**Universidad Centroamericana José Simeón Cañas**
**Departamento de Electrónica e Informática**
**Facultad de Ingeniería y Arquitectura**
**Programación Declarativa** **Guía de Ejercicios No. 2**
**Ciclo 02/2024** **Racket**

1. Colocar en pantalla una secuencia de cadenas de caracteres como si se tratara de una sola (No use string-append).
2. Calcular la longitud de una lista (No use length).
3. Encontrar el índice de un elemento en una lista usando sin usar `list-index`.
4. Ordenar una lista de números en orden ascendente con y sin la función `sort`.
5. Convertir un número a cadena y concatenarlo con un texto.
6. Sin usar `remove`, elimine un elemento específico de una lista.
7. Generar una lista de los primeros números pares.
8. Obtener los primeros 5 elementos de una lista.
9. Sin usar `substring`, extraer una parte específica de una cadena.
10. Crear una lista de listas anidadas que represente una matriz, dadas sus dimensiones.
11. Usar `random` para generar una secuencia de números aleatorios entre 1 y
12. Leer tres números del usuario y calcular su promedio.
13. Leer un número y determinar si es primo.
14. Escribir un programa que lea cadenas hasta que el usuario escriba "FIN".
15. Escribir las primeras $n$ potencias de un número dado por el usuario.
16. Leer una lista de cadenas y mostrar las únicas que comienzan con una vocal.
17. Leer una matriz y calcular su transpuesta.
18. Leer un número entero del usuario e imprimir su representación binaria.
19. Pedir al usuario un número y mostrar la tabla de multiplicar de ese número.
20. Leer dos listas de números y guardar su suma elemento a elemento.
21. Usar `let` para calcular el área de un círculo dado su radio.
22. Usar `let*` para calcular el volumen de un cilindro.
23. Implementar una función que devuelva el máximo de tres números usando `let`.
24. Usar `let` para dividir una cadena en dos partes según su longitud.
25. Usar `let-values` para intercambiar las coordenadas de un par ordenado.
26. Calcular el interés compuesto dado el capital, tasa y tiempo con `let*`.
27. Usar `let` para calcular el promedio ponderado de tres números.
28. Implementar una función que devuelva una lista con todos los valores $x$ y $y$, tal que $x + y = n$.
29. Usar `let-values` para separar una cadena en su primera palabra y el resto.
30. Implementar un programa que calcule la energía cinética con `let`.
31. Usar `let*` para calcular la suma de una serie como $1 + 2 + 3 + \ldots + n$.
32. Calcular el tiempo promedio de tres eventos usando `let-values`.
33. Usar `let` para calcular el área de un triángulo con la fórmula de Herón.

34. Implementar un programa que transforme coordenadas cartesianas a polares usando `let`.
35. Usar `let` para calcular el punto medio entre dos puntos.
36. Crear un indicador mutable que almacene una lista y agregarle elementos usando `set!`.
37. Usar mutación para implementar un contador decreciente.
38. Modificar un indicador para almacenar el producto acumulado de una serie de números.
39. Implementar un programa que use mutación para alternar entre dos valores predeterminados.
40. Crear un indicador mutable que guarde la longitud de una lista dinámica.
41. Implementar un programa que simule un reloj aumentando la hora con `set!`.
42. Usar mutación para actualizar los valores de una lista al cuadrado.
43. Escribir una función que utilice mutación para intercambiar los valores de dos indicadores.
44. Usar mutación para implementar un historial de valores en un indicador.
45. Implementar un programa que use mutación para encontrar el número más grande de una lista.
46. Crear un acumulador mutable que sume los valores ingresados por el usuario.
47. Implementar un programa que almacene las palabras leídas y las ordene al final.
48. Implementar un sistema mutable que cuente las veces que una función es llamada.
49. Determinar si un número es divisible entre 3 y 5 usando `if`.
50. Usar `cond` para determinar el rango de edad de una persona (niño, adolescente, adulto).
51. Implementar un programa que indique si una cadena es vacía usando `if`.
52. Usar `cond` para identificar la categoría de un triángulo según sus lados.
53. Escribir un programa que verifique si un número está dentro de un rango dado.
54. Usar `cond` para determinar si un carácter es una vocal, consonante o no alfabético.
55. Usar `if` para verificar si una lista está ordenada en forma ascendente.
56. Implementar un programa que determine el tipo de un número (entero, real, racional).
57. Usar `cond` para clasificar un número como negativo, cero o positivo.
58. Implementar una función que devuelva el mayor divisor común usando `if`.
59. Usar `if` para verificar si una lista contiene elementos duplicados.
60. Implementar una función que use `cond` para calcular el salario con horas extras.
61. Escribir un programa que determine si un número es perfecto usando `if`.
62. Usar `cond` para identificar la estación del año según un mes dado.
63. Calcular el número de elementos negativos en una lista recursivamente.
64. Implementar una función recursiva que devuelva una lista de todos los divisores de un número.
65. Determinar recursivamente si una lista está ordenada.
66. Escribir una función que sume las diagonales de una matriz representada como lista de listas.
67. Implementar una función recursiva que devuelva el último elemento de una lista.
68. Crear una función recursiva que elimine todos los elementos iguales a un valor dado.

69. Implementar una función que genere todas las permutaciones de una lista recursivamente.
70. Escribir una función recursiva que devuelva el número de sublistas en una lista.
71. Implementar una función recursiva que encuentre el subarreglo más largo en una lista de listas.
72. Crear una función que recursivamente divida una lista en dos mitades.
73. Implementar un programa que recorra dos listas en paralelo recursivamente.
74. Crear una función recursiva que encuentre el menor número en una lista de listas.
75. Implementar una función recursiva que devuelva una lista con las posiciones de un valor dado en una lista.
76. Escribir una función recursiva que reemplace todos los ceros en una lista con unos.
77. Crear una función que cuente recursivamente los elementos mayores a n en una lista.
78. Implementar una función recursiva para calcular el enésimo término de la sucesión de Fibonacci.
79. Escribir una función recursiva que calcule la suma de los dígitos de un número.
80. Implementar una función que recursivamente determine si un número es primo.
81. Crear una función recursiva que convierta un número decimal en binario.
82. Implementar una función recursiva que devuelva el mínimo común múltiplo de dos números.
83. Escribir una función recursiva que elimine los elementos pares de una lista.
84. Implementar una función recursiva que genere todos los subconjuntos de una lista.
85. Crear una función recursiva que calcule la potencia de un número a^b .
86. Implementar una función recursiva que cuente los caracteres en una cadena (sin usar string-length)
87. Escribir una función recursiva que determine si una cadena tiene todos los caracteres únicos.
88. Implementar una función recursiva que revierta una cadena sin usar funciones predefinidas.
89. Crear una función recursiva que divida un número en sus factores primos.
90. Implementar una función recursiva que encuentre el menor elemento en un árbol binario.
91. Crear una función recursiva que simule una pila (stack) para almacenar y recuperar datos.
92. Usar `map` para sumar 1 a cada elemento de una lista.
93. Implementar una función que devuelva las cadenas de más de 5 caracteres usando `filter`.
94. Usar `map` para convertir una lista de números en sus cuadrados.
95. Implementar una función que elimine los números impares de una lista usando `filter`.
96. Usar `map` para convertir una lista de cadenas a mayúsculas.
97. Implementar una función que calcule el módulo de cada elemento en una lista con un divisor dado usando `map`.
98. Usar `filter` para obtener todas las palabras que empiezan con una vocal en una lista de cadenas.

99. Implementar una función que utilice `map` para convertir una lista de temperaturas en grados Fahrenheit a Celsius.
100. Usar `map` para generar una lista de pares (x, y) donde y = 2x.
101. Implementar una función que elimine todas las cadenas vacías de una lista usando `filter`.
102. Usar `map` para agregar un prefijo a cada cadena en una lista.
103. Implementar una función que filtre los números primos de una lista usando `filter`.
104. Usar `filter` para extraer todos los números negativos de una lista.
105. Implementar una función que multiplique dos listas elemento a elemento usando `map`.
106. Usar `map` para calcular el logaritmo base de una lista de números.
107. Contar cuántas vocales tiene una cadena.
108. Convertir todas las vocales de una cadena a mayúsculas.
109. Crear una función que elimine los espacios de una cadena.
110. Revertir una cadena sin usar funciones predefinidas.
111. Contar cuántas veces aparece una palabra en una cadena.
112. Dividir una cadena en palabras y ordenarlas alfabéticamente.
113. Reemplazar todas las apariciones de una palabra en una cadena por otra.
114. Crear una función que devuelva una subcadena hasta un carácter específico.
115. Determinar si una cadena es un palíndromo.
116. Contar el número de palabras en una cadena.
117. Crear una función que devuelva las palabras que comienzan con un carácter dado.
118. Implementar una función que elimine los números de una cadena.
119. Crear una función que convierta una cadena en formato "snake_case".
120. Determinar si dos cadenas son anagramas.
121. Usar `foldl` para calcular el producto de una lista de números.
122. Implementar una función que concatene una lista de cadenas usando `foldr`.
123. Usar `foldl` para calcular el máximo de una lista de números.
124. Implementar una función que cuente los elementos en una lista usando `foldr`.
125. Usar `foldl` para calcular la suma acumulada de una lista.
126. Implementar una función que devuelva una lista invertida usando `foldr`.
127. Usar `foldr` para determinar si todos los elementos de una lista son pares.
128. Implementar una función que combine dos listas en una lista de pares usando `foldl`.
129. Usar `foldl` para calcular el promedio de una lista de números.
130. Implementar una función que elimine duplicados de una lista usando `foldr`.
131. Crear una función que combine varias cadenas con un separador usando `foldl`.
132. Usar `foldr` para contar cuántos elementos en una lista cumplen una condición.
133. Implementar una función que sume los elementos impares de una lista usando `foldl`.
134. Crear una función que genere una lista de pares acumulados usando `foldr`.
135. Usar `foldl` para implementar una versión personalizada de `map`.
136. Implementar una función que calcule la profundidad de un árbol binario.
137. Crear una función que sume los valores de todos los nodos en un árbol.
138. Implementar una función que encuentre el nodo con el valor más grande en un árbol.

139. Crear una función que determine si un valor está presente en un árbol binario.
140. Implementar una función que cuente los nodos hoja de un árbol binario.
141. Crear una función que invierta un árbol binario.
142. Implementar una función que convierta un árbol binario en una lista en orden in-orden.
143. Crear una función que verifique si dos árboles binarios son iguales.
144. Implementar una función que elimine todos los nodos con un valor menor a un umbral dado.
145. Crear una función que encuentre el nivel más profundo de un árbol binario.
146. Implementar una función que cuente los nodos en cada nivel de un árbol.
147. Crear una función que construya un árbol binario a partir de una lista en orden.
148. Implementar una función que convierta un árbol binario en su representación de cadena.
149. Crear una función que recorra un árbol binario en orden post-orden.
150. Implementar una función que determine si un árbol es un árbol binario de búsqueda.