Intro to Racket & Java

Please take a handout &
Sit in row H or lower

How do we learn a new programming language?

What kinds of things do we want to know first?

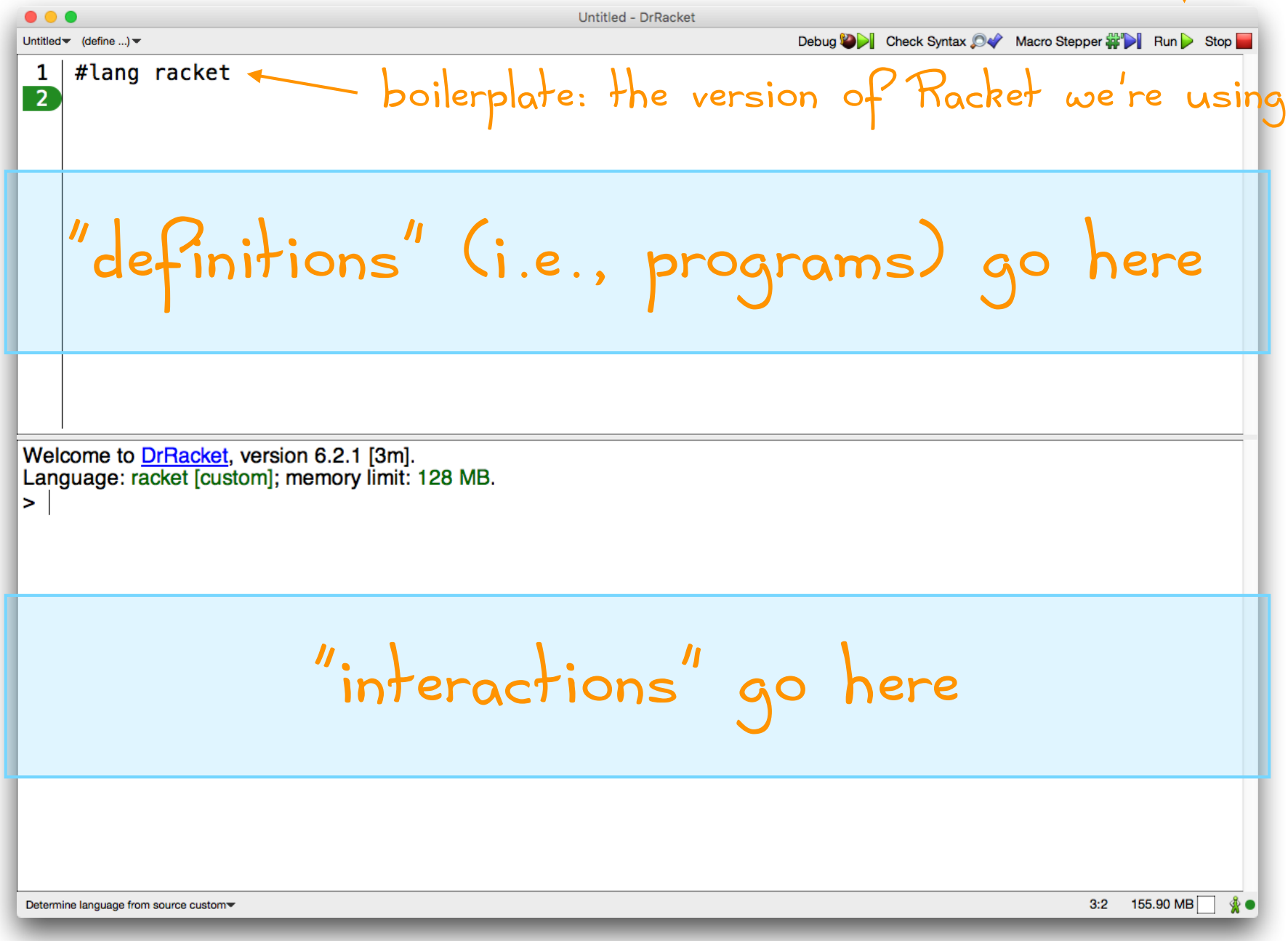
What does it feel like as you learn a new PL?

Racket

Why are we learning Racket?

Dr. Racket

an Integrated Development Environment (IDE) for Racket



Racket: “primitive” values

- integers
- booleans
  only `false` is false
- real numbers
- strings
- (lots more, non-primitive values)

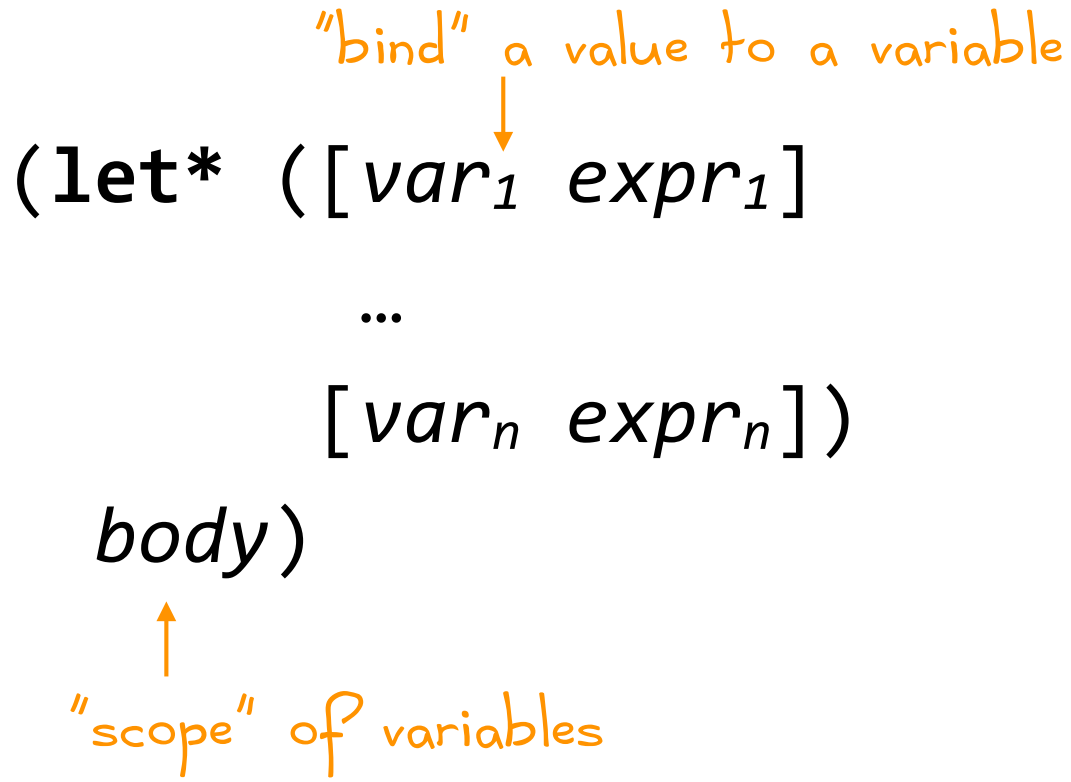
Racket: operations

(*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?`)

Racket: "variables"

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



Racket: conditionals

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

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

this is the most common form

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

Racket: functions

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

Exercise: write factorial in Racket

$$n! = \begin{cases} 1 & : n = 0 \\ n * (n-1)! & : \text{otherwise} \end{cases}$$

```
(define (factorial n)
   …
   your code here
   … )
```

Syntax reminders:

```
(op arg1 arg2 … argn)

(let* ([var1 expr1]
         …
        [varn exprn])
  body)

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

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

Exercise: write factorial in Racket

$$n! = \begin{cases} 1 & : n = 0 \\ n * (n-1)! & : \text{otherwise} \end{cases}$$

```
(define (fact n)
  (if (= n 0)
      1
      (* n (fact (- n 1)))))
```

sum in Racket

$$\mathrm{sum}(n) = \begin{cases} 0 & : n = 0 \\ n + \mathrm{sum}(n-1) & : \mathrm{otherwise} \end{cases}$$

```
(define (sum n)
  (if (= n 0)
      0
      (+ n (sum (- n 1)))))
```

fibonacci in Racket

$$fib(n) = \begin{cases} 1 & : n = 1 \\ 1 & : n = 2 \\ fib(n-2) + fib(n-1) & : n > 2 \end{cases}$$

```
(define (fib n)
  (cond [(= n 1) 1]
        [(= n 2) 1]
        [else (+ (fib (- n 1)) (fib (- n 2)))]))
```

upload.wikimedia.org/wikipedia/commons/8/8c/Key_break.jpg

Java

Expressions vs. Statements

Expressions produce values

```
2+2
```

```
Math.sin(5*y)
```

```
"he" + "llo"
```

Statements change something

```
System.out.println(“Hello, world!”);
```

```
count = count+1;
```

Java: Variables

Imperative variables vary!

But you have to "introduce" them before you can use them.

new variable with initial value

```
type var = expr;
```

```
type var;
```

new variable with

introducing new variables (declaration statements)

new value for the variable

```
var = expr;
```

modifying existing variables (assignment statements)

Recall

```
int LIMIT = 1000000;          ← declaration
int count = 0;                ← declaration
while (count < LIMIT) {
    System.out.println("I'm excited about CS 60!");
    count = count + 1;        ← assignment
}
```

Java: conditional statements

```
if (condition) {
   statements
}
```

```
if (condition) {
   statements1
} else {
   statements2
}
```

```
if (condition1) {
   statements1
} else if (condition2) {
   statements2
} else {
   statements3
}
```

and so on

Java: while loops

```
while (condition) {
    statements
}
```

```
int LIMIT = 1000000;
int count = 0;
while (count < LIMIT) {
    System.out.println("I'm excited about CS 60!");
    count = count + 1;
}
```

What's the difference?

```
if (condition) {
   statements
}
```

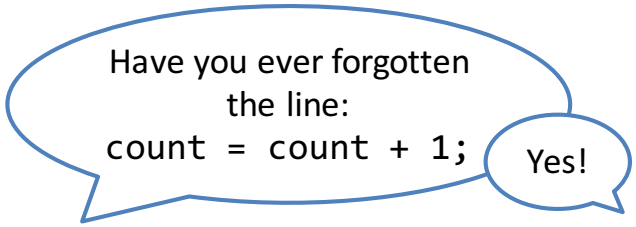
```
while (condition) {
   statements
}
```

```
if (count < LIMIT) {
    System.out.println("I'm excited about CS 60!");
    count = count + 1;
}
```

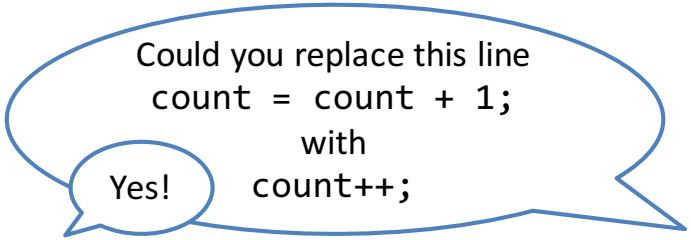
```
while (count < LIMIT) {
    System.out.println("I'm excited about CS 60!");
    count = count + 1;
}
```

Java: for loops

```
int count = 0;
while (count < LIMIT) {
    System.out.println("I'm excited about CS 60!");
    count = count + 1;
}
```




```
for (int count = 0; count < LIMIT; count = count+1) {
    System.out.println("I'm excited about CS 60!");
}
```




```
for (int count = 0; count < LIMIT; count++) {
    System.out.println("I'm excited about CS 60!");
}
```

Java: Non-Recursive Factorial

Since we haven't said how to define functions yet!

The goal: set answer to be n!

```
int answer = ?;
for (int i = ?; i ? n; i++) {
    answer = answer * i;
}
```

Exercise: sum n numbers in Java

$$answer = \sum_{i=1}^{n} i$$
$$= 1 + 2 + \cdots + n$$

```
int answer = 0;
for (int i = 1; i <= n; i++) {
    answer = answer + i;
}
```

Tracing Execution in Java

Suppose n is initially 3. What answer is computed?

```
int thisFib = 1;
int nextFib = 1;
for (int i = 1; i < n; i++) {
    int prevFib = thisFib;
    thisFib = nextFib;
    nextFib = prevFib + thisFib;
}
int answer = thisFib;
```

Compare & Contrast

Conditionals - Why the difference?

Conditional expressions in Racket always have 3 parts

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

Conditional statements in Java are more varied.

```
if (condition) {
   statements
}
```

```
if (condition) {
   statements1
} else {
   statements2
}
```

Is Java more Loopy than Racket?

Loops are used all the time in Java.

You won't see while loops in Racket.

```
(while … …)
```

Why not?

What does Racket do instead?