

## **Manual Técnico**



## **Título del Trabajo**

Manual Técnico

## **Asignatura**

Teoría de Lenguajes de Programación

## **Profesor**

M. en C. Luis R. Basto Diaz

## **Integrantes:**

- Víctor Manuel Lara Lopez (152126359)
- Antonio Alfonso Cetzel Patron (14000516)
- Miguel Angel Quiñones Ramirez (17115665)

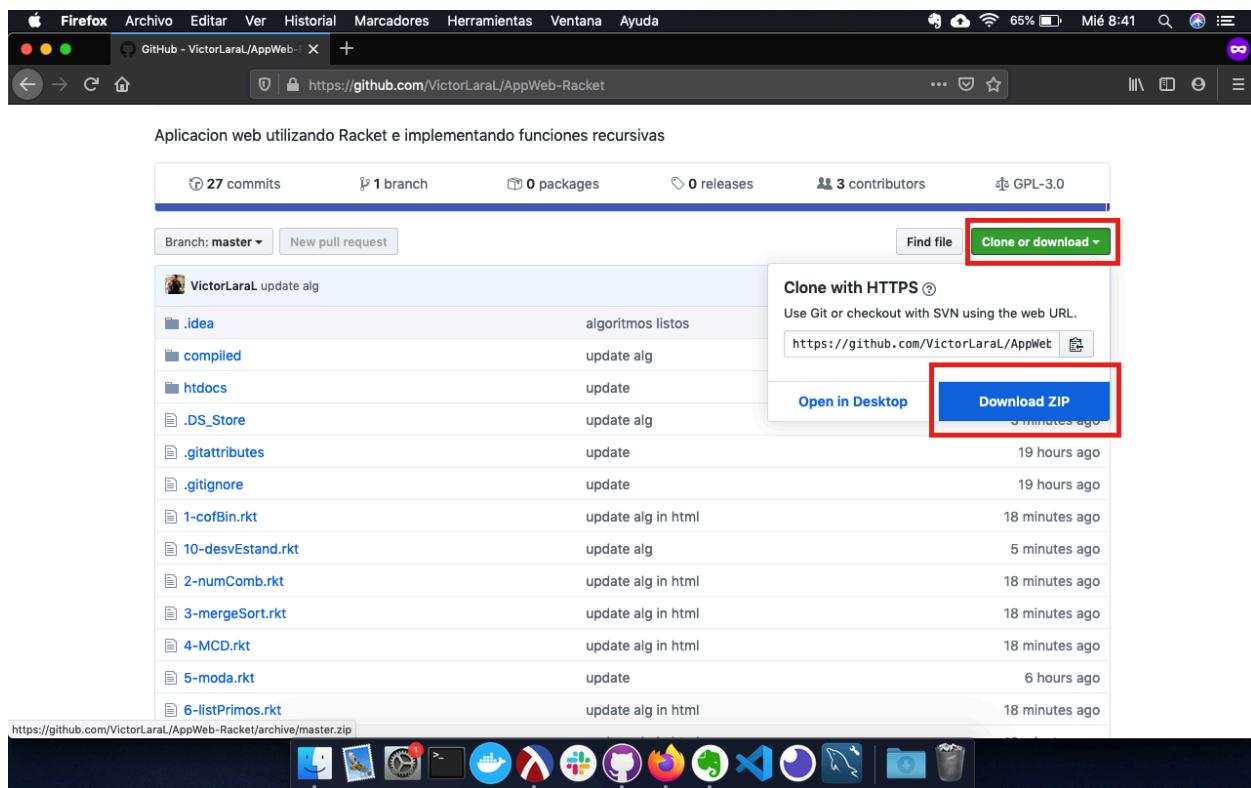
**entrega:** 27/11/19

---

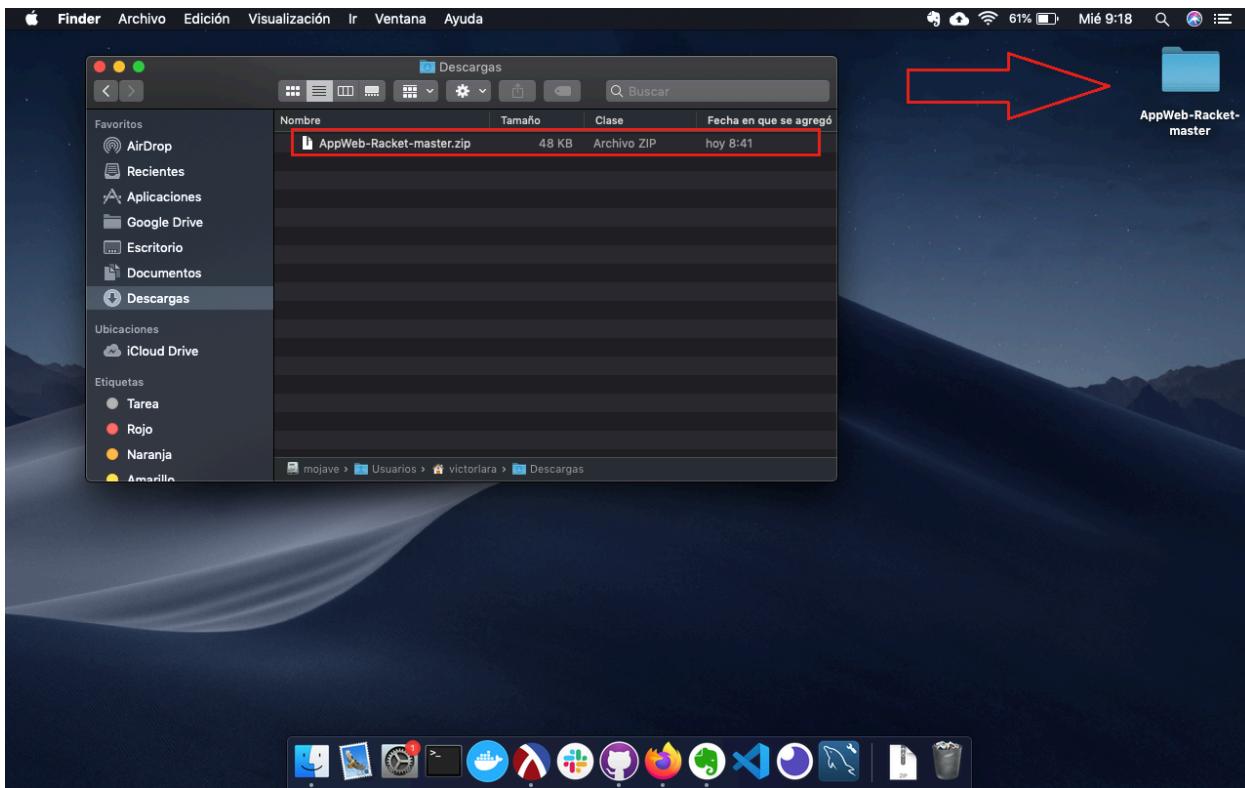
# Descarga e instalación

**Nota:** Antes de la descarga del paquete es necesario tener instalado Racket y DrRacket en el equipo.

Ir a la ruta: <https://github.com/VictorLaraL/AppWeb-Racket> y dentro de la pagina seleccionar descargar ZIP:

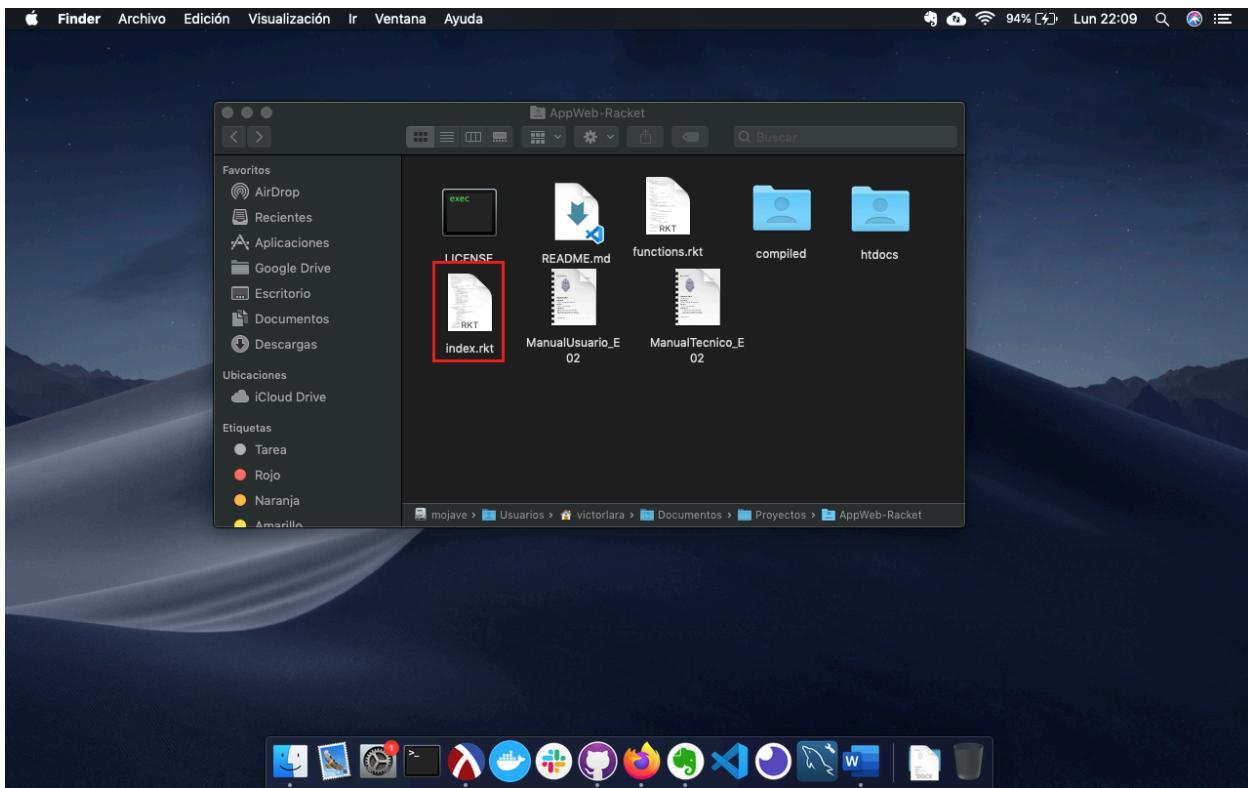


Una vez descargado el paquete lo extraemos en la carpeta de nuestra preferencia:



## Ejecución de los ficheros

Ahora nos situamos dentro de la carpeta donde se tienen los archivos .rkt cada uno es un algoritmo diferente, para acceder a ellos solo tenemos que darle doble click al algoritmo que queríamos utilizar:



Al hacer doble click sobre algún archivo .rkt el equipo nos abrirá el IDE DrRacket, una vez situados en el IDE hacemos click en “ejecutar”:

Al ejecutar el archivo lanzara el servidor con un archivo HTML en una dirección local, si el IDE no despliega nuestro navegador web, podemos hacerlo manual copiando la dirección que aparece en DrRacket y pegándola en nuestro navegador de preferencia:

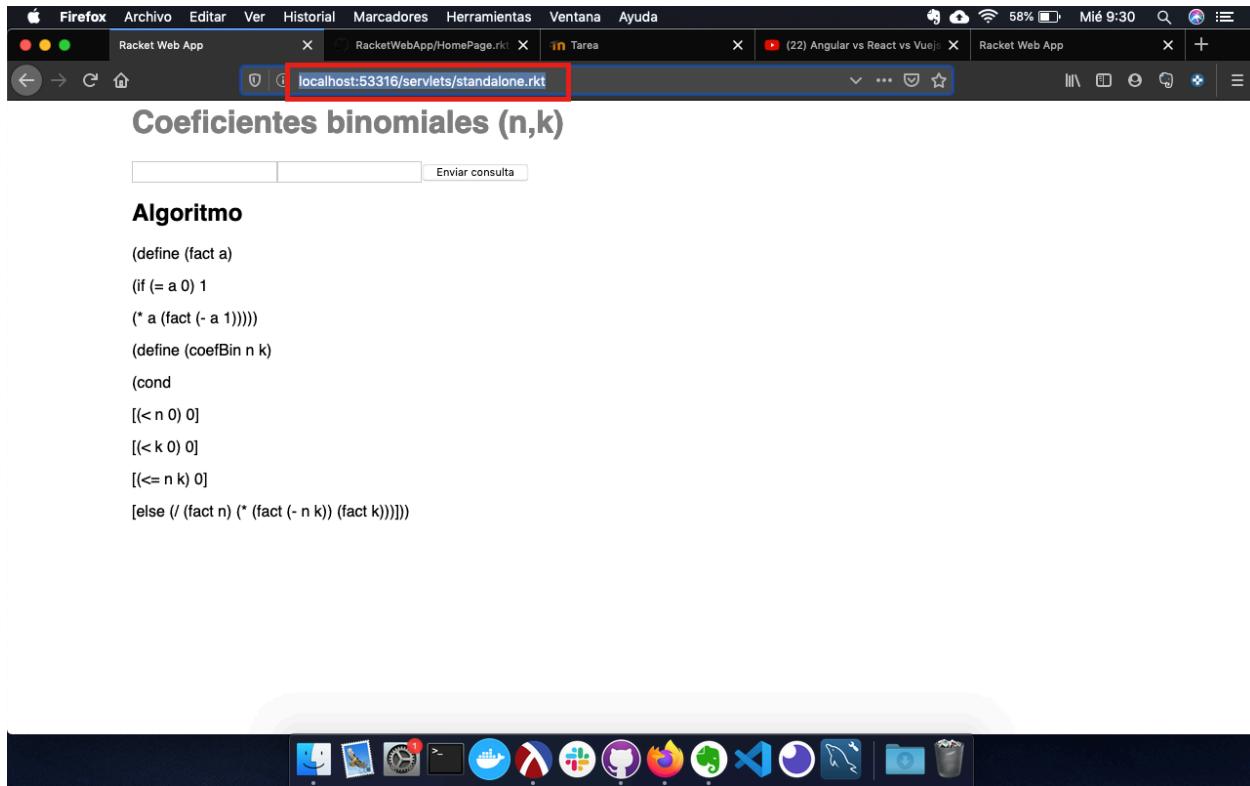
A screenshot of the DrRacket IDE on a Mac OS X desktop. The window title is "DrRacket" and the file name is "1-cofBin.rkt". The menu bar includes Archivo, Edición, Muestra, Lenguaje, Racket, Insert, Scripts, Ventana, Ayuda. The top right shows battery at 59%, network, and the date Mié 9:24. The code editor contains Racket code for a web server. The status bar at the bottom shows "Determine language from source ▾" and system icons.

```
#lang web-server/insta
(require "funciones.rkt")
;; Estructura del post
(struct post (n k) #:transparent)
;; Lista de resultados
(define RESULTS
  (list ))
;; Arrancamos la pagina y hacemos una peticion para corroborar si existen datos para ingresar
(define (start request)
  (define a-calculate
    (cond [(can-parse-post? (request-bindings request))
           (cons (parse-post (request-bindings request)) RESULTS)]
          [else RESULTS]))
  (render-page a-calculate request))
;; Render of the page
(define (render-page results request)
  (response/xexpr
    '(html (head (title "Racket Web App")
                 (link ((rel "stylesheet")
                        (href "/test-static.css"))
                  (type "text/css"))))
        (body (h1 "Coeficientes binomiales (n,k)"),
              (render-results results)
              (form
                (input ((name "calculate-n")))
                (input ((name "calculate-k")))
                (input ((type "submit"))))
              )
              (h2 "Algoritmo")
              (p "(define (fact a)")
              (p "  (if (= a 0) 1")
              (p "  (* a (fact (- a 1))))")
              (p "  ")
              (p "(define (coefBin n k)"))
              (p "  (cond")
              (p "    ([(< n 0) 0]")
              (p "    ([(< k 0) 0]")
              (p "    ([(<= n k) 0]))"

```

A screenshot of the DrRacket IDE on a Mac OS X desktop, similar to the first one but with a context menu open over the status bar. The menu options include Undo, Redo, Copy, Cut, Paste, Clear, and Select All. The status bar also displays "Busca en el Módulo de Ayuda "http://localhost:53316/servlets/standalone.rkt"".

```
Determine language from source ▾
```



## Descripción de los Algoritmos

La manera de realizar el proyecto fue a través de una serie de algoritmos almacenados en aplicaciones independientes que despliegan un servidor de manera local y a través de funciones que tomamos de la biblioteca "#lang web-server/insta" renderizamos utilizando HTML y CSS la UI para utilizar los algoritmos, la implantación y distribución del código es la siguiente:

Para todo el sistema fueron 3 archivos clave los que utilizamos para crear toda la aplicación (o cada una de ellas):

fichero de funciones ("funciones.rkt"), fichero para algoritmos de 1 o 2 datos (Ej. "1-cofBin"), fichero para algoritmos que necesitaban una lista (Ej. "7-maxList"). Dichos ficheros los describimos a continuación:

### funciones.rkt

The screenshot shows the DrRacket IDE interface. The top menu bar includes 'Revisar sintaxis' (Check Syntax), 'Debug' (with icons for step, run, and stop), 'Macro Stepper' (with icons for step, run, and stop), 'Ejecutar' (Run), and 'Interrumpir' (Interrupt). The code editor displays a file named '1-cofBin.rkt'. The code is written in Racket and defines several functions:

```
1 #lang racket
2
3 ;; Funcion para pasar una lista de numeros a un string
4 (define (slist->string slst)
5   (cond ((empty? slst) "")
6         ((empty? (rest slst)) (number->string (first slst)))
7         (else (string-append (number->string (first slst))
8                           " "
9                           (slist->string (rest slst)))))))
10
11 ;; 1) Coeficientes Binomiales
12 (define (fact a)); Funcion para calcular factorial (regresa un numero)
13   (if (= a 0) 1
14       (* a (fact (- a 1)))))
15
16 (define (coefBin n k)); Funcion para los coeficientes binomiales (regresa un numero)
17   (cond
18     [(< n 0) 0] ; Condiciones que debe cumplir
19     [(< k 0) 0]
20     [(<= n k) 1]
21     [else (/ (* (fact n) (* (fact (- n k)) (fact k))))])
22
23 ;; 2) Numero Combinatorio
24 (define (combinaciones n k)); Funcion para las combinaciones entre dos numeros (regresa un numero)
25   (cond [= (k n) 1]
26         [= (k 0) 1]
27         [(> k n) 0]
28         [else (+ (combinaciones (- n 1) (- k 1)) (combinaciones (- n 1) k))]))
29
30 ;; ideamos dos formas de resolver el algoritmo
31 ;; 3) MergeSort
32 (define (mergeListas l1 l2)
33   (cond
34     [(null? l1) l2];; Condiciones que debe cumplir
35     [(null? l2) l1]
36     [(< (car l1) (car l2)) (cons (car l1) (mergeListas (cdr l1) l2))]
37     [else (cons (car l2) (mergeListas l1 (cdr l2)))])))
38
39 (define (mergesort lista); Funcion que ordena una lista dada (regresa la lista ordenada)
40   (cond [(or (null? lista) (null? (cdr lista))) lista];; Condiciones que debe cumplir
41         [(null? (cdr (cdr lista))) (mergeListas (list (car lista)) (cdr lista))]
42         [else (let ([x (floor (/ (length lista) 2))])
43                (mergeListas (mergesort (take lista x)) (mergesort (drop lista x)))))])
44
45 ;; 3) MergeSort
46 ;/define (mergesort l1 l2)
```

Determine language from source ▾

158:119 541.18 MB

Lo primero que podemos denotar en este fichero es que debe ser un archivo **#lang racket** puesto que para ser llamado necesita ser de este tipo.

después nos topamos con una función muy utilizada en los algoritmos para convertir un lista a un string y de esa manera imprimir de forma horizontal en los HTML.

```

1 #lang racket
122 (define (maximoL l); Calculo del elemento maximo en una lista dada (regresa un numero)
123   (cond
124     [(null? l) 0]
125     [= (length l) 1] (car l)]
126     [else (mayor (car l) (maximoL (cdr l))))]))
127
128 ;; 8) Minimo en una lista
129 (define (menor a b)
130   (if (< a b) a b))
131
132 (define (minimoL l); Calculo del elemento minimo en una lista dada (regresa un numero)
133   (cond
134     [(null? l) 0]
135     [= (length l) 1] (car l)]
136     [else (menor (car l) (minimoL (cdr l))))]))
137
138 ;; 9) Suma de n primeros impares
139 (define (impares a); calculo de la suma de los primeros n numeros impares (regresa un numero)
140   (if (= a 0) 0
141       (+ (- (* a 2) 1) (impares (- a 1))))))
142
143 ;; 10) Desviacion estandar
144 (define (media lista)
145   (/ (apply + lista) (length lista)))
146
147 (define (diferenciacuadrada lista)
148   (map (lambda (x) (* x x))
149         (map (lambda (x) (- x (media lista))) lista)))
150
151 (define (sumacuadrados lista)
152   (apply + (diferenciacuadrada lista)))
153
154 (define (varianza lista)
155   (/ (sumacuadrados lista) (length lista)))
156
157 (define (desviacionestandar lista); Calculo de la desviacion estandar de una lista de numeros dada (regresa un numero)
158   (cond
159     [(null? lista) 0]
160     [else (sqrt (varianza lista))]))
161
162 ;; Enviar datos
163 (provide (all-defined-out))
164

```

Por ultimo, despues de todas las funciones tenemos la ultima función que es la que indica que todo lo del fichero puede ser utilizado por un fichero externo.

## 1-cofBin.rkt

Este es un ejemplo de algoritmo en el que se ingresa solo dos datos y como resultado nos muestra un dato. sirve de referencia puesto que la estructura es idéntica en los ademas algoritmos que tambien ingresan y regresan la misma cantidad de datos

```

1 #lang web-server/insta
2
3 (require "funciones.rkt"); Importamos el fichero donde almacenamos nuestras funciones
4
5 ;; Estructura del post
6 (struct post (n k) #:transparent)
7
8 ;; Lista de resultados
9 (define RESULTS
10   (list ))
11
12 ; Arrancamos la pagina y hacemos una petición para corroborar si existen datos para ingresar
13 (define (start request)
14   (define a-calculate
15     (cond [(can-parse-post? (request-bindings request));; Condicional que corrobora si existen datos en el request para agregar a la lista
16       (cons (parse-post (request-bindings request)) RESULTS))
17     [else RESULTS]))
18   (render-page a-calculate request));; Enviamos la lista y el request
19
20 ;; Renderizamos la pagina
21 (define (render-page results request)
22   (response/xexpr
23     `(~(html (~(head (~(title "Racket Web App"))
24           (~(link ((rel "stylesheet"));; Estilos para la página (CSS)
25             (href "/test-static.css"))
26           (~(type "text/css")))))
27       (~(body (~(h1 "Coeficientes binomiales (n,k)"), (render-results results));; Llamamos a la función que renderiza los resultados
28         (~(form)
29           (~(input ((name "calculate-n"))); Datos de entrada
30           (~(input ((name "calculate-k"))))
31           (~(input ((type "submit"))))
32           )
33           (~(h2 "Algoritmo")); Algoritmo en la página
34           (~(p "(define (fact a)")
35             (~(p "(if (= a 0) 1")
36               (~(p "(* a (fact (- a 1))))"))
37             (~(p ""))
38             (~(p "(define (coefBin n k)"))
39               (~(p "(cond"))
40                 (~(p "[(< n 0) 0]"))
41                 (~(p "[(< k 0) 0]"))
42                 (~(p "[(<= n k) 0]"))
43                 (~(p "[else (/ (* (fact n) (* (fact (- n k)) (fact k))))]))"))
44               )
45           (~(static-files-path "htdocs")); Importamos la carpeta de los estilos
46
47   ;; Verificación de datos de entrada
48   (~(define (can-parse-post? hindrance)

```

Determine language from source ▾      60:33      602.43 MB □      🌐

Dos partes importantes a resaltar es cuando llama al fichero de funciones en la parte superior y cuando lanza la aplicación para después renderizar la página, al lanzar la aplicación hace la comprobación de los requests para saber si hay algún dato de entrada, en caso de haberlo lo pide en otra función que lo castea a un numero y lo agrega a la lista de resultados.

```

1 | #lang web-server/insta
24 |     (link ((rel "stylesheet")); Estilos para la pagina (CSS)
25 |         (href "/test-static.css"))
26 |         (type "text/css"))
27 |     (body (h1 "Coeficientes binomiales (n,k)", (render-results results)); Llamamos a la funcion que renderiza los resultados
28 |     (form
29 |         (input ((name "calculate-n"))); Datos de entrada
30 |         (input ((name "calculate-k")))
31 |         (input ((type "submit"))))
32 |     )
33 |     (h2 "Algoritmo"); Algoritmo en la pagina
34 |     (p "(define (fact a)")
35 |     (p "  (if (= a 0) 1")
36 |     (p "  (* a (fact (- a 1)))))")
37 |     (p "  ")
38 |     (p "  (define (coefBin n k)")
39 |     (p "    (cond"
40 |     (p "      [(< n 0) 0]")
41 |     (p "      [(< k 0) 0]")
42 |     (p "      [(=< n k) 1]")
43 |     (p "      [else (/ (* (fact n) (* (fact (- n k)) (fact k))))]))"))
44 |
45 |     (static-files-path "htdocs"); Importamos la carpeta de los estilos
46 |
47 |     ;; Verificacion de datos de entrada
48 |     (define (can-parse-post? bindings)
49 |         (and (exists-binding? 'calculate-n bindings)
50 |             (exists-binding? 'calculate-k bindings)))
51 |
52 |     ;; Obtencion y casteo de los datos (los regresamos como un post de numeros n,k)
53 |     (define (parse-post bindings)
54 |         (post (string->number
55 |             (extract-binding/single 'calculate-n bindings))
56 |             (string->number
57 |                 (extract-binding/single 'calculate-k bindings))))
58 |
59 |
60 |     ;; Renderizado de los resultados
61 |     (define (render-results algorithms)
62 |         `(~(div ((class "results"))
63 |             ,(map render-result algorithms)))
64 |
65 |     (define (render-result result)
66 |         `(~(div ((class "result"))
67 |             ,(number->string
68 |                 (coefBin (post-n result) (post-k result))))))
69 |
70 |

```

Por ultimo al final del código vemos la llamada al algoritmo que hará los cálculos y el resultado que regrese lo casteara a string para poder renderizarlo puesto que HTTP solo recibe y envía strings.

## 7-maxList.rkt

En los ejemplos de las aplicaciones que utilizan listas encontramos un modelo diferente de renderizado de la aplicación, puesto que en esta ocasión utilizamos una función que nos permite permanecer en la pagina sin volver a lanzarla con cada request y con esto evitar que se reinicen los datos (perder la lista de datos almacenados)

7-maxList.rkt ▾ (define ...) ▾

1: funciones.rkt

Revisar sintaxis Debug Macro Stepper Ejecutar Interrumpir

2: 7-maxList.rkt

```
1 #lang web-server/insta
2
3 (require "funciones.rkt")
4
5 ;; Lista de datos ingresados
6 (define ELEMENTS
7   (list ))
8
9 ;; Arrancamos la pagina, esta vez sin corroborar los datos
10 (define (start request)
11   (render-page ELEMENTS request))
12
13 ;; Renderizado de la pagina
14 (define (render-page listElements request)
15   (define (response-generator embed/url); esta funcion nos va a permitir no reiniciar la pagina y solo enviar y recibir datos
16     (response/xexpr
17       `(html (head (title "Racket Web App"))
18             (link ((rel "stylesheet")); Estilos para la pagina
19                   (href "/test-static.css"))
20             (type "text/css")))
21       (body (h1 "Max en lista"), (render-result listElements)); Imprimimos el resultado
22       (form ((action , (embed/url insert-post-handler))); Formulario para ingresar los datos
23         (input ((name "elemento")))
24         (input ((type "submit"))); Boton para ingresar los datos (a travez de un request)
25       ),
26       (render-list listElements); Impresion de la lista generada
27       (h2 "Algoritmo"); Algoritmo renderizado en la pagina
28       (p "(define (mayor a b)")
29         (p "(if (> a b) a b)")
30
31         (p "(define (maximoL l)")
32           (p "(cond")
33             (p "[(null? l) 0]")
34             (p "[(= (length l) 1) (car l)]")
35             (p "[else (mayor (car l) (maximoL (cdr l))))]))"))
36
37   (define (insert-post-handler request)
38     (render-page
39       (cons (parse-post (request-bindings request))
40             listElements)
41     request))
42   (send/suspend/dispatch response-generator)); Utilizamos este apartado para complementar los embed/url
43
44 (static-files-path "htdocs")
45
46 ;; Extraccion del dato y casteo a numero
```

Determine language from source ▾

21:71 607.16 MB

7-maxList.rkt ▾ (define ...) ▾

1: funciones.rkt

Revisar sintaxis Debug Macro Stepper Ejecutar Interrumpir

2: 7-maxList.rkt

```
1 #lang web-server/insta
2
3 (require "funciones.rkt")
4
5 ;; Lista de datos ingresados
6 (define ELEMENTS
7   (list ))
8
9 ;; Arrancamos la pagina, esta vez sin corroborar los datos
10 (define (start request)
11   (render-page ELEMENTS request))
12
13 ;; Renderizado de la pagina
14 (define (render-page listElements request)
15   (define (response-generator embed/url); esta funcion nos va a permitir no reiniciar la pagina y solo enviar y recibir datos
16     (response/xexpr
17       `(html (head (title "Racket Web App"))
18             (link ((rel "stylesheet")); Estilos para la pagina
19                   (href "/test-static.css"))
20             (type "text/css")))
21       (body (h1 "Max en lista"), (render-result listElements)); Imprimimos el resultado
22       (form ((action , (embed/url insert-post-handler))); Formulario para ingresar los datos
23         (input ((name "elemento")))
24         (input ((type "submit"))); Boton para ingresar los datos (a travez de un request)
25       ),
26       (render-list listElements); Impresion de la lista generada
27       (h2 "Algoritmo"); Algoritmo renderizado en la pagina
28       (p "(define (mayor a b)")
29         (p "(if (> a b) a b)")
30
31         (p "(define (maximoL l)")
32           (p "(cond")
33             (p "[(null? l) 0]")
34             (p "[(= (length l) 1) (car l)]")
35             (p "[else (mayor (car l) (maximoL (cdr l))))]))"))
36
37   (define (insert-post-handler request)
38     (render-page
39       (cons (parse-post (request-bindings request))
40             listElements)
41     request))
42   (send/suspend/dispatch response-generator)); Utilizamos este apartado para complementar los embed/url
43
44 (static-files-path "htdocs")
45
46 ;; Extraccion del dato y casteo a numero
47 (define (parse-post bindings)
48   (string->number
49     (extract-binding/single 'elemento bindings)))
50
51 ;; Impresion de la lista de elementos
52 (define (render-list listElements)
53   `(div ((class "list"))
54     ,(slist->string listElements)))
55
56
57 ;; Impresion de el resultado del algoritmo
58 (define (render-result listElements)
59   `(div ((class "result"))
60     ,(number->string
61       (maximoL listElements))))
```

Al final cómo vemos llamamos a la función que hará un calculo con la lista y nos regresara un dato que castearemos para poder renderizarlo.

## Funciones

### 1 Coeficientes Binomiales

**Nombre:** (coefBin n k)

**Input:**  $n \geq 0$   
 $0 \leq k \leq n$

**Output:** Combinaciones binomiales k de n

**Código:**

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

(define (coefBin n k)
  (cond
    [(< n 0) 0]
    [(< k 0) 0]
    [(<= n k) 0]
    [else (/ (fact n) (* (fact (- n k)) (fact k))))])
```

**Descripción:**

Se calculan los coeficientes binomiales usando la fórmula  $B(n, k) = n! / ((n - k)! * k!)$

### 2 Número combinatorio

**Nombre:** (combinaciones n k)

**Input:**  $k \geq 0$   
 $n \geq k$

**Output:** Combinaciones k de n

**Código:**

```
(define (combinaciones n k)
  (cond [(= k n) 1]
        [(= k 0) 1]
        [(> k n) 0]
        [else (+ (combinaciones (- n 1) (- k 1)) (combinaciones (- n 1) k))))])
```

### Descripción:

Se evalua el número combinatorio utilizando la definición recursiva  $C(n, 0) = 1$ ,  
 $C(n, n) = 1$ ,  $C(n, k) = C(n - 1, k - 1) + C(n - 1, k)$

## 3 Mergesort

**Nombre:** (mergesort lista)

**Input:** Lista *lista*  
                  La lista *lista*, después de ordenarse mediante el algoritmo mergesort

**Output:** Combinaciones k de n

### Código:

```
(define (mergelistas l1 l2)
  (cond
    [(null? l1) l2]
    [(null? l2) l1]
    [(< (car l1) (car l2)) (cons (car l1) (mergelistas (cdr l1) l2))]
    [else (cons (car l2) (mergelistas l1 (cdr l2))))])

(define (mergesort lista)
  (cond [(or (null? lista) (null? (cdr lista))) lista]
        [(null? (cdr (cdr lista))) (mergelistas (list (car lista)) (cdr lista))]
        [else (let ([x (floor (/ (length lista) 2))])
                (mergelistas (mergesort (take lista x)) (mergesort (drop lista x))))]))
```

### Descripción:

Se sigue el procedimiento:

```
procedure merge( $L_1, L_2$ : sorted lists)
 $L :=$  empty list
while  $L_1$  and  $L_2$  are both nonempty
  remove smaller of first elements of  $L_1$  and  $L_2$  from its list; put it at the right end of  $L$ 
  if this removal makes one list empty then remove all elements from the other list and
    append them to  $L$ 
return  $L$  { $L$  is the merged list with elements in increasing order}
```

```
procedure mergesort( $L = a_1, \dots, a_n$ )
if  $n > 1$  then
   $m := \lfloor n/2 \rfloor$ 
   $L_1 := a_1, a_2, \dots, a_m$ 
   $L_2 := a_{m+1}, a_{m+2}, \dots, a_n$ 
   $L := merge(mergesort(L_1), mergesort(L_2))$ 
{ $L$  is now sorted into elements in nondecreasing order}
```

Cortando las listas y uniendo los pedazos debidamente. Si la lista es vacía o tiene

un elemento, no se realiza ningún procedimiento.

## 4 Máximo Común Divisor

**Nombre:** (mcd a b)

**Input:** Enteros  $a$  y  $b$

**Output:** MCD de  $a$  y  $b$

**Código:**

```
(define (mcd a b)
  (cond [(< a 0) (let ([a (* a -1)]
                         [b (* b -1)])
              (mcd a b))]
        [(< b a) (mcd b (- a b))]
        [(< a b) (mcd a (- b a))]
        [else a]))
```

**Descripción:**

Se calcula el MCD de  $a$  y  $b$ , donde  $a < b$ , recursivamente usando  $\text{MCD}(a,b) = \text{MCD}(a, b-a)$ . Además, se incorpora el caso  $b < a$ , donde  $\text{MCD}(a,b) = \text{MCD}(b, a-b)$ . Para enteros negativos, se hacen las conversiones necesarias, tomando en cuenta que entre los divisores de los números negativos se encuentran los de sus valores positivos.

## 5 Moda de una lista de enteros

**Nombre:** (moda lista)

**Input:** Lista  $lista$

**Output:** Los valores más frecuentes dentro de  $lista$  y sus frecuencias

**Código:**

```
(define (calcularmoda lista hashf)
  (hash-update! hashf
    (car lista)
    (lambda (frec) (add1 frec))
    0))
```

```

(cond [(not (null? (cdr lista))) (calcularmoda (cdr lista) hashf)]
      [else (for/fold ([mod null]
                      [mfrec 0])
                     [(val cant) (in-hash hashf)])
             (cond [(> cant mfrec) (values (list val) cant)]
                  [(= cant mfrec) (values (cons val mod) mfrec)]
                  [else (values mod mfrec)])))))

(define (moda lista)
  (let ([frecuencias (make-hash)])
    (let-values([(x y) (calcularmoda lista frecuencias)])
      (list x y)))

```

### Descripción:

Se crea una tabla hash y se añade, recursivamente, cada valor dentro de la lista, modificando la frecuencia en caso de valores repetidos. Posteriormente se evalúa para encontrar, dentro de la tabla hash, el valor o valores con mayor frecuencia.

## 6 Números primos en un rango

**Nombre:** (listaprimos a b)

**Input:** Enteros  $a \geq 0$ ,  $b > a$

**Output:** Lista de números primos entre  $a$  y  $b$

### Código:

```

(define (esprimo n)
  (cond [(= (modulo (+ n 1) 4) 0) (let loop2 ([n n] [j (+ (/ (+ n 1) 2) 1)])
                                             (cond [(= j 1) true]
                                                   [ (= (modulo n j) 0) false]
                                                   [else (loop2 n (- j 2))])])
         [else (let loop3 ([n n] [j (/ (+ n 1) 2)])
                 (cond [(= j 1) true]
                   [ (= (modulo n j) 0) false]
                   [else (loop3 n (- j 2))]))])

```

```

(define (listaprimos a b)
  (cond [(= (modulo a 2) 0) (= a (+ a 1))])
        (let loop ([primos '() [i a]])
          (cond [(> i b) primos]
                [(or (= i 1) (= i 2)) (loop (append primos (list i)) (+ i 1))]
                [(esprimo i) (loop (append primos (list i)) (+ i 2))]]))

```

```
[else (loop primos (+ i 2))]))
```

#### Descripción:

Agrega 1 y 2 como primos, para posteriormente evaluar cada número impar  $n$  como primo, si existe algún número  $j$  tal que  $n \bmod j = 0$ . Los números primos se agregan a una lista.

## 7 Número mayor en una lista

**Nombre:** (maximoL l)

**Input:** Lista /

**Output:** Número mayor dentro de la lista /

#### Código:

```
(define (mayor a b)
  (if (> a b) a b))

(define (maximoL l)
  (cond
    [(null? l) 0]
    [(= (length l) 1) (car l)]
    [else (mayor (car l) (maximoL (cdr l))))])
```

#### Descripción:

Va disminuyendo la longitud de la lista confirme el primer elemento se evalúa si es el mayor.

## 8 Número menor en una lista

**Nombre:** (minimoL l)

**Input:** Lista /

**Output:** Número menor dentro de la lista /

#### Código:

```
(define (menor a b)
  (if (< a b) a b))
```

```
(define (minimoL l)
  (cond
    [(null? l) 0]
    [(= (length l) 1) (car l)]
    [else (menor (car l) (minimoL (cdr l)))]))
```

**Descripción:**

Va disminuyendo la longitud de la lista confirme el primer elemento se evalúa si es el menor.

## 9 Suma de los primeros elementos impares

**Nombre:** (impares a)

**Input:**  $a > 0$

**Output:** La suma de los primeros  $a$  elementos impares

**Código:**

```
(define (impares a)
  (if (= a 0) 0
      (+ (- (* a 2) 1) (impares (- a 1))))))
```

**Descripción:**

El valor del impar  $a$  está dado por  $\boxed{a * 2 - 1}$ , al cual se le agrega, para hacer la función recursiva, el valor del impar  $a - 1$ , hasta que  $a$  valga 0.

## 10 Desviación estándar de una lista

**Nombre:** (desviacionestandar lista)

**Input:** Una lista *lista*

**Output:** La desviación estándar de los elementos de la lista *lista*

**Código:**

```
(define (media lista)
  (/ (apply + lista) (length lista)))
```

```
(define (diferenciacuadrada lista)
  (map (lambda (x) (* x x))
```

```

(map (lambda (x) (- x (media lista))) lista))

(define (sumacuadrados lista)
  (apply + (diferenciacuadrada lista)))

(define (varianza lista)
  (/ (sumacuadrados lista) (length lista)))

(define (desviacionestandar lista)
  (cond
    [(null? lista) 0]
    [else (sqrt (varianza lista))]))

```

**Descripción:**

La desviación estándar  $s$  de un conjunto de elementos está dada por la fórmula:

$$s = \sqrt{\frac{1}{n} \sum_{i=1}^n (x_i - \text{media}(x))^2}$$

Por lo que se calcula secuencialmente, llamando así de una función a otra, la varianza, que consiste en la suma de los cuadrados de la diferencia entre cada elemento y la media de la lista, divididos entre la cantidad de elementos. Una vez con la varianza, su raíz cuadrada es la desviación estándar.