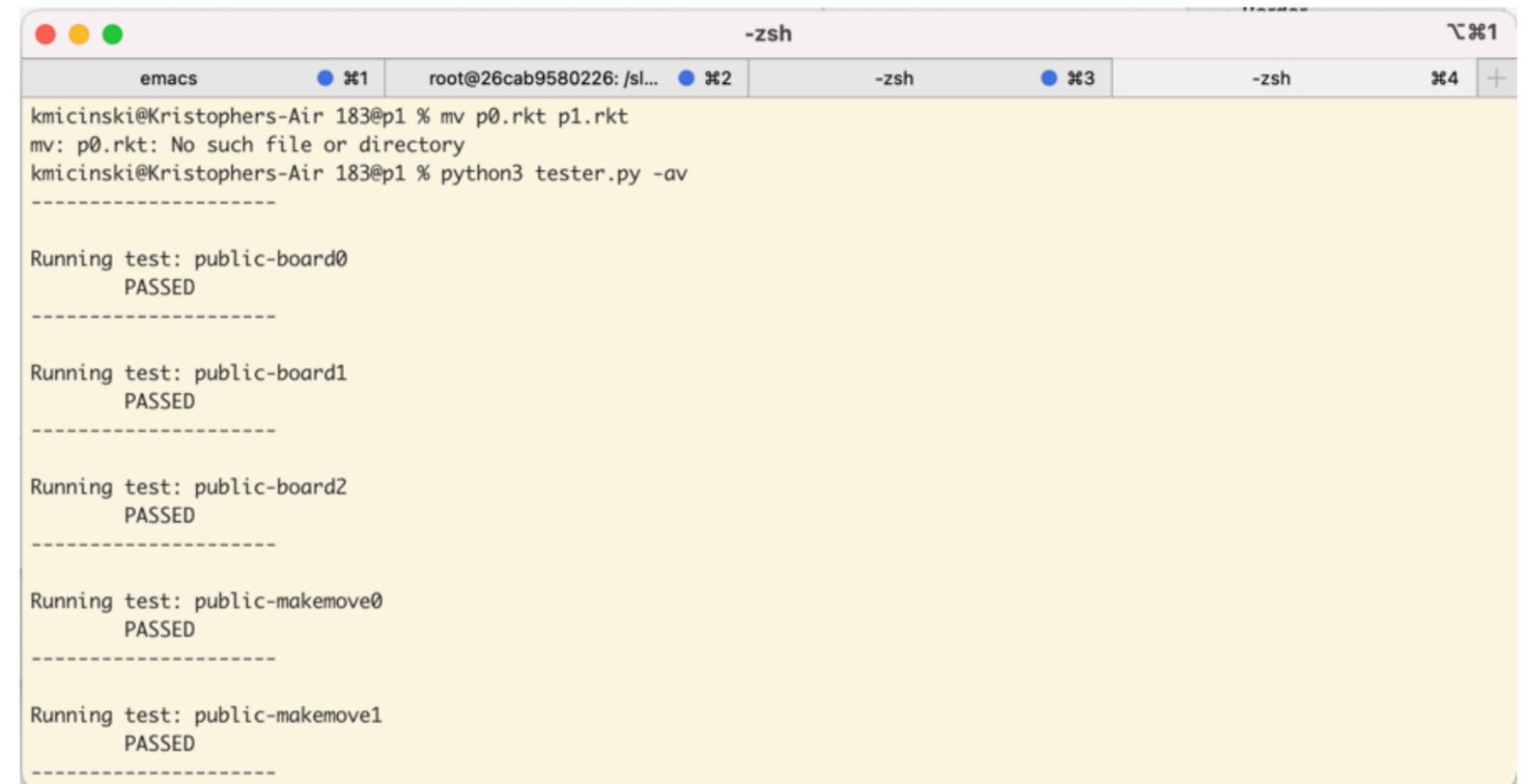
Untitled.rkt - DrRacket

```
#lang racket

;; we define the tree? data type
;;
;; algebraic data types are called "algebraic" because they are
;; built using sum (ors/cond) and product (ands/list) types.
;; We can use cond to express this--soon in the course we will
;; see how "match" can encapsulate this more succinctly.
(define (tree? t)
  (cond
    [(equal? t 'empty) #t]
    [(and (list? t)
          (= (length t) 4)
          (equal? (first t) 'node)
          (number? (second t))
          (tree? (third t)))])])
```

Welcome to **DrRacket**, version 8.0 [cs].
Language: racket, with debugging; memory limit: 128 MB.
>

Untitled.rkt:21:5: _: wildcard not allowed as an expression in: _

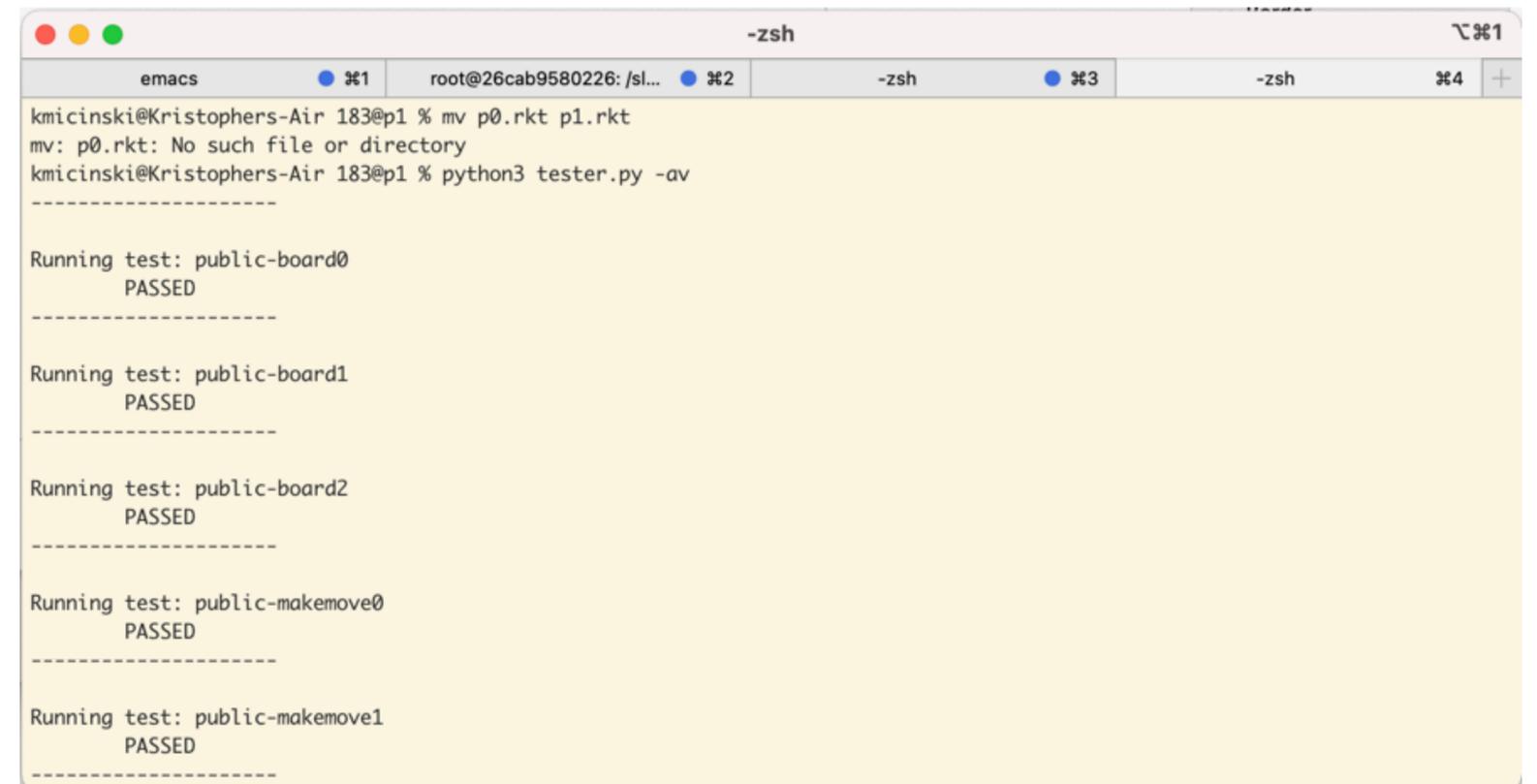


-zsh

```
kmicinski@Kristophers-Air 183@p1 % mv p0.rkt p1.rkt
mv: p0.rkt: No such file or directory
kmicinski@Kristophers-Air 183@p1 % python3 tester.py -av
-----
Running test: public-board0
PASSED
-----
Running test: public-board1
PASSED
-----
Running test: public-board2
PASSED
-----
Running test: public-makemove0
PASSED
-----
Running test: public-makemove1
PASSED
-----
```

Starting my development

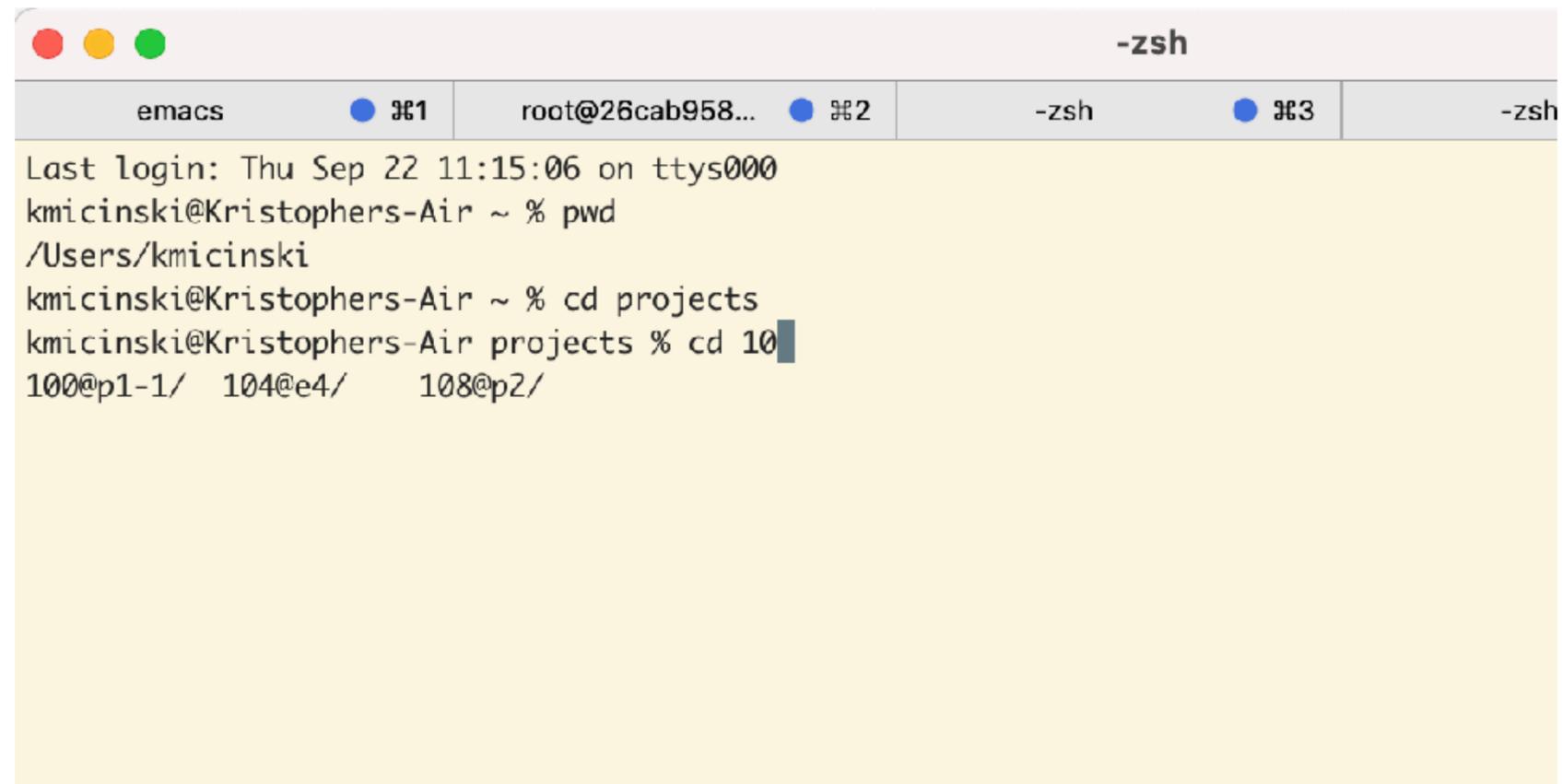
- Every day when I begin my work I:
 - (a) open a new tab in the command line
 - (b) navigate to the project folder I want
 - Everything kept in git, so this is a git repo

A terminal window with a yellow background and a title bar that reads '-zsh'. The window contains several tabs: 'emacs', 'root@26cab9580226: /sl...', and two '-zsh' tabs. The terminal output shows a user named 'kmicinski' at 'Kristophers-Air 183@p1' running a series of commands. The first command is 'mv p0.rkt p1.rkt', which fails with the error 'mv: p0.rkt: No such file or directory'. The second command is 'python3 tester.py -av', which runs a series of tests. Each test is preceded by a dashed line and followed by 'PASSED'. The tests are: 'public-board0', 'public-board1', 'public-board2', 'public-makemove0', and 'public-makemove1'.

```
kmicinski@Kristophers-Air 183@p1 % mv p0.rkt p1.rkt
mv: p0.rkt: No such file or directory
kmicinski@Kristophers-Air 183@p1 % python3 tester.py -av
-----
Running test: public-board0
PASSED
-----
Running test: public-board1
PASSED
-----
Running test: public-board2
PASSED
-----
Running test: public-makemove0
PASSED
-----
Running test: public-makemove1
PASSED
-----
```

Useful commands

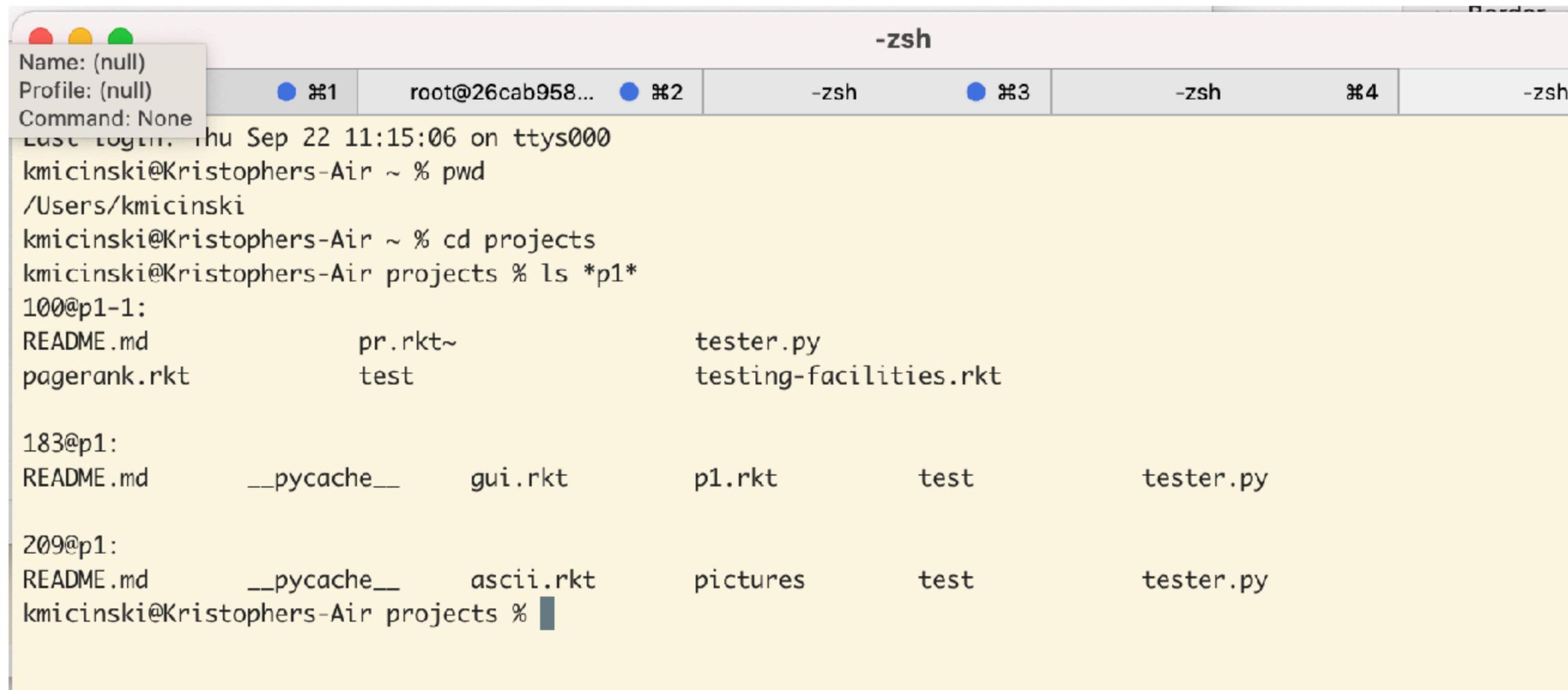
- When I open up the command line, I'm in my **home** directory
- Use **cd** to change into the directory I want
- Use **tab completion** *always* when I use the shell
- You should too!



```
emac  root@26cab958...  -zsh
Last login: Thu Sep 22 11:15:06 on ttys000
kmicinski@Kristophers-Air ~ % pwd
/Users/kmicinski
kmicinski@Kristophers-Air ~ % cd projects
kmicinski@Kristophers-Air projects % cd 10
100@p1-1/ 104@e4/ 108@p2/
```

Globs

- You can use search patterns (“globs”) with most commands
 - Regular-expression-like language (not standard)
- Lets me search ***p1*** to say “find anything with p1 in it”



A terminal window titled "-zsh" showing a user named kmicinski navigating to a directory and using the 'ls' command with a glob pattern '*p1*'. The output shows three directories: 100@p1-1, 183@p1, and 209@p1, each containing various files and subdirectories.

```
Name: (null)
Profile: (null)
Command: None
Thu Sep 22 11:15:06 on ttys000
kmicinski@Kristophers-Air ~ % pwd
/Users/kmicinski
kmicinski@Kristophers-Air ~ % cd projects
kmicinski@Kristophers-Air projects % ls *p1*
100@p1-1:
README.md          pr.rkt~           tester.py
pagerank.rkt       test              testing-facilities.rkt

183@p1:
README.md          __pycache__       gui.rkt           p1.rkt           test             tester.py

209@p1:
README.md          __pycache__       ascii.rkt         pictures         test             tester.py
kmicinski@Kristophers-Air projects %
```

Git status

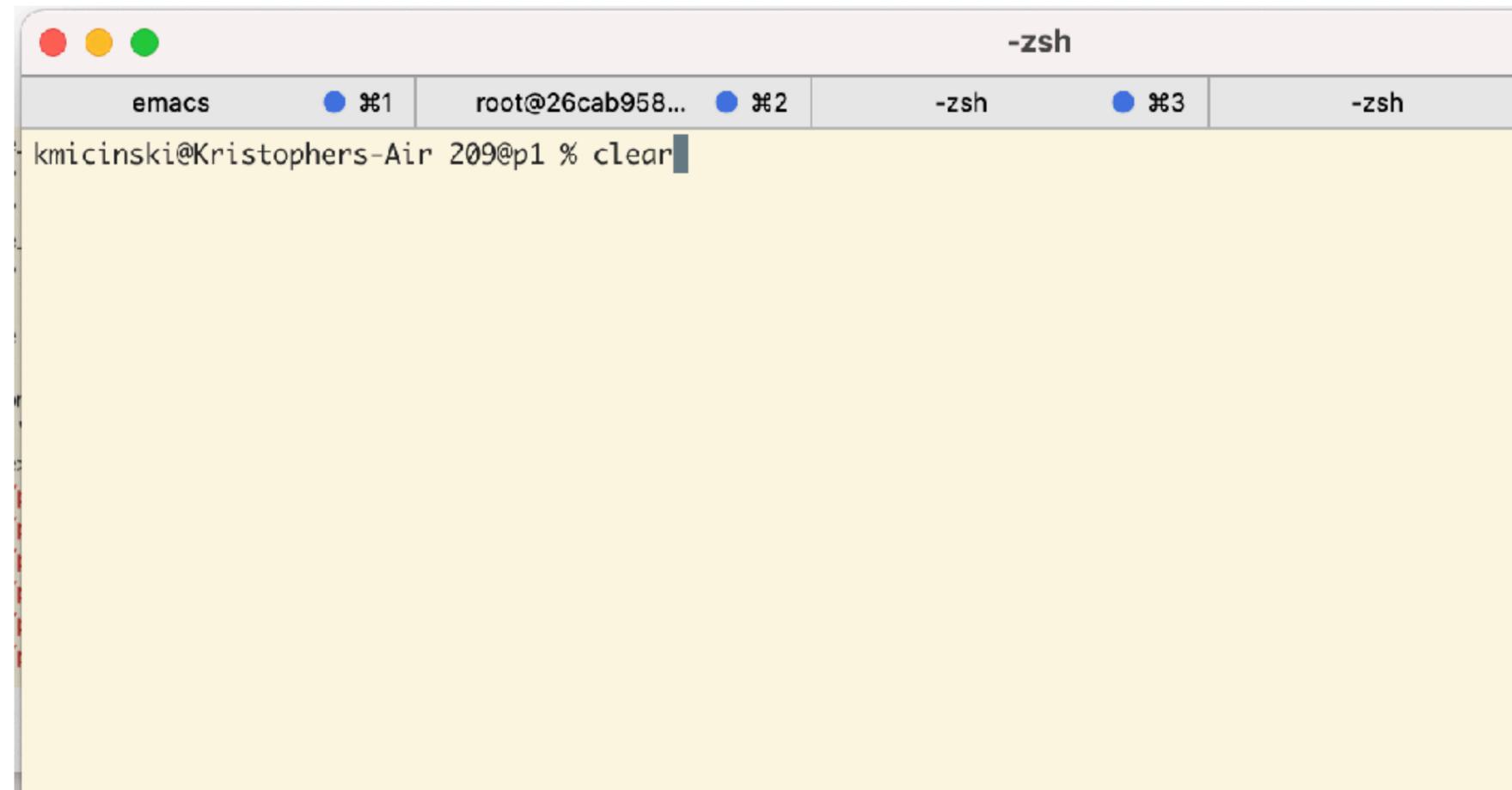
- After getting into the work directory, I use "git status" to see what's new
 - Shows any uncommitted work

```
README.md  __pycache__  ascii.rkt  pictures  test  tester.py
kmicinski@Kristophers-Air projects % cd 209@p1
kmicinski@Kristophers-Air 209@p1 % ls
README.md  __pycache__  ascii.rkt  pictures  test  tester.py
kmicinski@Kristophers-Air 209@p1 % git status .
On branch master
Your branch is up to date with 'origin/master'.

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   test/public-draw-0/output
        modified:   test/public-draw-1/output
        modified:   test/public-draw-line-0/output
        modified:   test/public-draw-line-1/output
        modified:   test/public-newlines-0/output
        modified:   test/public-newlines-1/output
```

clear

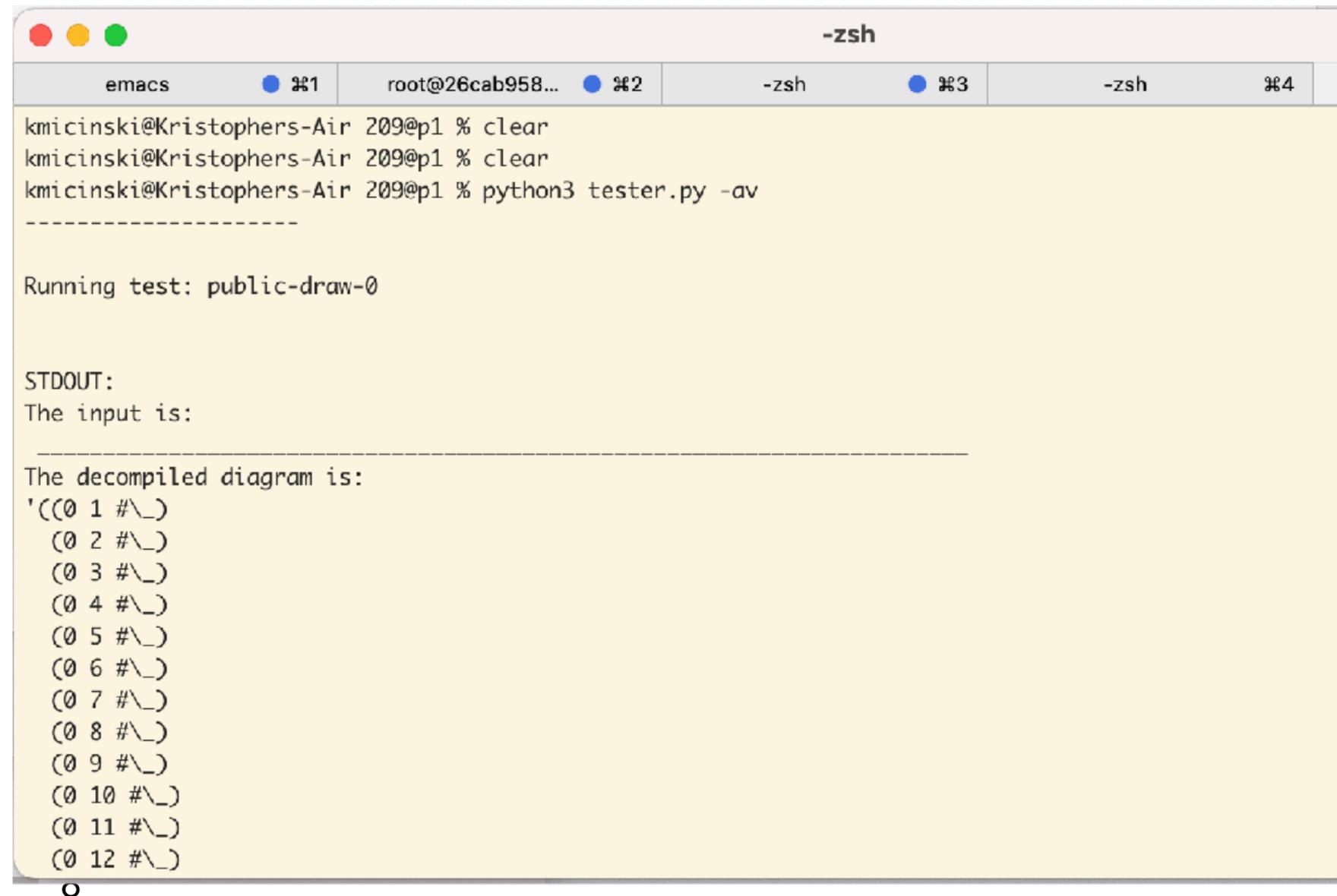
- I hate seeing too text on the screen



A screenshot of a macOS terminal window. The window title bar shows three colored buttons (red, yellow, green) and the text "-zsh". Below the title bar, there are four tabs: "emacs", "root@26cab958...", "-zsh", and "-zsh". The main content area of the terminal is a light yellow color. The prompt "kmicinski@Kristophers-Air 209@p1 %" is visible, followed by the command "clear" and a cursor. The rest of the terminal area is empty.

Running the tests

- Once in a while, I'll run the tests
 - Always use **python3**
 - Old python is python 2, it is now dead
 - Run it from the command line
 - Same project folder that holds our git repo
- I encourage you to go **read** tester.py
 - But it uses several helper scripts

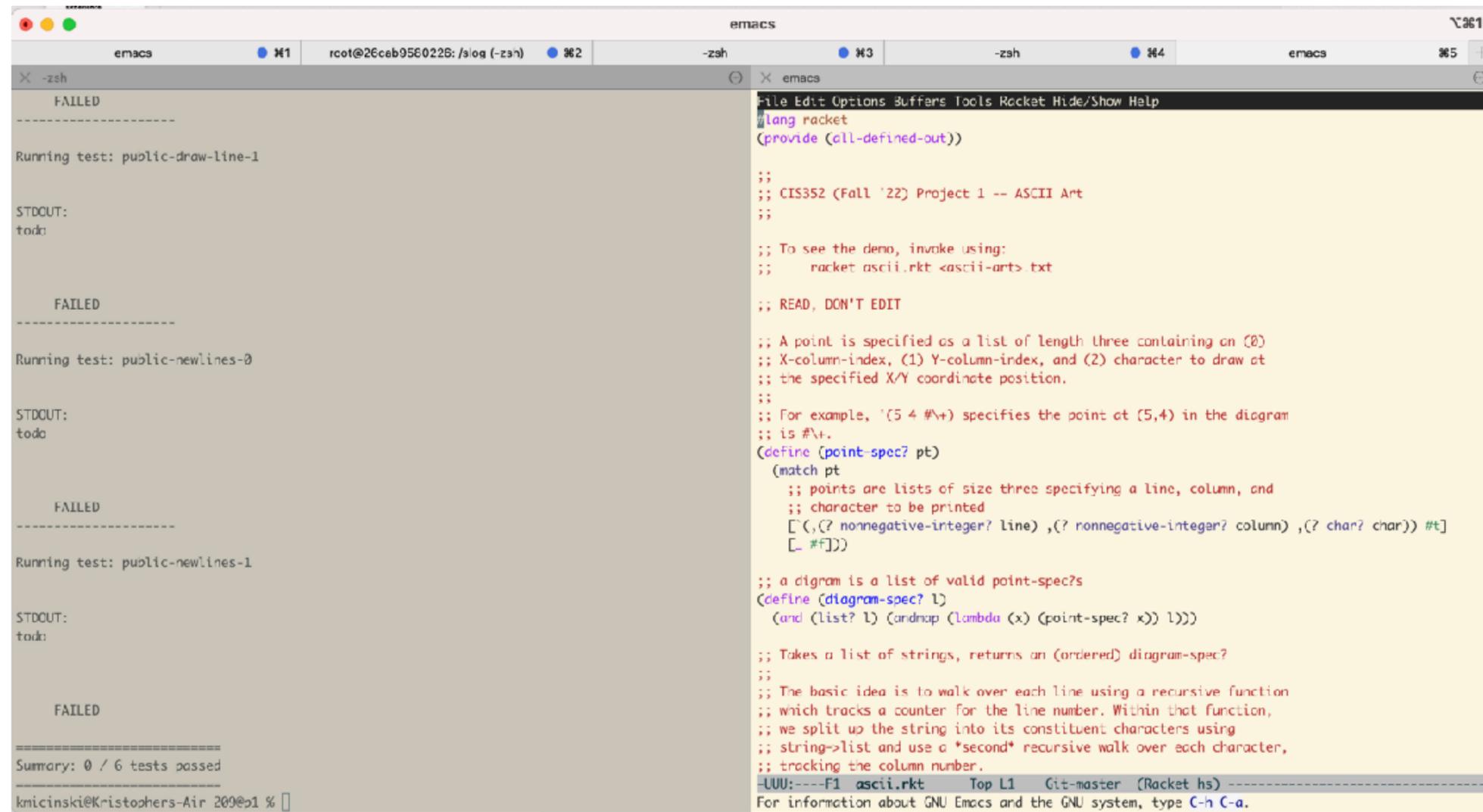


```
emac  root@26cab958...  -zsh  -zsh
kmicinski@Kristophers-Air 209@p1 % clear
kmicinski@Kristophers-Air 209@p1 % clear
kmicinski@Kristophers-Air 209@p1 % python3 tester.py -av
-----
Running test: public-draw-0

STDOUT:
The input is:
-----
The decompiled diagram is:
'((0 1 #\_)
(0 2 #\_)
(0 3 #\_)
(0 4 #\_)
(0 5 #\_)
(0 6 #\_)
(0 7 #\_)
(0 8 #\_)
(0 9 #\_)
(0 10 #\_)
(0 11 #\_)
(0 12 #\_)
)
```

Editing the code

- Several choices:
 - Emacs / vi in the terminal
 - Probably want side-by-side terms



The screenshot shows the Emacs editor interface. The left pane displays a terminal window with the following content:

```
FAILED
-----
Running test: public-draw-line-1

STDOUT:
todk:

FAILED
-----
Running test: public-newlines-0

STDOUT:
todk:

FAILED
-----
Running test: public-newlines-1

STDOUT:
todk:

FAILED
-----
Summary: 0 / 6 tests passed
knicinski@Kristophers-Air 209@p1 %
```

The right pane shows the Emacs code editor with the following content:

```
File Edit Options Buffers Tools Rocket Hide/Show Help
lang racket
(provide (all-defined-out))

;;
;; CIS352 (Fall '22) Project 1 -- ASCII Art
;;
;;
;; To see the demo, invoke using:
;; racket ascii.rkt <ascii-art>.txt
;;
;; READ, DON'T EDIT

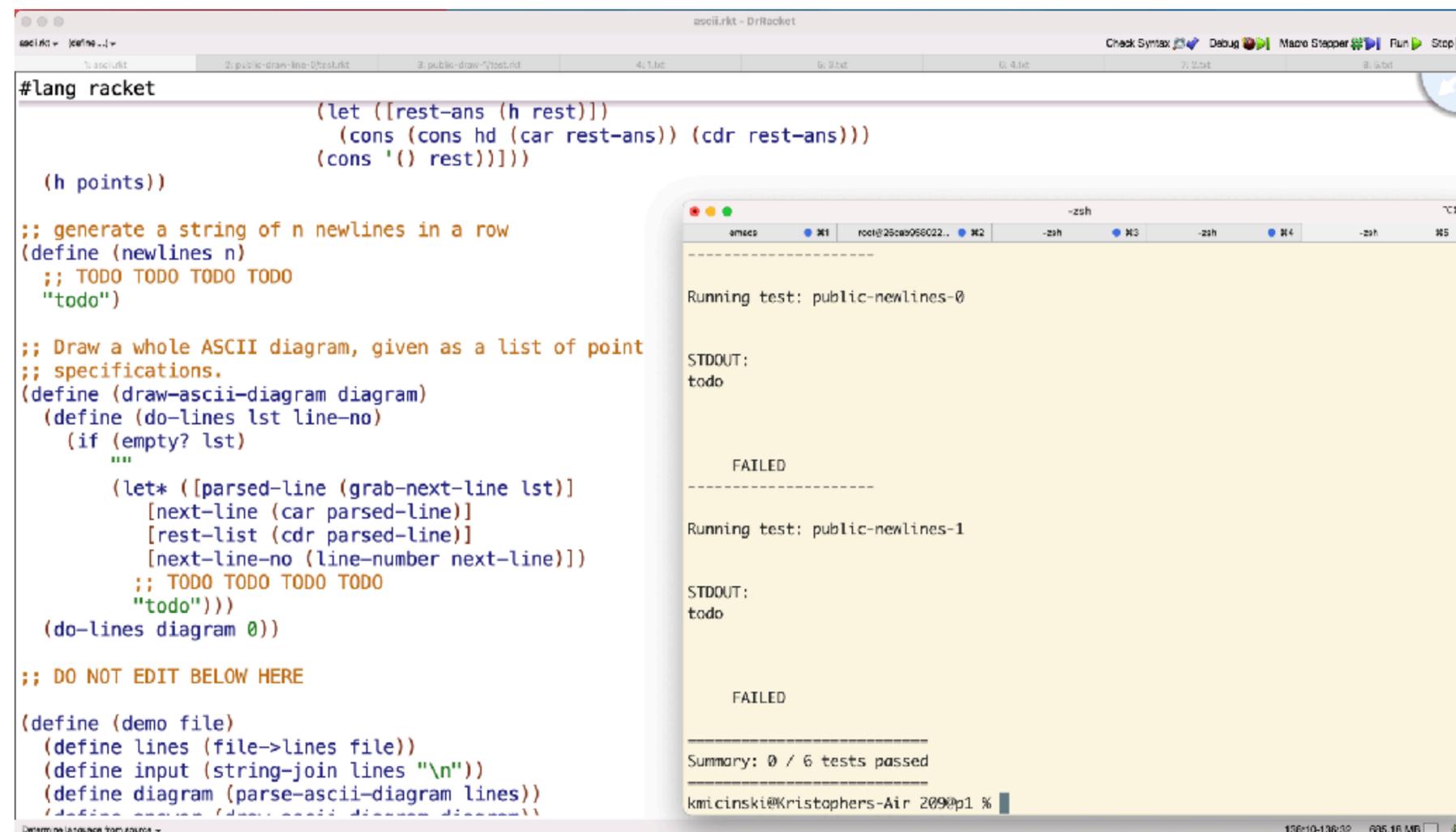
;; A point is specified as a list of length three containing an (0)
;; X-column-index, (1) Y-column-index, and (2) character to draw at
;; the specified X/Y coordinate position.
;;
;; For example, '(5 4 #\+) specifies the point at (5,4) in the diagram
;; is #\+.
(define (point-spec? pt)
  (match pt
    ;; points are lists of size three specifying a line, column, and
    ;; character to be printed
    [ `(,(? nonnegative-integer? line) ,(? nonnegative-integer? column) ,(? char? char)) #t]
    [_ #f]))

;; a diagram is a list of valid point-spec?s
(define (diagram-spec? l)
  (and (list? l) (andmap (lambda (x) (point-spec? x)) l)))

;; Takes a list of strings, returns an (ordered) diagram-spec?
;;
;; The basic idea is to walk over each line using a recursive function
;; which tracks a counter for the line number. Within that function,
;; we split up the string into its constituent characters using
;; string->list and use a *second* recursive walk over each character,
;; tracking the column number.
-LUU:---F1 ascii.rkt Top L1 Git-master (Racket hs) -----
For information about GNU Emacs and the GNU system, type C-h C-a.
```

Editing the code

- Most students will simply use Dr. Racket **and** a terminal
- This is fine—keep them both side-by-side
- (Switch between with command-tab on MacOS, ...)



The image shows a side-by-side view of a Dr. Racket editor window and a terminal window. The Dr. Racket window displays Racket code for testing a function. The terminal window shows the output of running tests, including 'FAILED' messages and a summary of 0 passed tests out of 6.

```
#lang racket

      (let ([rest-ans (h rest)])
        (cons (cons hd (car rest-ans)) (cdr rest-ans)))
      (cons '() rest))))

(h points))

;; generate a string of n newlines in a row
(define (newlines n)
  ;; TODO TODO TODO TODO
  "todo")

;; Draw a whole ASCII diagram, given as a list of point
;; specifications.
(define (draw-ascii-diagram diagram)
  (define (do-lines lst line-no)
    (if (empty? lst)
        ""
        (let* ([parsed-line (grab-next-line lst)]
                [next-line (car parsed-line)]
                [rest-list (cdr parsed-line)]
                [next-line-no (line-number next-line)])
          ;; TODO TODO TODO TODO
          "todo")))
    (do-lines diagram 0))

;; DO NOT EDIT BELOW HERE

(define (demo file)
  (define lines (file->lines file))
  (define input (string-join lines "\n"))
  (define diagram (parse-ascii-diagram lines))
  (define output (draw-ascii-diagram diagram))
  (displayln input)
  (displayln output))

(define (test-public-newlines n)
  (demo "public-newlines-0.rkt"))

(test-public-newlines 0)
(test-public-newlines 1)
```

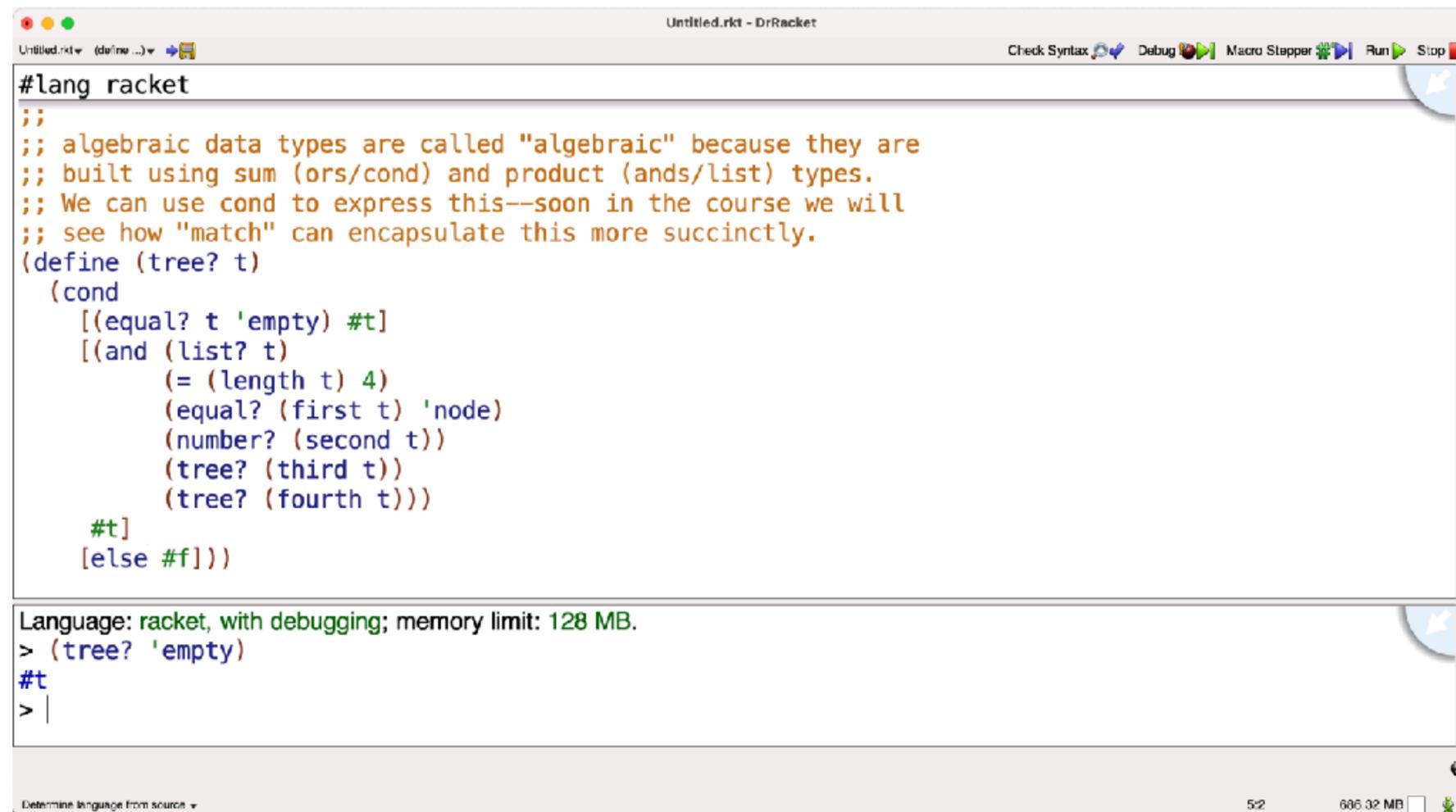
```
Running test: public-newlines-0
-----
STDOUT:
todo

-----
FAILED
-----
Running test: public-newlines-1
-----
STDOUT:
todo

-----
FAILED
-----
Summary: 0 / 6 tests passed
kmicinski@Kristophers-Air 2019 %
```

Test Always

- Whenever you do *something*, test it as **fast as possible**
 - Otherwise you will lose context, context is crucial for bug finding!
- Get in the habit of pressing “run” a bunch
 - Even if you run no tests, it does “rough check” of syntactic correctness
- Type tests in the REPL “manually” for small things, use the terminal to run larger tests



The screenshot shows the DrRacket IDE interface. The main editor window contains the following Racket code:

```
#lang racket
;;
;; algebraic data types are called "algebraic" because they are
;; built using sum (ors/cond) and product (ands/list) types.
;; We can use cond to express this--soon in the course we will
;; see how "match" can encapsulate this more succinctly.
(define (tree? t)
  (cond
    [(equal? t 'empty) #t]
    [(and (list? t)
          (= (length t) 4)
          (equal? (first t) 'node)
          (number? (second t))
          (tree? (third t))
          (tree? (fourth t)))
     #t]
    [else #f])))
```

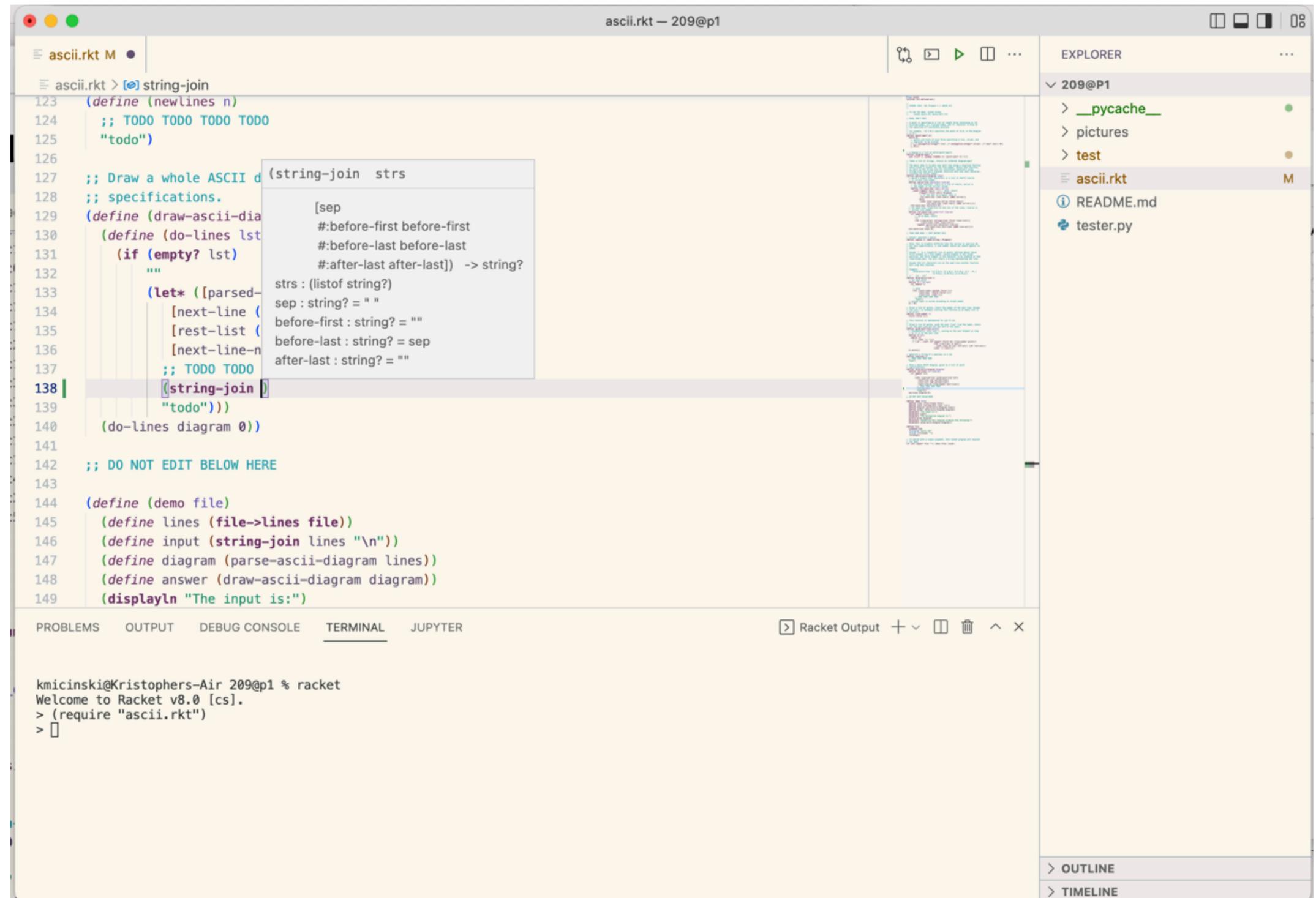
Below the code editor, the REPL shows the following interaction:

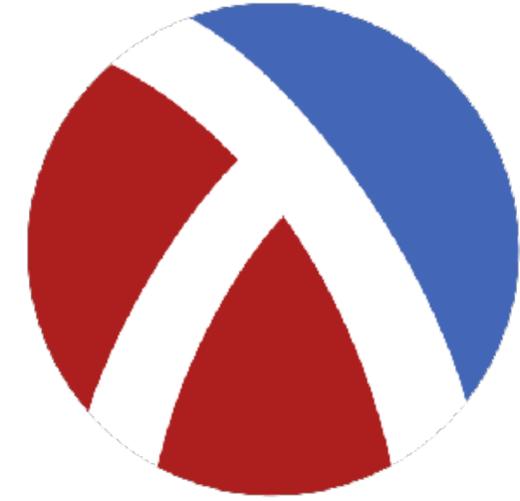
```
Language: racket, with debugging; memory limit: 128 MB.
> (tree? 'empty)
#t
> |
```

The IDE window title is "Untitled.rkt - DrRacket". The status bar at the bottom shows "Determine language from source", "5:2", and "606.32 MB".

Other Editors

VSCode is worth trying





Cons Diagrams and Boxes

CIS352 — Fall 2022

Kris Micinski

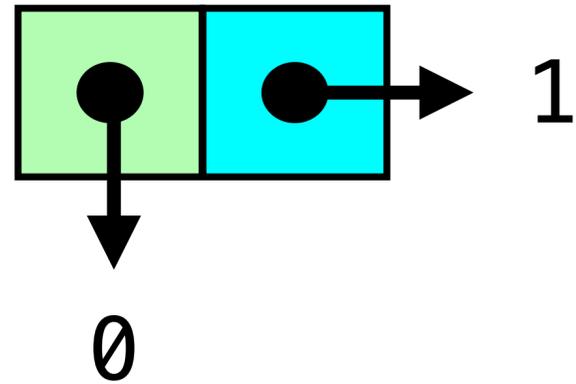
Derived Types

- **S-expressions** (**symbolic** expression)
 - Untyped lists that generalize neatly to trees:
`(this (is an) s expression)`
- Computer represents these as **linked** structures
 - Cons cells of head & tail (`cons 1 2`)

Derived Types

- Racket also has **structural** types
 - Defined via **struct**; aids robustness
 - We will usually prefer agility of “tagged” S-expressions
- Also an elaborate object-orientation system (we won't cover)

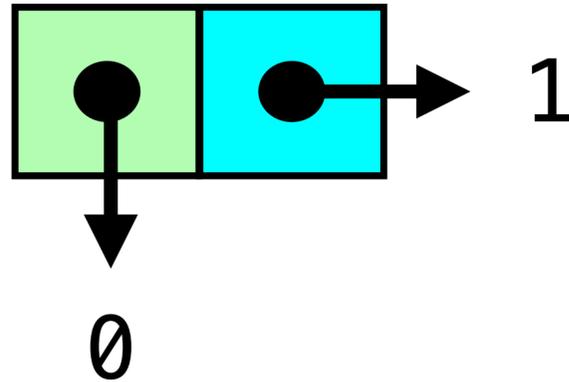
(cons 0 1)



The function **cons** builds a cons cell

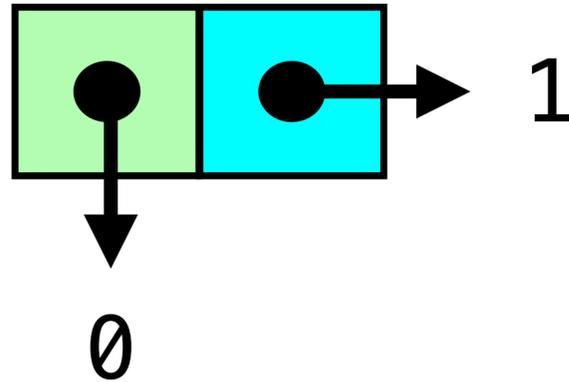
The function **car** gets the left element

(car (cons 0 1)) is 0



The function **cdr** gets the left element

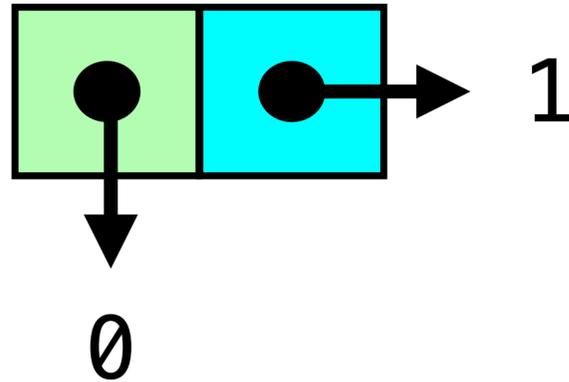
(cdr (cons 0 1)) is **1**



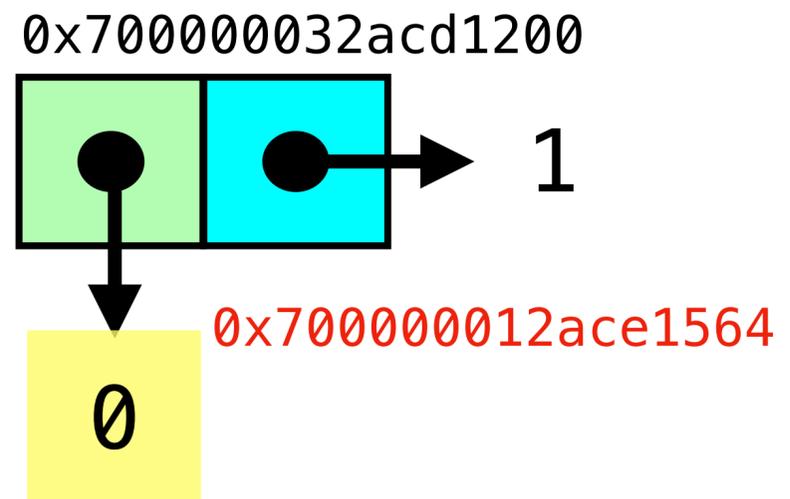
At runtime, each cons cell sits at an **address** in memory

`(cdr (cons 0 1))` is `1`

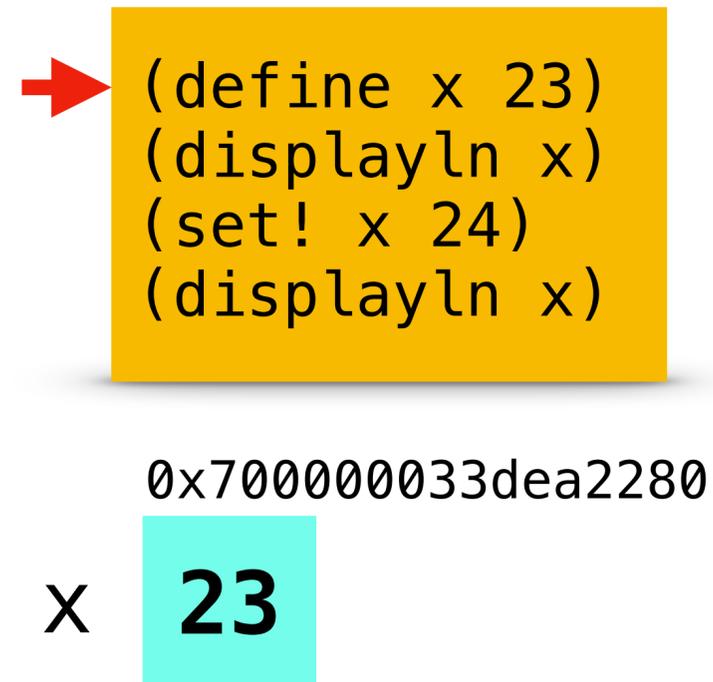
`0x70000032acd1200`



In fact, numbers are **also** stored in memory locations.
They are thus said to be a “boxed” type



Actually, every Racket variable stores a value in some "box" (i.e., memory location)



Actually, every Racket variable stores a value in some "box" (i.e., memory location)

```
→ (define x 23)
   (displayln x)
   (set! x 24)
   (displayln x)
```

0x700000033dea2280

x **23**

Console output...

> 23

Actually, every Racket variable stores a value in some "box" (i.e., memory location)

```
(define x 23)
(displayln x)
→ (set! x 24)
   (displayln x)
```

0x700000033dea2280

x **24**

x's value **changes** to 24

```
(define x (vector 1 2 3))  
(vector-set! x 1 0)  
x  
;; '#(1 0 3)
```

Vectors (similar to arrays) are mutable, and give $O(1)$ indexing and updating

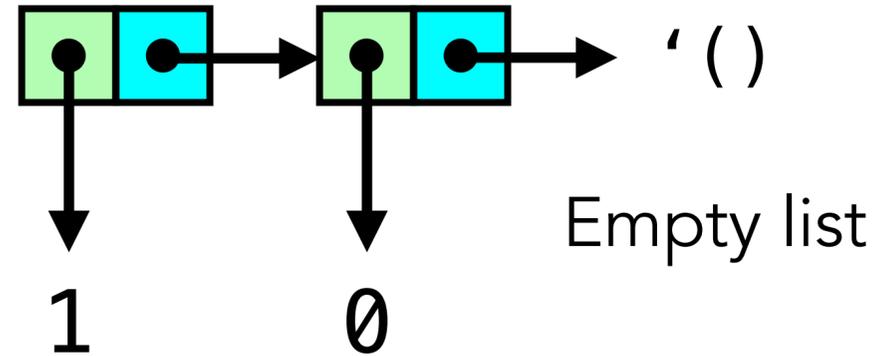
Unless we say otherwise, you should avoid using `set!`, any use will be at your own risk

Similarly, avoid `vector-set!`, `hash-set!`, ...

Using `set!` will, in CIS352, lead to hard-to-debug code that will make it much harder for instructors to understand your code

Pairs enable us to build **linked lists** of data

```
(cons 1 (cons 0 ' ( )))
```



This is how Racket represents lists in memory

Note that in Racket, the following are equivalent

```
(cons 2 (cons 1 (cons 0 '())))  
'(2 1 0)
```

But the following is called an **improper list**

```
(cons 2 (cons 1 0))  
'(2 1 . 0)
```

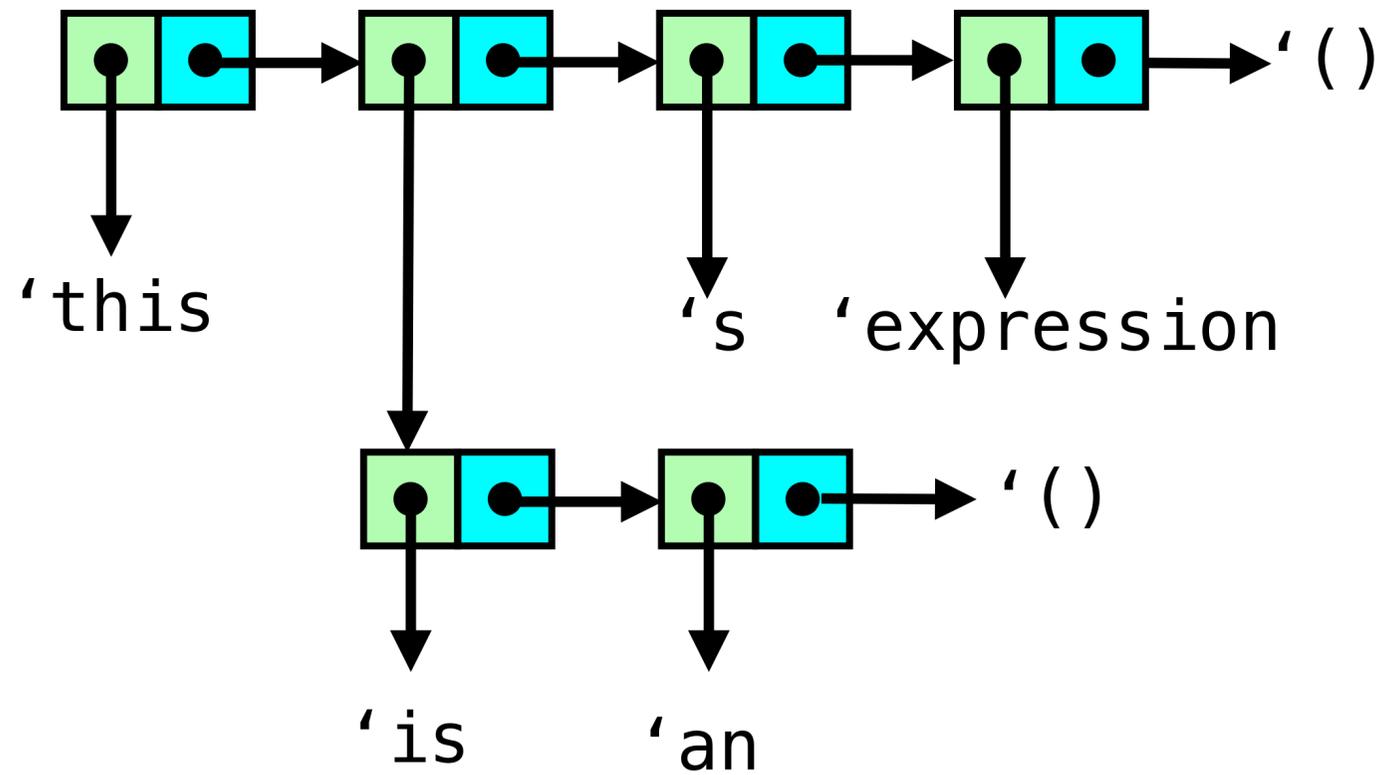
Dot indicates a cons cell of a left and right element

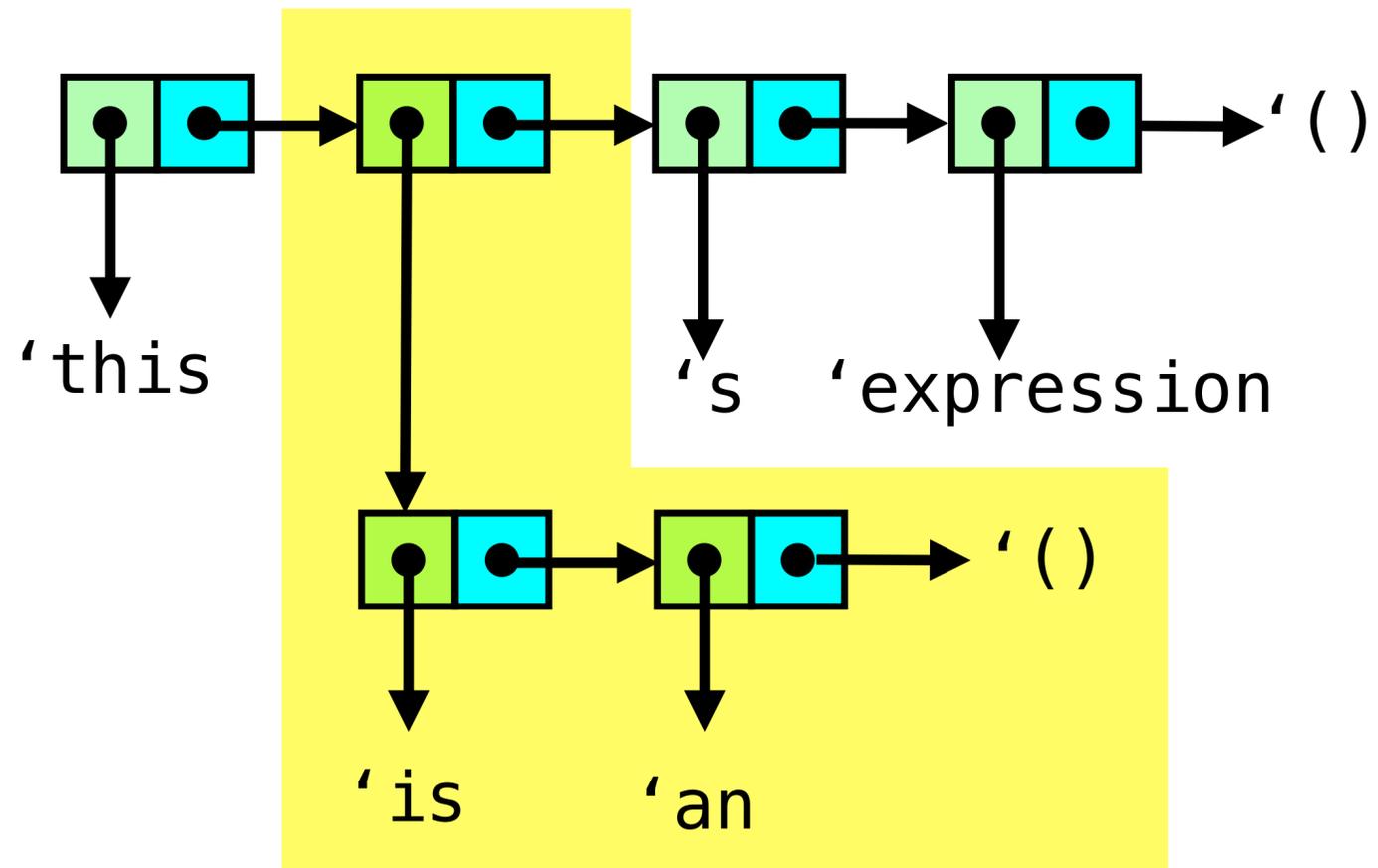
Also can build **compound** expressions

```
'(this (is an) s expression)'
```

Also can build **compound** expressions

'(this (is an) s expression)

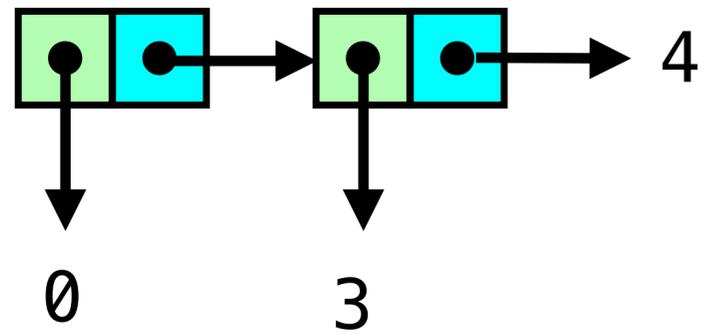




Draw the cons diagram for...

- `(cons 0 (cons 3 4))`
- Is this a list? If not, what is it?
- `(cons 0 (cons 3 (cons 4 '())))`
- Is this a list? If not, what is it?

(cons 0 (cons 3 4))



This is *not* a list (an improper list)

(cons 0 (cons 3 (cons 4 '())))

