Práctica 5: Procedimientos recursivos

Ejercicio 1

¿Conoces el algoritmo de ordenación *Quicksort*? Este algoritmo lo inventó *Hoare* en 1961 y originalmente se implementó en Algol. La versión de *Quicksort* que vamos a implementar en esta práctica es dividir la lista original en dos sublistas con los elementos mayores y menores que el primer elemento de la lista original y ordenarlos recursivamente.

Por ejemplo, si tenemos la lista `(5 4 9 4 6 3 3)` el algoritmo coge el primer elemento, 5, y divide la lista restante en dos sublistas, una conteniendo los elementos menores de 5 y otra conteniendo los mayores:

```
(4 4 3 3) 5 (9 6)
```

Ahora recursivamente ordenamos las dos listas mediante Quicksort:

```
(3 3 4 4) 5 (6 9)
```

Finalmente, ponemos todo en orden:

```
(3 3 4 4 5 6 9)
```

a) Escribe en notación matemática o pseudocódigo una solución recursiva.
b) Implementa el procedimiento recursivo `(quicksort lista)`.

Ejercicio 2

Vamos a trabajar con la recursión y los gráficos de tortuga. Recordamos las instrucciones para cargar la librería en Racket:

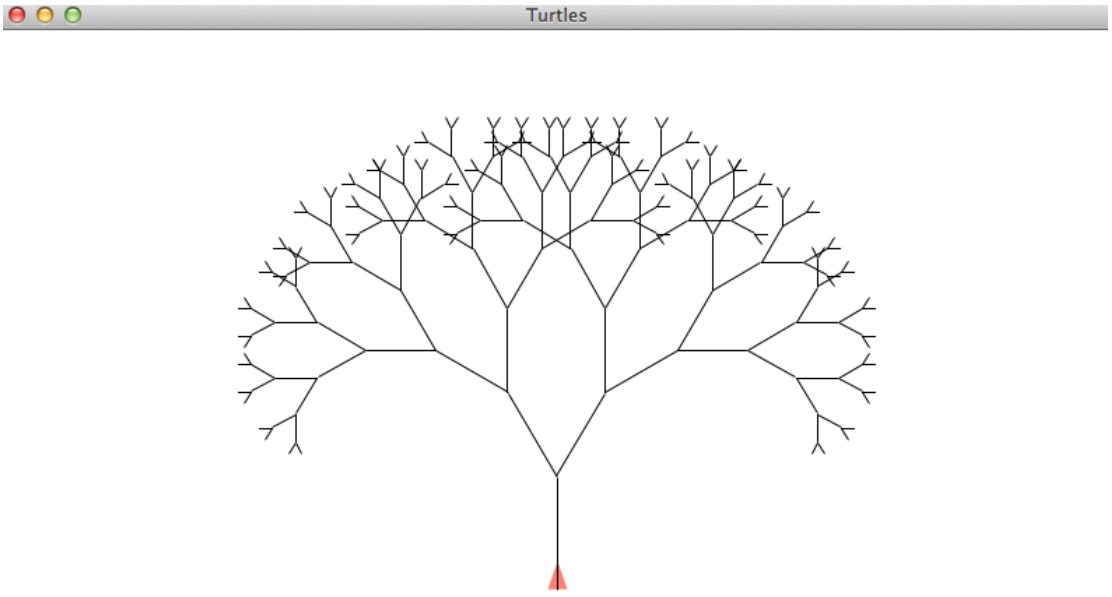
```
(require graphics/turtles)
(turtles #t)
```

Para mostrar varios ejemplos en la ejecución de un mismo fichero Racket, podemos llamar a la siguiente función que muestra la ventana y pide introducir algo para continuar:

```
(define (show-window-and-wait)
  (turtles #f)
  (turtles #t)
```

```
(display "Introduce algo para continuar")
(read))
```

Haciendo uso de los gráficos de tortuga, escribe una función `(arbol l)` que dibuje recursivamente el árbol siguiente:



image

Fíjate que un árbol está formado por dos subárboles más pequeños. El algoritmo es:

- Si `l` es mayor que un umbral:
    - Dibujar la rama (una línea recta de determinada longitud)
    - Girar la tortuga sobre sí misma un determinado número de grados a la izquierda, por ejemplo 30 para disponerte a dibujar el “subárbol” de la izquierda.
    - Dibujar un árbol un poco más pequeño que el tamaño actual
    - Girar la tortuga a la derecha para disponerte a dibujar el “subárbol” de la derecha (serán –60, ya que tienes que deshacer los 30 que giraste y girar otros 30 más)
    - Dibujar un árbol un poco más pequeño que el tamaño actual
    - Volver a girar a la izquierda 30 para quedarte mirando a la horizontal
    - Volver hacia atrás al punto de partida (en línea recta pero una cantidad negativa, usa la orden `move` en lugar de `draw` ya que no necesitas dibujar)

Ten en cuenta que la tortuga empieza mirando a la derecha, tendrás que girarla 90 grados a la izquierda antes de empezar a dibujar.

Prueba a variar la longitud de la rama o el número de grados, o a hacer árboles asimétricos en los que el número de grados que se gira para el subárbol izquierdo sea distinto al derecho.

Ejercicio 3

Vamos ahora a practicar con los gráficos de tortuga usando un enfoque iterativo.

a) Espiral

Veamos un primer ejemplo. Supongamos el siguiente procedimiento que hace avanzar la tortuga y la gira un ángulo:

```
(define (trazo lado angulo)
  (draw lado)
  (turn angulo))
```

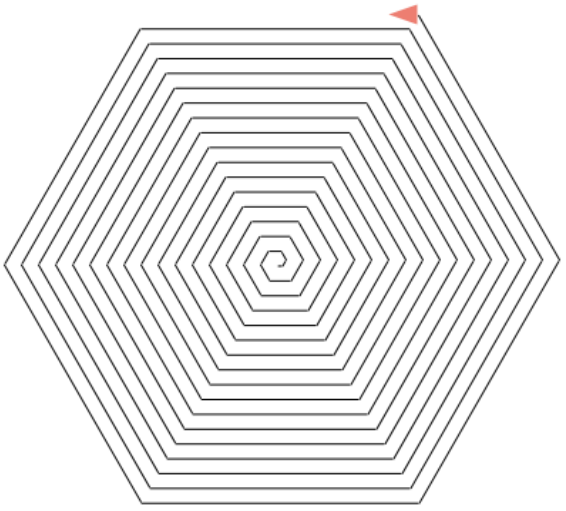
Si pruebas a llamar a `trazo` incrementando el lado, verás que se va dibujando una espiral:

```
(trazo 10 90)
(trazo 13 90)
(trazo 16 90)
(trazo 19 90)
(trazo 21 90)
(trazo 24 90)
(trazo 27 90)
(trazo 30 90)
```

Escribe, utilizando *tail recursion*, la función `(espiral lado inc angulo lado-max)` que invoque al procedimiento `trazo` de forma repetida, aumentando cada vez el lado hasta que el `lado` supere `lado-max`. Por ejemplo, si invocamos a

```
(espiral 3 2 60 200)
```

Obtendremos el siguiente dibujo:



Espiral

Prueba con distintos valores, para comprobar qué dibujos se realizan.

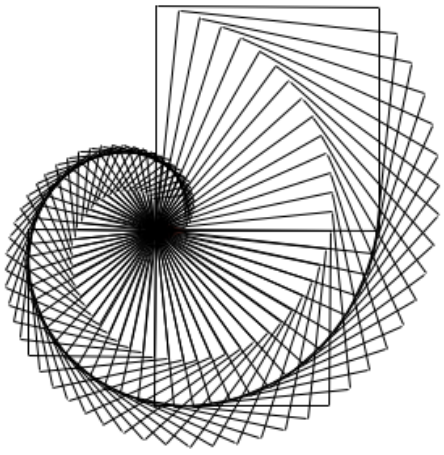
b) Girador

Definimos los siguientes procedimientos:

```
(define (cuadrado n)
  (draw n)
  (turn 90)
  (draw n)
  (turn 90)
  (draw n)
  (turn 90)
  (draw n)
  (turn 90))

(define (cuadrado-girado lado giro)
    (turn giro)
    (cuadrado lado))
```

Escribe una función recursiva que use *tail recursion* llamada `girador` que invoque a `cuadrado-girado` modificando repetidamente su parámetro `lado`, con el que podamos dibujar figuras como la siguiente. Terminamos la figura cuando el giro total del cuadrado es de 360 grados. Define tu mismo los parámetros que consideres necesarios para el procedimiento.



Girador

Ejercicio 4

a) Implementa la función `(suma-iter lista)` que sume los números de una lista mediante *tail recursion*.

Ejemplo:

```
(suma-iter '(1 2 3 4 5))
⇒ 15
```

b) Implementa la función `(fold-iter f base lista)` que implemente mediante *tail recursion* una versión modificada de *fold* en la que se comienzo aplicando la función `f` al parámetro `base` y al primer elemento de la lista. La recursión sería la siguiente:

```
(fold-iter f base '(1 2 3 4))
⇒ (f (f (f (f base 1) 2) 3) 4)
```

Por ejemplo:

```
(fold-iter string-append "-" '("Hola" "que" "tal")) ⇒ "-Holaquetal"
(fold-iter + 0 '(1 2 3 4)) ⇒ 10
```

Ejercicio 5

Implementa la función `cumple-predicados` de la práctica anterior mediante *tail recursion*.

Ejemplo:

```
(cumple-predicados (list even? mayor-5? menor-10?) 6)
⇒ (#t #t #t)
```

Ejercicio 6

La función `(list-reverse lista)` invierte una lista. Implementa sus dos versiones: recursiva e iterativa.

```
(list-reverse '(1 2 3 4))
⇒ (4 3 2 1)
```
