

# Ejercicios tomados en exámenes finales

## Programación I

(08.07.2020)

Con el objetivo de mostrar el tipo de ejercicios que son tomados en exámenes finales, este documento agrupa algunos de aquellos utilizados en mesas de examen de años anteriores. **Este documento NO es un examen final.**

### 1 Evaluando Expresiones

*Estos ejercicios son para entregar en papel.*

**Ejercicio 1.** Dada la función:

```
(define (fun x y z) (cond [(> z (expt x y)) (* x y z)]
                          [(< (expt x z) y) (* x y z)]
                          [else x]))
```

Muestre cómo *DrRacket* evalúa  $(\text{fun } 2 \ 1 \ 4)$  y  $(\text{fun } 6 \ 1 \ 4)$ . Recuerde incluir todos los pasos involucrados en el cálculo.

**Ejercicio 2.** Se pide, para cada frase, escribir una expresión que la represente y calcular su valor de verdad:

- El área de un rectángulo de lados 5 y 25 es 60
- El sexto caracter de la cadena que se obtiene al concatenar "aBC", "dEf", "ghi" es "F"
- El cuarto elemento de la lista  $(\text{list } \#t \ 5 \ \text{"hola"} \ \#f \ \text{"mundo"})$  es un string.

**Ejercicio 3.** Dada la función:

```
(define (f n m b) (cond [(< n (* m 2)) (* n m)]
                        [(and (< n m) b) (* n m 3)]
                        [else (* 2 n)]))
```

- Dar la signatura de  $f$ .
- Muestre cómo *DrRacket* evalúa  $(\text{fun } 4 \ 2 \ \#t)$  y  $(\text{fun } 5 \ 6 \ \#f)$ . Recuerde incluir todos los pasos involucrados en el cálculo.

**Ejercicio 4.** Considere las siguientes definiciones:

```
(define (f a b c d)
  (+ (- 3 a) (if c 1 (string-length b)) (image-width d)))

(define (g x y z)
  (if (> x (string-length y))
      (if (> 3 z) (+ x 24) 13)
      (if (= (string-length y) 4) z "Error")))
```

- Determine la signatura de cada función.
- Escriba un caso de test (con la notación *check-expect*) para cada una.
- Elija cuatro valores  $(a, b, c, d)$  compatibles con la signatura de  $f$  y evalúe paso a paso (con lápiz y papel) la expresión  $(f \ a \ b \ c \ d)$ .  
**Aclaración:** Al decir *compatibles* queremos indicar que esos valores pertenecen al dominio de la función que determinó en el ítem 1.
- Defina una función  $g2$  que no use expresiones *if*, y que se comporte exactamente igual que  $g$  en todo su dominio. No hace falta dar el diseño, alcanza sólo con la definición.

### 2 Programando con listas y/o estructuras

Para resolver estos ejercicios **NO UTILICE map, filter ni fold**.

**Ejercicio 5.** Diseñe una función *replicar* que tome una lista y un número entero  $n$  y replique cada elemento de la lista  $n$  veces.

*Ejemplo:*

```
| (replicar (list 4 6 3 9) 3) = (list 4 4 4 6 6 6 3 3 3 9 9 9)
```

**Ejercicio 6.** En este ejercicio programaremos un gestor de pedidos de almuerzos de una oficina.

Cada pedido individual consta del nombre de la persona que lo pide, el plato y el postre que ha elegido. Los menús constan siempre de tres opciones y los postres de dos opciones.

Representaremos un pedido individual con una estructura que conste de:

- Un string que represente el nombre.
- Un número de 1 a 3 que represente la opción de menú elegida.
- Un número entre 1 y 2 que represente la opción de postre elegida.

Con el objetivo de simplificar el trabajo al restaurante que toma los pedidos, se requiere procesar un lista de pedidos individuales y convertirlo en un único gran pedido final. Representaremos un pedido final como una estructura con los siguientes campos:

- La cantidad de menús 1.
- La cantidad de menús 2.
- La cantidad de menús 3.
- La cantidad de postres 1.
- La cantidad de postres 2.

Defina estructuras *pedido* y *pedido-final*. Diseñe una función *armar-pedido-final* que dada una lista de pedidos individuales *pedido* construya un *pedido-final*.

**Ejercicio 7.** Diseñe la función *sublistas-vacias?* que dada una lista compuesta por listas devuelva *true* si todas las sublistas están vacías o *false* en caso contrario. Por ejemplo:

```
| (check-expect (sublistas-vacias? (list '() '() '())) #true)
| (check-expect (sublistas-vacias? (list '() (list 5 7))) #false)
```

**Ejercicio 8.** En este ejercicio representaremos un trabajador con una estructura con los siguientes campos:

- 1er campo: Apellido
- 2do campo: Estado Civil
- 3er campo: Cantidad de hijos
- 4to campo: Sueldo Bruto

Teniendo en cuenta esto se pide:

- Diseñe una estructura *trabajador* que contenga los campos descriptos más arriba.
- Diseñe una función *impuesto* que tome como entrada un valor de tipo *trabajador* y calcule el impuesto a pagar. En caso que no reciba como entrada una estructura de tipo *trabajador* deberá mostrar el siguiente mensaje de error: "Tipo de dato inválido".

El impuesto a pagar por un trabajador es del 5% de su sueldo bruto. Sin embargo, hay algunos trabajadores que están exentos del pago, y son los casos que se detallan a continuación:

- Todo trabajador soltero que cobre un sueldo menor o igual a \$15.000 estará exento del pago del impuesto.
- Todo trabajador casado que cobre un sueldo menor o igual a \$18.000 estará exento del pago del impuesto.
- Todo trabajador que posea un hijo verá incrementado el mínimo no imponible en \$1.000, o sea, no pagará impuestos hasta un sueldo bruto de \$16.000 en el caso de ser soltero o de \$19.000 en el caso de ser casado
- Todo trabajador que posea dos hijos verá incrementado el mínimo no imponible en \$2.000, o sea, no pagará impuestos hasta un sueldo bruto de \$17.000 en el caso de ser soltero o de \$20.000 en el caso de ser casado
- Todo trabajador que posea tres hijos o más quedará exento de pagar impuesto, sin importar su estado civil.
- En caso de cobrarse el impuesto, este será el 5% del sueldo bruto del trabajador

**Ejemplo 1:** El impuesto a pagar por un trabajador casado, con 4 hijos y un sueldo bruto de \$290.800 es \$0.

**Ejemplo 2:** El impuesto a pagar por un trabajador soltero, con 1 hijo y un sueldo bruto de \$18.500 es \$925.

**Ayuda:** Recuerde que cada estructura que define viene acompañada de un predicado que determina si un objeto es o no una estructura de ese tipo.

**Ejercicio 9.** Diseñe la función *subcadena?* que dada una lista de strings  $l$  y un string  $s$  devuelva otra lista que contenga únicamente los strings de  $l$  que contienen como subcadena a  $s$ . Por ejemplo:

```
| (check-expect (subcadena? (list "cadena" "holena" "casa") "na") (list "cadena" "helena"))
| (check-expect (subcadena? (list "Hola" "mundo") "eso") '())
| (check-expect (subcadena? '() "altos") '())
```

**Ayuda:** *Racket* provee la función *string-contains?*. Puede leer la documentación si no la conoce.

**Ejercicio 10.** Diseñe una función *shortest-longest* que dada una lista de strings, devuelva una estructura *peso* que contenga el último string de menor longitud y el último string de mayor longitud de la lista. En caso de que la lista recibida sea vacía, deberá mostrar un mensaje indicándolo.

Por ejemplo:

```
| (check-expect (shortest-longest (list "a" "acb" "b" "xf" "asdf")) (make-posn "b" "asdf"))
| (check-expect (shortest-longest (list "a")) (make-posn "a" "a"))
```

Incluya en su diseño al menos dos ejemplos adicionales.

**Ejercicio 11.** El índice de masa corporal (IMC) se calcula a partir del peso  $p$  y de la altura  $a$  de una persona como:  $p/(a^2)$ .

- Diseñe una estructura *pers* que permita representar la información asociada a una persona.
- Diseñe una función *prom-inc* que dada una lista  $l$  de estructuras *pers* devuelva el promedio del IMC de todas las personas de la lista  $l$ .

Por ejemplo:

```
| (check-within (prom-inc (list (make-pers 44 1.55) (make-pers 50 1.56))) 19.43 0.01)
```

Incluya en su diseño otros ejemplos apropiados.

- Se puede clasificar a cada persona, según su IMC, en las siguientes categorías:

- Bajo Peso, si  $\text{IMC} < 18.5$
- Peso Normal, si  $18.5 \leq \text{IMC} < 25$
- Sobrepeso, si  $25 \leq \text{IMC} < 30$
- Obesidad, si  $30 \leq \text{IMC}$

Diseñe una función *clasif* que dada una lista  $l$  de estructuras *pers* devuelva una lista con la clasificación de las personas de la lista  $l$ .

Por ejemplo:

```
| (check-expect (clasif (list (make-pers 44 1.55) (make-pers 50 1.56))) (list "Bajo Peso" "Peso Normal"))
```

Incluya en su diseño otros ejemplos apropiados.

**Ejercicio 12.** Complete el diseño del programa interactivo que se le provee.

(Modifique el archivo *ejercicio-plantillaA.rkt*)

Este programa comienza con el fondo de color verde y un cuadrado de color rojo en el centro de la escena.

El estado del sistema estará compuesto por:

- un color, que sera el color correspondiente al cuadrado en el centro de la escena
- un angulo que indica la rotacion con la cual se dibuja el cuadrado en el centro de la escena
- un string que indica si el cuadrado rota automaticamente con cada tick o no. El string "on" indica una rotacion automatica. El string "off" indica que el cuadrado no se mueve.

Defina constantes para representar el ancho y el alto de la escena, así como el color de fondo. Defina cualquier otra constante que considere necesaria (por ejemplo, el lado del cuadrado a dibujar).

La expresión *big-bang* deberá comportarse de manera tal que:

- El estado inicial del sistema sea: color rojo, angulo 0 y el movimiento "on".
- La función que responde a la cláusula *to-draw* dibuje el cuadrado en el centro de la escena con la rotación indicada por el angulo.
- Si se presiona la barra espaciadora deberá modificar el movimiento. O sea, si al presionar la barra espaciadora el movimiento tenía un valor "on" el valor pasara a ser "off". Y viceversa.
- Con cada tick del reloj si el movimiento tiene un valor "on" se modifica en 5 grados el angulo del cuadrado. Los valores posibles que puede asumir el angulo son valores entre 0 y 359. Si el angulo en un determinado momento es de 359, en el tick siguiente tendrá un valor de 3. Si el movimiento tiene un valor "off" no se modifica el angulo.
- Si se presiona la tecla "b" el color del cuadrado pasa a ser azul, si se presiona la tecla "y" pasa a ser amarillo y si se presiona "r" pasa a ser rojo.

**Ayuda:** Recuerde que *DrRacket* provee la función *rotate*. Si queremos dibujar un cuadrado de lado 20 con una rotación de 45 grados usamos la siguiente expresion:

```
| (rotate 45 (rectangle 20 20 "solid" "red"))
```

**Ejercicio 13.** Diseñe un programa que permita agregar mediante el mouse círculos a una escena. Al principio, se dibujará una escena vacía. Cada vez que se haga click con el mouse, se agregará en esa posición un círculo de un determinado radio y color.

(Modifique el archivo *ejercicio-plantillaB.rkt*)

El estado del sistema estará compuesto por:

- una imagen (que representará la escena completa en un momento dado)
- un número que indica el radio de círculo que se dibujará al hacer click con el mouse
- un string que indica de qué color se dibujarán los círculos.

Defina constantes para representar el ancho y el alto de la escena, así como el color de fondo. Defina cualquier otra constante que considere necesaria.

La expresión *big-bang* deberá comportarse de manera tal que:

- El estado inicial del sistema sea: como imagen, una escena vacía del color de fondo elegido. El radio inicial será 20 y el color rojo.
- La función que responde a la cláusula *to-draw* dibuje la imagen guardada dentro del estado. Esta función debería ser muy sencilla de definir, dado que simplemente es poner en pantalla una componente del estado!
- Al hacer click sobre la escena, se modifique el estado agregando a la imagen (primera componente) un círculo de radio y color que corresponda (según la segunda y tercera componente) en las coordenadas donde se realizó el click.
- Responda al teclado de la siguiente forma: Al presionar "b" el color de los próximos círculos será azul, "m" para magenta y "g" para verde.
- Al presionar "up", el estado debe modificarse para que el radio de los próximos círculos se duplique. Al presionar "down", el radio de los próximos círculos debe reducirse a la mitad.
- Al presionar "r", la imagen debe ser la inicial.

**Ayuda:** Observe que el manejador del mouse sólo modifica la primer componente del estado, aunque para hacerlo necesita saber cuáles son la segunda y la tercera.

**Ejercicio 14.** En este ejercicio escribiremos una versión reducida de un programa que permita al usuario dibujar figuras sobre un lienzo vacío.

Inicialmente tendremos un lienzo blanco.

Cuando se haga click con el mouse en el lienzo, se dibujará una figura en la posición donde ocurrió el evento. Las opciones de color y forma para la figura a dibujar están dadas a continuación y se controlan con el teclado. Si se presiona:

- "r" el color del lápiz será rojo.
- "g" el color del lápiz será verde.
- "b" el color del lápiz será azul.
- "s" el lápiz dibujará un cuadrado.
- "c" el lápiz dibujará un círculo.

**Ejercicio 15.** Complete el diseño del programa interactivo que se le provee.

(Modifique el archivo *ejercicio-plantillaC.rkt*)

Este programa comienza con el fondo vacío de color amarillo. Cuando se presione la barra espaciadora se agregará un círculo color negro en la escena en coordenadas aleatorias (el círculo no puede aparecer cortado, debe caber íntegramente en la escena).

El estado del sistema estará compuesto por una lista de *posn*. Cada uno de estos *posn* representa la posición de cada círculo.

Defina constantes para representar el ancho y el alto de la escena, así como el color de fondo. Defina cualquier otra constante que considere necesaria (por ejemplo, el radio de los círculos a dibujar).

La expresión *big-bang* deberá comportarse de manera tal que:

- El estado inicial del sistema sea la lista vacía.
- La función que responde a la cláusula *to-draw* dibuje la escena con todos los círculos en las posiciones que le corresponden.
- Si se presiona la tecla "b" deberán borrarse todos los círculos y volver a estado inicial.
- Si se presiona la tecla "u" deberá borrarse únicamente el último círculo agregado.

### 4 Programando con funciones de alto orden

Para resolver estos ejercicios, en las funciones que se trabajen sobre listas se pide que utilice *fold*, *map* y *filter*. Recuerde cargar el lenguaje **Estudiante Intermedio**, y que en dicho lenguaje *fold* se llama *foldr*.

**Ejercicio 16.** Los sistemas recomendadores de libros que puede encontrar en la web se basan generalmente en los datos y las puntuaciones que introducen los miles de usuarios que utilizan estos sistemas. En este ejercicio programaremos la base de un muy sencillo recomendador de libros.

- Entre los datos que nos interesan de un libro se encuentran el isbn (identificador), nombre, autor y puntaje (número natural entre 1 y 10). Diseñe una estructura *libro* que permita representar esta información.
- Cada usuario puede verse como una lista de libros que ha leído y puntuado. Defina una constante de ejemplo que represente a un usuario.

- Dadas dos puntuaciones de dos usuarios distintos sobre un mismo libro decimos que las lecturas coinciden si la diferencia entre los puntajes es menor a 2. Diseñe una función *coincide?* que dados dos estructuras *libro* decida si las lecturas fueron coincidentes. Considerar el caso donde los dos libros argumento son distintos.

- Dado un usuario de nuestra plataforma, nos interesa poder recomendarle libros que ya hayan leído otros usuarios similares a él. Diseñe una función *similares?* que dados dos usuarios  $a$  y  $b$  (listas de usuarios) decida si  $b$  es similar a  $a$  y por lo tanto puede recomendarle lecturas. Un usuario  $b$  puede recomendar lecturas a otro usuario  $a$  si más del 10% de las lecturas de  $a$  son coincidentes con las de  $b$ .

**Ejercicio 17.** Diseñe una función *suma-positivos-es-mayor?* que toma una lista  $l$  con objetos de cualquier clase, y devuelve *#t* si el valor absoluto de la suma de los números positivos es mayor al valor absoluto de la suma de los números negativos.

Por ejemplo:

```
| (check-expect (suma-positivos-es-mayor?
               (list 5 "abc" 2 #t -3 "def")) #t)
| (check-expect (suma-positivos-es-mayor?
               (list (circle 10 "solid" "red") 12 1 -2 -45)) #f)
```

Presente al menos dos ejemplos más en su diseño.

**Ejercicio 18.** Diseñe una función *perfr-sqrt* que dada una lista  $l$  de números enteros devuelva el producto de las raíces cuadradas de aquellos elementos de  $l$  que son cuadrados perfectos.

Por ejemplo:

```
| (check-expect (perfr-sqrt (list 12 25 19 144 7)) 60)
```

Incluya en su diseño otros ejemplos apropiados.

**Ejercicio 19.** Diseñe una función *par-par?* que dada una lista  $l$  de números enteros devuelva *#t* si hay una cantidad par de números pares y *#f* en caso contrario.

Por ejemplo:

```
| (check-expect (par-par? (list 2 5 6)) #t)
| (check-expect (par-par? (list 2 5 6 0 7)) #f)
```

Incluya en su diseño al menos dos ejemplos más.

Para la definición de *par-par?* se pide que complete adecuadamente lo siguiente:

```
| (define (par-par? l) (foldr _ _ l))
```

**Ejercicio 20.** Diseñe la función *suma-cubos*, que dada una lista de números  $l$ , devuelva la suma de los cubos de los números positivos de  $l$ .

```
| (check-expect (suma-cubos (list 1 2 3 -1)) 36)
| (check-expect (suma-cubos '()) 1)
| (check-expect (suma-cubos (list 1 0 -3 5 -12 -16)) 126)
```

**Ejercicio 21.** Diseñe la función *long-lists*, que toma una lista de listas y devuelve *#true* si y sólo si las longitudes de todas las sublistas son mayores a 4.

```
| (check-expect (long-lists (list (list 1 2 3 4 5) (list 1 2 3 4 5 6) (list 87 73 78 83 33))) #t)
| (check-expect (long-lists (list '() '() (list 1 2 3))) #f)
| (check-expect (long-lists (list (list 1 2 3 4 5) '())) #f)
```

### 5 Números naturales

**Ejercicio 22.** Considere la función  $f$  definida como sigue:

- $f(0) = 0$
- $f(1) = 2$
- $f(2) = 5$
- $f(n) = f(n-1) + 2 * f(n-3)$ , para todo  $n$  mayor o igual a 3.

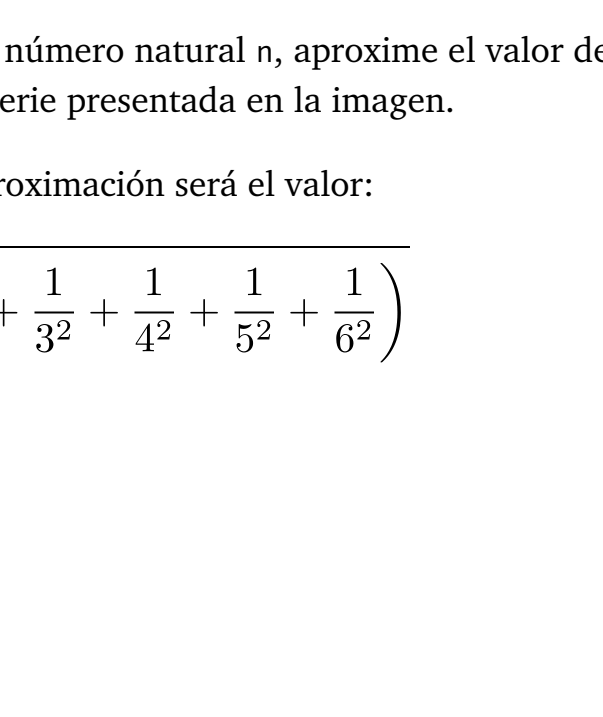
Diseñe una función  $f5$  que dado un número  $n$  devuelva la lista con todos los valores entre  $f(0)$  y  $f(n)$ .

```
| (check-expect (f5 5)
               (list 0 2 5 5 9 19))
| (check-expect (f5 0)
               (list 0))
```

Presente al menos dos ejemplos más en su diseño.

**Ejercicio 23.** Diseñe una función *dibujar-elipses* que tome un número natural  $n$  y devuelva una imagen de  $300 \times 300$  con  $n$  elipses de contorno azul centradas en el centro de la imagen. Los tamaños de las elipses serán:  $(10^\circ n)$ ,  $(5^\circ n)$ ,  $(10^\circ (n-1))$ ,  $(5^\circ (n-1))$ , ...,  $20^\circ 10$ ,  $10^\circ 5$ . El ángulo de rotación de cada elipse coincide con el valor del doble de  $n$ .

Ejemplo:

```
| (check-expect (dibujar-elipses 30)
               
```

**Ejercicio 24.** La conjetura de Goldbach postula que todo entero par, positivo y mayor a 2 puede expresarse como la suma de dos números primos. Por ejemplo:  $22 = 3 + 19$ . Esta conjetura es uno de los más famosos resultados de teoría de números que no ha podido ser demostrado correcto. Sin embargo, ha sido confirmado empíricamente hasta números muy grandes. Escriba una función que dado un número par  $n \gg 4$  encuentre dos números primos cuya suma resulte  $n$ . Por ejemplo:

$| (\text{goldbach } 22) = (\text{make-posn } 3 \ 19)$

**Ayuda:** Puede utilizar la función *prime?* definida a continuación que decide si un número dado es primo.

```
(define (f n c)
  (cond [(< n (+ c c)) #t]
        [(zero? (modulo n c)) #f]
        [else (f n (add1 c))]))
```

(define (prime? n) (f n 2))

**Ejercicio 25.** Es conocido que

$$\pi = \sqrt{6 \left( \frac{1}{1^2} + \frac{1}{2^2} + \frac{1}{3^2} + \frac{1}{4^2} + \dots \right)}$$

Diseñe una función *aprox-pi* que, dado un número natural  $n$ , aproxime el valor de  $\pi$  calculando los  $n$  primeros sumandos de la serie presentada en la imagen.

Por ejemplo, si tomamos  $n = 6$ , nuestra aproximación será el valor:

$$\pi \approx \sqrt{6 \left( \frac{1}{1^2} + \frac{1}{2^2} + \frac{1}{3^2} + \frac{1}{4^2} + \frac{1}{5^2} + \frac{1}{6^2} \right)}$$