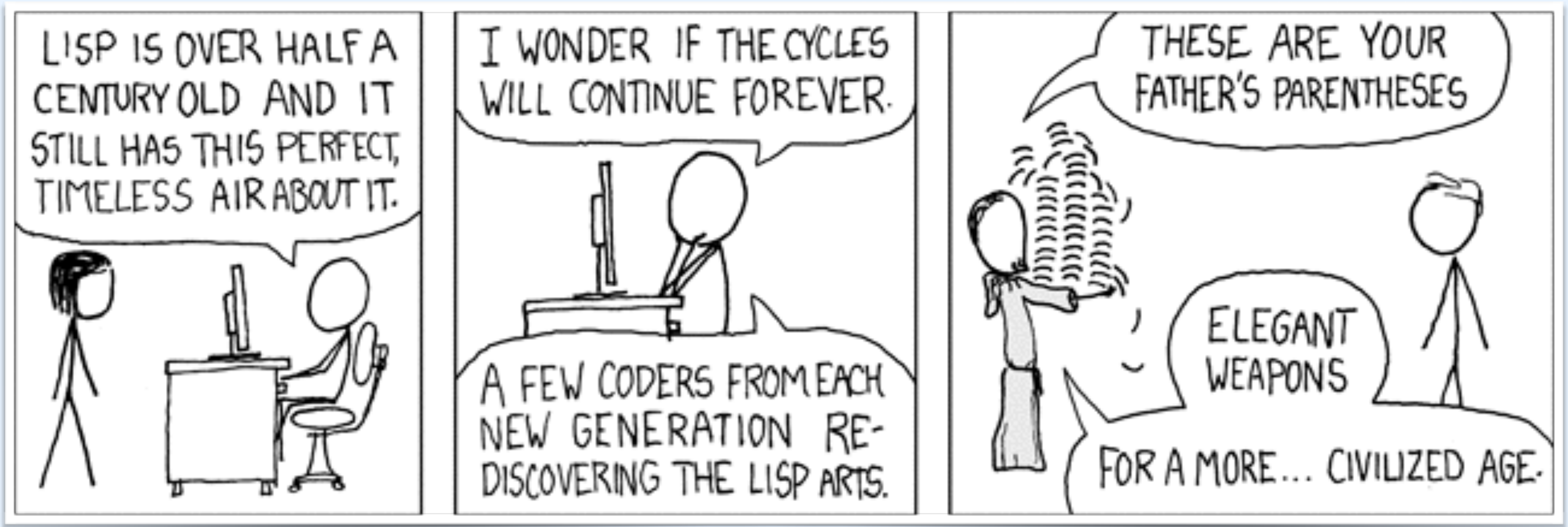
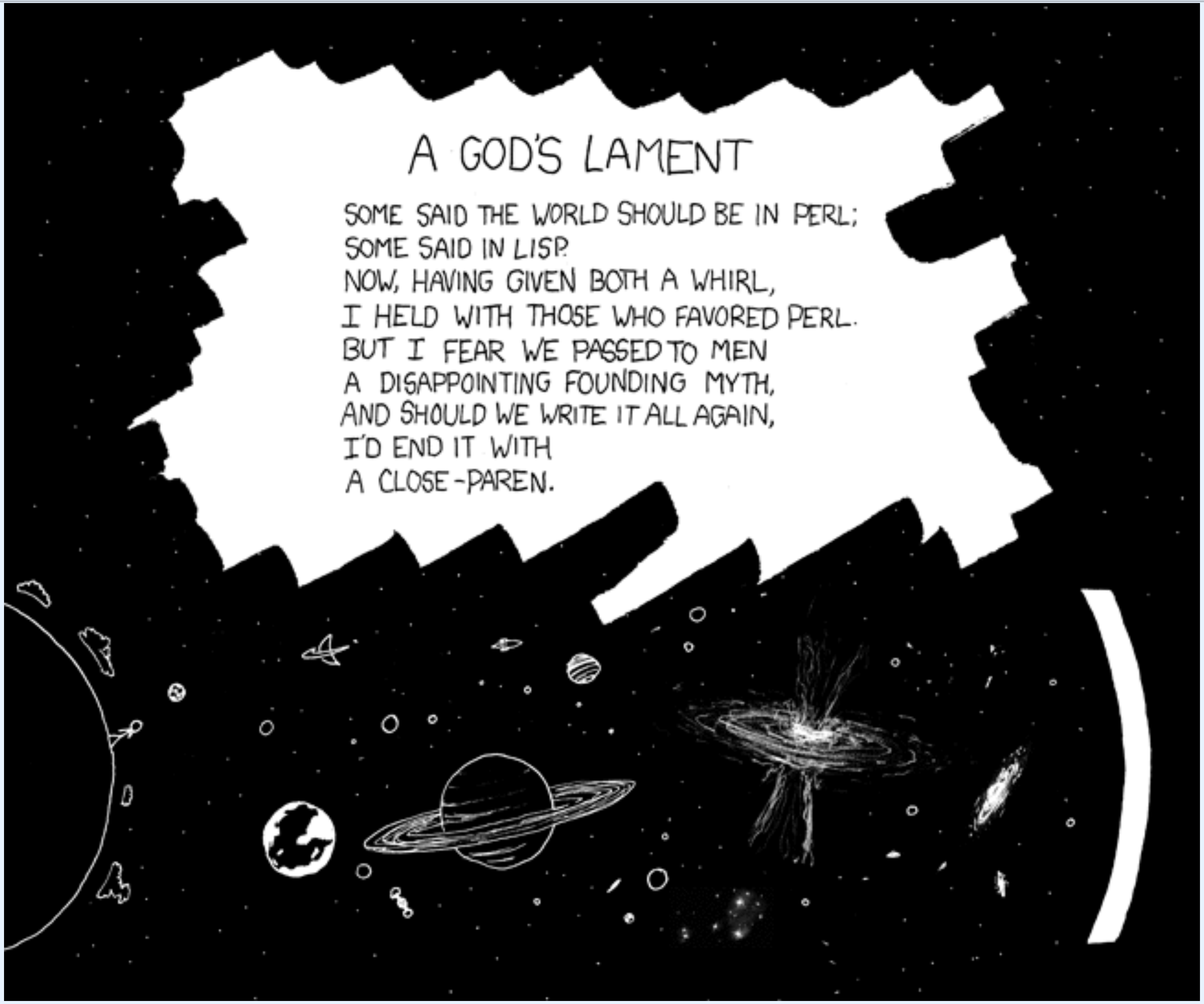

LISP IS OVER HALF A CENTURY OLD AND IT STILL HAS THIS PERFECT, TIMELESS AIR ABOUT IT.
I WONDER IF THE CYCLES WILL CONTINUE FOREVER.
A FEW CODERS FROM EACH NEW GENERATION RE-DISCOVERING THE LISP ARTS.
THESE ARE YOUR FATHER'S PARENTHESES
ELEGANT WEAPONS
FOR A MORE... CIVILIZED AGE.

xkcd comic 297

Introduction to Racket

September 13, 2018

Why are there so many parentheses?


A GOD'S LAMENT
SOME SAID THE WORLD SHOULD BE IN PERL;
SOME SAID IN LISP.
NOW, HAVING GIVEN BOTH A WHIRL,
I HELD WITH THOSE WHO FAVORED PERL.
BUT I FEAR WE PASSED TO MEN
A DISAPPOINTING FOUNDING MYTH,
AND SHOULD WE WRITE IT ALL AGAIN,
I'D END IT WITH
A CLOSE-PAREN.

xkcd comic 312

Euclid's algorithm for GCD

Find greatest common divisor of r1 and r2:

base case:

```
If r1 = 0:
    return r2
If r2 = 0:
    return r1
```

kth step:

```
If r1 and r2 are greater than 0:
    r1 / r2
    GCD(r2, remainder)
```

Local binding

A let expression binds a set of variables for use in the body of the let block.

```
(define (greet str)
        (let ((greeting (string-append “hi “ str))
             printf(greeting))
```

Lists

```
(list “apple” “banana” “carrot”)

(list 1 2 3)

(list 1 “carrot”  3   #t   “cucumber”)
```

Lists are recursively defined

A list is either null, or a pair whose second item is a list

Two key methods:

```
> (first (list 1 2 3))
1

> (rest (list 1 2 3))
(list 1 2)
```

Local binding

A let expression binds a set of variables for use in the body of the let block.

```
(define (greet str)
        (let ((greeting (string-append “hi “ str))
              printf(greeting))
```

Anonymous Functions

A lambda expression is an anonymous function.
(define (fn)) is really short for (define fn (lambda))

```
(define (hello-world) (printf “hello world!”))
(define hello-world (lambda () (printf “hello world!)))
```

Arguments **Function body**

Local binding, take two

In a let expression, the right-hand side of a declaration can't refer to the left-hand side.

If we write:

```
(let ((a (+ a 5))))
```

if the a is not defined outside the scope of the let, then the let will throw an error.

Letrec

This is a problem for declaring recursive functions, since they refer to themselves!

Racket has another local binding environment for this reason: letrec.

If we write:

```
(letrec ((a (+ a 5))))
```

The a in the right-hand side refers to whatever value the a on the left-hand side has.

Lab 1

- **Due Sunday, September 22nd at 10pm**
- Submit through Moodle
- Generally labs will be released during 4th hour and due the following Sunday.
- 2 parts: 6 finger exercises in Part 1
  merge-sort in Part 2
- Bring questions to 4th hour on Monday!