GUIA_2 (GUIA 1 ESTA EN EL CUADERNO)

1. **Construir la función Circunferencia cuya formula es: 2*Pi*Radio , dato de entrada Radio**

```
#lang racket
(define (circunferencia radio)
  (* 2 3.1415 radio)) ; 2 * Pi * radio

; Ejemplo de uso
(circunferencia 5) ; Esto devolverá 31.415, que es el valor de 2 * Pi * 5
```

2. **Construya una función para calcular la suma de: f(X,Y)=X^3+Y^2**

```
#lang racket

; Función para calcular la suma de potencias

(define (suma-potencias x y)
  (+ (expt x 3) (expt y 2))) ; X^3 + Y^2

; Solicitar los valores al usuario
(display "Ingrese el valor de X: ")
(define x (read)) ; Leer el valor de X

(display "Ingrese el valor de Y: ")
(define y (read)) ; Leer el valor de Y

; Calcular y mostrar el resultado
(display "El resultado de f(X,Y) = X^3 + Y^2 es: ")
(displayln (suma-potencias x y))
```

```
Welcome to DrRacket, version 6.8 [3m].
Language: racket, with debugging; memory limit: 128 MB.
Ingrese el valor de X: 2
Ingrese el valor de Y: 3
El resultado de f(X,Y) = X^3 + Y^2 es: 17
>
```

3. **Construir una función para calcular la siguiente expresión f(a=(a+1) ^2 + (a-1) ^2**

```
#lang racket

; Función para calcular f(a) = (a+1)^2 + (a-1)^2
(define (calcular-f a)
  (+ (expt (+ a 1) 2) (expt (- a 1) 2))) ; (a+1)^2 + (a-1)^2

; Solicitar el valor de a al usuario
(display "Ingrese el valor de a: ")
(define a (read)) ; Leer el valor de a

; Calcular y mostrar el resultado
(display "El resultado de f(a) = (a+1)^2 + (a-1)^2 es: ")
(displayln (calcular-f a))
```

```
Welcome to DrRacket, version 6.8 [3m].
Language: racket, with debugging; memory limit: 128 MB.
Ingrese el valor de a: 3
El resultado de f(a) = (a+1)^2 + (a-1)^2 es: 20
>
```

4. **Construir una función para establecer el mayor entre tres números diferentes**

```
#lang racket

; Función para determinar el mayor entre tres números diferentes
(define (mayor-entre-tres a b c)
  (cond
    [(and (> a b) (> a c)) a] ; Si a es mayor que b y c, entonces a es el mayor
    [(and (> b a) (> b c)) b] ; Si b es mayor que a y c, entonces b es el mayor
    [else c]))                ; En cualquier otro caso, c es el mayor

; Solicitar los valores al usuario
(display "Ingrese el valor de a: ")
(define a (read)) ; Leer el valor de a

(display "Ingrese el valor de b: ")
(define b (read)) ; Leer el valor de b

(display "Ingrese el valor de c: ")
(define c (read)) ; Leer el valor de c

; Calcular y mostrar el resultado
(display "El mayor entre los tres números es: ")
(displayln (mayor-entre-tres a b c))
```

```
Welcome to DrRacket, version 6.8 [3m].
Language: racket, with debugging; memory limit: 128 MB.
Ingrese el valor de a: 5
Ingrese el valor de b: 9
Ingrese el valor de c: 3
El mayor entre los tres números es: 9
>
```

5. **Elaborar un algoritmo que lea los 3 lados de un triángulo cualquiera y calcule su área, considerar: Si A,B y C son los lados , y Sel semi perímetro. A= √S* (S-A) (S-B) (S-C)**

```
#lang racket

; Función para calcular el área de un triángulo usando la fórmula de Herón
(define (area-triangulo a b c)
  (let* ([s (/ (+ a b c) 2)] ; Calcular el semiperímetro
         [area (sqrt (* s (- s a) (- s b) (- s c)))]) ; Aplicar la fórmula de Herón
    area))

; Solicitar los valores de los lados al usuario
(display "Ingrese el valor del lado a: ")
(define a (read)) ; Leer el valor del lado a
```

```
(display "Ingrese el valor del lado b: ")
(define b (read)) ; Leer el valor del lado b

(display "Ingrese el valor del lado c: ")
(define c (read)) ; Leer el valor del lado c

; Calcular y mostrar el resultado
(display "El área del triángulo es: ")
(displayln (area-triangulo a b c))
```

```
Welcome to DrRacket, version 6.8 [3m].
Language: racket, with debugging; memory limit: 128 MB.
Ingrese el valor del lado a: 3
Ingrese el valor del lado b: 4
Ingrese el valor del lado c: 5
El área del triángulo es: 6
>
```

6. **Se tiene registrada la producción(unidades) logradas por un operario a lo largo de la semana (lunes a sábado). Elabore un algoritmo que nos muestre si el operario recibirá incentivos sabiendo que el promedio de producción mínima es de 100 unidades. Si el promedio de producción es mayor o igual a 100 unidades recibe incentivos.**

```
#lang racket

; Función para determinar si un operario recibe incentivos
(define (incentivo-operario lunes martes miercoles jueves viernes sabado)
  (let* ([total (+ lunes martes miercoles jueves viernes sabado)]
         [promedio (/ total 6)])
    (if (>= promedio 100)
        (displayln "El operario recibe incentivos")
        (displayln "El operario no recibe incentivos"))))

; Solicitar los valores al usuario
(display "Ingrese la producción del lunes: ")
(define lunes (read)) ; Leer la producción del lunes

(display "Ingrese la producción del martes: ")
(define martes (read)) ; Leer la producción del martes

(display "Ingrese la producción del miércoles: ")
(define miercoles (read)) ; Leer la producción del miércoles

(display "Ingrese la producción del jueves: ")
(define jueves (read)) ; Leer la producción del jueves

(display "Ingrese la producción del viernes: ")
(define viernes (read)) ; Leer la producción del viernes

(display "Ingrese la producción del sábado: ")
(define sabado (read)) ; Leer la producción del sábado
```

; Calcular y mostrar el resultado
(incentivo-operario lunes martes miercoles jueves viernes sabado)

```
Welcome to DrRacket, version 6.8 [3m].
Language: racket, with debugging; memory limit: 128 MB.
Ingrese la producción del lunes: 110
Ingrese la producción del martes: 95
Ingrese la producción del miércoles: 100
Ingrese la producción del jueves: 105
Ingrese la producción del viernes: 90
Ingrese la producción del sábado: 120
El operario recibe incentivos
>
```

GUIA_3_ESTRUCTURAS DE CONTROL

1 Dado un número natural entre 1 -7, imprimir los números de la semana según el número: 1=Lunes,.. . . ., 7=Domingo.

```
#lang racket

; Función para imprimir el día de la semana según el número ingresado
(define (dia-semana num)
  (cond
    [(= num 1) (displayln "Lunes")]
    [(= num 2) (displayln "Martes")]
    [(= num 3) (displayln "Miércoles")]
    [(= num 4) (displayln "Jueves")]
    [(= num 5) (displayln "Viernes")]
    [(= num 6) (displayln "Sábado")]
    [(= num 7) (displayln "Domingo")]
    [else (displayln "Número fuera de rango (1-7)")]))

; Solicitar el número al usuario
(display "Ingrese un número del 1 al 7: ")
(define num (read)) ; Leer el número ingresado

; Mostrar el día correspondiente
(dia-semana num)
```

```
Welcome to DrRacket, version 6.8 [3m].
Language: racket, with debugging; memory limit: 128 MB.
Ingrese un número del 1 al 7: 3
Miércoles
>
```

2. Realizar un programa recursivo para dividir dos números, con restas sucesivas

```
#lang racket

; Función recursiva para realizar la división por restas sucesivas
(define (division-resta dividendo divisor)
  (cond
```

```
    [(< dividendo divisor) 0] ; Si el dividendo es menor que el divisor, el cociente es 0
    [else (+ 1 (division-resta (- dividendo divisor) divisor))])) ; Resta sucesiva y cuenta

; Solicitar los valores al usuario
(display "Ingrese el dividendo: ")
(define dividendo (read)) ; Leer el dividendo

(display "Ingrese el divisor: ")
(define divisor (read)) ; Leer el divisor

; Verificar si el divisor es válido y calcular el resultado
(if (= divisor 0)
    (displayln "Error: División por cero no está permitida")
    (begin
      (display "El cociente es: ")
      (displayln (division-resta dividendo divisor))))
```

```
Welcome to DrRacket, version 6.8 [3m].
Language: racket, with debugging; memory limit: 128 MB.
Ingrese el dividendo: 15
Ingrese el divisor: 3
El cociente es: 5
>
```

3. Generar los números los números naturales hasta n.

```
#lang racket

; Función principal para verificar si n es natural y generar los números hasta n
(define (naturales-hasta n)
  (if (< n 0)
      (displayln "El valor de n debe ser un número natural (n ≥ 0)")
      (begin
        (naturales-desde-0 n))))

; Función recursiva para imprimir números naturales desde 0 hasta n
(define (naturales-desde-0 n)
  (if (= n 0)
      (displayln 0) ; Imprime 0 cuando n es 0
      (begin
        (naturales-desde-0 (- n 1)) ; Llama recursivamente con n-1
        (displayln n)))) ; Imprime n después de cada llamada recursiva

; Solicitar el valor de n al usuario
(display "Ingrese el valor de n: ")
(define n (read)) ; Leer el valor de n
```

; Generar e imprimir los números naturales hasta n
(naturales-hasta n)

```
Welcome to DrRacket, version 6.8 [3m].
Language: racket, with debugging; memory limit: 128 MB.
Ingrese el valor de n: 5
0
1
2
3
4
5
>
```

4. Generar la siguiente sumatoria:

$$\sum_{i=a}^{n} i$$

#lang racket

; Función recursiva para calcular la sumatoria desde a hasta n
(define (sumatoria a n)
  (if (> a n)
      0 ; Si a es mayor que n, se devuelve 0
      (+ a (sumatoria (+ a 1) n)))) ; Suma a y llama recursivamente con a+1

; Solicitar los valores al usuario
(display "Ingrese el valor de a (inicio): ")
(define a (read)) ; Leer el valor de a

(display "Ingrese el valor de n (fin): ")
(define n (read)) ; Leer el valor de n

; Calcular y mostrar el resultado de la sumatoria
(if (> a n)
    (displayln "El valor de a debe ser menor o igual a n")
    (let ([resultado (sumatoria a n)]) ; Calcular el resultado
      (displayln resultado))) ; Mostrar el resultado

```
Welcome to DrRacket, version 6.8 [3m].
Language: racket, with debugging; memory limit: 128 MB.
Ingrese el valor de a (inicio): 3
Ingrese el valor de n (fin): 7
25
```

Por esto da 25 → 3 + 4 + 5 + 6 + 7 = 25

5. Generar la siguiente Sumatoria

$$\sum_{i=a}^{n} \frac{1}{i^2+1}$$

```
#lang racket

(define (sumatoria a n)
  (define (suma acumulador i)
    (if (> i n)
        acumulador
        (suma (+ acumulador (/ 1 (+ (* i i) 1))) (+ i 1))))
  (suma 0 a))

; Solicitar el valor inicial a
(display "Ingrese el valor inicial (a): ")
(define a (read))

; Solicitar el valor final n
(display "Ingrese el valor final (n): ")
(define n (read))

; Calcular la sumatoria y mostrar el resultado en formato decimal
(define resultado (sumatoria a n))
(display "El resultado de la sumatoria es: ")
(displayln (exact->inexact resultado)) ; Convierte
```

```
Welcome to DrRacket, version 6.8 [3m].
Language: racket, with debugging; memory limit: 128 MB.
Ingrese el valor inicial (a): 4
Ingrese el valor final (n): 7
El resultado de la sumatoria es: 0.1443120949003302
>
```

GUIA_4 EJERCICIOS DE INICIO

1. **Escribir un programa que reciba un número e indique si se trata de un número par:**

```
#lang racket

; Función para verificar si un número es par
(define (es-par? n)
  (if (= (modulo n 2) 0)           ; Verifica si el resto de n dividido por 2 es 0
      (display "El número es par.")  ; Imprime que el número es par
      (display "El número es impar."))) ; Imprime que el número es impar

; Llamada a la función principal
(display "Ingrese un número para verificar si es par: ")
```

```
(define num-par (read))          ; Lee el número ingresado por el usuario
(es-par? num-par)                ; Llama a la función para verificar si es par
```

```
Welcome to DrRacket, version 6.8 [3m].
Language: racket, with debugging; memory limit: 128 MB.
Ingrese un número para verificar si es par: 1
El número es impar.
>
```

2. **Calcular la suma de enteros hasta un número**

```
#lang racket

; Función para calcular la suma de enteros hasta n
(define (suma-enteros n)
  (if (= n 1)                    ; Caso base: si n es 1
      1                          ; La suma de enteros hasta 1 es 1
      (+ n (suma-enteros (- n 1))))  ; Sumar n y llamar recursivamente con n-1

; Función para mostrar la suma de enteros y los números involucrados
(define (suma-y-numeros n)
  (define resultado (suma-enteros n)) ; Llama a la función de suma
  (display "Sumando los números: ")
  (for ([i (in-range 1 (+ n 1))])     ; Itera desde 1 hasta n
    (display i)                  ; Imprime cada número
    (when (< i n)                ; Si no es el último número
      (display ", ")))           ; Imprime una coma
  (newline)                      ; Salto de línea
  (printf "La suma de enteros entre 1 y ~a es: ~a\n" n resultado)) ; Imprime el resultado

; Llamada a la función principal
(display "Ingrese un número para calcular la suma de enteros: ")
(define num-suma (read))         ; Lee el número ingresado por el usuario
(suma-y-numeros num-suma)        ; Llama a la función para calcular la suma y mostrar los números
```

```
Welcome to DrRacket, version 6.8 [3m].
Language: racket, with debugging; memory limit: 128 MB.
Ingrese un número para calcular la suma de enteros: 5
Sumando los números: 1, 2, 3, 4, 5
La suma de enteros entre 1 y 5 es: 15
```

3. **Contar números naturales entre dos números**

```
#lang racket
```

```
; Función para calcular la suma de enteros hasta n
(define (suma-enteros n)
  (if (= n 1)                    ; Caso base: si n es 1
      1                          ; La suma de enteros hasta 1 es 1
      (+ n (suma-enteros (- n 1))))  ; Sumar n y llamar recursivamente
con n-1

; Función para mostrar la suma de enteros y los números
involucrados
(define (suma-y-numeros n)
  (define resultado (suma-enteros n)) ; Llama a la función de suma
  (display "Sumando los números: ")
  (for ([i (in-range 1 (+ n 1))])     ; Itera desde 1 hasta n
    (display i)                      ; Imprime cada número
    (when (< i n)                    ; Si no es el último número
      (display ", ")))               ; Imprime una coma
  (newline)                          ; Salto de línea
  (printf "La suma de enteros entre 1 y ~a es: ~a\n" n resultado)) ;
Imprime el resultado

; Llamada a la función principal
(display "Ingrese un número para calcular la suma de enteros: ")
(define num-suma (read))             ; Lee el número ingresado por el
usuario
(suma-y-numeros num-suma)            ; Llama a la función para calcular
la suma y mostrar los números
```

```
Welcome to DrRacket, version 6.8 [3m].
Language: racket, with debugging; memory limit: 128 MB.
Ingrese un número para calcular la suma de enteros: 5
Sumando los números: 1, 2, 3, 4, 5
La suma de enteros entre 1 y 5 es: 15
>
```

4. **Obtener el nombre del día de la semana**

```
#lang racket

; Función para obtener el nombre del día de la semana
(define (nombre-dia n)
  (cond
    [(= n 1) (display "Lunes")]
    [(= n 2) (display "Martes")]
    [(= n 3) (display "Miércoles")]
    [(= n 4) (display "Jueves")]
    [(= n 5) (display "Viernes")]
    [(= n 6) (display "Sábado")]
    [(= n 7) (display "Domingo")]
    [else (display "Número de día inválido")]))
```

```
; Llamada a la función principal
(display "Ingrese el número del día de la semana (1-7): ")
(define dia (read))              ; Lee el número del día
(nombre-dia dia)                 ; Llama a la función para obtener el
nombre del día
```

```
Welcome to DrRacket, version 6.8 [3m].
Language: racket, with debugging; memory limit: 128 MB.
Ingrese el número del día de la semana (1-7): 5
Viernes
>
```

GUIA_5

1. **Construir una función que reciba un parámetro y devuelva Verdad si es un carácter. Falso si no lo es.**

```
#lang racket

; Función para verificar si el parámetro es un carácter
(define (es-caracter parametro)
  (if (and (char? parametro)          ; Verifica si es un carácter
           (not (char-numeric? parametro)))  ; Verifica que no sea un
número
      "Verdad"                    ; Si es un carácter (no numérico),
devuelve "Verdad"
      "Falso"))                   ; Si no es un carácter, devuelve "Falso"

; Llamada a la función principal
(display "Ingrese un carácter: ")
(define caracter (read-char))       ; Lee un solo carácter desde el
teclado
(displayln (es-caracter caracter))  ; Muestra el resultado
```

```
Language: racket, with debugging; memory limit: 128 MB.
Ingrese un carácter: A
Verdad
>
```

2. **Construir una función que reciba un parámetro. Si el parámetro es un carácter alfabético, determinar si está en minúscula y pasarlo a mayúscula y retornar este valor. Hacer lo mismo en caso contrario.**

```
#lang racket

; Función para cambiar el caso del carácter alfabético
(define (cambiar-caso caracter)
  (if (char? caracter)              ; Verifica si es un carácter
      (if (char-alphabetic? caracter) ; Verifica si el carácter es alfabético
          (if (char-lower-case? caracter) ; Si es minúscula
              (char-upcase caracter)   ; Cambia a mayúscula
```

(char-downcase caracter)) ; Si es mayúscula, cambia a minúscula
"El parámetro no es un carácter alfabético.") ; Si no es alfabético
"El parámetro no es un carácter.")) ; Si no es un carácter

; Llamada a la función principal
(display "Ingrese un carácter alfabético: ")
(define caracter (read-char)) ; Lee un solo carácter desde el teclado
(displayln (cambiar-caso caracter)) ; Muestra el resultado

```
Welcome to DrRacket, version 6.8 [3m].
Language: racket, with debugging; memory limit: 128 MB.
Ingrese un carácter alfabético: a
A
>
```

3. **Realice un programa que pida un número y saque por pantalla su tabla de Sumas (del a-10).**

#lang racket

(define (tabla-sumas numero)
(for ([i (in-range 1 11)])
(displayln (string-append (number->string numero) " + " (number->string i) " = " (number->string (+ numero i))))))

;; Ejemplo de uso
(display "Ingrese un número: ")
(define numero (read))
(tabla-sumas numero)

```
Welcome to DrRacket, version 6.8 [3m].
Language: racket, with debugging; memory limit: 128 MB.
Ingrese un número: 4
4 + 1 = 5
4 + 2 = 6
4 + 3 = 7
4 + 4 = 8
4 + 5 = 9
4 + 6 = 10
4 + 7 = 11
4 + 8 = 12
4 + 9 = 13
4 + 10 = 14
```

5. **Tarea de la multiplicación**

#lang racket
(define (tabla-multiplicar n)
(for ([i (in-range 1 11)])
(printf "~a x ~a = ~a\n" n i (* n i))))

```
(define (main)
  (display "Ingrese un número: ")
  (define num (read))
  (tabla-multiplicar num))

(main)
```

```
Ingrese un número: 5
5 x 1 = 5
5 x 2 = 10
5 x 3 = 15
5 x 4 = 20
5 x 5 = 25
5 x 6 = 30
5 x 7 = 35
5 x 8 = 40
5 x 9 = 45
5 x 10 = 50
>
```

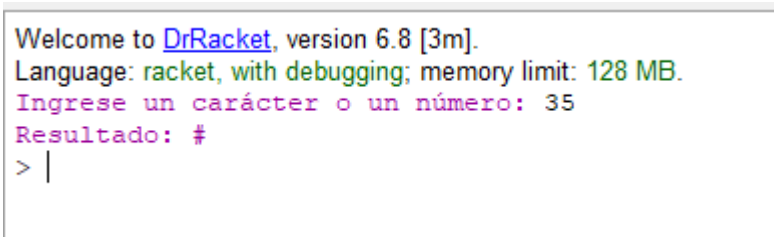
4. **Construir una función que reciba un parámetro. Si el parámetro es un carácter devolver el número que corresponda en la tabla del código ASCII y si es número devolver el carácter que corresponda en la tabla. Nota: la función (number? n),retorna verdadero si n es un número y falso de lo contrario**

```
#lang racket

(define (ascii-o-caracter parametro)
  (cond
    [(char? parametro) (char->integer parametro)]  ; Si es un carácter, devuelve el código ASCII
    [(and (string? parametro) (= (string-length parametro) 1))  ; Si es una cadena de un solo carácter
     (char->integer (string-ref parametro 0))] ; Convierte el carácter de la cadena al código ASCII
    [(and (string? parametro) (string->number parametro)) ; Si es una cadena que representa un número
     (integer->char (string->number parametro))] ; Convierte el número en su correspondiente carácter
    [else "El parámetro no es un carácter ni un número."])) ; Si no es ninguno de los anteriores

(display "Ingrese un carácter o un número: ")
(define entrada (read-line))  ; Lee la entrada como una línea de texto

;; Verifica si la entrada es un número o un carácter
(display "Resultado: ")
(display (ascii-o-caracter entrada)) ; Muestra el resultado de la función
```



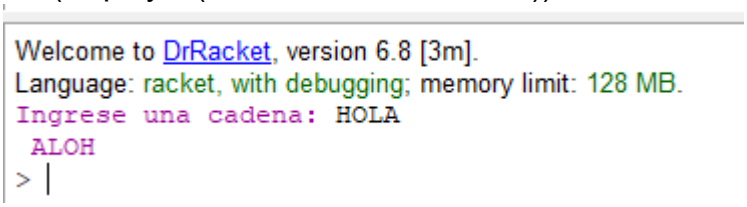
Ejercicio:

1. **Construir una función que reciba una cadena y la devuelva invertida.**

```
#lang racket

(define (invertir-cadena cadena)
  (list->string (reverse (string->list cadena))))

;; Ejemplo de uso
(display "Ingrese una cadena: ")
(define cadena (read-line))
(displayln (invertir-cadena cadena))
```

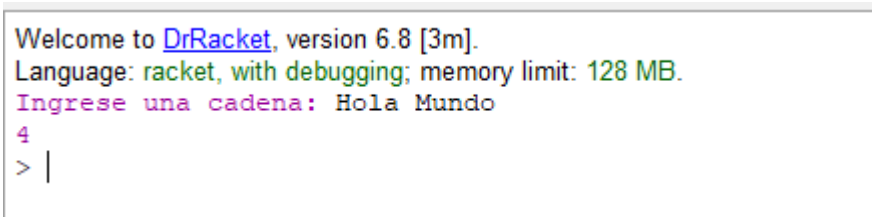


2. **Construir una función que reciba una cadena y devuelva cuantas vocales tiene**

```
#lang racket

(define (contar-vocales cadena)
  (define vocales '(#\a #\e #\i #\o #\u #\A #\E #\I #\O #\U)) ; Definimos las vocales (mayúsculas y minúsculas)
  (define (es-vocal? c) (member c vocales)) ; Función que verifica si un carácter es una vocal
  (define caracteres (string->list cadena)) ; Convertimos la cadena en una lista de caracteres
  (define vocales-en-cadena (filter es-vocal? caracteres)) ; Filtramos las vocales
  (length vocales-en-cadena)) ; Contamos cuántas vocales hay

;; Ejemplo de uso
(display "Ingrese una cadena: ")
(define cadena (read-line))
(displayln (contar-vocales cadena))
```


Welcome to DrRacket, version 6.8 [3m].
Language: racket, with debugging; memory limit: 128 MB.
Ingrese una cadena: Hola Mundo
4
>

EJERCICIOS DE VECTORES

1. **Dado un vector con diferentes valores imprimir dichos valores del vector leyéndolos uno por uno de manera recursiva.**

```
#lang racket

(define (imprimir-vector-recursivo vec)
  ; Definimos una función recursiva para imprimir el vector
  (define (imprimir-recursivo idx)
    ; Comprobamos si el índice es menor que la longitud del vector
    (if (< idx (vector-length vec))
        (begin
          (display (vector-ref vec idx)) ; Imprime el valor en la posición idx
          (newline)                      ; Imprime un salto de línea
          (imprimir-recursivo (+ idx 1)) ; Llamada recursiva al siguiente
índice
        ) ; Cierra el 'begin'
        (void)                     ; Si el índice es igual o mayor, termina la
recursión
    ) ; Cierra el 'if'
  ) ; Cierra la definición de 'imprimir-recursivo'

  ; Llamamos a la función recursiva empezando desde el índice 0
  (imprimir-recursivo 0)
)

; Función para leer n valores del usuario y almacenarlos en un vector
(define (leer-valores n)
  (define valores '()) ; Lista vacía para almacenar los valores
  (define (leer-valores-recursivo idx)
    (if (< idx n)
        (begin
          (display "Ingrese un valor: ")
          (let ([val (read)])  ; Usamos let para definir val dentro de la función
            (set! valores (append valores (list val)))) ; Agregamos el valor a la
lista
          (leer-valores-recursivo (+ idx 1)) ; Llamamos recursivamente
        )
        (vector (apply vector valores)) ; Convertimos la lista a un vector
    )
  )
  (leer-valores-recursivo 0) ; Llamamos a la función recursiva
comenzando desde el índice 0
)

; Pedimos al usuario el tamaño del vector
(display "Ingrese el tamaño del vector: ")
```

```
(define n (read))

; Leemos los valores
(define mi-vector (leer-valores n))

; Imprimimos el vector ingresado
(imprimir-vector-recursivo mi-vector)
```

```
Welcome to DrRacket, version 6.8 [3m].
Language: racket, with debugging; memory limit: 128 MB.
Ingrese el tamaño del vector: 5
Ingrese un valor: 4
Ingrese un valor: 4
Ingrese un valor: 7
Ingrese un valor: 1
Ingrese un valor: 9
#(4 4 7 1 9)
>
```

2. **Dado un número n, crear un vector de tamaño n y luego ingresar en el vector los números del 1 hasta n, e imprimir el vector. Ej: Dado el número 4 ingresar en el vector (vector 1 2 3 4).**

```
#lang racket

; Función para crear el vector y llenarlo con los números del 1 hasta n
(define (crear-vector n)
  (define (llenar-vector idx vec)
    (if (< idx n)  ; Aseguramos que el índice sea menor que n
        (begin
          (vector-set! vec idx (+ idx 1)) ; Establecemos el valor como idx + 1
(porque el índice comienza en 0)
          (llenar-vector (+ idx 1) vec)  ; Llamada recursiva para llenar el
siguiente índice
        )
        vec))  ; Devolvemos el vector cuando hemos llegado a n
  (let ([vec (make-vector n)])  ; Creamos un vector vacío de tamaño n
    (llenar-vector 0 vec)  ; Llenamos el vector comenzando desde el
índice 0
    vec))  ; Devolvemos el vector lleno

; Función para imprimir el vector
(define (imprimir-vector vec)
  (define (imprimir-recursivo idx)
    (if (< idx (vector-length vec))
        (begin
          (display (vector-ref vec idx)) ; Imprime el valor en la posición idx
          (display " ")                  ; Agrega un espacio para separar los
números
          (imprimir-recursivo (+ idx 1)) ; Llamada recursiva al siguiente
índice
        )
```

```
      (newline))) ; Salto de línea al finalizar la impresión
  (imprimir-recursivo 0))  ; Comenzamos desde el índice 0

; Solicitar al usuario el tamaño del vector
(display "Ingrese el tamaño del vector: ")
(define n (read))

; Crear el vector con números del 1 hasta n
(define mi-vector (crear-vector n))

; Imprimir el vector
(display "El vector creado es: ")
(imprimir-vector mi-vector)
```

```
Welcome to DrRacket, version 6.8 [3m].
Language: racket, with debugging; memory limit: 128 MB.
Ingrese el tamaño del vector: 5
El vector creado es: 1 2 3 4 5
>
```

3. **Llenar un vector V de 10 elementos con cuadrados de los 10 primeros elementos. Ejemplo: (vector 1 4 9 1 6 25 36 49 64 81 100).**

```
#lang racket

; Función para crear un vector de tamaño n y llenarlo con los cuadrados
de los números del 1 hasta n
(define (crear-vector-cuadrados n)
  (define (llenar-vector idx vec)
    (if (< idx n)  ; Aseguramos que el índice esté dentro del rango
        (begin
          (vector-set! vec idx (* (+ idx 1) (+ idx 1))) ; Asignamos el cuadrado
de (idx + 1)
          (llenar-vector (+ idx 1) vec)) ; Llamada recursiva al siguiente índice
        vec))  ; Devolvemos el vector cuando hemos llenado todos los
elementos
  (let ([vec (make-vector n)])  ; Creamos un vector vacío de tamaño n
    (llenar-vector 0 vec)  ; Llenamos el vector comenzando desde el
índice 0
    vec))  ; Devolvemos el vector lleno

; Función para imprimir el vector
(define (imprimir-vector vec)
  (define (imprimir-recursivo idx)
    (if (< idx (vector-length vec))
        (begin
          (display (vector-ref vec idx)) ; Imprime el valor en la posición idx
          (display " ")                  ; Espacio entre los números
          (imprimir-recursivo (+ idx 1)) ; Llamada recursiva al siguiente
índice
```

```
      )
      (newline))) ; Salto de línea al finalizar la impresión
  (imprimir-recursivo 0))  ; Comenzamos desde el índice 0

; Solicitar al usuario el tamaño del vector
(display "Ingrese el tamaño del vector (debe ser 10 o más): ")
(define n (read))

; Verificar que el tamaño sea al menos 10
(if (< n 10)
    (display "El tamaño debe ser al menos 10. Se establecerá un tamaño
de 10.")
    (set! n 10))

; Crear el vector con los cuadrados de los números del 1 hasta n
(define mi-vector (crear-vector-cuadrados n))

; Imprimir el vector
(display "El vector creado con los cuadrados es: ")
(imprimir-vector mi-vector)
```

```
Welcome to DrRacket, version 6.8 [3m].
Language: racket, with debugging; memory limit: 128 MB.
Ingrese el tamaño del vector (debe ser 10 o más): 10
El vector creado con los cuadrados es: 1 4 9 16 25 36 49 64 81 100
>
```

4. **Dado un vector V de enteros y un número X, devolver el valor que corresponde al número de veces que está el valor X en el vector.**

```
#lang racket

; Función para contar cuántas veces aparece el número X en el vector
(define (contar-valor-en-vector vec x)
  ; Función recursiva para recorrer el vector y contar las veces que
aparece X
  (define (contar-recursivo idx count)
    (if (< idx (vector-length vec)) ; Si el índice es menor que la longitud del
vector
        (if (= (vector-ref vec idx) x) ; Si el valor en la posición idx es igual a
X
            (contar-recursivo (+ idx 1) (+ count 1)) ; Aumentamos el contador
            (contar-recursivo (+ idx 1) count)) ; No aumentamos el contador
        count))  ; Devolvemos el contador cuando llegamos al final del
vector
  (contar-recursivo 0 0))  ; Comenzamos desde el índice 0 y el contador
en 0

; Función para leer un vector de tamaño n ingresado por el usuario
(define (leer-vector n)
```

```
  (define vec (make-vector n))  ; Creamos el vector vacío de tamaño n

  ; Función recursiva para llenar el vector con los valores ingresados
  (define (leer-recursivo idx)
    (if (< idx n)  ; Mientras el índice sea menor que el tamaño del vector
        (begin
          (display "Ingrese el valor para el índice ")
          (display idx)
          (display ": ")
          (let ((valor (read)))  ; Leemos el valor usando let para evitar 'define' en expresión
            (vector-set! vec idx valor)) ; Asignamos el valor al vector
          (leer-recursivo (+ idx 1)) ; Llamamos a la función recursiva para el siguiente índice
        )
        vec))  ; Devolvemos el vector cuando se haya llenado
  (leer-recursivo 0))  ; Comenzamos desde el índice 0

; Ejemplo de uso

; Solicitar al usuario el tamaño del vector
(display "Ingrese el tamaño del vector: ")
(define n (read))

; Crear un vector de tamaño n y llenarlo con los valores ingresados por el usuario
(define mi-vector (leer-vector n))

; Mostrar el vector ingresado
(display "El vector ingresado es: ")
(display mi-vector)
(newline)

; Solicitar al usuario el valor X para contar cuántas veces aparece en el vector
(display "Ingrese el valor de X: ")
(define x (read))

; Contar cuántas veces aparece X en el vector
(define resultado (contar-valor-en-vector mi-vector x))

; Mostrar el resultado
(display "El número X aparece en el vector: ")
(display resultado)
(newline)
```

```
Welcome to DrRacket, version 6.8 [3m].
Language: racket, with debugging; memory limit: 128 MB.
Ingrese el tamaño del vector: 5
Ingrese el valor para el índice 0: 1
Ingrese el valor para el índice 1: 1
Ingrese el valor para el índice 2: 4
Ingrese el valor para el índice 3: 6
Ingrese el valor para el índice 4: 2
El vector ingresado es: #(1 1 4 6 2)
Ingrese el valor de X: 1
El número X aparece en el vector: 2
>
```

EJERCICIOS DE LABORATORIO

1. **Construir una función que reciba un parámetro. Si el parámetro es un carácter devolver el número que corresponda en la tabla del código ASCII y si es un número devolver el carácter que corresponda en la tabla. Tomar en cuenta que la función (number? n), retorna verdadero si n es un número y falso de lo contrario.**

```
#lang racket

(define (ascii-converter input)
  (cond
    [(number? input) (integer->char input)]  ; Si es número, convertir a carácter.
    [(char? input) (char->integer input)]   ; Si es carácter, convertir a código ASCII.
    [else "Entrada inválida"]))            ; Para cualquier otro caso.

; Solicitar datos al usuario:
(display "Introduce un número o un carácter: ")
(define user-input (read)) ; Leer entrada del usuario.

; Mostrar el resultado:#\A 65 HOLA SALE INVALIDO
(displayln (ascii-converter user-input))
```

```
Welcome to DrRacket, version 6.8 [3m].
Language: racket, with debugging; memory limit: 128 MB.
Introduce un número o un carácter: 65
A
```

2. **Construir un programa que permita recibir una cadena y contar cuantos espacios en blanco tiene (emplear recursividad).**

```
#lang racket

(define (contar-espacios cadena)
  (define (recursivo pos contador)
```

```
    (if (>= pos (string-length cadena)) ; Caso base: si hemos llegado al final de la
cadena.
        contador
        (recursivo (+ pos 1)           ; Avanzar al siguiente carácter.
                (if (char=? (string-ref cadena pos) #\space) ; Si es un espacio,
aumentar contador.
                    (+ contador 1)
                    contador))))
  (recursivo 0 0)) ; Iniciar la recursión desde la posición 0 con contador en 0.

; Solicitar cadena al usuario:
(display "Introduce una cadena: ")
(define user-input (read-line)) ; Leer cadena del usuario.

; Contar espacios y mostrar resultado:
(define resultado (contar-espacios user-input))
(displayln (string-append "Número de espacios en blanco: " (number->string
resultado)))
```

```
Welcome to DrRacket, version 6.8 [3m].
Language: racket, with debugging; memory limit: 128 MB.
Introduce una cadena: "Hola mundo desde Racket"
Número de espacios en blanco: 3
```

3. **Construir un programa que reciba un vector de enteros y un número X. Se requiere que el programa busque el número x en el vector y devuelva la posición donde se encuentra por primera vez en el vector. En caso de no estar debe devolver #f**

```
#lang racket

; Función para buscar el número en el vector
(define (buscar-en-vector vector x)
  (define len (vector-length vector)) ; Calcula la longitud del vector

  (define (buscar-posicion i)
    (cond
      [(= i len) #f] ; Si llega al final del vector sin encontrar x, devuelve
#f
      [(= (vector-ref vector i) x) i] ; Si encuentra x, devuelve la posición i
      [else (buscar-posicion (+ i 1))])) ; Llama recursivamente para la
siguiente posición

  (buscar-posicion 0)) ; Inicia la búsqueda desde la posición 0

; Solicitar al usuario los valores del vector
(display "Ingrese el número de elementos del vector: ")
(define n (read)) ; Leer la cantidad de elementos
```

```
; Crear un vector vacío y llenarlo con valores proporcionados por el
usuario
(define mi-vector (make-vector n))
(for ([i n])
  (display (string-append "Ingrese el elemento " (number->string (+ i
1)) ": "))
  (vector-set! mi-vector i (read)))

; Realizar dos búsquedas consecutivas
(for ([j 2])
  (display "Ingrese el número a buscar en el vector: ")
  (define x (read)) ; Leer el número a buscar
  (displayln (buscar-en-vector mi-vector x))) ;
```

```
Welcome to DrRacket, version 6.8 [3m].
Language: racket, with debugging; memory limit: 128 MB.
Ingrese el número de elementos del vector: 5
Ingrese el elemento 1: 4
Ingrese el elemento 2: 3
Ingrese el elemento 3: 4
Ingrese el elemento 4: 2
Ingrese el elemento 5: 7
Ingrese el número a buscar en el vector: 3
1
Ingrese el número a buscar en el vector: 1
#f
```

4. Generar la siguiente Sumatoria

$$\sum_{i=a}^{n} \frac{1}{i^2+1}$$

```
#lang racket

(define (sumatoria a n)
  (define (suma acumulador i)
    (if (> i n)
        acumulador
        (suma (+ acumulador (/ 1 (+ (* i i) 1))) (+ i 1))))
  (suma 0 a))

; Solicitar el valor inicial a
(display "Ingrese el valor inicial (a): ")
(define a (read))

; Solicitar el valor final n
(display "Ingrese el valor final (n): ")
(define n (read))

; Calcular la sumatoria y mostrar el resultado en formato decimal
(define resultado (sumatoria a n))
```

```
(display "El resultado de la sumatoria es: ")
(displayln (exact->inexact resultado)) ; Convierte
```

```
Welcome to DrRacket, version 6.8 [3m].
Language: racket, with debugging; memory limit: 128 MB.
Ingrese el valor inicial (a): 4
Ingrese el valor final (n): 7
El resultado de la sumatoria es: 0.1443120949003302
>
```

EJERCICIOS DE LABORATORIO DEL OTRO LAB

1. **Construya una función que reciba una cadena y la devuelva invertida**

```
#lang racket

(define (cadenaInvertida)
  (display "Introduzca una cadena: ")        ; Solicita la entrada al usuario.
  (define cadena (read-line))            ; Lee toda la línea como una cadena.
  (define lon (string-length cadena))       ; Calcula la longitud de la cadena.
  (display "Cadena invertida: ")              ; Imprime el encabezado de la salida.
  (displayln (revertir cadena (- lon 1) "")) ; Imprime la cadena invertida.

  ; Devuelve la cadena invertida como resultado de la función (opcional).
  (revertir cadena (- lon 1) ""))

(define (revertir cadena pos nuevo)
  (if (< pos 0)                          ; Si se alcanza la posición inicial (< 0),
      nuevo                              ; devolver la cadena invertida.
      (revertir cadena                   ; De lo contrario, continuar recursión:
                (- pos 1)                ; disminuir posición.
                (string-append nuevo     ; añadir carácter actual al resultado.
                               (string (string-ref cadena pos))))))
```

```
Welcome to DrRacket, version 6.8 [3m].
Language: racket, with debugging; memory limit: 128 MB.
> (cadenaInvertida)
Introduzca una cadena: hola mundo
Cadena invertida: odnum aloh
"odnum aloh"
```

2. **Construir un programa que reciba un vector de enteros y obtenga el promedio de los números del vector. Necesariamente utilizar recursividad**

```
#lang racket

; Función principal
(define (promedio-vector)
  (display "Introduce los números del vector separados por espacios: ")
```

```
  (define entrada (read-line))              ; Leer una línea de entrada
como cadena.
  (define numeros                           ; Convertir la cadena en una lista
de números.
    (map string->number (string-split entrada)))

  (define suma (sumar-lista numeros))       ; Llama a la función
recursiva para sumar.
  (define cantidad (length numeros))        ; Obtiene la cantidad de
números.

  (if (= cantidad 0)                        ; Verifica si el vector está vacío.
      (displayln "El vector está vacío. No se puede calcular el
promedio.")
      (displayln (string-append
                  "El promedio es: "
                  (number->string (/ suma cantidad))))))

; Función recursiva para sumar los elementos de la lista.
(define (sumar-lista lista)
  (if (null? lista)                         ; Caso base: si la lista está vacía.
      0                                     ; La suma es 0.
      (+ (first lista) (sumar-lista (rest lista))))) ; Sumar primer elemento
con el resto.
```

```
Welcome to DrRacket, version 6.8 [3m].
Language: racket, with debugging; memory limit: 128 MB.
> (promedio-vector)
Introduce los números del vector separados por espacios: 10 20 30
El promedio es: 20
>
```

3. **Genera la sumatoria Σ^n _(i=a)(1/i^3+1)**

```
#lang racket

; Función principal para solicitar los límites de la sumatoria.
(define (calcular-sumatoria)
  (display "Introduce el valor de a (límite inferior): ")
  (define a (read)) ; Leer el límite inferior.

  (display "Introduce el valor de n (límite superior): ")
  (define n (read)) ; Leer el límite superior.

  (if (> a n)
      (displayln "Error: el límite inferior (a) no puede ser mayor que el
límite superior (n).")
      (let ((resultado (sumatoria a n))) ; Usamos 'let' para definir la
variable de resultado.
        (displayln (string-append
                    "La sumatoria es: "
```

```
                (number->string resultado)))))) ; Mostrar el resultado.

; Función recursiva para calcular la sumatoria.
(define (sumatoria i n)
  (if (> i n) ; Caso base: si el índice supera el límite superior, la suma
termina.
      0.0      ; Retornar 0.0 como número decimal.
      (+ (/ 1.0 (+ (expt i 3) 1)) ; Sumar el término actual: 1 / (i^3 + 1) en
decimal.
         (sumatoria (+ i 1) n)))) ; Llamada recursiva para el siguiente
índice.

; Ejecutar la función para probar.
(calcular-sumatoria)
```

```
Welcome to DrRacket, version 6.8 [3m].
Language: racket, with debugging; memory limit: 128 MB.
Introduce el valor de a (límite inferior): 1
Introduce el valor de n (límite superior): 3
La sumatoria es: 0.6468253968253968
>
```

4. **Escribe un programa que cuente el número de vocales y consonantes que contiene necesariamente usar recursividad**

```
#lang racket

; Función principal para solicitar los límites de la sumatoria.
(define (calcular-sumatoria)
  (display "Introduce el valor de a (límite inferior): ")
  (define a (read)) ; Leer el límite inferior.

  (display "Introduce el valor de n (límite superior): ")
  (define n (read)) ; Leer el límite superior.

  (if (> a n)
      (displayln "Error: el límite inferior (a) no puede ser mayor que el
límite superior (n).")
      (let ((resultado (sumatoria a n))) ; Usamos 'let' para definir la
variable de resultado.
        (displayln (string-append
                    "La sumatoria es: "
                    (number->string resultado)))))) ; Mostrar el resultado.

; Función recursiva para calcular la sumatoria.
(define (sumatoria i n)
  (if (> i n) ; Caso base: si el índice supera el límite superior, la suma
termina.
      0.0      ; Retornar 0.0 como número decimal.
      (+ (/ 1.0 (+ (expt i 3) 1)) ; Sumar el término actual: 1 / (i^3 + 1) en
decimal.
```

(sumatoria (+ i 1) n)))) ; Llamada recursiva para el siguiente índice.

; Ejecutar la función para probar.
(calcular-sumatoria)

```
Welcome to DrRacket, version 6.8 [3m].
Language: racket, with debugging; memory limit: 128 MB.
Introduce el valor de a (límite inferior): 1
Introduce el valor de n (límite superior): 3
La sumatoria es: 0.6468253968253968
>
```