Introducción

Tipos primitivos, imágenes y expresiones en Racket

Laura Pomponio

Departamento de Ciencias de la Computación (DCC)

Escuela de Ciencias Exactas y Naturales (ECEN)

2022

Content

¿Qué es Racket?

Racket es

- un lenguaje de programación (basado en Lisp y Scheme)
- una familia de lenguajes de programación (hay variantes de Racket)
- una familia de herramientas, las principales son:
  - `racket`: leguaje, compilador e intérprete
  - `drracket` (DrRacket): entorno de programación
  - `raco`: herramienta de línea de comando para instalar paquetes

https://racket-lang.org/

Como todo lenguaje tiene

- vocabulario (palabras o símbolos)
  Por ejemplo: `(, ), +,-,*,/,<,>,=, and, or, not, 1,2,...`
- sintaxis (reglas que indican cómo se combinan los elementos del lenguaje)
  `(* 8 3)` esta expresión es válida en Racket
  `8 * 3` esta expresión NO es válida en Racket
- semántica (significado de las expresiones)
  `(* 8 3)` significa que 8 es multiplicado por 3 y por tanto reduce a 24
  "`(* 8 3)`" significa que (* 8 3) es una secuencia de caracteres (un texto).

Comencemos con el lenguaje **Estudiante Principiante**.

Expresiones en Racket

¿Qué expresiones podemos escribir?

- **datos**
  - números (Number)
  - valores de verdad (Boolean)
  - cadenas de caracteres (Strings)
  - imágenes (Image)
- **expresiones utilizando operadores (o funciones)**

Racket utiliza notación **prefija** y **paréntesis**.

(<*operador*> <*operando1*> **...** <*operandoN*>)

matemática	Racket
6×5	`(* 6 5)`

- otras expresiones que veremos más adelante

Expresiones en Racket

¿Qué expresiones podemos escribir?

- **datos**
  - números (Number)
  - valores de verdad (Boolean)
  - cadenas de caracteres (Strings)
  - imágenes (Image)
- **expresiones utilizando operadores (o funciones)**

Racket utiliza notación **prefija** y **paréntesis**.

(*<operador>* *<operando1>* **...** *<operandoN>*)

matemática	Racket
6×5	`(* 6 5)`

- otras expresiones que veremos más adelante

Expresiones en Racket

¿Qué expresiones podemos escribir?

- **datos**
  - números (Number)
  - valores de verdad (Boolean)
  - cadenas de caracteres (Strings)
  - imágenes (Image)
- **expresiones utilizando operadores (o funciones)**

Racket utiliza notación **prefija** y **paréntesis**.

(<*operador*> <*operando1*> ... <*operandoN*>)

matemática	Racket
6×5	`(* 6 5)`

- otras expresiones que veremos más adelante

Expresiones en Racket

¿Qué expresiones podemos escribir?

- **datos**
  - números (Number)
  - valores de verdad (Boolean)
  - cadenas de caracteres (Strings)
  - imágenes (Image)
- **expresiones utilizando operadores (o funciones)**

Racket utiliza notación **prefija** y **paréntesis**.

(<*operador*> <*operando1*> **...** <*operandoN*>)

matemática	Racket
6×5	`(* 6 5)`

- otras expresiones que veremos más adelante

Expresiones aritméticas

¿Cómo escribiríamos en Racket las siguientes expresiones?

- $4 - 2 \times 5$
- $(4 - 2) \times 5$
- $1 + 2 \times 3 + \sqrt{\frac{14-5}{2+7}}$
- $10 \div 3$
- 16^2
- $\sqrt{16}$
- $\sqrt[5]{12}$ recordemos que esto es igual a $12^{\frac{1}{5}}$
- $-2, 1/2, \pi$

Lápiz y papel: pasos de reducción

```
(+ 1 (* 2 3) (sqrt (/ (- 14 5) (+ 2 7))))
==<definición de *>
(+ 1 6 (sqrt (/ (- 14 5) (+ 2 7))))
== <definición de - >
(+ 1 6 (sqrt (/ 9 (+ 2 7))))
==<definición de + >
(+ 1 6 (sqrt (/ 9 9)))
==<definición de / >
(+ 1 6 (sqrt 1))
==<definición de sqrt >
(+ 1 6 1)
==<definición de + >
(+ 7 1)
==<definición de + >
8
```

Además de **hacer cálculos** utilizando

+, -, *, /, sqr, sqrt, y otros;

podemos **compara valores numéricos** con

<, >, <=, >=, =.

Cadenas de caracteres (String)

- Las doble comillas (“) son las que me permiten construir un String.
- “A” es un String de 1 caracter.
- “8 + 4” es un String de 5 caracteres.

8		+		4

- “memoria” es un String de 7 caracteres.

String	m	e	m	o	r	i	a
posición/índice	0	1	2	3	4	5	6

Observar que la longitud es 7 pero el último elemento está en la posición 6.

Cadenas de caracteres (String)

Si queremos unir “La longitud de la palabra ”, “memoria ”, “es ” y “7”, para obtener:
“La longitud de la palabra memoria es 7”
podemos utilizar string-append.

Algunas otras operaciones muy utilizadas sobre String.
string-append, string-length, substring, number->string, string->number.

Podríamos combinarlas...

Lápiz y papel: pasos de reducción

```
(string-append "La longitud de la palabra "
               "memoria" "es "
               (number->string (string-length "memoria")))
==<def. de string-length>
(string-append "La longitud de la palabra "
               "memoria" "es "
               (number->string 7))
==<def. number->string>
(string-append "La longitud de la palabra "
               "memoria" "es " "7")
==<def. string-append>
"La longitud de la palabra memoria es 7"
```

Valores de verdad (Boolean)

Recordemos que una **proposición** es una expresión que asume un único valor de verdad, es **verdadero** o es **falso**. Por ejemplo: $(3 \geq 5)$ es una proposición cuyo valor es falso.

- verdadero: #true o #t
- falso: #false o #f

```
Evaluamos la siguiente expresión.
(>= 3 5)
==
#false
```

Valores de verdad (Boolean)

- verdadero: #true o #t
- falso: #false o #f
- Operadores
  - conjunción ($\wedge$): and
  - disyunción ($\vee$): or
  - negación ($\neg$): not

Recordemos que

- los operadores quedan definidos por las **tablas de verdad**
- para que una **conjunción** sea **verdadera**, **todas** las proposiciones que la componen deben ser **verdaderas**
- para que una **disyunción** sea **verdadera**, **al menos una** proposición debe ser **verdadera**

Lapiz y papel: pasos de reducción (1)

```
(and #true (and #true (and #true #true)))
==< def. de and >
(and #true (and #true #true))
==< def. de and >
(and #true #true)
==< def. de and >
#true
```

```
(and (and #true #true) #true (and #true #true))
==< def. de and >
(and #true #true (and #true #true))
==< def. de and >
(and #true #true #true)
==< def. de and >
#true
```

Lapiz y papel: pasos de reducción (2)

```
(or #false (or #false (or #false #true)))
==< def. de or >
(or #false (or #false #true))
==< def. de or >
(or #false #true)
==< def. de or >
#true
```

```
(or (or #false #false) #false (or #true #false))
==< def. de or >
(or #false #false (or #true #false))
==< def. de or >
(or #false #false #true)
==< def. de or >
#true
```

Lapiz y papel: pasos de reducción (3)

Evaluación de cortocircuito

```
(and (and #true #false) #true (or #true #true))
==< def. de and >
(and #false #true (or #true #true))
==<def. de and, evaluación de cortocircuito>
#false
```

```
(or (or #true #false) #false (and #true #true))
==< def. de or >
(or #true #false (and #true #true))
==<def. or, evaluación de cortocircuito>
#true
```

Lapiz y papel: pasos de reducción (4)

¿Cuáles serían los pasos de reducción de la siguiente expresión?

```
(and (< (+ 5 2) 10)
     (not (or (= (+ 2 1) 3) (< 2 0)))
     (= 7 7) )
```

Imágenes (Image)

Colores: “red”, “blue”, “yellow”, “gray”, “violet”, “lime”, y muchos otros.
Hay varias funciones definidas para crear imágenes y manipularlas.
circle, rectangle, overlay, place-image, image-width, image-height

En DrRacket se pueden pegar imágenes png,jpg y demás.

Tipos e igualdades

¿de qué tipo es el elemento? ¿son iguales los elementos?

Tipo	Predicado	Comparación
Number	number?	=
String	string?	string=?
Boolean	boolean?	boolean=?
Image	image?	image=?

Hay varios predicados sobre tipos y sobre relaciones de orden.

Constantes y funciones

Expresión constante

```
( define <identificador> <expresión> )
```

```
( define SALUDO "Les damos la bienvenida")
```

¿de qué tipo es **SALUDO**?

```
; SALUDO : String
```

Por convención usaremos MAYÚSCULAS en los identificadores/nombres de constantes.

Constantes y funciones

Funciones

```
(define (<identificador>
          <argumento1>...<argumentoN>)
          <expresión> )
```

```
; f: Number Number Number -> Number
( define (f x y z) (+ x y z))
```

```
; elValorEs: Number -> String
( define (elValorEs n)
         (string-append "El valor es "
                        (number->string n)))
```

Ejemplo: Un reintegro del %30

Un sistema de pago, establece el reintegro del %30 en una compra.
Definir una función que dado el valor de una compra, calcule el valor final de la misma, luego de haber recibido el reintegro.

Reglas de reducción: función

- Definición de una función
  `(define (f x) e )`
  siendo **`e`** una expresión válida.
- Evaluación/invocación de una función en cierto valor
  `(f a )`
  siendo **`a`** una expresión que reduce a un valor.
- Regla de reducción (lápiz y papel)

```
(f a)
== <por definición de f>
e[a/x]
```

Reemplazamos en **`e`** *todas las ocurrencias de* **`x`** *por* **`a`**.

Papel y lápiz: pasos de reducción

- Definición de una función

```
(define (f x)
        (- x (sqrt x))
)
```

- Lápiz y papel: pasos de reducción

```
(f 16)
== <por definición de f>
(- 16 (sqrt 16))
==<por def. de sqrt>
(- 16 4)
==<por def. de - >
12
```