The 2018 Chemistry Laureates

The Royal Swedish Academy of Sciences has decided to award the Nobel Prize in Chemistry 2018 with one half to Frances H. Arnold "for the directed evolution of enzymes" and the other half jointly to George P. Smith and Sir Gregory P. Winter "for the phage display of peptides and antibodies".
Read the press release

Ill. Niklas Elmehed. © Nobel Media

Press Release: The Nobel Prize in Chemistry 2016

5 October 2016

The Royal Swedish Academy of Sciences has decided to award the Nobel Prize in Chemistry 2016 to

Jean-Pierre Sauvage
University of Strasbourg, France

Sir J. Fraser Stoddart
Northwestern University, Evanston, IL, USA

and

Bernard L. Feringa
University of Groningen, the Netherlands

"for the design and synthesis of molecular machines"

They developed the world's smallest machines

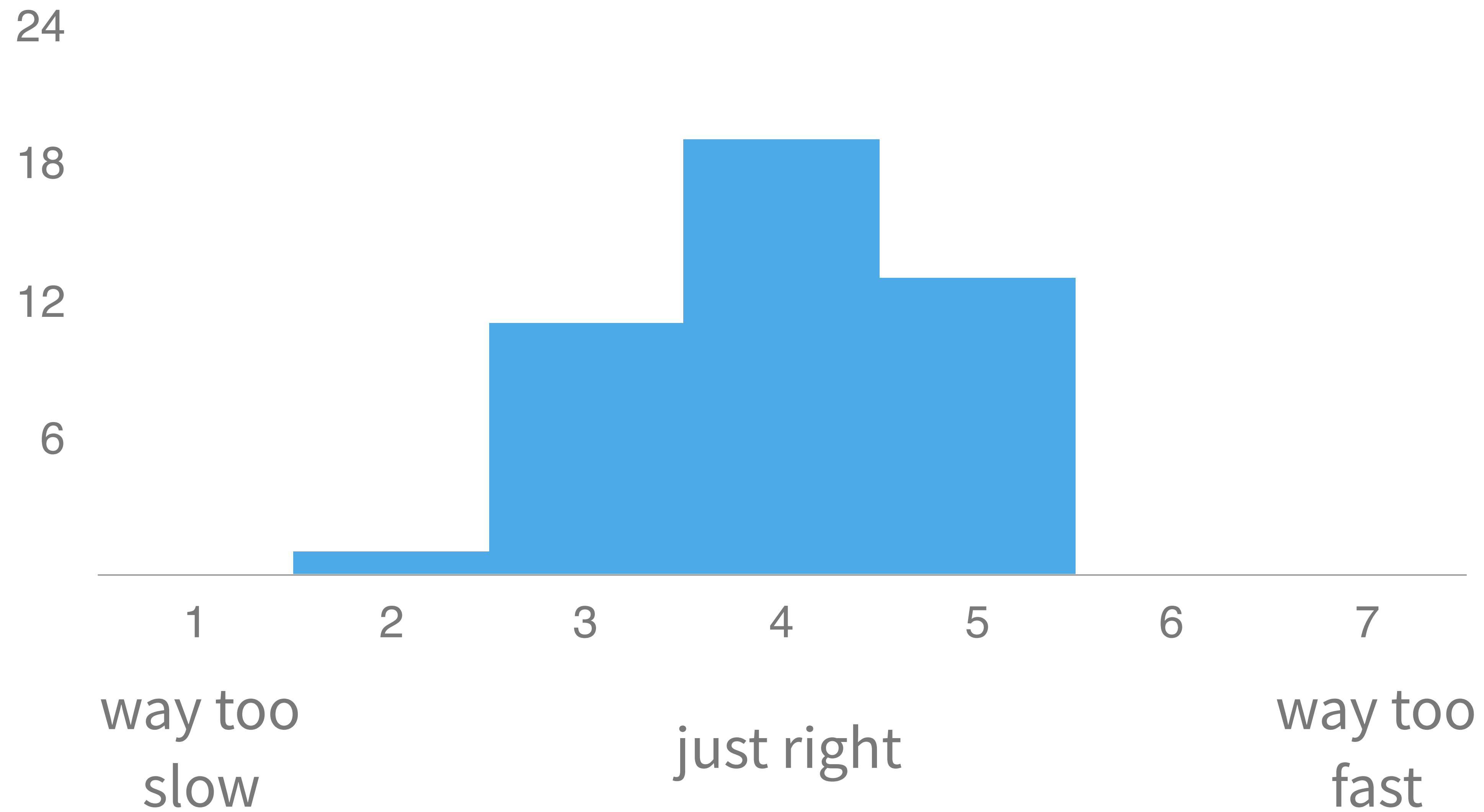
A tiny lift, artificial muscles and miniscule motors. The Nobel Prize in Chemistry 2016 is awarded to **Jean-Pierre Sauvage, Sir J. Fraser Stoddart** and **Bernard L. Feringa** for their design and production of molecular machines. They have developed molecules with controllable movements, which can perform a task when energy is added.

The development of computing demonstrates how the miniaturisation of technology can lead to a revolution. The 2016 Nobel Laureates in Chemistry have miniaturised machines and taken chemistry to a new dimension.

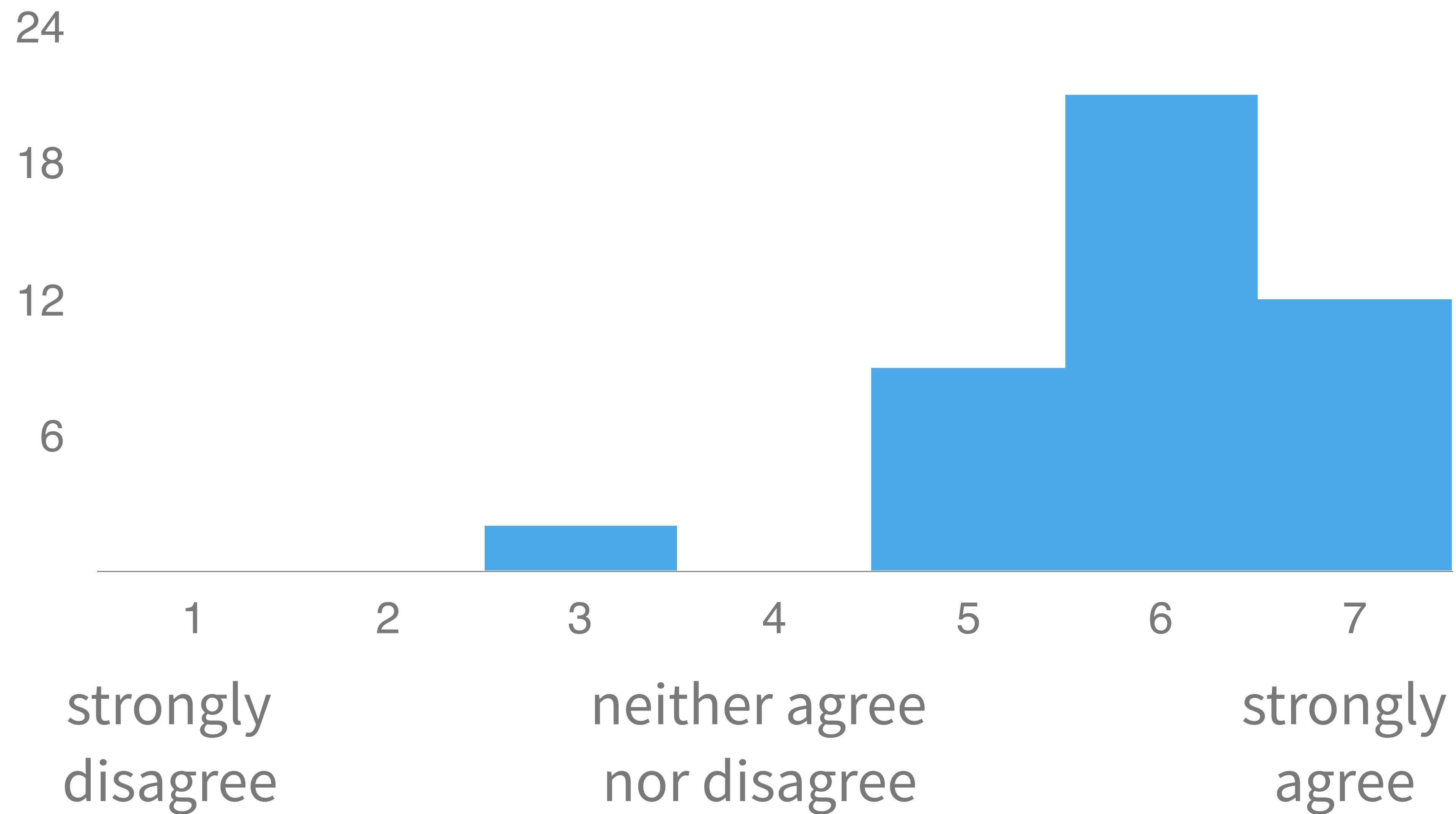
The first step towards a molecular machine was taken by Jean-Pierre Sauvage in 1983, when he succeeded in linking two ring-shaped molecules together to form a chain, called a *catenane*. Normally, molecules are joined by strong covalent bonds in which the atoms share electrons, but in the chain they were instead linked by a freer *mechanical bond*. For a machine to be able to perform a task it must consist of parts that can move relative to each other. The two interlocked rings fulfilled exactly this requirement.

The second step was taken by Fraser Stoddart in 1991, when he developed a *rotaxane*. He threaded a molecular ring onto a thin molecular axle and

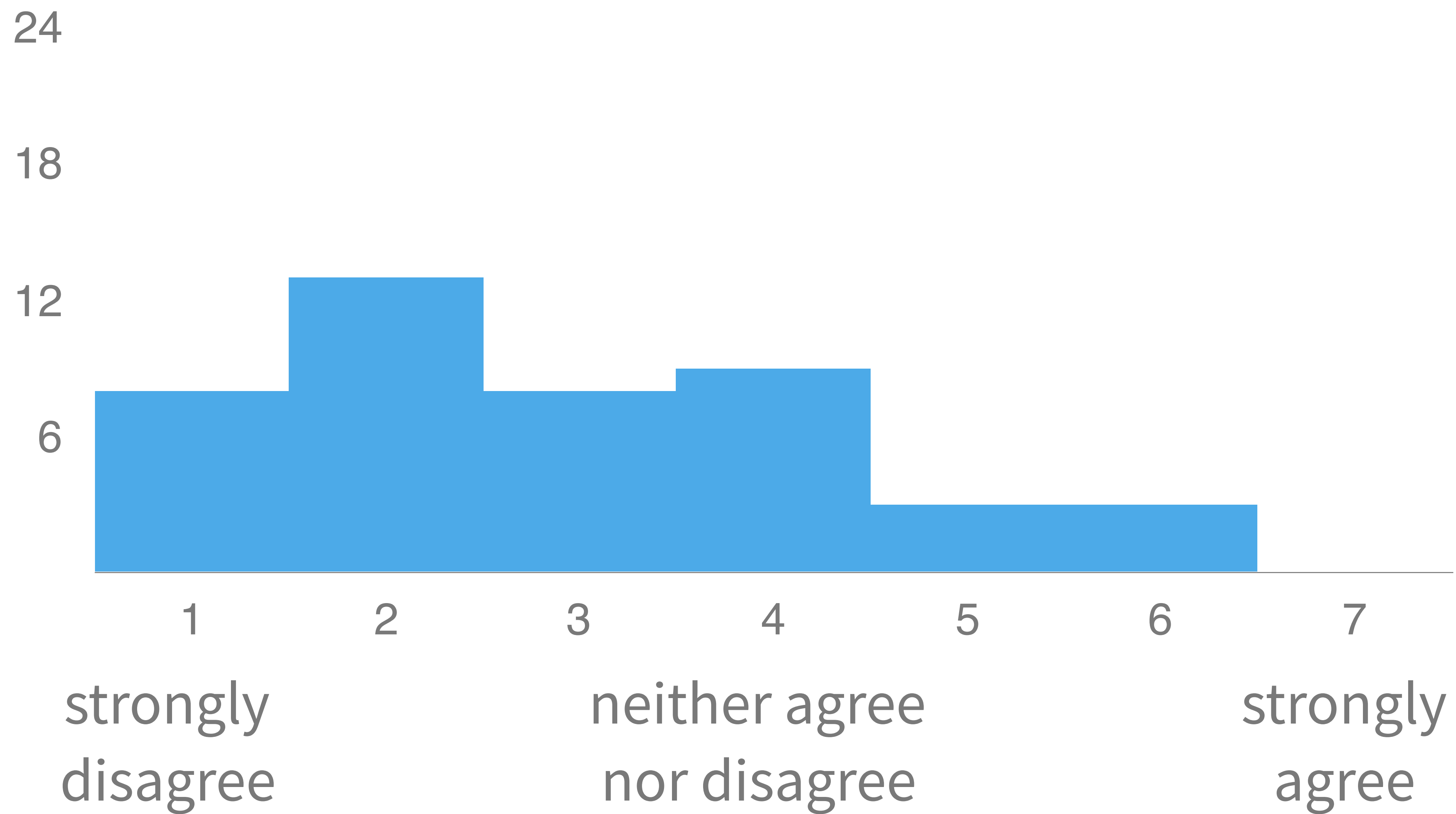
The pace of this class is…



I’m learning a lot in CS 42.



When it comes to workload, so far, this is my hardest course.



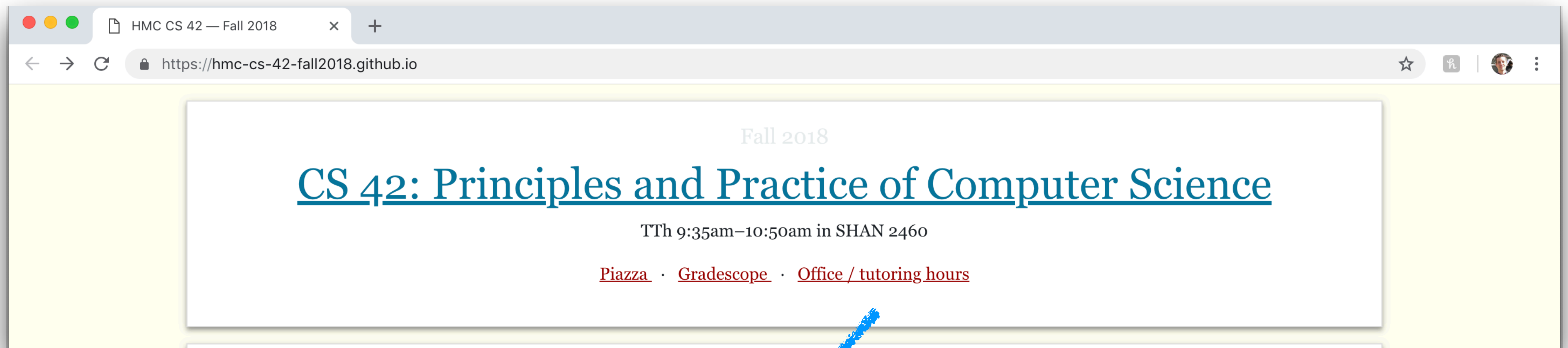
Three kinds of work

In-class
Why? introduce new skills and concepts, provide context, discuss implications
How? lectures, small-group discussions, exercises

Assignments
Why? practice skills and concepts
How? usually by making things

Exams
Why? build deeper understanding of concepts
How? apply familiar concepts in new contexts

Help outside of class



CS 42

https://piazza.com/class/jjvi5vnxlan4yn?cid=13

piazza CS 42 Q & A Resources Statistics Manage Class Ben Wiedermann

hw1 hw2 hw3 hw4 exam logistics other hw0 submission-system grutors

Note History:

note

stop following 59 views

Actions

Office hours and tutoring hours

Office hours with Ben, in Olin B161C

- Mondays 3–5pm
- Tuesdays 2–4pm
- By appointment

If the door is open, come on in! Even–especially–if there are other people in there.

Tutoring hours with the grutors in the LAC (Roland Maxine Linde Activities Center):

- Generally, there are grutoring hours in the evening Sunday–Wednesday, as well as Friday, Saturday, and Sunday afternoons.
- For the most-up-to-date information, browse this Google doc.

If there are a lot more people than grutors, we'll form a queue to make sure every gets a turn.

Racket conditionals

```
(if conditional-expr
    true-expr
    false-expr)
```

```
(cond  [condition₁ expr₁]
       …
       [conditionₙ exprₙ]
       [else else-expr])
```

this is the most common form of `cond`

idiom: if you have more than one condition, use `cond`

Racket: functions

```
(define (function-name parameter₁ … parameterₙ)
   body)
```

Write tests first!

using `rackunit`

average.rkt - DrRacket

average.rkt (define ...) Check Syntax Debug Macro Stepper Run Stop

```racket
#lang racket

(require rackunit) ; this line gives us access to the testing library

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; int-average
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;; int-average: computes the average of two numbers, using integer division
;;   inputs: x & y, two integers
;;   outputs: the integer average of the two inputs
(define (int-average x y)
  0)

;  tests
(check-equal? (int-average 0 0) 0)
(check-equal? (int-average 0 2) 1)
(check-equal? (int-average 4 6) 5)
(check-equal? (int-average 1 1) 1)
(check-equal? (int-average 1 2) 1)
```

```
--------------------
--------------------
FAILURE
actual:     0
expected:   1
name:       check-equal?
location:   (#<path:/Users/ben/Documents/work/teaching/courses/CS42/fall 2016/class/05_1 -
Racket Intro/code/average.rkt> 19 0 497 34)
expression: (check-equal? (int-average 1 1) 1)

Check failure
--------------------
--------------------
FAILURE
actual:     0
```

Write tests first!

using rackunit

average.rkt - DrRacket

average.rkt (define ...) Check Syntax Debug Macro Stepper Run Stop

```
#lang racket

(require rackunit) ; this line gives us access to the testing library

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; int-average
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;; int-average: computes the average of two numbers, using integer division
;;   inputs: x & y, two integers
;;   outputs: the integer average of the two inputs
(define (int-average x y)
  (quotient (+ x y) 2))

;  tests
(check-equal? (int-average 0 0) 0)
(check-equal? (int-average 0 2) 1)
(check-equal? (int-average 4 6) 5)
(check-equal? (int-average 1 1) 1)
(check-equal? (int-average 1 2) 1)
```

Welcome to DrRacket, version 6.6 [3m].
Language: racket, with debugging; memory limit: 128 MB.
>

Use trace to help investigate / debug

fact.rkt - DrRacket

fact.rkt ▾ (define ...) ▾ Check Syntax Debug Macro Stepper Run Stop

```
1  #lang racket
2
3  (provide fact) ; this line "exports" fact
4
5  (require racket/trace)
6
7  ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
8  ;; fact
9  ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
10
11 ;; fact: computes n!
12 ;;   inputs: n, a non-negative integer
13 ;;   outputs: n!
14 (define (fact N)
15   (if (= N 0)
16       1
17       (+ N (fact (- N 1)))))
18 (trace fact)
```

Welcome to DrRacket, version 6.12 [3m].
Language: racket, with debugging; memory limit: 128 MB.

```
> (fact 3)
>(fact 3)
> (fact 2)
> >(fact 1)
> > (fact 0)
< < 1
< <2
< 4
<7
7
> |
```

Determine language from source ▾ 13:2 410.50 MB

Separate tests from code

using `provide` and `require`

fact.rkt - DrRacket

```
#lang racket

(provide fact) ; this line "exports" fact

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; fact
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;; fact: computes n!
;;   inputs: n, a non-negative integer
;;   outputs: n!
(define (fact N)
  (if (= N 0)
      1
      (* N (fact (- N 1)))))
```

fact_tests.rkt - DrRacket

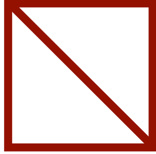
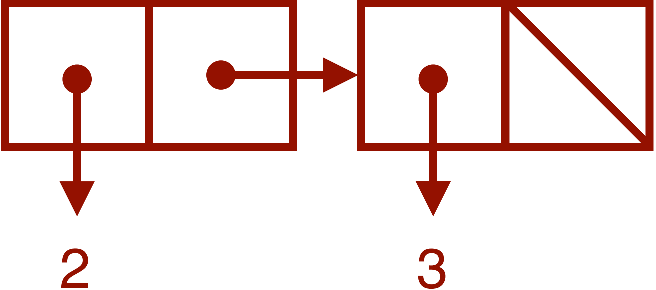
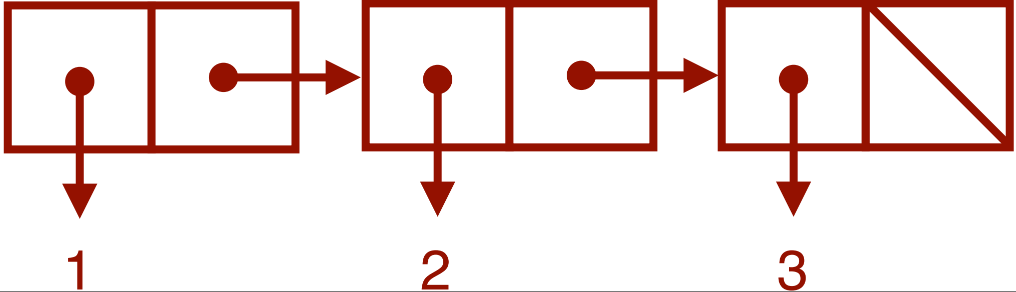
```
#lang racket
(require rackunit)
(require "fact.rkt")

(check-equal? (fact 0) 1)
(check-equal? (fact 1) 1)
(check-equal? (fact 3) 6)
(check-equal? (fact 4) 24)
(check-equal? (fact 5) 121)
```

https://www.teepublic.com/tote/2541248-funny-duck-billed-platypus-playing-tennis-cartoon

Racket:
Functions & Lists
(& Recursion)

Creating lists in Racket

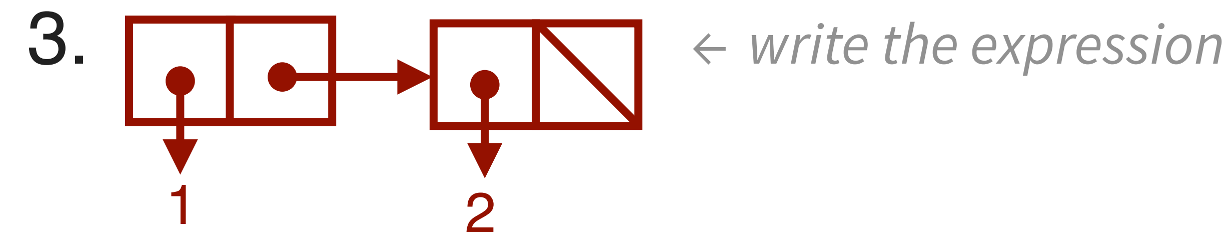
syntax what we *write*	**printed representation** what Racket *prints*	**semantics** what it *means*
empty *make an empty list*	> empty '()	
(list <value$_1$> ... <value$_N$>) *or* '(<value$_1$> ... <value$_N$>) *make a list with N values*	> (list 2 3) '(2 3) > '(2 3) '(2 3)	
(cons *<value> <list>*) *add an element to the front of a list*	> (cons 1 (list 2 3)) '(1 2 3)	

Creating lists: let's practice

write down the answers as either a drawing or a Racket expression

1. (list 3 2 1) ← *draw the picture*

2. (cons 3 (list 2 1)) ← *draw the picture*

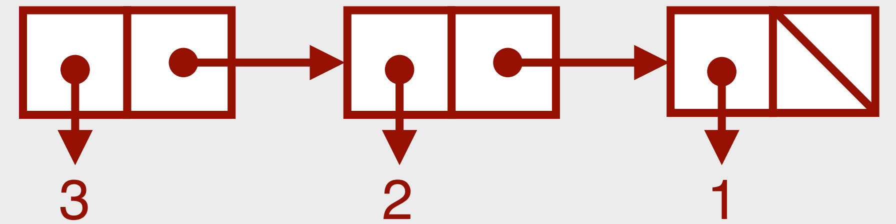
3. 

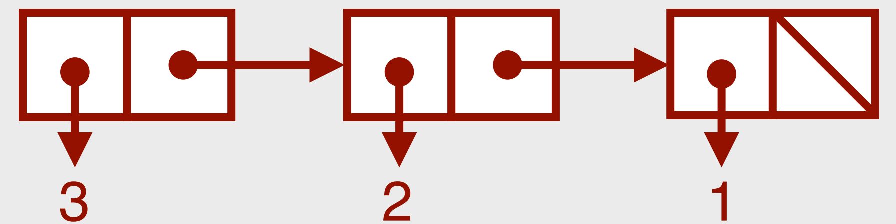
← *write the expression*

4. '(1) ← *write the expression that makes Racket display this*

5. (list 3 (list 2 1) 0) ← *draw the picture*

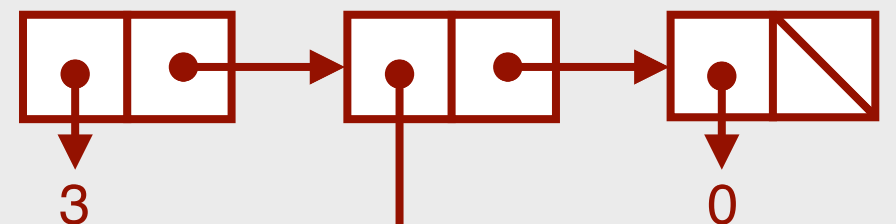
Full name Th. 10/4

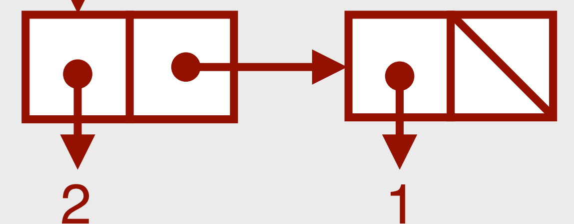




(list 1 2)

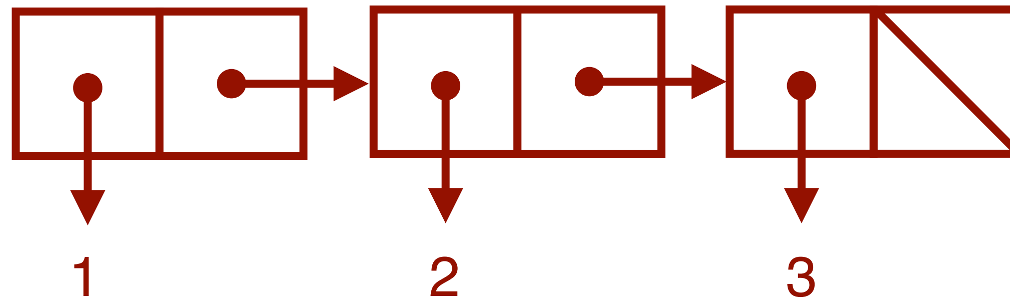
(list 1)





Aside: we don't actually *need* list!

list is "syntactic sugar" for one or more calls to cons

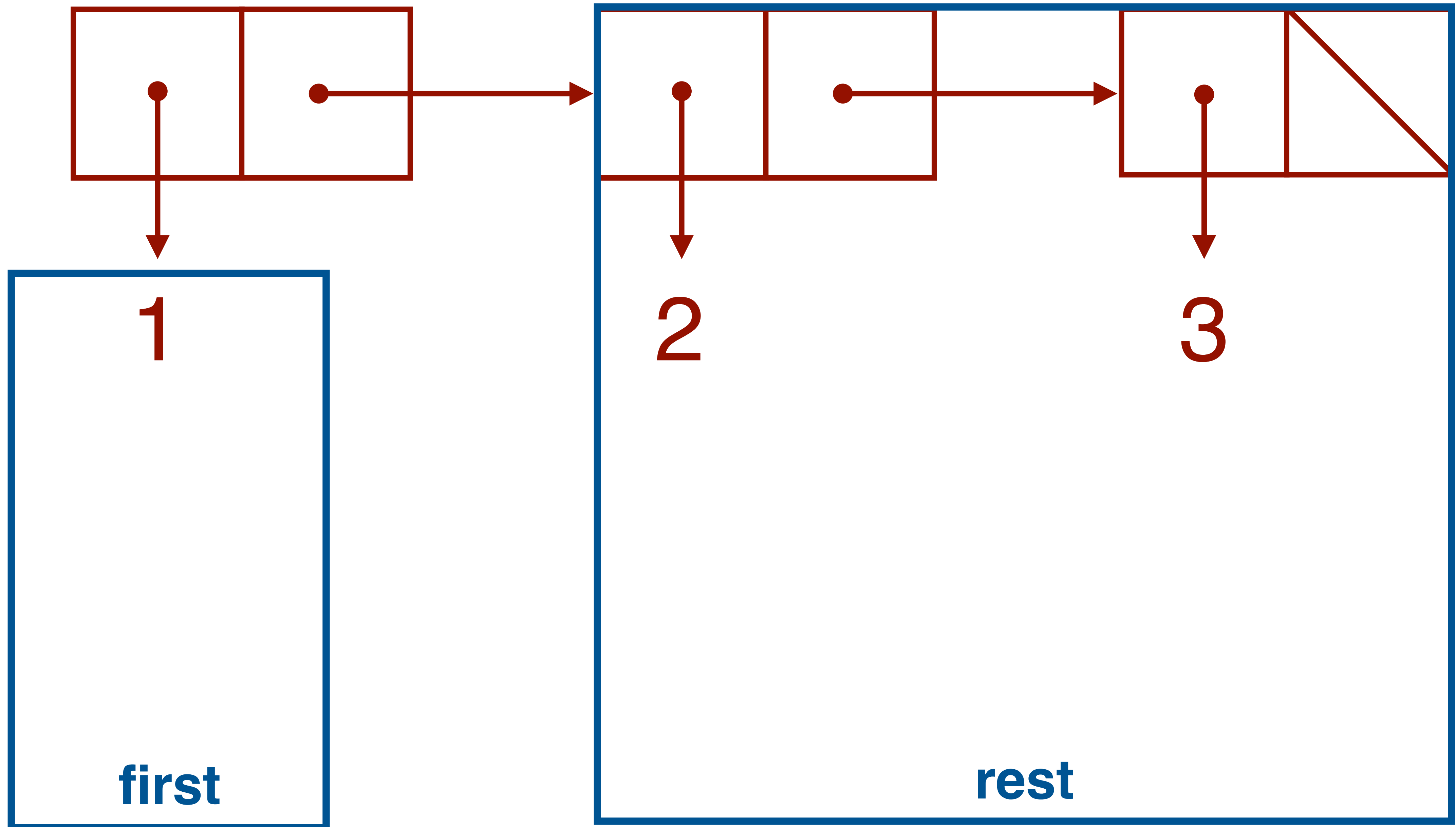


(list 1 2 3)

is the same as

(cons 1 (cons 2 (cons 3 empty)))

Accessing Racket lists



Accessing lists: let's practice

Assume the variable **L** has the value '(1 2 3). Fill in the table.

result	expression that uses **L** to compute result
1	(first L)
'(2 3)	
2	
'(3)	

Accessing lists: let's practice

Assume the variable **L** has the value '(1 2 3). Fill in the table.

result	expression that uses **L** to compute result
1	(first L)
'(2 3)	(rest L)
2	(first (rest L))
'(3)	(rest (rest L))