Lenguajes de Programación

2020 — 2º Semestre

Clase 2:

- Introducción a Racket

Federico Olmedo

Introducción a Racket

Racket is a functional language

Functional style

```
(define (factorial n)
  (if (zero? n)
      1
      (* n (factorial (- n 1)))))
```

FACTORIAL PROGRAM IN RACKET

Built around the **evaluation of expressions and the application of functions (on inmutable data)**

Closer to expressing **what** to compute

Closer to **mathematics**

Imperative style

```
long factorial(int n)
{
  int c;
  long result = 1;
  for (c = 1; c <= n; c++)
    result = result * c;
  return result;
}
```

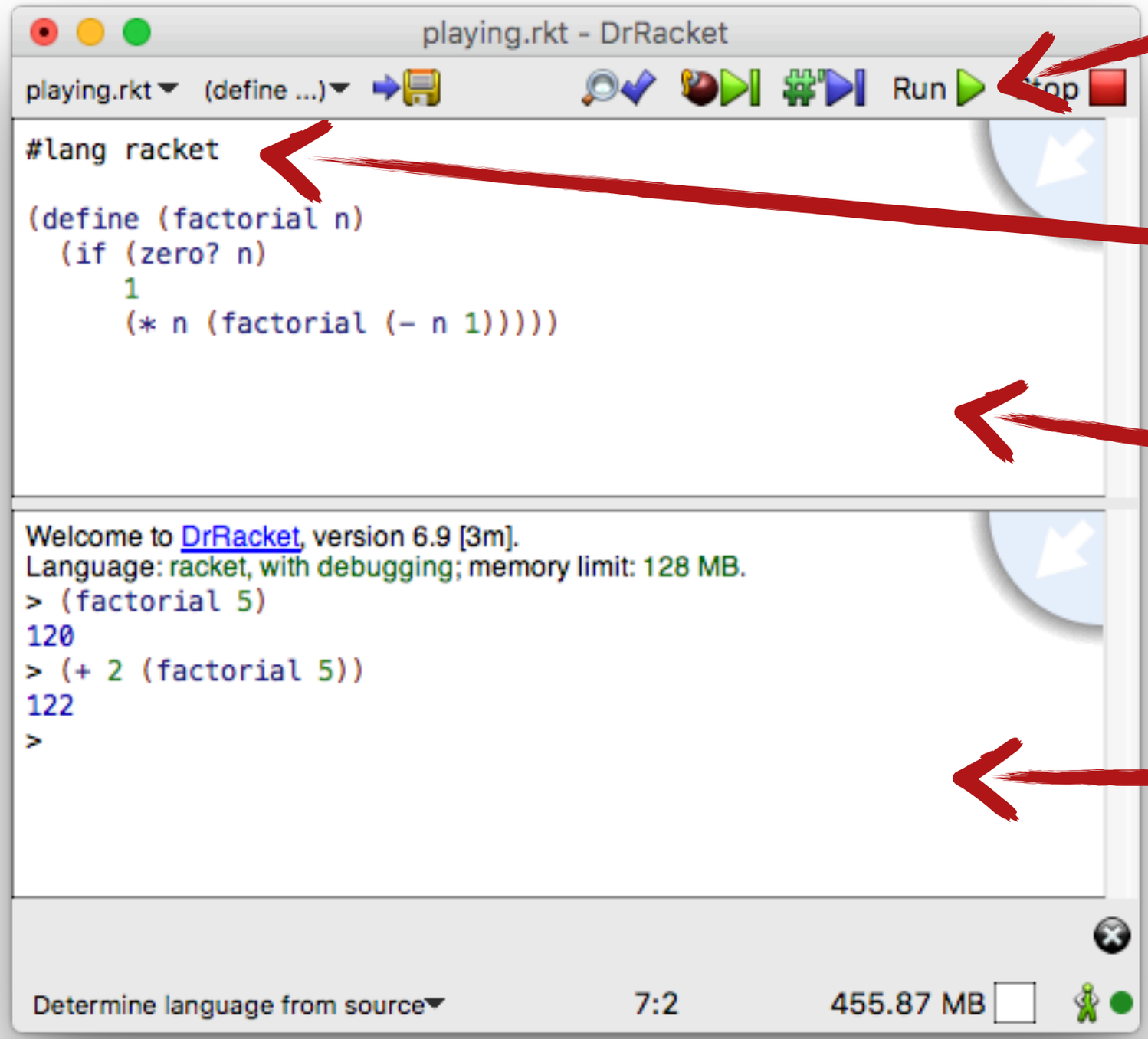
FACTORIAL PROGRAM IN C

Built around the execution of **sequences of commands** for their **effects on mutable storage**

Closer to expressing **how** to compute it

Closer to **computer hardware**

DrRacket: Our Development Environment



Evaluates programs in the DW, making these defs available in the IW (also removes old defs)

LANGUAGE SELECTION: Determines available primitives

DEFINITION WINDOW (DW)

INTERACTION WINDOW (IW) REPL: read–evaluate–print loop

5 Introducing basic elements

Introduction to Racket

Define functions

```
(my-max a b)
(pick-random a b)
```

Hint: use function `(random)` that returns a random number between 0 and 1

Primitive datatypes and operators

- `number` `(+, -, *, /, quotient, sqrt, …, <, <=, =, zero?, …)`
- `boolean` `(and, or, not, …)`
- `string` `(string-length, string-append, substring, …)`
- `symbol` `(equal?, string->symbol, …)`

Conditionals

- `(if` *`guard t-brach f-branch`*`)`
- `(cond [`$guard_1$ $expr_1$`] … [`$guard_n$ $expr_n$`])`

Global and local definition of identifiers

- `(define` *`id expr`*`)`
- `(let ([`id_1 $expr_1$`] … [`id_n $expr_n$`])` *`body`*`)`

Function definition

- `(define (`*`func-name`* arg_1 … arg_n`)` *`func-body`*`)`

Introduction to Racket

Inmutable data structures:

Pairs

- `(cons a b)` — `(car, cdr)`

List

- `(cons a1 (cons a2 (… (cons an empty)…)))`
- `(list a1 a2 … an)`

`(append, length, first, rest, reverse, list-ref, …)`

Mutable data structures:

Vectors

- `(vector a1 a2 … an)` — `(vector-ref, vector-set!, vector-length)`

Define function

```
(pick-random-vector v)
```

that returns a random element from vector `v`. To this end, use function `(random k)`, which returns a random integer between `0` and `k-1`

8 Takeaways

- Prefix notation
- Dynamically type-checked
- Standard primitive and compound datatypes.
- Difference between mutable and inmutable data structures.
- Secure access to list and vectors.

9 Lecture material

Bibliography

- PrePLAI: Introduction to functional programming in Racket [Sections 1-2]

For a more detailed reference, see the online Racket documentation:

- Racket Guide: tutorial
- Racket Reference: reference manual