Mac OS Assembly language

numbers

registers

```
Google Chrome:
(__TEXT,__text) section
0000000100000ef0        pushq   $0x0
0000000100000ef2        movq    %rsp, %rbp
0000000100000ef5        andq    $-0x10, %rsp
0000000100000ef9        movq    0x8(%rbp), %rdi
0000000100000efd        leaq    0x10(%rbp), %rsi
0000000100000f01        movl    %edi, %edx
0000000100000f03        addl    $0x1, %edx
0000000100000f06        shll    $0x3, %edx
0000000100000f09        addq    %rsi, %rdx
0000000100000f0c        movq    %rdx, %rcx
0000000100000f0f        jmp     0x100000f15
0000000100000f11        addq    $0x8, %rcx
0000000100000f15        cmpq    $0x0, (%rcx)
0000000100000f19        jne     0x100000f11
0000000100000f1b        addq    $0x8, %rcx
0000000100000f1f        callq   _main
0000000100000f24        movl    %eax, %edi
0000000100000f26        callq   0x100000f46     ## symbol stub for: _exit
0000000100000f2b        hlt
0000000100000f2c        nop
0000000100000f2d        nop
0000000100000f2e        nop
0000000100000f2f        nop
```

The Principles in CS 42

Theory of computation & Machines (~4 weeks)
What is a computer?

no code!

Functional programming (~ 4 weeks)
There is no difference between functions and variables.

no loops!
no assignments!

Problem-solving techniques (~ 3 weeks)
Algorithms & Data structures
What is Computer Science?

Object-oriented programming (~ 3 weeks)
How do we design a program so that it can grow and change?

How’s CS 42 going?

(1) The pace of this class is...
1 = way too slow; 4 = just right; 7 = way too fast

(2) I’m learning a lot in CS 42.
1 = strongly disagree; 4 = neither agree nor disagree; 7 = strongly agree

(3) I find the handouts helpful.
1 = strongly disagree; 4 = neither agree nor disagree; 7 = strongly agree

(4) I can get help / support from (e.g., Ben, grutors, Piazza), if and when I need it.
1 = strongly disagree; 4 = neither agree nor disagree; 7 = strongly agree

(5) When it comes to workload, so far, this is my heaviest course this semester.
1 = strongly disagree; 4 = neither agree nor disagree; 7 = strongly agree

---

Full name **T. 10/2**

What does it mean “to compute”?

An Engineer’s Viewpoint

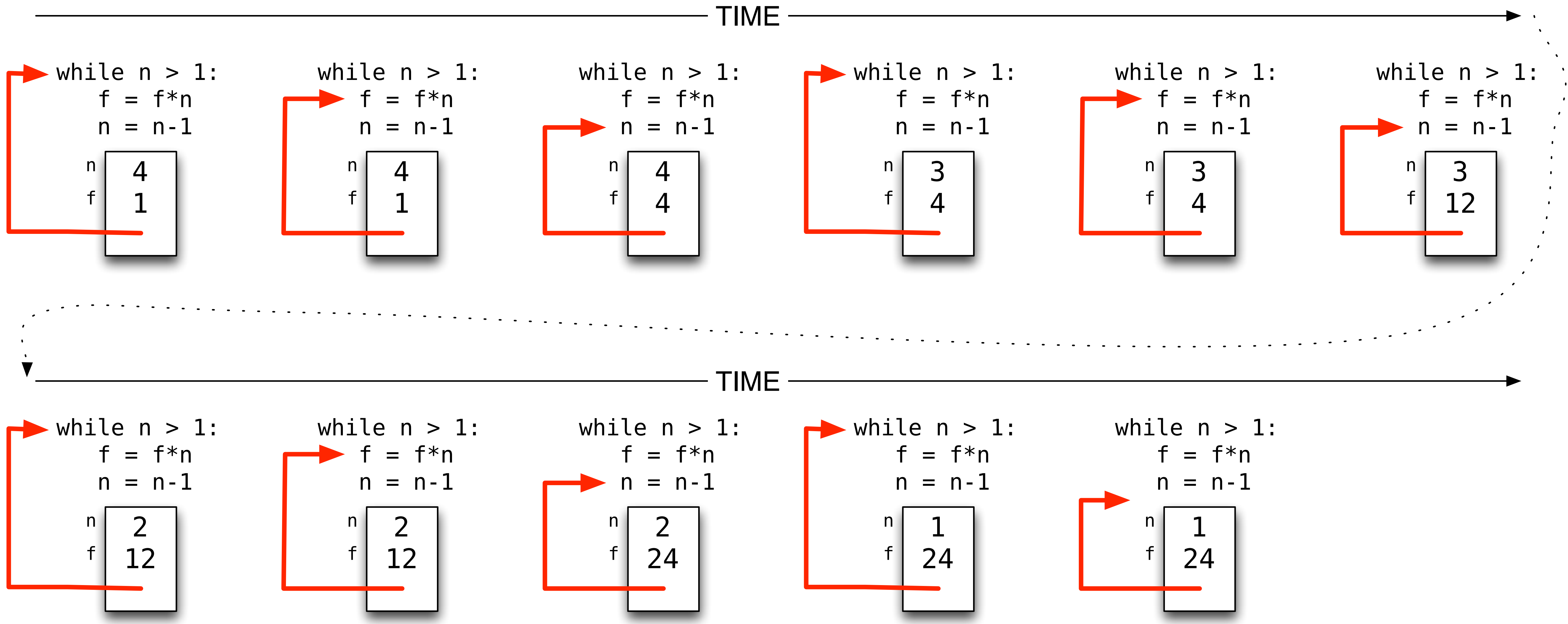
Computation means **modifying** the bits in memory & registers, step-by-step until we’re done.

```
0 read r1         # read dividend from the user
1 write r1        # echo the input
2 read r2         # read divisor from the user
3 jeqz r2 7       # jump to 7 if trying to divide by 0
4 div r3 r1 r2    # divide user's parameters
5 write r3        # print the result
6 halt
7 setn r3 0       # 0 is the return for division by 0
8 write r3        # print the result
9 halt
```

Imperative programming

Step-by-step instructions for updating memory (data)

```
while n > 1:
    f = f*n
    n = n-1
```



TIME

```
while n > 1:
    f = f*n
    n = n-1
```

n 4
f 1

```
while n > 1:
    f = f*n
    n = n-1
```

n 4
f 1

```
while n > 1:
    f = f*n
    n = n-1
```

n 4
f 4

```
while n > 1:
    f = f*n
    n = n-1
```

n 3
f 4

```
while n > 1:
    f = f*n
    n = n-1
```

n 3
f 4

```
while n > 1:
    f = f*n
    n = n-1
```

n 3
f 12

TIME

```
while n > 1:
    f = f*n
    n = n-1
```

n 2
f 12

```
while n > 1:
    f = f*n
    n = n-1
```

n 2
f 12

```
while n > 1:
    f = f*n
    n = n-1
```

n 2
f 24

```
while n > 1:
    f = f*n
    n = n-1
```

n 1
f 24

```
while n > 1:
    f = f*n
    n = n-1
```

n 1
f 24

A Mathematician’s viewpoint

Computation means **evaluating** an expression to get its value.

$$2 + 2$$

$$8 \sin^3 x + y^2$$

$$\frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} f(t)\ e^{-i\omega t}\ dt$$

$$\{\ x \in \mathbb{R}\ |\ x^3 > 5\ \}\ \cap\ \{\ y \in \mathbb{R}\ |\ y^2 < 5\}$$

Functional programming

Calculating answers (by repeatedly evaluating sub-calculations)

$$\mathrm{fact}(n) := \begin{cases} 1 & \text{if } n = 0 \\ n \times \mathrm{fact}(n-1) & \text{otherwise} \end{cases}$$

$$\begin{aligned} \therefore \qquad \mathrm{fact}(4) &= 4 \times \mathrm{fact}(3) \\ &= 4 \times (3 \times \mathrm{fact}(2)) \\ &= 4 \times (3 \times (2 \times \mathrm{fact}(1)))) \\ &= 4 \times (3 \times (2 \times (1 \times \mathrm{fact}(0)))) \\ &= 4 \times (3 \times (2 \times (1 \times 1))) \\ &= 4 \times (3 \times (2 \times 1)) \\ &= 4 \times (3 \times 2) \\ &= 4 \times 6 \\ &= 24 \end{aligned}$$

Features of functional programming

A functional program contains no assignment statements.
A variable’s value, once initialized, never changes.

Functional programs use a somewhat limited set of language features.
variables, primitive values, conditionals, function definitions & calls

A function’s only purpose is to compute its result; it has **no side effects**.

Functional programs have **referential transparency**.
An expression *always* evaluates to the same result, given the same input.

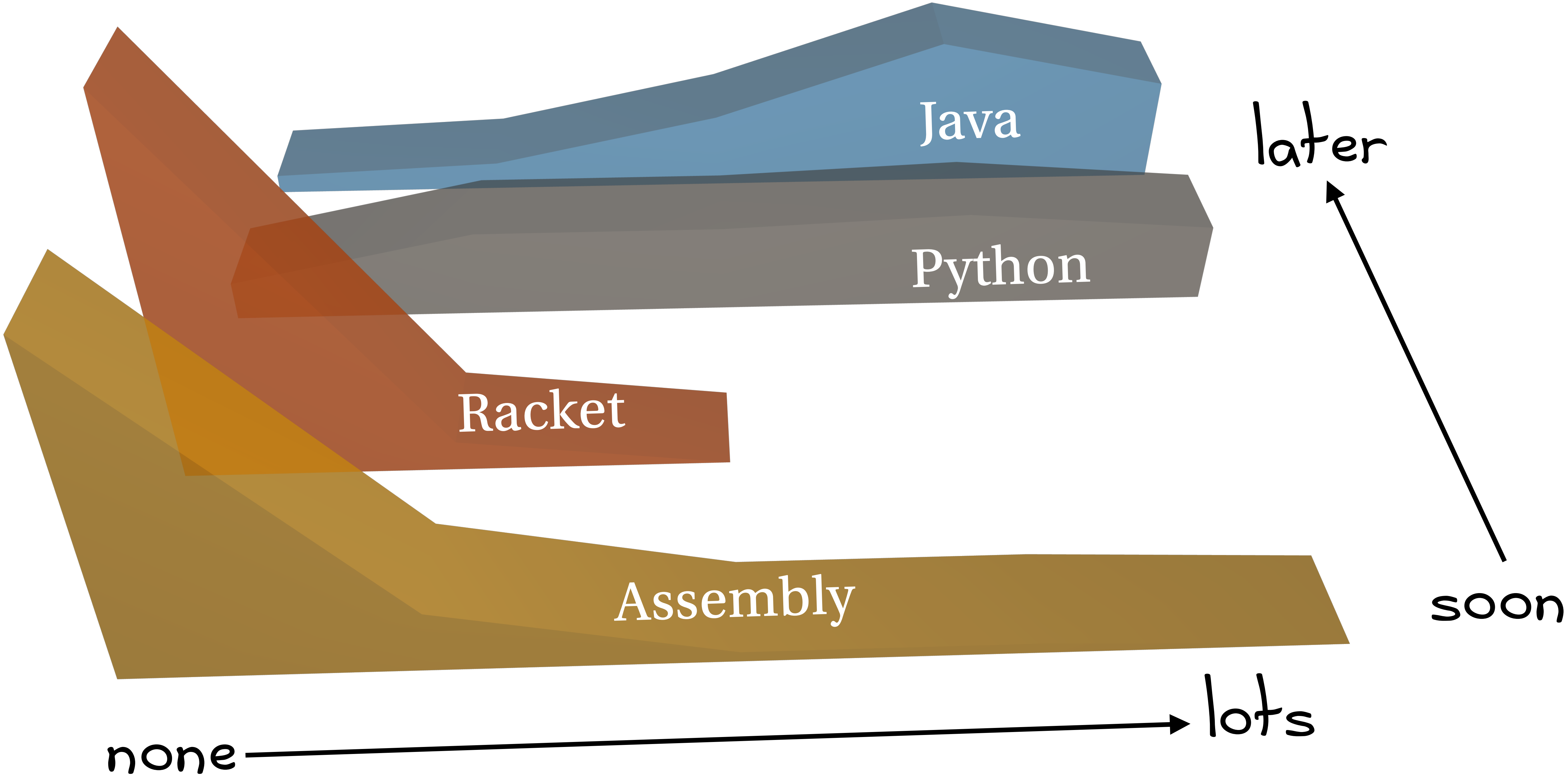
Why are we learning functional programming?

It can teach us something about computation.

Most modern language are a hybrid of imperative & functional styles.

It helps us learn how to choose the right tool for the right job.

Prior experience: programming languages



Math notation is not consistent

$\sin x$

$x + y$

x^2

$-y$

$|-3|$

$\sqrt{2}$

$(a + b) - c$

Racket notation *is* consistent

$\sin x$

```
(sin x)
```

$x + y$

```
(+ x y)
```

x^2

```
(sqr x)
```

$-y$

```
(- y)
```

$|-3|$

```
(abs -3)
```

$\sqrt{2}$

```
(sqrt 2)
```

$(a + b) - c$

```
(- (+ a b) c)
```

Racket: operations (*s-expressions*)

(op arg$_1$ *arg*$_2$ … *arg*$_n$*)*

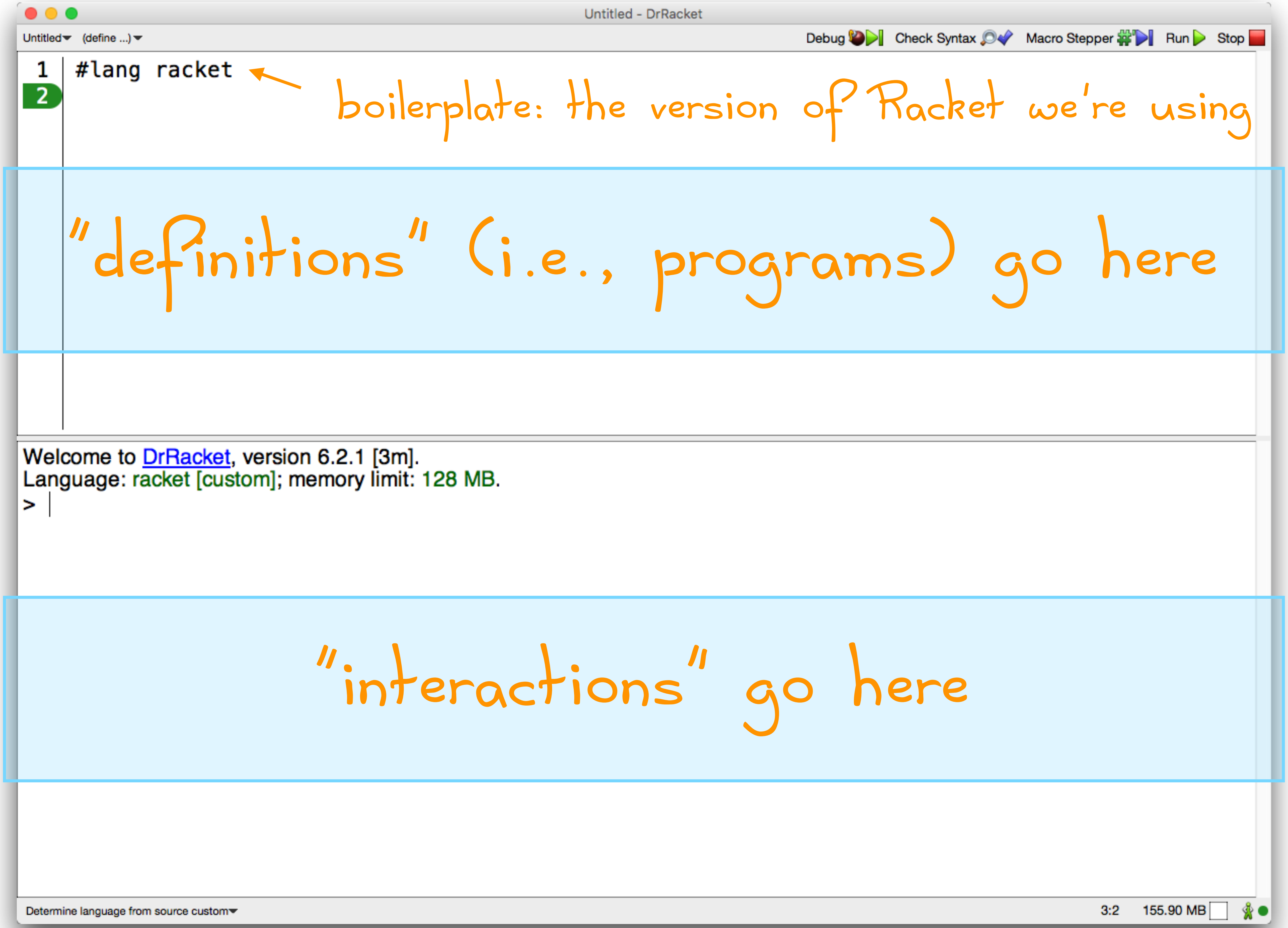
- **Rules:**
  - the operation always comes first
  - its arguments (if there are any) follow the operation
  - no commas between arguments
  - everything goes between parentheses
- **Common mistakes:**
  - forgetting parentheses
  - rational *vs.* integer division (`/` *vs.* `quotient`)
  - equality (= *vs.* `equal?`)

https://en.wikipedia.org/wiki/File:Perry_Platypus.png

Dr. Racket

an **I**ntegrated **D**evelopment **E**nvironment (**IDE**) for Racket

Run the program!



```
#lang racket
```

boilerplate: the version of Racket we're using

"definitions" (i.e., programs) go here

```
Welcome to DrRacket, version 6.2.1 [3m].
Language: racket [custom]; memory limit: 128 MB.
> 
```

"interactions" go here

Racket: “variables”

They’re called variables, but we won’t vary them (i.e., their values are constant).

"bind" a value to a variable

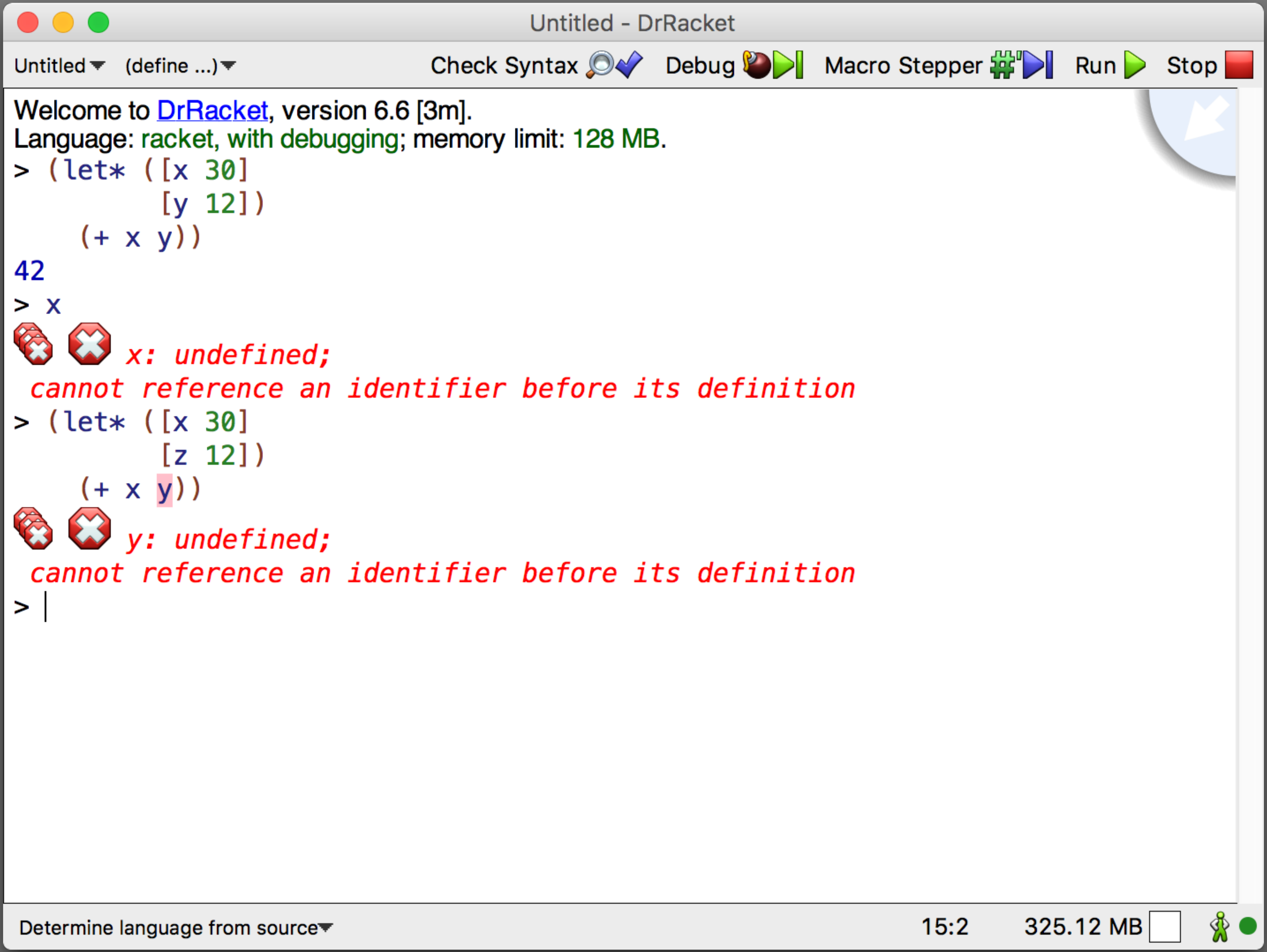
(let* ([*var*$_1$ *expr*$_1$]

...

[*var*$_n$ *expr*$_n$])

body)

"scope" of variables



```
Welcome to DrRacket, version 6.6 [3m].
Language: racket, with debugging; memory limit: 128 MB.
> (let* ([x 30]
         [y 12])
    (+ x y))
42
> x
x: undefined;
 cannot reference an identifier before its definition
> (let* ([x 30]
         [z 12])
    (+ x y))
y: undefined;
 cannot reference an identifier before its definition
> 
```