EBook Gratis

APRENDIZAJE racket

Free unaffiliated eBook created from
Stack Overflow contributors.

#racket

Tabla de contenido

Acerca de 1

Capítulo 1: Empezando con la raqueta 2

Observaciones 2

Versiones 2

Examples 3

¡Hola Mundo! 3

Raqueta 4

Instalación 4

Definición de función recursiva simple 4

Encuentra fuentes de raqueta en todos los subdirectorios 4

Instalación o configuración 5

Capítulo 2: Cierres 6

Introducción 6

Observaciones 6

Examples 6

Cierre con entorno estático 6

Capítulo 3: Comentarios 8

Observaciones 8

Examples 8

Comentarios de una sola línea 8

Bloquear comentarios 8

S-expresión de comentarios 8

Comentarios en at-exps 8

Capítulo 4: Funciones 10

Sintaxis 10

Examples 10

Llamadas de función simple 10

Argumentos de palabras clave 10

La función `apply` 11

Definiciones de funciones 11

Capítulo 5: Funciones de orden superior ... 13
Examples ... 13
Mapa ... 13
Doblez ... 13
Filtrar ... 13
Componer ... 14
Curry ... 14
Capítulo 6: Garabato ... 16
Introducción ... 16
Examples ... 16
Párrafos y secciones ... 16
Documentando un enlace proporcionado por un paquete ... 16
Capítulo 7: Pasos de instalación (Linux) ... 18
Examples ... 18
Instalación o configuración ... 18
Pasos de instalación para Linux: ... 18
Descargando ... 18
Iniciando el instalador ... 18
Instalación ... 19
Empezando DrRacket ... 19
Ejecutando tu primer programa ... 19
Capítulo 8: Pasos de instalación (macOS) ... 21
Examples ... 21
Instalación o configuración ... 21
Pasos de instalación para macOS: ... 21
Descargando ... 21
Iniciando el instalador ... 21
Configuración de herramientas de línea de comandos ... 21
Ejecutando tu primer programa ... 21
Capítulo 9: Pasos de instalación (Windows) ... 23
Examples ... 23

Instalación o configuración ... 23
Pasos de instalación para Windows: ... 23
Descargando ... 23
Iniciando el instalador ... 23
Configuración de herramientas de línea de comandos ... 23
Ejecutando tu primer programa ... 24
Capítulo 10: Recursion ... 25
Examples ... 25
Utilizando define ... 25
Usando let-rec ... 25
Usando una let con nombre ... 25
Usando rec ... 26
Usando funciones de orden superior en lugar de recursión ... 26
Creditos ... 28

Acerca de

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: racket

It is an unofficial and free racket ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official racket.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Capítulo 1: Empezando con la raqueta

Observaciones

Esta sección proporciona una descripción general de qué es la raqueta y por qué un desarrollador puede querer usarla.

También debe mencionar cualquier tema grande dentro de la raqueta y vincular a los temas relacionados. Dado que la Documentación para la raqueta es nueva, es posible que deba crear versiones iniciales de los temas relacionados.

Versiones

Versión	Documentación	Fecha de lanzamiento
Construcciones nocturnas	Última documentación	2999-12-31
Versión 6.9	Documentación	2017-04-01
Versión 6.8	Documentación	2017-01-01
Versión 6.7	Documentación	2016-10-01
Versión 6.6	Documentación	2016-07-01
Versión 6.5	Documentación	2016-04-01
Versión 6.4	Documentación	2016-02-01
Versión 6.3	Documentación	2015-11-01
Versión 6.2.1	Documentación	2015-08-01
Versión 6.2	Documentación	2015-06-01
Versión 6.1.1	Documentación	2014-11-01
Versión 6.1	Documentación	2014-08-01
Versión 6.0.1	Documentación	2014-05-01
Versión 6.0	Documentación	2014-02-01
Versión 5.93	Documentación	2014-01-01
Versión 5.92	Documentación	2014-01-01
Versión 5.3.6	Documentación	2013-08-01

Versión	Documentación	Fecha de lanzamiento
Versión 5.3.5	Documentación	2013-06-01
Versión 5.3.4	Documentación	2013-05-01
Versión 5.3.3	Documentación	2013-02-01
Versión 5.3.2	Documentación	2013-02-01
Versión 5.3.1	Documentación	2012-11-01
Versión 5.3	Documentación	2012-08-01
Versión 5.2.1	Documentación	2012-03-01
Versión 5.2	Documentación	2011-11-01
Versión 5.1.3	Documentación	2011-08-01
Versión 5.1.2	Documentación	2011-08-01
Versión 5.1.1	Documentación	2011-04-01
Versión 5.1	Documentación	2011-02-01
Versión 5.0.2	Documentación	2010-11-01
Versión 5.0.1	Documentación	2010-08-01
Versión 5.0	Documentación	2010-06-01

Examples

¡Hola Mundo!

El siguiente ejemplo declara que un fragmento de código se escribe en Racket y luego imprime la cadena `Hello, world`.

```
#lang racket
"Hello, world!"
```

El código de la raqueta se puede ejecutar directamente desde la línea de comandos o en el IDE de DrRacket. Al escribir `racket` en la línea de comando, se iniciará una REPL, y al escribir `racket` seguida de un nombre de archivo se evaluará el contenido del archivo. Por ejemplo, supongamos que el archivo `hello.rkt` contiene el código anterior. Este es un ejemplo de la ejecución de Racket en la línea de comandos.

```
$ racket
```

```
Welcome to Racket v6.5.
> "Hello, world!"
"Hello, world!"
> (exit)
$ racket hello.rkt
"Hello, world!"
```

Raqueta

Racket es un lenguaje de programación de espectro completo. Va más allá de Lisp y Scheme con dialectos que admiten objetos, tipos, holgazanería y más. Racket permite a los programadores vincular componentes escritos en diferentes dialectos, y permite a los programadores crear nuevos dialectos específicos del proyecto. Las bibliotecas de Racket admiten aplicaciones desde servidores web y bases de datos hasta GUI y gráficos.

La documentación oficial, completa y muy bien escrita se puede encontrar en [http://docs.racket-lang.org/◆◆1] . En este sitio (Documentación de desbordamiento de pila) puede encontrar ejemplos aportados por usuarios.

Instalación

Vaya a http://racket-lang.org y haga clic en el botón de descarga.

Definición de función recursiva simple

En Racket, usamos la recursión muy frecuentemente. Aquí hay un ejemplo de una función que suma todos los números desde cero hasta el parámetro, `n` .

```
(define (sum n)
    (if (zero? n)
        0
        (+ n (sum (sub1 n)))))
```

Tenga en cuenta que hay muchas funciones útiles basadas en la conveniencia utilizadas aquí, como `zero?` y `sub1` . Cada uno, respectivamente, hace exactamente lo que podrías esperar: `zero?` devuelve un valor booleano que indica si el número dado era igual a cero y `sub1` resta uno de su argumento.

Encuentra fuentes de raqueta en todos los subdirectorios.

```
#lang racket
(for ([path (in-directory)]
  #:when (regexp-match? #rx"[.]rkt$" path))
  (printf "source file: ~a\n" path))
```

La línea `#lang` especifica el lenguaje de programación de este archivo. `#lang racket` estamos usando el lenguaje de programación de Raqueta, línea de base y batería incluida. Hay otros idiomas que van desde Racket flavors como Type Racket (`#lang typed/racket`) o el lenguaje de

documentación Scribble (`#lang scribble`), hasta lenguajes pequeños como el idioma para definir paquetes (`#lang info`).

La función `in-directory` construye una secuencia que recorre un árbol de directorios (comenzando con el directorio actual, de manera predeterminada) y genera rutas en el árbol. La forma `for` vincula la `path` a cada ruta en la secuencia y `regexp-match?` Aplica un patrón al camino.

Para ejecutar el ejemplo, instale Racket, inicie DrRacket, pegue el programa de ejemplo en el área superior de DrRacket y haga clic en el botón Ejecutar. Alternativamente, guarde el programa en un archivo y ejecute la `racket` desde la línea de comandos en el archivo.

Instalación o configuración

La instalación es muy sencilla. Si está acostumbrado a este tipo de cosas, simplemente vaya a https://download.racket-lang.org . Si lo prefiere, hay instrucciones de instalación paso a paso más detalladas para los siguientes sistemas:

- Pasos de instalación (Windows)
- Pasos de instalación (Linux)
- Pasos de instalación (macOS)

Lea Empezando con la raqueta en línea: https://riptutorial.com/es/racket/topic/1134/empezando-con-la-raqueta

Capítulo 2: Cierres

Introducción

Desde la etiqueta de cierre StackOverflow:

Un cierre es una función de primera clase que hace referencia a las variables del ámbito en el que se definió. Si el cierre aún existe después de que finalice su ámbito de definición, las variables sobre las que se cierra también continuarán existiendo.

Observaciones

A veces es útil considerar cierres y objetos como similares.

El venerable maestro Qc Na caminaba con su alumno, Anton. Con la esperanza de incitar al maestro a una discusión, Anton dijo: "Maestro, he escuchado que los objetos son algo muy bueno, ¿es cierto?" Qc Na miró a su alumno con lástima y respondió: "Foolish pupil - los objetos son simplemente los cierres de un hombre pobre". Castigado, Anton se despidió de su maestro y regresó a su celda, con la intención de estudiar los cierres. Leyó detenidamente toda la serie de documentos "Lambda: The Ultimate ..." y sus primos, e implementó un pequeño intérprete de Scheme con un sistema de objetos basado en el cierre. Aprendió mucho y esperaba informar a su maestro sobre su progreso.

En su próxima caminata con Qc Na, Anton intentó impresionar a su maestro diciendo: "Maestro, he estudiado diligentemente el asunto, y ahora entiendo que los objetos son realmente los cierres de un hombre pobre". Qc Na respondió golpeando a Anton con su bastón, diciendo "¿Cuándo aprenderás? Los cierres son un objeto de los pobres". En ese momento, Anton se iluminó.

Fuente: http://c2.com/cgi/wiki?ClosuresAndObjectsAreEquivalent

Examples

Cierre con entorno estático.

Un cierre es un procedimiento que mantiene el estado interno:

Definir un procedimiento que devuelva un cierre.

El procedimiento `make-an-adder` toma un argumento `x` y devuelve una función que se cierra sobre el valor. O para decirlo de otra manera, `x` está dentro del alcance léxico de la función devuelta.

```
#lang racket
(define (make-an-adder x)
  (lambda (y)
    (+ y x)))
```

Uso

Al llamar al procedimiento `make-an-adder` devuelve un procedimiento que es un cierre.

```
Welcome to DrRacket, version 6.6 [3m].
Language: racket, with debugging; memory limit: 128 MB.
> (define 3adder (make-an-adder 3))
> (3adder 4)
7
> (define 8adder (make-an-adder 8))
> (8adder 4)
12
```

Lea Cierres en línea: https://riptutorial.com/es/racket/topic/7176/cierres

Capítulo 3: Comentarios

Observaciones

Los tipos de comentarios más comunes son los comentarios de línea y s-expresión (usando `;` y `#;` respectivamente).

Es común utilizar de 1 a 3 puntos y coma en función del tipo de comentario realizado. Referirse a ???

Examples

Comentarios de una sola línea

```
; We make single line comments by writing out text after a semicolon
```

Bloquear comentarios

```
#| We make
block comments
like this |#
```

S-expresión de comentarios

```
#;(define (commented-out-function x)
    (print (string-append "This entire "
            "s-expression is commented out!")))
```

Comentarios en at-exps

Cuando un módulo está usando en expresiones, tales como:

```
#lang at-exp racket/base
```

o

```
#lang scribble/manual
```

Tienes acceso a los siguientes tipos de comentarios:

```
@;{Block text that goes
   until the closing
   brace.}
```

Tanto como:

```
@; Single line text.
```

Tenga en cuenta que si está usando un idioma que solo usa at-exps (como la mayoría de los idiomas de garabatos), necesitará usar uno de estos tipos de comentarios.

Lea Comentarios en línea: https://riptutorial.com/es/racket/topic/3147/comentarios

Capítulo 4: Funciones

Sintaxis

- (definir (nombrar argumentos ...) cuerpo)
- (argumentos de función ...)

Examples

Llamadas de función simple

Puede llamar a una función en Racket envolviéndola entre paréntesis con los argumentos detrás de ella. Esto parece `(function argument ...)` .

```
> (define (f x) x)
> (f 1)
1
> (f "salmon")
"salmon"
> (define (g x y) (string-append x y))
> (g "large" "salmon")
"largesalmon"
> (g "large " "salmon")
"large salmon"
```

Las operaciones como `+` y `*` son funciones, y usan la misma sintaxis que la llamada `f` o `g` .

```
> (+ 1 2)
3
> (* 3 4)
12
> (+ (* 3 3) (* 4 4))
25
```

Para obtener más información y ejemplos, consulte Llamadas de funciones en la Guía de la raqueta.

Argumentos de palabras clave

Las funciones de raqueta también pueden tener *argumentos de palabras clave* , que se especifican con una palabra clave seguida de la expresión de argumento. Una palabra clave comienza con los caracteres `#:` por lo que un argumento de palabra clave se ve como `#:keyword arg-expr` . Dentro de una llamada de función, este aspecto se ve `(function #:keyword arg-expr)` .

```
> (define (hello #:name n)
    (string-append "Hello " n))
> (hello #:name "John")
"Hello John"
```

```
> (hello #:name "Sarah")
"Hello Sarah"
> (define (kinetic-energy #:mass m #:velocity v)
    (* 1/2 m (sqr v)))
> (kinetic-energy #:mass 2 #:velocity 1)
1
> (kinetic-energy #:mass 6 #:velocity 2)
12
```

Para obtener más información y ejemplos, consulte Argumentos de palabras clave en la Guía de Racket.

La función `apply`

Si tiene una lista y desea utilizar los elementos de esa lista como argumentos para una función, se `apply` lo que desea:

```
> (apply string-append (list "hello" " " "and hi" " " "are both words"))
"hello and hi are both words"
> (apply + (list 1 2 3 4))
10
> (apply append (list (list "a" "b" "c") (list 1 2 3) (list "do" "re" "mi")))
(list "a" "b" "c" 1 2 3 "do" "re" "me")
```

`apply` toma dos argumentos. El primer argumento es la función a aplicar, y el segundo argumento es la lista que contiene los argumentos.

Una llamada de `apply` como

```
(apply + (list 1 2 3 4))
```

Es equivalente a

```
(+ 1 2 3 4)
```

La principal ventaja de `apply` es que funciona en listas calculadas arbitrarias, incluidas listas agregadas y listas que provienen de argumentos de función.

```
> (apply + (append (list 1 2 3 4) (list 2 3 4)))
19
> (define (sum lst)
    (apply + lst))
> (sum (list 1 2 3 4))
10
> (sum (append (list 1 2 3 4) (list 2 3 4)))
19
```

Para obtener más información y ejemplos, consulte La función de `apply` en la Guía de raquetas.

Definiciones de funciones

Las funciones en Racket se pueden crear con el formulario `lambda` . La forma toma una lista de argumentos y un cuerpo.

```
(lambda (x y) (* x y))
```

En el ejemplo anterior, la función toma dos argumentos y devuelve el resultado de multiplicarlos.

```
> ((lambda (x y) (* x y)) 4 4)
16
> ((lambda (x y) (* x y)) 3 2)
6
```

Es tedioso volver a escribir la función y su cuerpo cada vez que queremos multiplicar dos números, así que démosle un nombre. Para darle un nombre, usa la forma de `define` . Esto enlazará funciones a un nombre.

```
(define multiply (lambda (x y) (* x y)))
```

Ahora podemos referirnos a nuestra función llamando a `multiply`

```
> (multiply 5 2)
10
```

Dado que es muy común vincular procedimientos a nombres, Racket proporciona una forma abreviada para definir funciones utilizando la forma de definir.

```
(define (multiply x y) (* x y))
```

Para obtener más información y ejemplos, consulte Funciones: lambda en la Guía de Racket.

Lea Funciones en línea: https://riptutorial.com/es/racket/topic/1935/funciones

Capítulo 5: Funciones de orden superior

Examples

Mapa

El mapa aplica una función a cada elemento de una lista:

```
map: (a -> b) (listof a) -> (listof b)

> (map (lambda (x) (* x 2)) (list 1 2 3 4 5)
(list 2 4 6 8 10)

> (map sqrt (list 1 4 9))
(list 1 2 3)

> (map (lambda (x) (if (even? x) "even" "odd")) (list 1 2 3))
(list "odd" "even" "odd")
```

Doblez

Fold Right aplica sucesivamente una función de dos argumentos a cada elemento de una lista de izquierda a derecha, comenzando con un valor base:

```
foldr: (a b -> b) b (listof a) -> b

> (foldr + 0 (list 1 2 3 4))
10

> (foldr string-append "" (list "h" "e" "l" "l" "o"))
"hello"

> (foldr cons empty (list 1 2 3 4))
(list 1 2 3 4)
```

Fold Left realiza la misma acción en la dirección opuesta:

```
foldl: (a b -> b) b (listof a) -> b

> (foldl + 0 (list 1 2 3 4)
10

> (foldl string-append "" (list "h" "e" "l" "l" "o"))
"olleh"

> (foldl cons empty (list 1 2 3 4))
(list 4 3 2 1)
```

Filtrar

`filter` devuelve una lista de cada elemento en la lista dada para la cual el predicado dado

devuelve un valor que no es `#f` .

```
;; Get only even numbers in a list
> (filter even? '(1 2 3 4))
'(2 4)

;; Get all square numbers from 1 to 100
> (filter (lambda (n) (integer? (sqrt n))) (range 1 100))
'(1 4 9 16 25 36 49 64 81)
```

Componer

Te permite componer varias funciones f_0 f_1 … f_n . Devuelve una función que aplicará sucesivamente f_n a sus argumentos, luego f_{n-1} al resultado de f_n y así sucesivamente. Las funciones se aplican de derecha a izquierda, como en la composición de funciones matemáticas: $(f \circ g \circ h)(x) = f(g(h(x)))$.

```
> ((compose sqrt +) 16 9)
5
> ((compose - sqrt) 16)
-4
```

La aridad de cada función debe incluir el número de valores devueltos de la función inmediatamente a su derecha. La función más a la derecha determina la aridad de toda la composición. La función compose1 impone que las funciones devuelvan 1 valor y esperen 1 argumento. Sin embargo, compose1 no restringe la aridad de entrada de la última función, ni la aridad de salida de la primera función.

```
[n input]--> first-function -->[1 output]--> ... last function -->[m output].

((compose + values) 1 2 3 4)
10
> ((compose1 + values) 1 2 3 4)
XX result arity mismatch;
 expected number of values not received
  expected: 1
  received: 4
  values...:
```

Curry

Devuelve una función parcialmente aplicada.

```
> ((curry + 10) 20)
30
```

`curryr` se puede utilizar cuando los argumentos deben insertarse al final. En otras palabras, `(curryr list 1 2)` producirá una función que espera algunos `new-arguments ...` Cuando se le llame, esa nueva función llamará a su vez `(list new-arguments ... 1 2)` .

```
> (((curryr list) 1 2) 3 4)
```

```
'(3 4 1 2)
> ((curryr list 1 2) 3 4)
'(3 4 1 2)
> ((curryr - 30) 40)
10
> (((curryr -) 30 40))
10
```

Lea Funciones de orden superior en línea: https://riptutorial.com/es/racket/topic/4433/funciones-de-orden-superior

Capítulo 6: Garabato

Introducción

Scribble es la herramienta utilizada para crear la documentación de Racket , ¡y también puedes documentar tus propios paquetes! Cuando se publique, su documentación aparecerá en https://docs.racket-lang.org/ , junto con la documentación principal de Racket.

Scribble se implementa como un lenguaje para la plataforma Racket. Por lo tanto, los documentos de Scribble generalmente comienzan con `#lang scribble/manual`

Examples

Párrafos y secciones

```
#lang scribble/manual

@section{Introduction}

First paragraph. Some text, some text, some text,
some text, some text, some text.

@section{More stuff}

@subsection{This is a subsection}

Second paragraph. More text, more text, more text,
more text, more text, more text.
```

Documentando un enlace proporcionado por un paquete

```
#lang scribble/manual

@; Make sure that code highlighting recognises identifiers from my-package:
@require[@for-label[my-package]]

@; Indicate which module is exporting the identifiers documented here.
@defmodule[my-package]

@defproc[(my-procedure [arg1 number?] [arg2 string?]) symbol?]{
  The @racket[my-procedure] function repeats the @racket[arg2] string
  @racket[arg1] times, and transforms the result into a symbol.

  @history[#:added "1.0"
           #:changed "1.1" @elem{Improved performance,
                                 from @tt{O(n²)} to @tt{O(n)}}]
}
```

Como regla de oro, un módulo `.scribble` debe documentar un módulo (algo que podría aparecer a la derecha de un `(require foo/bar)` , es decir, `foo/bar`). Un archivo `.scribble` puede documentar

varios módulos, siempre que cada uno esté documentado en una `@section` separada.

Lea Garabato en línea: https://riptutorial.com/es/racket/topic/9881/garabato

Capítulo 7: Pasos de instalación (Linux)

Examples

Instalación o configuración

Visite https://download.racket-lang.org y elija entre las dos distribuciones disponibles:

- `Racket` es la distribución principal, viene con varios paquetes adicionales como matemática / teoría de números y el IDE DrRacket.
- `Minimal Racket` es mucho más pequeño y viene solo con los paquetes necesarios.

Pasos de instalación para Linux:

La instalación es muy sencilla. Si está acostumbrado a este tipo de cosas, simplemente siga estos cuatro pasos. Más adelante, si lo prefiere, se detalla un tutorial paso a paso más detallado.

1. descárguelo desde https://download.racket-lang.org
2. `chmod +x racket-6.6-x86_64-linux.sh`
3. `./racket-6.6-x86_64-linux.sh`
4. Responda las preguntas y posiblemente actualice su `$PATH` .

Para obtener una guía paso a paso más detallada, consulte a continuación.

Descargando

1. Vaya a https://download.racket-lang.org .
2. Seleccione `Plataforma: Linux i386` si tiene un sistema de 32 bits o `Plataforma: Linux x86_64` .
3. Haga clic en el botón de descarga etiquetado `raqueta-6.9-x86_64-linux.sh (113M)` (la etiqueta puede ser ligeramente diferente según la versión).

Iniciando el instalador

4. Abre una terminal.
5. Si descargaste el archivo a `/home/YOUR_USER_NAME/Downloads` , escribe el siguiente comando:

```
cd / home / YOUR_USER_NAME / Descargas
```

Asegúrese de reemplazar `YOUR_USER_NAME` por su nombre de usuario real y `/Downloads` por la ruta real a la carpeta en la que descargó Racket.

6. Escriba `chmod + x racket-6.6-x86_64-linux.sh` (cambie el número de versión y el `x86_64` para que coincida con el archivo que descargó).
7. Si desea instalar Racket en todo el sistema, escriba `sudo ./racket-6.6-x86_64-linux.sh` (cambie el número de versión y el `x86_64` para que coincida con el archivo que descargó).

De lo contrario, si no es un administrador en la computadora, simplemente escriba `./racket-6.6-x86_64-linux.sh` para instalarlo en su propio directorio de inicio (cambie el número de versión y el `x86_64` para que coincida con el archivo que descargó).

Instalación

El instalador le hará las siguientes preguntas:

8. `Do you want a Unix-style distribution?`

Responda `no` (el predeterminado).

9. `Where do you want to install the "racket-6.6.0.4" directory tree?`

Seleccione `/usr/racket` (escriba `1 Enter ␣`) o `/usr/local/racket` (escriba `2 Enter ␣`) si está instalando Racket en todo el sistema. De lo contrario, para instalarlo en su propio directorio personal (por ejemplo, si no es un administrador), seleccione `~/racket (/home/YOUR_USER_NAME/racket)` (escriba `3 Enter ␣`).

10. `If you want to install new system links within the "bin", "man" and "share/applications" subdirectories…`

Si está realizando una instalación en todo el sistema, es una buena idea escribir `/ usr / local` o `/ usr` aquí (para saber cuál, verifique cuál está presente en su `PATH` , escribiendo `echo $ PATH` en otra ventana de terminal). Si lo está instalando en su propio directorio de inicio, deje la respuesta en `blanco` y solo presione `Enter ␣` .

Empezando DrRacket

Dependiendo de su respuesta a los pasos 9 y 10, debe escribir uno de los siguientes comandos en un terminal para iniciar DrRacket:

- `Drracket` (si el paso 10 fue exitoso)
- `/ usr / raqueta / bin / drracket`
- `/ usr / local / raqueta / bin / drracket`
- `/ home / YOUR_USER_NAME / racket / bin / drracket` (reemplaza `YOUR_USER_NAME` por tu nombre de usuario real, o simplemente escribe `~ / racket / bin / drracket`)

Para evitar escribir un comando tan largo cada vez, puede agregar el siguiente comando al archivo `~/.bashrc` , donde `/path/to/the/containing/folder/` debe ser uno de `/usr/racket/bin/` , `/usr/local/racket/bin/` or `/home/YOUR_USER_NAME/racket/bin/` :

```
export PATH="/path/to/the/containing/folder/:$PATH"
```

Ejecutando tu primer programa

Para ejecutar un programa, abra DrRacket como se explicó anteriormente, ingrese el programa que comienza con `#lang racket` y haga clic en el botón `Run` cerca de la esquina superior derecha.

Aquí hay un primer programa de ejemplo:

```
#lang racket
(displayln "Hello Racket!")
```

Lea Pasos de instalación (Linux) en línea: https://riptutorial.com/es/racket/topic/9870/pasos-de-instalacion--linux-

Capítulo 8: Pasos de instalación (macOS)

Examples

Instalación o configuración

Visite https://download.racket-lang.org y elija entre las dos distribuciones disponibles:

- `Racket` es la distribución principal, viene con varios paquetes adicionales como matemática / teoría de números y el IDE DrRacket.
- `Minimal Racket` es mucho más pequeño y viene solo con los paquetes necesarios.

Pasos de instalación para macOS:

La instalación es muy sencilla. Si está acostumbrado a este tipo de cosas, solo vaya a https://download.racket-lang.org , luego descargue e instale el archivo `.dmg` . Más adelante, si lo prefiere, se detalla un tutorial paso a paso más detallado.

Descargando

1. Vaya a https://download.racket-lang.org .
2. Seleccione `Plataforma: Mac OS (Intel de 32 bits)` si tiene un sistema de 32 bits, o `Plataforma: Mac OS (Intel de 64 bits)` si tiene un sistema de 64 bits.
3. Haga clic en el botón de descarga etiquetado `racket-6.9-x86_64-macosx.dmg (106M)` (la etiqueta puede ser ligeramente diferente según la versión).

Iniciando el instalador

4. `FIXME: If you have macOS, please fill in this section`

Configuración de herramientas de línea de comandos

En **Mac OS X** , puede visitar el menú de Ayuda de DrRacket y usar "Configurar línea de comandos para raquetas ..." para configurar las herramientas de raquetas para el uso de la línea de comandos. En **Windows** , deberá agregar la carpeta de instalación de Racket a su variable `PATH` .

Ejecutando tu primer programa

Para ejecutar un programa, abra DrRacket, ingrese el programa que comienza con `#lang racket` y haga clic en el botón `Run` cerca de la esquina superior derecha. Aquí hay un primer programa de ejemplo:

```
#lang racket
(displayln "Hello Racket!")
```

Lea Pasos de instalación (macOS) en línea: https://riptutorial.com/es/racket/topic/9872/pasos-de-instalacion--macos-

Capítulo 9: Pasos de instalación (Windows)

Examples

Instalación o configuración

Visite https://download.racket-lang.org y elija entre las dos distribuciones disponibles:

- `Racket` es la distribución principal, viene con varios paquetes adicionales como matemática / teoría de números y el IDE DrRacket.
- `Minimal Racket` es mucho más pequeño y viene solo con los paquetes necesarios.

Para ejecutar un programa, abra DrRacket, ingrese el programa que comienza con `#lang racket` y haga clic en el botón `Run` cerca de la esquina superior derecha.

Pasos de instalación para Windows:

La instalación es muy sencilla. Si está acostumbrado a este tipo de cosas, solo vaya a https://download.racket-lang.org , luego descargue y ejecute el instalador. Más adelante, si lo prefiere, se detalla un tutorial paso a paso más detallado.

Descargando

1. Vaya a https://download.racket-lang.org .
2. Seleccione `Plataforma: Windows (x86, 32 bits)` si tiene un sistema de 32 bits, o `Plataforma: Windows (x64, 64 bits)` si tiene un sistema de 64 bits. En caso de duda, elija la versión de 32 bits.
3. Haga clic en el botón de descarga etiquetado `racket-6.9-i386-win32.exe (73M)` (la etiqueta puede ser ligeramente diferente según la versión).

Iniciando el instalador

4. Abra el directorio donde se descargó el archivo y haga doble clic en el archivo de `racket-….exe` .
5. Siga las instrucciones del instalador.

Configuración de herramientas de línea de comandos

Para configurar las herramientas de la línea de comandos, abra DrRacket, haga clic en el menú Ayuda y haga clic en "Configurar línea de comandos para raqueta". Esto instalará los comandos de `racket` y `raco` . (En Windows, el comando `racket` es `Racket.exe`).

Ejecutando tu primer programa

Para ejecutar un programa, abra DrRacket, ingrese el programa que comienza con `#lang racket` y haga clic en el botón `Run` cerca de la esquina superior derecha. Aquí hay un primer programa de ejemplo:

```
#lang racket
(displayln "Hello Racket!")
```

Lea Pasos de instalación (Windows) en línea: https://riptutorial.com/es/racket/topic/9871/pasos-de-instalacion--windows-

Capítulo 10: Recursion

Examples

Utilizando define

```
#lang racket
(define (sum-of-list l)
  (if (null? l)
      0
      (+ (car l)
         (sum-of-list (cdr l)))))
(sum-of-list '(1 2 3 4 5)) ;; => 15
```

Usando let-rec

```
#lang racket
(letrec ([sum-of-list (λ (l)
                        (if (null? l)
                            0
                            (+ (car l) (sum-of-list (cdr l)))))])
  (sum-of-list '(1 2 3 4 5)))
;; => 15
```

Es posible escribir funciones recursivas entre sí con `letrec` :

```
#lang racket
(letrec ([even? (λ (n) (if (= n 0) #t (odd?  (sub1 n))))]
         [odd?  (λ (n) (if (= n 0) #f (even? (sub1 n))))])
  (list (even? 3)
        (odd? 5)))
;; => '(#f #t)
```

Usando una let con nombre

Una forma normal de `let` une cada valor a su identificador correspondiente, antes de ejecutar el cuerpo. Con una " `let` nombre", el cuerpo puede volver a ejecutarse recursivamente, pasando un nuevo valor para cada identificador.

```
#lang racket
(let sum-of-list ([l '(1 2 3)])
  (if (null? l)
      0
      (+ (car l) (sum-of-list (cdr l)))))
;; => 15
```

Es común usar `rec` como el nombre para let, que da:

```
#lang racket
```

```
(let rec ([l '(1 2 3 4 5)])
  (if (null? l)
      0
      (+ (car l) (rec (cdr l)))))
;; => 15
```

Usando rec

```
#lang racket
(require mzlib/etc)
((rec sum-of-list
   (λ (l)
     (if (null? l)
         0
         (+ (car l) (sum-of-list (cdr l))))))
 '(1 2 3 4 5))
;; => 15

;; Outside of the rec form, sum-of-list gives an error:
;; sum-of-list: undefined;
;;  cannot reference an identifier before its definition
```

Esto es similar a `define` , pero el identificador de `sum-of-list` no es visible fuera del formulario de `rec` .

Para evitar el uso de un `λ` explícito, es posible reemplazar la `sum-of-list` con `(sum-of-list args ...)` :

```
#lang racket
(require mzlib/etc)
((rec (sum-of-list l)
   (if (null? l)
       0
       (+ (car l) (sum-of-list (cdr l)))))
 '(1 2 3 4 5))
;; => 15
```

Usando funciones de orden superior en lugar de recursión

Es una práctica común utilizar funciones de orden superior en lugar de recursión, si hay una función de orden superior que expresa el patrón de recursión correcto. En nuestro caso, la `sum-of-numbers` se puede definir usando `foldl` :

```
#lang racket
(define (sum-of-numbers l)
  (foldl + 0 l))
(sum-of-numbers '(1 2 3 4 5)) ;; => 15
```

Es posible llamar a `foldl` directamente en la lista:

```
#lang racket
(foldl + 0 '(1 2 3 4 5)) ;; => 15
```

Lea Recursion en línea: https://riptutorial.com/es/racket/topic/6465/recursion

Creditos

S. No	Capítulos	Contributors
1	Empezando con la raqueta	Alex Knauth, belph, bitrauser, Community, eyqs, Georges Dupéron, Guillaume Marceau, John Gallagher, Kronos, Leif Andersen, mnoronha, soegaard
2	Cierres	ben rudgers
3	Comentarios	eyqs, Leif Andersen, pvdsp, soegaard
4	Funciones	Alex Knauth, eyqs, Jason Yeo
5	Funciones de orden superior	4444, Anjali Pal, Brendan, Georges Dupéron, Majora320, mathk
6	Garabato	Georges Dupéron
7	Pasos de instalación (Linux)	Alex Knauth, bitrauser, eyqs, Georges Dupéron
8	Pasos de instalación (macOS)	Georges Dupéron
9	Pasos de instalación (Windows)	Georges Dupéron
10	Recursion	Georges Dupéron