**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

1. **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

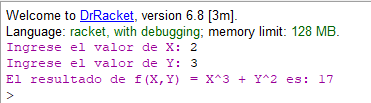
(display "Ingrese el valor de Y: ")

(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))



1. **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

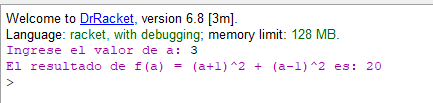
(display "Ingrese el valor de a: ")

(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))



1. **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

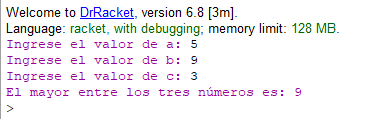
(display "Ingrese el valor de c: ")

(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))



1. **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

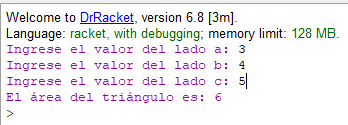
(display "Ingrese el valor del lado c: ")

(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))



1. **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: ")

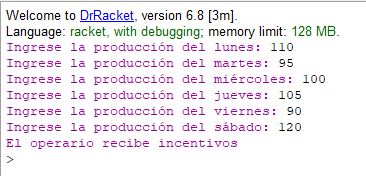
(define viernes (read)) ; Leer la producción del viernes

(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)



**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)")]))

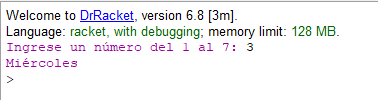
; Solicitar el número al usuario

(display "Ingrese un número del 1 al 7: ")

(define num (read)) ; Leer el número ingresado

; Mostrar el día correspondiente

(dia-semana num)



**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

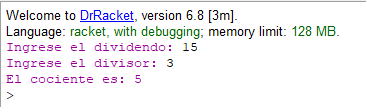
(if (= divisor 0)

(displayln "Error: División por cero no está permitida")

(begin

(display "El cociente es: ")

(displayln (division-resta dividendo divisor))))



**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

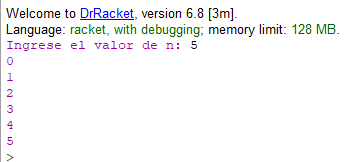
; Solicitar el valor de n al usuario

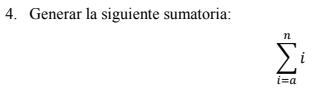
(display "Ingrese el valor de n: ")

(define n (read)) ; Leer el valor de n

; Generar e imprimir los números naturales hasta n

(naturales-hasta n)





#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

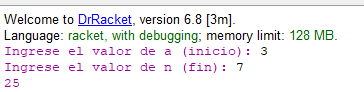
; Calcular y mostrar el resultado de la sumatoria

(if (> a n)

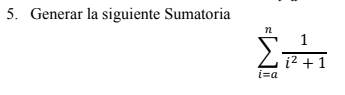
(displayln "El valor de a debe ser menor o igual a n")

(let ([resultado (sumatoria a n)]) ; Calcular el resultado

(displayln resultado))) ; Mostrar el resultado



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



#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): ")

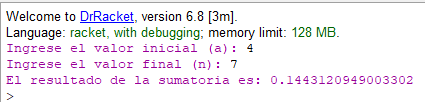
(define n (read))

; 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



**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

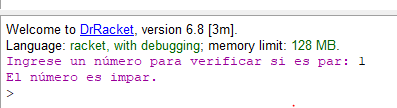
(display "El número es impar."))) ; Imprime que el número es impar

; 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



1. **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

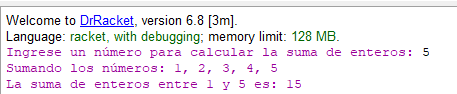
(printf "La suma de enteros entre 1 y ~a es: ~a\n" n resultado)) ; Imprime el resultado

; 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



1. **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

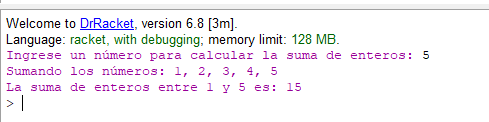
(printf "La suma de enteros entre 1 y ~a es: ~a\n" n resultado)) ; Imprime el resultado

; 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



1. **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")]

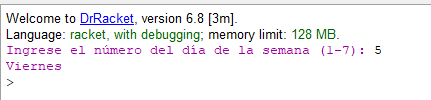
[else (display "Número de día inválido")]))

; 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



**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"

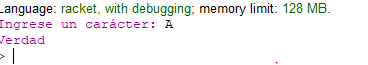
"Falso")) ; Si no es un carácter, devuelve "Falso"

; 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



1. **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

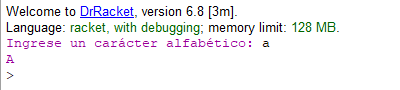
"El parámetro no es un carácter.")) ; Si no es un carácter

; 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



1. **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)])

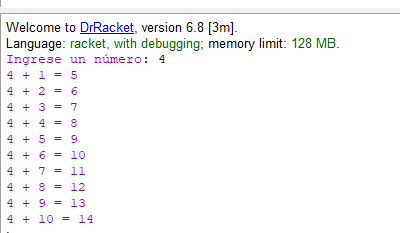
(displayln (string-append (number->string numero) " + " (number->string i) " = " (number->string (+ numero i))))))

;; Ejemplo de uso

(display "Ingrese un número: ")

(define numero (read))

(tabla-sumas numero)



1. **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))))

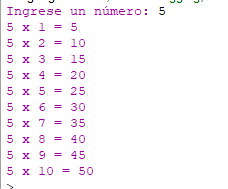
(define (main)

(display "Ingrese un número: ")

(define num (read))

(tabla-multiplicar num))

(main)



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 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

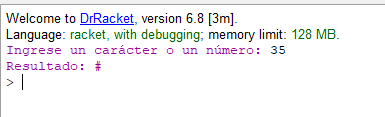
(display "Ingrese un carácter o un número: ")

(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)

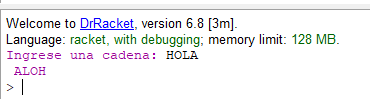
(list->string (reverse (string->list cadena))))

;; Ejemplo de uso

(display "Ingrese una cadena: ")

(define cadena (read-line))

(displayln (invertir-cadena cadena))



1. **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))

