

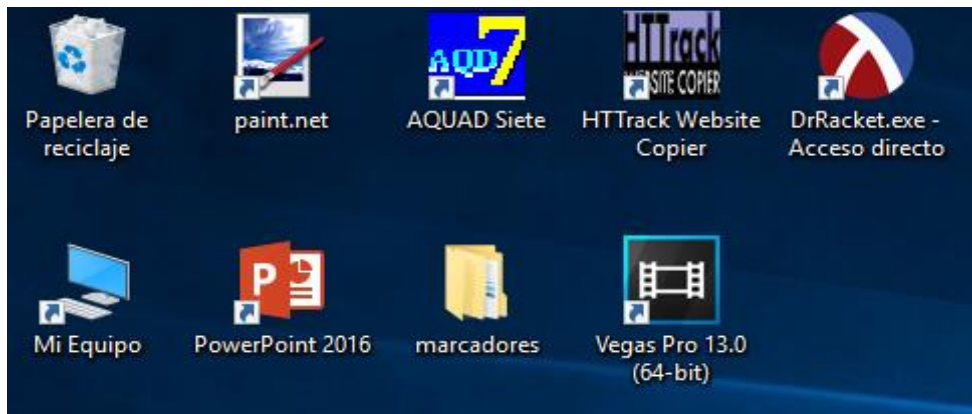


GUIA DrRacket

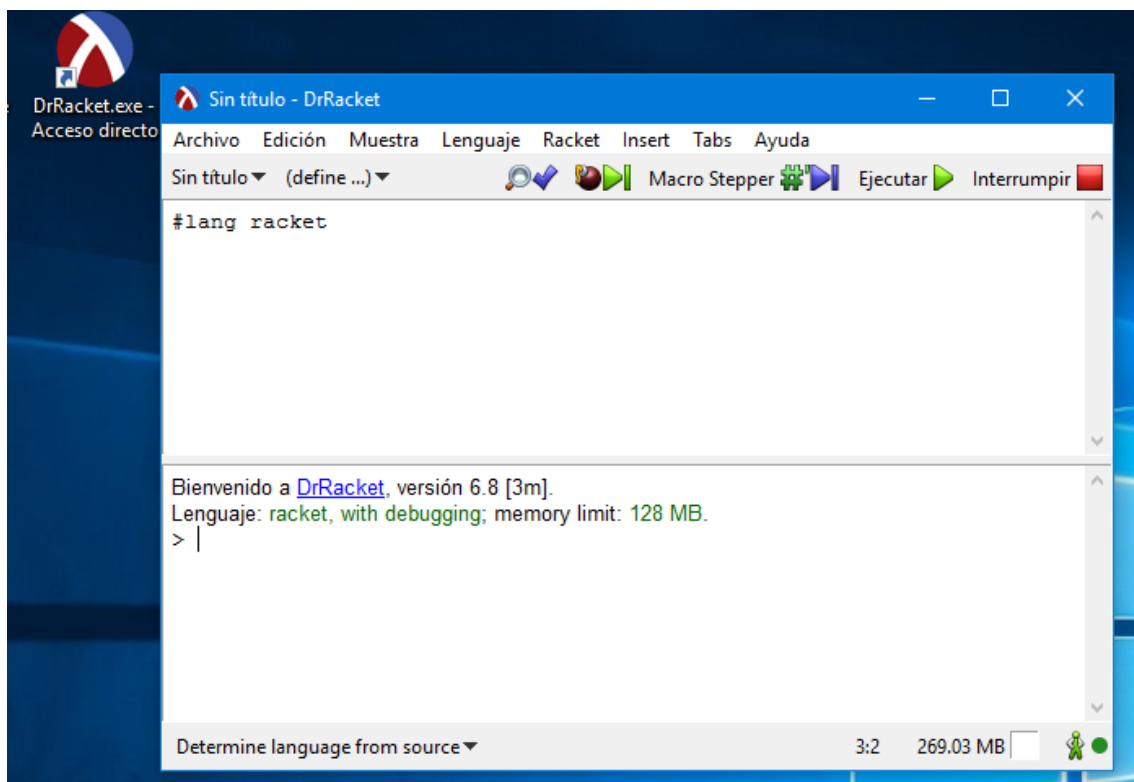
1. Entorno Dr. Racket.-

Es un lenguaje de programación SCHEME (lenguaje de programación funcional y un dialecto de Lisp). Fue desarrollado por Guy L. Steele y Gerald Jay Sussman.

Para iniciar hacemos doble clic en icono de DrRacket



Nos aparecerá la ventana de DrRacket , aparecerá una ventana dividida en dos la ventana de definiciones y la ventana de interacciones(donde se ejecutaran nuestros códigos).





2. Expresiones en DrRacke.-

La evaluación de expresiones emplea la notación de preorden como se muestra en la figura siguiente

Recorrer el subárbol izquierdo en Inorden
Examinar la raíz
Recorrer el subárbol derecho en Inorden



Preorden:

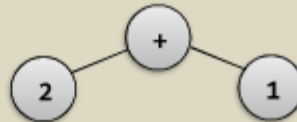
Examinar la raíz
Recorrer el subárbol izquierdo en Preorden
Recorrer el subárbol derecho en Preorden

Postorden:

Recorrer el subárbol izquierdo en Postorden
Recorrer el subárbol derecho en Postorden
Examinar la raíz

La notación de una expresión puede representarse con un Árbol Binario, si aplicamos el algoritmo correspondiente para recorrer dicho árbol obtendremos nuestra expresión en la notación deseada.

Inorden: 2 + 1
Preorden: + 2 1
Postorden: 2 1 +



Evaluemos las siguientes expresiones en DrRacket

- a) (+ (* 3 (- 5 3)) (/ 8 4))
- b) (* 3 2 1 (- 5 3))
- c) (/ (+ 3 2 1) (- 5 3))
- d) ((+) (* 2 (- 5 3)) (/ 8 4))
- e) (2 + 2)



```
#lang racket

Bienvenido a DrRacket, versión 6.8 [3m].
Lenguaje: racket, with debugging; memory limit: 128 MB.
> (+ 2 3)
5
> (+ (* 3 (- 5 3)) (/ 8 4))
8
> (* 3 2 1 (- 5 3))
12
> (/ (+ 3 2 1) (- 5 6))
-6
> ((+) (* 2 (- 5 3)) (/ 8 4))
application: not a procedure;
expected a procedure that can be applied to arguments
given: 0
arguments...:
> (2 + 2)
application: not a procedure;
expected a procedure that can be applied to arguments
given: 2
```

3. Tipos de Datos.-

En el siguiente cuadro se resume los tipos de datos que se maneja en DrRacket

TIPO	EJEMPLO
Booleano	#t, #f
Entero	1, -2, 3, 97, 0
Racional	1/4, 36/8
Real	3.1532, 1.2e+4
Complejo	0+i 15.9405, 2+4i
Carácter	#\c, #\a, #\i, #\3, #\1, #\space, #\tab, #\newline
Símbolo	'var 'xyz 'programación
Cadena	"¡Hola- Mundo!"
Lista	'(1 2 3 4 5 6)
Vector	#(1 2 3 4 5 6)
Estructura	(define-struct persona (nombre cc dirtel))



En la mayoría de los datos no es necesario declarar los tipos de datos que vamos a introducir es decir:

```
#lang racket

Bienvenido a DrRacket, versión 6.8 [3m].
Lenguaje: racket, with debugging; memory limit: 128 MB.
> 8;numero
8
> (- 9 3); numero
6
> 'Prueba ;simbolo
'Prueba
> 'A1 ;simbolo
'A1
> "Ejemplo de cadena" ;cadena string
"Ejemplo de cadena"
> #A ;Caracter
read: bad syntax `#A'
> #\A ; este si es caracter valido
#\A
> (> 4 2) ;Booleano
#t
> (< 4 2) ; booleano
#f
> |
```

3. Funciones.-

Se contempla las funciones definidas por el propio lenguaje (Funciones Primitivas) y definidas por nosotros mismos.

Algunas Funciones primitivas son:



PROGRAMACION FUNCIONAL

Función	Entrada -> Salida	Descripción
*	(num num ... -> num)	Multiplicación
+	(num num ... -> num)	Suma
-	(num num ... -> num)	Resta
/	(num num ... -> num)	Division
<	(real real ... -> bool)	Compara el primer valor con los demás, si este es el menor devuelve: #t, sino #f
<=	(real real ... -> bool)	Compara el primer valor con los demás, si este es menor o igual a
>	(real real ... -> bool)	los demás devuelve: #t, sino #f Compara el primer valor con los siguientes, si este es el mayor obtendremos: #t, de lo contrario #f
>=	(real real ... -> bool)	Compara el primer valor con los demás, si este es mayor o igual devuelve: #t, sino #f
abs	(real -> real)	Valor absoluto de un número real
acos	(num -> num)	arco-coseno
add1	(num -> num)	Aumenta en uno un numero
angle	(num -> real)	Calcula el Angulo de un #real
asin	(num -> num)	arco-seno
atan	(num -> num)	arco-tangente
cos	(num -> num)	Coseno
cosh	(num -> num)	
max	(real real... -> real)	Evalúa dos o mas numeros y nos dice cuál es el mayor de todos
min	(real real... -> real)	Evalúa dos o más números y nos dice cuál es el menor de todos
modulo	(int int-> int)	Devuelve el modulo de la division entre dos números
negative?	(num-> bool)	Evalúa un número y dice si es negativo o no
number?	(any -> bool)	Evalúa un valor y nos dice si es un numero

**PROGRAMACION FUNCIONAL**

quotient	(int int -> int)	Devuelve el cociente de la división entre dos números
random	(int -> int)	Devuelve un número aleatorio
rational?	(any -> bool)	
real-part	(num -> real)	
real?	(any -> bool)	Evalúa un valor y nos dice si es real
round	(real -> int)	Redondea un numero
sgn	(real -> union)	
sin	(num -> num)	Devuelve el seno de un numero
sinh	(num -> num)	
sqr	(num -> num)	
sqr	(num -> num)	Calcula la raíz cuadrada de un numero

Ejercicios**1. Pasar a notación Preorden:**

- a. -3
- b. $8 = -3$
- c. $(2 * 3) + 5$
- d. $((5 + 2) * 3)$
- e. $(1 + 4) * (4 + 6)$
- f. $(3 * 3) / (8 * 2)$
- g. $5 + (6 / 2) + 3$
- h. $5 + (3 * 8) + 1$
- i. $((3 + 4) * 8) + 2$
- j. $(3 + ((8 - 2) - 4)) / 6$
- k. $(5 * (75 / 15)) + (4 * (4 - 1)) + (2 * (7 + 4))$
- l. $((15 / (8 - 3)) + (4 * (6 + 2))) * 2$
- m. $(8 + 3) * (40 - (7 * 4))$

2. Evaluar

- a. $2 * |-3|$
- b. $2^3 + \sqrt{16}$
- c. Hallar el máximo de 3, 2, 8, 4



PROGRAMACION FUNCIONAL

Funciones Definidas por el Usuario

(Lectura-Escritura)

(Parte 1)

Funciones Construidas. -

Definidas por el usuario

Define.-

Es una función que asocia valores a nombres. Se puede definir identificadores tal que queden globalmente disponibles para ser utilizados.

Formato

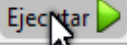
(Define <identificador> <expresión>

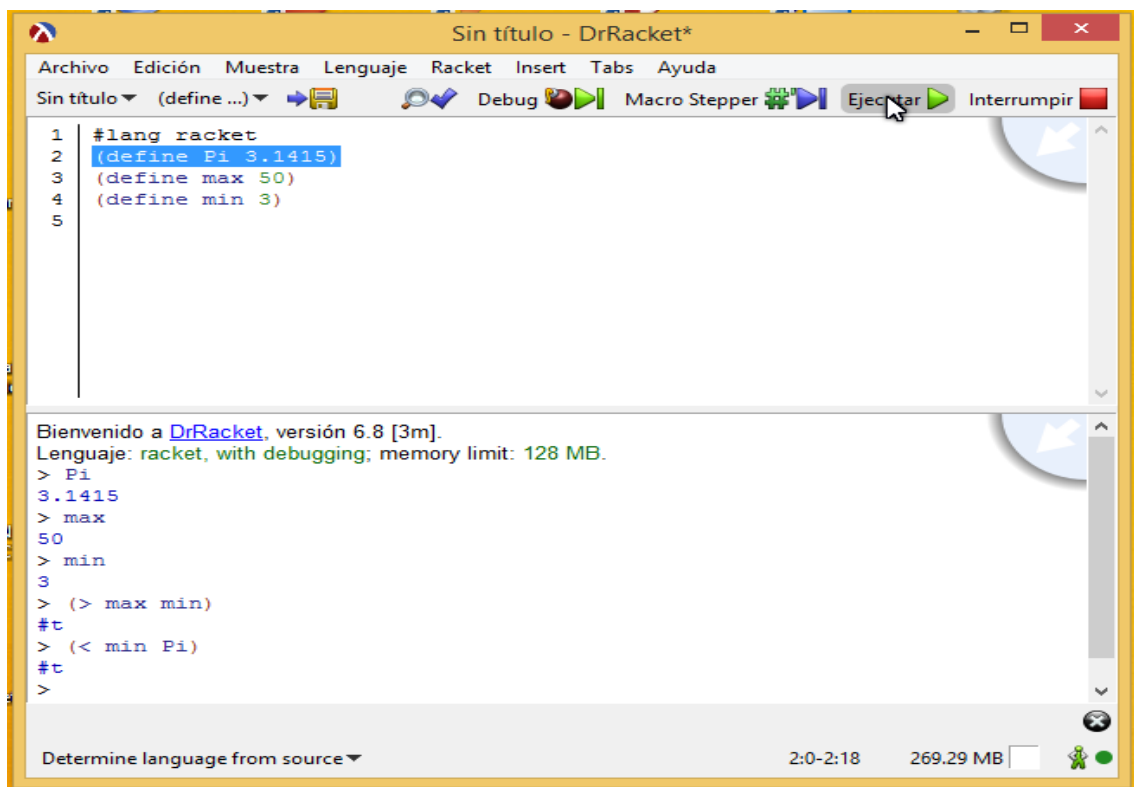
Ejemplo

(define Pi 3.1415)

(define Max 50)

(define Min 30)

Para definir los identificadores utilizamos la ventana de definiciones: escribimos la definición del identificador y luego hacemos Click en Ejecutar  como se muestra en la siguiente figura:





Definición de Funciones. -

Definidas por el usuario

Formato

(define (Nombre_Funcion Parametros_Formales)

Sentencias

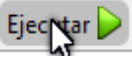
)

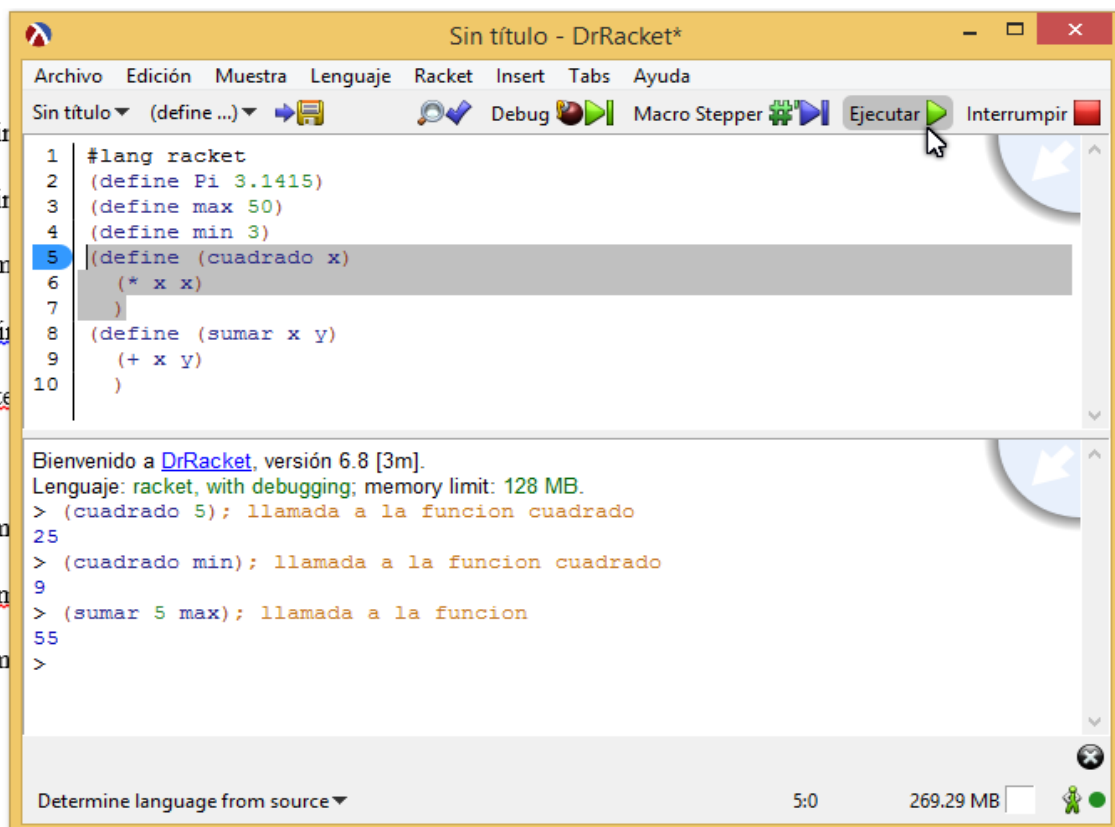
Llamada a la Función

(Nombre_Funcion Argumentos)

Ejemplo:

Del mismo modo que en la definición de Identificadores para definir una función vamos a utilizar la ventana de definición: escribimos la definición de la función y hacemos Click

en Ejecutar , como se muestra en la siguiente figura.





Sentencias de Control de Flujo.-

Control de Selectiva. -

Ejecuta las sentencias en funcion a la condición por verdad o por falso.

(if (Condición)

(Expresion_Verdad_Entonces)

(Expresion_Falso_SINO)

)

Ejemplo dado un numero establecer si es menor a 10 o mayor a 10), el ejemplo se muestra en la siguiente figura.

```
Sin título - DrRacket*
Archivo  Edición  Muestra  Lenguaje  Racket  Insert  Tabs  Ayuda
Sin título (define ...) [Icons] Debug [Icons] Macro Stepper [Icons] Ejecutar [Icon] Interrumpir [Icon]

1  #lang racket
11 (define (Menora10 x)
12   (if (< x 10)
13       (display "menor")
14       (display "mayor")))
15

Lenguaje: racket, with debugging; memory limit: 128 MB.
> (Menora10 5); llamada a la funcion
menor
> (Menora10 11); llamada a la funcion
mayor
>

Determine language from source ▼ 11:0 269.29 MB [Icon]
```

Imprimir. –

En racquet podemos emplea tres formas de imprimir: print, write, display, los tres son similares emplearemos display para trabajar en nuestros ejemplos. Sus sintaxis son:

(print Argumento)

(write Argumento)

(display Argumento)

**Ejemplos:**

(print 2) imprime 2

(write 2) imprime 2

(display 2) imprime 2

(print “saludos”) imprime “saludos”

(write “saludos”) imprime “saludos”

(display “saludos”) imprime saludos

(print #\a) imprime #\a

(write #\a) imprime #\a

(display #\a) imprime a

(define m 5)

(print m) imprime 5

(write m) imprime 5

(display m) imprime 5

Leer. -

Para leer se utiliza read, la sintaxis:

(read)

Almacena en el lugar que se le llame. Como se muestra en el grafico siguiente

```
Untitled - DrRacket*
File Edit View Language Racket Insert Tabs Help
Untitled (define ...) [save icon]

1 | #lang racket
2 |
3 | (define (ejemplo n)
4 | (display n)
5 | )

Welcome to DrRacket, version 6.8 [3m].
Language: racket, with debugging; memory limit: 128 MB.
> (ejemplo (read))
7|

Welcome to DrRacket, version 6.8 [3m].
Language: racket, with debugging; memory limit: 128 MB.
> (ejemplo (read))
7
7
>
```



PROGRAMACION FUNCIONAL

El primer 7 es de (read).

El segundo 7 es de la función **ejemplo**.

Otro ejemplo empleado una función para leer:

```
Untitled - DrRacket*
File Edit View Language Racket Insert Tabs Help
Untitled (define ...)
1 | #lang racket
2 |
3 | (define (sumauno n)
4 |   (display (+ n 3))
5 |   )
6 |
7 | (define (leer)
8 |   (begin
9 |     (display "Introduzca Numero:")
10 |    (read)
11 |    )
12 |   )
13 |
14 |

Welcome to DrRacket, version 6.8 [3m].
Language: racket, with debugging; memory limit: 128 MB.
> (sumauno (leer))
Introduzca Numero:10
13
> |
```

Ejercicios

1. Construir la función Circunferencia cuya formula es: $2 \cdot \pi \cdot \text{Radio}$, dato de entrada Radio

2. Construya una función para calcular la suma de:

$$f(X,Y)=X^3+Y^2$$

3. Construir una función para calcular la siguiente expresión

$$f(a)=(a+1)^2 + (a-1)^2$$

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



PROGRAMACION FUNCIONAL

5. Elaborar un algoritmo que lea los 3 lados de un triangulo cualquiera y calcule su área, considerar: Si A,B y C son los lados , y S el semi perímetro.

$$A = \sqrt{S * (S - A) * (S - B) * (S - C)}$$

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.

Fuentes de Información. –

Muñoz Luis & Rios Jorge & Solarte Guillermo (2014). Programación Funcional Con Racket. Universidad Tecnológica de Pereira.

Flatt Matthew & Bruce Robert & PLT(). The Racket Guia.

<https://docs.racket-lang.org/guide/index.html>

Guia2Funciones – Parte 2**Estructuras de Control o Control de Flujo****Ejemplo IF**

Crear una función que diga si el resultado de la suma de dos números es mayor a 10

```
(define (Mayor n1 n2)
```

```
(if (> (+ n1 n2) 10)
```

```
(begin
```

```
(display (+ n1 n2))
```

```
(display " Es mayor que 10")
```

```
);begin por verdad
```

```
(begin
```

```
(display (+ n1 n2))
```

```
(display " no es mayor que 10")
```

```
);begin por falso
```

```
);if
```

```
);Mayor
```

Se utiliza begin para agrupar el código en partes el primer begin es para el bloque por verdad y el segundo para el bloque de códigos por falso.

4.2 Condicional Cond.-

Su formato:

```
(cond
```

```
[(Condicion1) (Respuesta_Condicion1)]
```

```
[(Condicion2) (Respuesta_Condicion2)]
```

```
.
```



[(CondicionN) (Respuesta_CondicionN)]

(else (Respuesta_Falso)

)

Ejemplo : sobre el mismo ejemplo anterior.

(define (mayor n1 n2)

(cond

[(> (+ n1 n2) 10)

(begin

(display (+ n1 n2))

(display " es mayor que 10")

); begin

]; para >

[(= (+ n1 n2) 10) (begin (display (+ n1 n2)) (display " es igual que 10"))]

(else

(begin

(display (+ n1 n2))

(display " es menor que 10 ")

);begin del else

);else

); cond

); mayor

**Ejemplo**

Crear una función en la que el usuario ingrese un número y el programa devuelva si dicho número se encuentra entre 1 y 3, de lo contrario imprima que el número no está en el intervalo de 1 a 3

```
(define (leer)
  (begin
    (display "Ingres Numero: ")
    (read)
  )
)
(define (intervalo a)
  (cond
    [(and (<= a 3) (>= a 1)) (display "El numero se encuentra en el intervalo 1-3")]
    (else "El numero no se encuentra en el intervalo 1-3")
  )
)
```

Estructura de Control Repetitiva. -**Recursividad. -**

En programación funcional no existe el While , o Repeat , para ello empleamos la Recursividad. En el siguiente ejemplo 1 se plantea el criterio base y el criterio de recursividad, que son los elementos importantes para que haya recursividad.

$$1 \quad \text{si } n=0 \text{ o } n=1$$
$$N! = \text{Factorial}(n)$$
$$N * \text{Factorial}(n-1) \quad \text{si } n > 1$$

Llevar al DrRacket el siguiente código de solución

```
(define (factorial n)
  (if (or (= n 0) (= n 1))
      1
      (* n (factorial (- n 1)))))
```



PROGRAMACION FUNCIONAL

Ejemplo 2 Sea la función fib planteada, con su criterio base y criterio de recursividad

$$\text{fib}(n) = \begin{cases} 0 & \text{si } n=0 \\ 1 & \text{si } n=1 \\ \text{fib}(n-1) + \text{fib}(n-2) & \text{si } n>1 \end{cases}$$

Realizar: llevar la implementación al lenguaje DrRacket y ejecutar

```
(define (fib n)
  (cond ((= n 0) 0)
        ((= n 1) 1)
        (else (+ (fib (- n 1)) (fib (- n 2))))))
```

Ejercicios:

Realizar un algoritmo para:

1. Dado un numero natural entre 1 -7, imprimir los números de la semana según el numero: 1=Lunes,..., 7=Domingo.
2. Realizar un programa recursivo para dividir dos números, con restas sucesivas
3. Generar los números los números naturales hasta n.
4. Generar la siguiente sumatoria:

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

5. Generar la siguiente Sumatoria

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

Fuentes de Información. –

Muñoz Luis & Rios Jorge & Solarte Guillermo (2014). Programación Funcional Con Racket. Universidad Tecnológica de Pereira.

Flatt Matthew & Bruce Robert & PLT. The Racket Guia. <https://docs.racket-lang.org/guide/index.html>, <https://racket-lang.org/>

Programación Funcional Lisp-DrScheme. Dr. Oldemar Rodriguez Rojas



Recursividad.-

Ejemplo 3 Imprimir los números del uno al diez recursivamente

1	(define (PrintNum-10 x)
2	(if (= x 10) ;El caso base se dará cuando x sea igual a 10
3	(display x) ;Cuando el caso base se cumpla se imprimirá "x"
4	(begin ;Casos generales, cuando x no es igual a 10
5	(display x) ;si x diferente de 10 imprimir "x"
6	(PrintNum-10 (+ 1 x))) ;si x diferente de 10, volver a
	llamar la función estableciendo a "x" como x+1
7)
8)
9	(PrintNum-10 0) ;Llamado de la función

Realizar: llevar la implementación al lenguaje DrRacket y ejecutar

Realizar los siguientes ejercicios:

1. Escribir un programa que reciba un numero e indique si se trata de un numero par
2. Construir una función que reciba como parámetro un numero N y calcule la suma de todos los enteros comprendidos entre 1 y el numero recibido.
3. Escriba una función que calcule cuantos números naturales hay entre 2 números dados, incluyéndolos.
4. Escriba un programa que dado el día (en formato numérico) de la semana despliegue el nombre del día.

CARACTERES

CARACTERES

Son un tipo de dato los caracteres se utilizan para mostrarlos en pantalla y así comunicarnos con el usuarios, no solo las letras que conocemos del alfabeto son caracteres, también existen otro tipo de caracteres llamados “caracteres especiales”.

Formato

Los caracteres en Racket tienen una sintaxis especial, se escriben usando la notación:

`#\<caracter>`

Caracteres Especiales.-

Son especiales en dos sentidos:

- a) Porque no están contenidos en el alfabeto
- b) Porque los caracteres se representan con una única letra, en cambio los especiales, se representan con palabras completas.

Ejemplo:

`#\space` ; imprime un espacio en blanco

`#\newline` ; imprime un salto de línea

Operaciones con Caracteres. -

Racket provee de varias funciones para operar con caracteres, algunas de ellas son:

`Char?`: Es una función que determina si un dato es un carácter

Formato

`(char? X)`

Ejemplos: Verificar en Racket los siguientes ejemplos

`(char? 'a) -> #f` Devuelve falso porque ‘a es un símbolo y no un carácter

`(char? #\a) -> #t` Devuelve verdad porque a es un carácter \

`(char? "auxiliar") -> #f` Devuelve falso porque “auxiliar” es cadena y no un carácter

`(char? #\space) -> #t` Devuelve verdadero porque space es un carácter

Hay funciones que determinan si un carácter va antes o después de otro carácter en el código ASCII, o si son iguales, estas funciones son:

```
(char=? ch1 ch2) ; Es ch1 el mismo caracter que ch2?  
(char<? ch1 ch2) ; ch1 está antes que ch2 en el alfabeto?  
(char>? ch1 ch2) ; ch1 está después que ch2 en el alfabeto?  
(char<=? ch1 ch2) ; ch1 está antes que ch2 en el alfabeto o son los mismos?  
(char>=? ch1 ch2) ; ch1 está después que ch2 en el alfabeto o son los mismos?
```

-ci, se usa para un caso sensible, es decir las mayúsculas y minúsculas Racket las tomara como iguales, su formato:

```
(char-ci=? ch1 ch2) ; Lo mismo que char=? Pero insensible  
(char-ci<? ch1 ch2) ; Lo mismo que char<? Pero insensible  
(char-ci>? ch1 ch2) ; Lo mismo que char>? Pero insensible  
(char-ci<=? ch1 ch2) ; Lo mismo que char<=? Pero insensible  
(char-ci>=? ch1 ch2) ; Lo mismo que char>=? Pero insensible
```

Ejemplo: Realizar y verificar en Racket los siguientes ejemplos

- `(char=? #\s #\S)` -> #f Compara basándose en el código ASCII, si la s minúscula está antes que la S mayúscula.
- `(char-ci=? #\s #\S)` -> #t Devuelve verdadero porque -ci hace que no se distinga la mayúscula de la minúscula por lo tanto Racket las evalúa como iguales.
- `(char<? #\a #\b)` -> #t Devuelve verdadero pues la "a" va antes que la "b" en el código ASCII.
- `(char<? #\a #\A)` -> #f Devuelve falso pues las mayúsculas van antes que las minúsculas en el código ASCII.

Racket, tiene funciones para encontrar que tipo de carácter se está evaluando, puede ser un carácter alfabético, espacio en blanco, mayúscula o minúscula, es decir:

Formato

```
(char-alphabetic? ch)
(char-numeric? ch)
(char-whitespace? ch)
(char-upper-case? ch)
(char-lower-case? ch)
```

Ejemplo Realizar y verificar en Racket los siguientes ejemplos:

- `(char-alphabetic? #\a)` `-> #t`
- `(char-numeric? #\a)` `-> #f`
- `(char-numeric? #\2)` `-> #t`
- `(char-whitespace? #\space)` `-> #t`
- `(char-upper-case? #\A)` `-> #t`
- `(char-lower-case? #\A)` `-> #f`

Racket también tiene funciones para convertir caracteres en su equivalente entero basándose en el conjunto de datos del código ASCII y viceversa.

Formato

```
(char->integer ch)
(integer->char n)
```

Ejemplos: Realizar y verificar en Racket los siguientes ejemplos:

- `(char->integer #\3)` `-> 51`
- `(char->integer #\a)` `-> 97`
- `(integer->char 97)` `-> #\a`

En Raquet podemos usar dos procedimientos para convertir caracteres de minúsculas a mayúsculas y viceversa.

Formato

```
(char-upcase ch)
(char-downcase ch)
```

Ejemplo Realizar y verificar en Racket los siguientes ejemplos:

- `(char-upcase #\a)` `-> #\A`
- `(char-downcase #\B)` `-> #\b`
- `(char-upcase #\b)` `-> #\B`
- `(char-upcase (char-downcase #\B))` `-> #\B`

Ejemplos

1. Recibir un carácter que representa una nota. Si es la letra E(mayúscula) o la letra e (minúscula) imprimir “Excelente”, si es la letra S o s imprimir “Sobresaliente”, si es la letra A o a imprimir “Aceptable”, si es la letra es I o i imprimir “Insuficiente”, y si la letra es D o d imprimir “Deficiente”. Realizar una función

```
(define (Notas x)
  (cond
    [(char-ci=? x #\e) (display "Excelente")]
    [(char-ci=? x #\s) (display "Sobresaliente")]
    [(char-ci=? x #\a) (display "Aceptable")]
    [(char-ci=? x #\i) (display "Insuficiente")]
    [(char-ci=? x #\d) (display "Deficiente")]
    (else (display "No es nota valida"))
  )
)
```

2. Imprimir los caracteres que equivalen a los números hasta el 1024.

```
(define (CodASCII Cont)
  (if(= Cont 1024) ;Cont aumentara hasta 1024 y se dará el caso
  base
    (display (integer->char Cont)) ;(caso base)
    (begin ;Para todos los casos generales
      (display (integer->char Cont)) ;Imprimir Cont como un
      caracter
      (display " ") ;Imprimir un espacio para separar los
      caracteres
      (CodASCII(+ 1 Cont)) ;Llamado recursivo, aumenta Cont en
      1
    )
  )
)
(CodASCII 1)
```

Ejercicio sobre Caracteres:

1. Construir una función que reciba un parámetro y devuelva Verdad si es un carácter. Falso si no lo es.
2. Construir una función que reciba un parámetro. Si el parámetro es un carácter alfabético, determinar si esta en minúscula y pasarlo a mayúscula y retornar este valor. Hacer lo mismo en caso contrario.
3. Realice un programa que pida un número y saque por pantalla su tabla de Sumas (del a-10).
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

CADENA

Una cadena es un conjunto de caracteres, en Racket una cadena se representa entre comillas dobles, ejemplo “Esta es una cadena”, podemos referirnos a cada uno de los caracteres que componen la cadena con su índice, por ejemplo la cadena “ESTO” contiene cuatro caracteres y sus índices son los siguiente

Cadena	E	S	T	O
índice	0	1	2	3

Funciones Primitivas.-

Existen diferente funciones para manejar cadenas, estas son:

Creación y Modificación de Cadenas.-

Para crear una cadena basta con declararla lo cual se hace usando comillas dobles. Si no hay nada escrito entre las comillas se considera una cadena vacía.

Racket proporciona la función `string?` Para comprobar si un dato es una cadena, su formato:

`(string? n)`

Ejemplos

```
(string? "Casa"); -> #t, Devuelve verdadero porque es una cadena de caracteres
(string? 'bobo);  -> #f, Porque 'bobo es un símbolo
(string? 5);      -> #f, Porque 5 es un número
(string? "");     -> #t, Es verdadero pues es una cadena vacía
```

Racket también proporciona varias funciones para crear y manipular cadenas. Estas se pueden crear con procedimientos `make-string` y `string`.

Formato

```
(make-string n)
(make-string n ch)
(string ch1 ch2 ...)
```


Ejemplos

- `(make-string 3);` `-> "\u0000\u0000\u0000"`
Crea una cadena de 3 posiciones y la llena con ceros.
- `(make-string 3 #\a);` `-> "aaa"`
Crea una cadena de 3 posiciones y la llena con el caracter #\a.
- `(string #\R #\a #\c #\k #\e #\t);` `-> "Racket"`
Crea una cadena con los caracteres dados.

Racket cuenta con una función para calcular la longitud de una cadena: `string-length`. La longitud de una cadena es el número de caracteres que la componen, la longitud de la cadena vacía es 0, la longitud de "Hola" es cuatro.

Formato

(string-length cadena)

Toma una cadena como argumento y retorna un numero entero que corresponde al tamaño de la cadena

Ejemplo

- `(string-append "futbol" " villa" "deporte");` `-> "futbol villadeporte"`
- `(string-append);` `-> ""` Devuelve la cadena vacía

Cadenas modificables.-

Una cadena es modificable por medio de operaciones que pueden cambiar su contenido.

Por ejemplo la cadena "Maria" podría cambiarse por "Mario"

Racket provee una función que puede modificar una cadena : `string-set!`

Formato

`(string-set! cadena n caracter)` en donde *cadena* es la cadena a modificar, *n* es el índice del carácter a modificar y *caracter* es el carácter por el cual será reemplazado.

Ejemplo

```
(string-set! (string #\e #\j #\e) 2 #\a)
```

Para obtener caracteres de una cadena se usa `(string-ref cadena n)`.

Toma a “cadena” y devuelve el carácter que se encuentra en el índice “n”.

Ejemplo

```
(string-ref "América" 0); -> #\A, retorna el caracter de la cadena en la posición 0.
```

(substring cadena comienzo final)

Donde comienzo y final son números, y retorna una subcadena que empieza en el carácter de la posición comienzo y terminado en el carácter de la posición final restándole 1 .

Ejemplo

```
(substring "futbol" 1 3); -> "ut", Empieza en el caracter 1 y termina en el caracter 2
```

(string-set! Cadena n carácter)

Almacena el carácter en el elemento n de la cadena. En otras palabras reemplaza el carácter de cadena cuyo índice es n con carácter.

Ejemplo

```
(define str (string #\f #\u #\t #\b #\o #\l)) ; "futbol"  
(string-set! str 2 #\d); reemplaza d en el caracter no. 2 de cadena
```

Operaciones de Comparación en Cadenas.-

Formato

```
(string=? cadena1 cadena2) ; Es cadena1 igual que cadena2?  
(string<? cadena1 cadena2) ; Es cadena1 lexicográficamente menor que cadena2?  
(string>? cadena1 cadena2) ; Es cadena1 lexicográficamente mayor que cadena2?  
(string<=? cadena1 cadena2) ; Es cadena1 lexicográficamente menor o igual cadena2?  
(string>=? cadena1 cadena2) ; Es cadena1 lexicográficamente mayor o igual cadena2?  
(string-ci=? cadena1 cadena2) ; Igual que string=? Pero insensible  
(string-ci<? cadena1 cadena2) ; Igual que string<? Pero insensible  
(string-ci>? cadena1 cadena2) ; Igual que string>? Pero insensible  
(string-ci<=? cadena1 cadena2) ; Igual que string<=? Pero insensible  
(string-ci>=? cadena1 cadena2) ; Igual que string>=? Pero insensible
```

Ejemplos

```
(string=? "hola" "HOLA")           ;-> #f Se consideran como cadenas diferentes
(string-ci=? "hola" "HOLA")         ;-> #t El procedimiento es insensible
```

El lenguaje de programación Racket también proporciona una variedad de funciones o predicados para determinar la el orden de dos cadenas.

Si dos cadenas son de diferente longitud y una cadena es prefijo de la otra cadena, la cadena más corta es considerada lexicográficamente menor que la otra. Ejemplo: la cadena "carro" es prefijo de la cadena "carrotanque" por lo tanto la cadena "carro" es considerada menor que la otra cadena.

La función (**string-set! cadena n caracter**) almacena el *caracter* en el elemento *n* de la *cadena*. En otras palabras reemplaza el carácter de *cadena* cuyo índice es *n* con *carácter*.

Ejemplos:

```
(define str (string #\f #\u #\t #\b #\o #\l)) ; "futbol"
(string-set! str 2 #\d); reemplaza d en el caracter no. 2 de cadena
```

La función (**substring cadena comienzo final**), en donde *comienzo* y *final* son números, y retorna una subcadena que empieza en el carácter de la posición *comienzo* y terminando en el

Para obtener caracteres de una cadena se usa el procedimiento (**string-ref cadena n**), que toma a "cadena" y devuelve el carácter que se encuentre en el índice "n". Recordar que el primer carácter tiene el índice 0.

Cadenas modificables:

Una cadena es modificable por medio de operaciones que pueden cambiar su contenido. Ejemplo: la cadena "Maria" podría cambiarse por "Mario".

Los procedimientos (**string ch1 ch2 ...**), (**make-string n ch**) y (**string-append cadena1 cadena2...**) pueden usarse para crear cadenas modificables pero una cadena creada mediante comillas dobles no es modificable.

Racket provee una función que puede modificar una cadena: **string-set!**

Sintaxis:

(**string-set! cadena n caracter**) en donde *cadena* es la cadena a modificar, *n* es el índice del carácter a modificar y *caracter* es el carácter por el cual será reemplazado.

Ejemplo:

```
(string-set! (string #\e #\j #\e) 2 #\a)
```

Ejemplo 1

Escribir un programa que permita generar códigos de usuario por el procedimiento siguiente:

Tiene que leer el nombre y los dos apellidos de una persona y devolver un código de usuario formado por las tres primeras letras del primer apellido, las tres primeras letras del segundo apellido y las tres primeras letras del nombre , ejemplo ALEX RUIZ SANCHEZ , debe devolver RUISANALE.

```
(define (Codigo nom ap1 ap2)
  (display (substring nom 0 3))
  (display (substring ap1 0 3))
  (display (substring ap2 0 3))
  )
(Codigo (read) (read) (read))
```

Ejemplo 2

Construir un programa que reciba una cadena y devuelva una cadena equivalente pero sin las vocales.

```
(define (Cadena read) ;Lee y almacena nuestra Cadena.
  (Read)
)
(define (Tamaño read) ;Define y almacena el tamaño de nuestra Cadena.
  (string-length (Cadena read))
)
(define (Sinvocls CAD LENGTH POS)
  ;Devuelve Cadena sin vocales, CAD referencia a la cadena para
```

```
evaluar, LENGTH a su tamaño
  (if (= LENGTH POS)
    (display "")
    (begin
      (if (or (char-ci=? (string-ref (Cadena CAD) POS) #\a)
              ;Evalúa si el carácter de Cadena en la posición POS es igual a una de las vocales.
              (char-ci=? (string-ref (Cadena CAD) POS) #\e)
              (char-ci=? (string-ref (Cadena CAD) POS) #\i)
              (char-ci=? (string-ref (Cadena CAD) POS) #\o)
              (char-ci=? (string-ref (Cadena CAD) POS) #\u))
          (display "") ;De cumplirse, devolver vacío.
          (display (string-ref (Cadena CAD) POS))
        )
      (Sinvocls CAD LENGTH (+ POS 1)) ;Llamado recursivo.
    )
  )
)
(define (Principal usrdef) ;Esta función simplifica el llamado.
  (Sinvocls (Cadena usrdef) (Tamaño usrdef) 0)
)
(Principal (read))
```

Ejemplo de Cadena

Construir un programa que reciba una cadena y devuelva una cadena equivalente pero sin las vocales.

```
(define (Cadena read)
```

```
  read ; el error es (read)
```

```
)
```

```
(define (tamaño read)
```

```
  (string-length (Cadena read))
```

```
)
```

```
(define (sinvocls CAD LENGTH POS)
```

```
  (if (= LENGTH POS)
```

```
    (display ""))
```

```
  (begin
```

```
    (if(or (char-ci=? (string-ref(Cadena CAD) POS) #\a)
```

```
        (char-ci=? (string-ref(Cadena CAD) POS) #\e)
```

```
        (char-ci=? (string-ref(Cadena CAD) POS) #\i)
```

```
        (char-ci=? (string-ref(Cadena CAD) POS) #\o)
```

```
        (char-ci=? (string-ref(Cadena CAD) POS) #\u));cond
```

```
    (display ""));verdad
```

```
    (display (string-ref (Cadena CAD) POS));falso
```

```
  );if OR
```

```
  (sinvocls CAD LENGTH(+ POS 1))
```

```
  );begin
```

```
);if LENGTH
```

```
);funcion
```

```
(define (Principal usrdef)
```

```
  (sinvocls (Cadena usrdef) (tamaño usrdef) 0)
```

```
)
```

Ejercicio:

1. Construir una función que reciba una cadena y la devuelva invertida.
2. Construir una función que reciba una cadena y devuelva cuantas vocales tiene

VECTORES

EL tipos de datos vector, es conocido como un tipo de datos compuesto de uno o más tipos de datos básico o primitivo o de otros tipos de datos compuestos, inclusive vectores mismos. Los elementos de un vector pueden ser a su vez vectores, estamos en el caso de vectores multidimensionales.

En Racket un vector es un procedimiento que nos permite almacenar en si mismo determinada cantidad de datos (como enteros, cadenas, caracteres, vector, etc.) establecida por el usuario. Además otras funciones o procedimientos pueden hacer referencia a un elemento del vector mediante índices, siendo el primer elemento del vector de índice 0 y el ultimo de índice n.

Ejemplo: Vector unidimensional de 10 elementos

0	1	2	3	4	5	6	7	8	9

Formato

(vector Elem1 Elem2 ElemM)

Ejemplo.-

Este vector tienen cinco elementos, el primero de ellos es la función (+ 2 2), el segundo es el número 1, el tercero es el carácter q, el cuarto es el símbolo Woow “¡Hola!”.

```
{vector (+ 2 2) 1 #\q 'Woow "¡Hola!" }
```

Creación, Modificación y Manipulación de Vectores.-

make-vector es una función para crear un vector con uno o dos parámetros, el primero siempre debe ser un valor numérico que indica la cantidad de posiciones y el segundo(opcional) es un tipo de dato cualquiera para llenar todas las posiciones del vector.

Formato

(make-vector Num TipDato)

Ejemplo

```
(make-vector 6 #\w);-> #(#\w #\w #\w #\w #\w #\w)
(make-vector 5)      ;-> #(0 0 0 0 0)
```

Vector, crea un vector con una cantidad de posiciones como parámetros. Tiene n parámetros y puede ser de cualquier tipo.

Ejemplo:

```
(vector "X" #\a 'b 2 3 1 1 (+ 2 1));->("X" #\a b 2 3 1 1 3)
```

vector-ref, devuelve el dato ubicado en la posición referenciada del vector. Tienen dos parámetros: el primero es el vector o el nombre de la función donde esta definido, el segundo es el índice del vector donde se encuentra el dato que queremos obtener.

Ejemplo

```
(define vect (vector 12 "X" "Hola" 23))
(vector-ref vect 2) ;-> "Hola"
(vector-ref (vector "x" 1 23) 2) ;-> 23
```

vector-set!, es un procedimiento por el cual se toma un elemento de un vector y se modifica cambiando por otro. Tienen tres argumentos: el primero es el vector o nombre de la función donde esta definido, el segundo es el índice del dato a modificar y el tercero es el nuevo dato para esa posición.

Ejemplo

```
;Se define una constante "vec2" para guardar el vector:
(define vec2 (vector 1 2 3))
;Ingresa 876 en la posición 0 del vector "vec2":
(vector-set! vec2 0 876)
vec2                      ;-> #(876 2 3)
```

vector->list, convierte un vector en una lista, tiene un solo parámetro el vector a convertir.

Ejemplo

```
(define vec (vector 12 "1" "Hola" "amigos"))
(vector->listvec) ;-> (12 "1" "Hola" "amigos")
```


vector-fill!, permite ingresar a un dato dado en todas las posiciones del vector. Tienen dos parametros, el nombre del vector y el dato a introducir en las posiciones de este.

Ejemplo:

```
(define vec (vector 12 "1" "Hola" "amigos") )  
(vector-fill! vec "hola")          ;-> #("hola" "hola" "hola"  
"hola")  
(vector-fill! vec 56)              ;-> #(56 56 56 56)
```

vector-length, esta funcion cuenta la cantidad de posiciones de un vector. Tiene solo un argumento que es el vector a evaluar.

Ejemplo

```
(define vec (vector 12 "1" "Hola" "amigos") )  
(vector-length vec) ;-> 4
```

Ejemplo:

Dado un vector con diferentes valores imprimir dichos valores leyéndolos uno por uno de manera recursiva:

Análisis:

Debemos definir un vector en una función y llenar éste con determinado número de datos. Luego, usando otra función debemos leer la primera posición del vector (el número que indique esta posición debe ser un argumento en la función) y mostrarla por pantalla. En el llamado recursivo debemos aumentar en uno la posición a leer del vector hasta haber leído y visualizado todas las posiciones del vector, pero antes se debe conocer el número de posiciones en el vector; de otra forma no podríamos finalizar correctamente la recursión (este número también debe ser un argumento).

```

(define Vect (vector "Esto" " " "Es" " " "Un" " " "Vector" " "
"Leído" " " "Recursivamente" ".") )
(define (DisplayVectLength Pos)
  (if (= (- Length 1) Pos)
;El tamaño del vector es 12, pero su última posición es 11
(porque empiezan a contar en 0) por lo tanto Length se debe
restar en 1
    (display(vector-refVect Pos))
    (begin
      (display (vector-refVect Pos))
      (display VectLength (+ 1 Pos))
    )
  )
)
(DisplayVect (vector-lengthVect) 0)

```

Ejercicios

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

Dado un numero 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).

Llenar un vector V de 10 elementos con cuadrados de los 10 primeros elementos. Ejemplo: (vector 1 4 9 16 25 36 49 64 81 100).

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

PARES y LISTAS

PARES.-

Un par es una estructura de datos con dos campos llamados cabeza y cola. Los pares son creados con el procedimiento cons. Se puede ingresar cualquier tipo de datos a los pares.

Funciones Asociadas.-

Cons.- Esta función permite crear un par, tiene dos argumentos. El primero: la cabeza y el segundo: la cola

Formato

```
(cons a b) ;-> (a . b) La cabeza es a y la cola es b.
```

Car.- Extrae la cabeza de un par. Su argumento es el par a evaluar, ejemplo

```
(define par (cons 1 "hola"))  
(display (car par)) ;-> 1
```

Cdr.- Extrae la cola de un par. Su argumento es el par a evaluar, ejemplo

```
(define par (cons 1 "hola"))  
(display (cdr par)) ;-> "hola"
```

Una de las diferencias del par y lista porque la cola puede contener otros pares (pares anidados), ejemplo

```
(define par (cons 1 (cons 2 (cons 3 (cons 4 5)))))  
(display (cdr par)) ;-> (2 3 4 . 5)  
(display (car par)) ;-> 1
```

Otra de las diferencias entre listas y pares es que los pares admiten dos elementos, mientras que las listas admiten más de 2 elementos y terminan en un elemento vacío (empty), los pares no. Se puede decir que una lista es un par pero un par no es una lista.

Pair?.- Procedimiento que determina si un objeto es par o no. Su argumento es el objeto a evaluar, ejemplo

```
(pair? (cons 1 "hola")) ;-> #t
```

LISTAS.-

Es un tipo de dato en Racket dinámico que puede añadir información, en una lista podemos incluir un número de datos y estos pueden ser de cualquier tipo. Una lista siempre termina en un espacio vacío (empty), aunque este no sea visible.

Funciones Asociadas a la Lista.-

List.- Crea una lista, tiene tantos argumentos como elementos queramos incluir en nuestra lista, ejemplo.

```
{list 5 4 6 "Hola" 'bl #\a (vector 4 5 6 1) (list 2 3 4) (+ 2 2)}  
;-> (5 4 6 "Hola" 'bl #\a #(4 5 6 1) (2 3 4) 4)
```

Car,Cdr.- Tienen el mismo uso que en los pares, sirven para extraer la cabeza y la cola respectivamente de una lista.

```
{car (list 5 4 6)};-> 5  
{cdr (list 5 4 6)};-> (4 6)
```

Null?.- Es el procedimiento que determina si una lista está vacía. Su argumento es la lista a evaluar, devuelve un valor booleano.

```
{define Lista (list 5 4 6)}  
{null? Lista} ;-> #f  
  
{define Lista (list)}
```

```
{null? Lista} ;-> #t
```

Append.- Es el procedimiento que nos permite añadir un elemento a la lista, ejemplo

```
{define (AdicionarNumeros Lista n)  
  (if (<= n 10)  
      (AdicionarNumeros (append Lista (list n)) (+ n 1))  
      Lista)  
}  
{AdicionarNumeros (list) 1} ; -> (1 2 3 4 5 6 7 8 9 10)  
  
{define Lista (list 1 2 3)}  
{append Lista 4} ; -> (1 2 3 . 4)  
  
{define Lista (list 1 2 3)}  
{append Lista (list 4)} ; -> (1 2 3 4)}
```

List?.- este procedimiento determina si un elemento es una lista. Su argumento es la lista a evaluar.

```
(define Lista (list 1 2 3))  
(list? Lista) ; -> #t
```

Length.- Es el procedimiento que determina el tamaño de una lista. Su argumento es la lista a evaluar, ejemplo:

```
(define Lista (list 1 2 3))  
(length Lista) ; -> 3
```

Reverse.- Permite invertir una lista. Su argumento es la lista a invertir, ejemplo :

```
(define Lista (list 1 2 3))  
(reverse Lista) ; -> (3 2 1)
```

List-tail.- Es una función que devuelve una subcola de una lista, tiene dos argumentos el primero es la lista y el segundo es el índice donde comienza la cola. Los índices en la lista empiezan en cero y terminan en un elemento vacío, ejemplo:

```
(define Lista (list "Lunes" "Martes" "Miércoles" "Jueves"  
"Viernes"))  
(list-tail Lista 2) ;-> ("Miércoles" "Jueves" "Viernes")
```

Ejemplo.- Construir una función que reciba una lista e imprima uno por uno todos los valores. Usar la función null? para saber si llego al final de una lista, ejemplo:

La única forma de recorrer una lista o un par es usando la función car y cdr , es por eso que en el llamado recursivo de la función es argumento Lista es remplazado por la cola de la misma, para que no se tome el primer elemento, pues este ya se visualizó, y quede en la cabeza el siguiente elemento.

```
(define (Printlist Lista)  
  (if (not (null? Lista))  
      (begin  
        (display (car Lista))  
        (newline)  
        (Printlist (cdr Lista))  
      )  
      )  
  )  
(Printlist (list 5 4 3 2 0 12 93))
```

Ejemplo.- Construir una función que devuelva una lista con los datos que el usuario digite por teclado. La entrada de datos termina cuando el usuario digite el número -1.

Análisis: Se debe tomar en cuenta el uso de equal? Pues este es necesario para poder introducir cualquier tipo de dato a la lista, en el llamado recursivo a la función remplazar leer con (read) es de vital importancia, pues de no hacerlo leer siempre tomara el valor que ingresamos en el primer read sin importar cuantos o cuales mas ingresemos.

```
(define (Agregar lista leer)
  (if (equal? leer -1)
```

```
    (display lista)
    (Agregar (append lista (list leer)) (read))
  )
)
(Agregar (list) (read))
```

Bibliografía.-

Muños Guerrero Luis Eduardo (2014). “Programación funcional con Raquet” Universidad Tecnológica de Pereira.

Navas Eduardo (2010).Programando en Rackete 5. Universidad Centro Americana José Simeón Cañas.

Fokker Jeroen(1996). “Programación Funcional”. Universidad de Utrecht Departamento de Informática.

Rodríguez Rojas Oldemar . “Programación Funcional Lisp-DrScheme”. Escuela de Informática Universidad de Nacional.