Práctica 1, 1ra parte

Programación I

1 Proposiciones y Condicionales

Una *proposición* es una expresión sobre la cual se puede afirmar que es *verdadera* o *falsa*. En la Práctica 0 - sección 4 vimos que, a través de expresiones que evalúan a valores de verdad, podemos representar proposiciones en *racket* . Por ejemplo:

`(< 3 4)` es una proposición cuyo valor es `#true`

Otro ejemplo es la siguiente proposición cuyo valor es falso: "*La palabra azul tiene más de cuatro letras.*"

En *DrRacket* esto se representa y evalúa como sigue.

```
> (> (string-length "azul") 4)
#f
```

Las proposiciones juegan un rol clave en programación, ya que permiten que los programas se comporten de diferentes formas de acuerdo a si se cumplen o no determinadas condiciones.

1.1 Un programa que necesita tomar decisiones

La librería *El lápiz curioso* tiene una promoción para la venta de cuadernos. El precio de cada cuaderno es $60, pero comprando 4 o más realiza un 10% de descuento.

Estamos interesados en definir una función que, dado un número que representa la cantidad de cuadernos a comprar, nos calcule el importe a pagar.

Dado que la promoción de la librería establece una condición en los precios, nuestro programa tendrá que evaluar lo siguiente:

"*Si la cantidad de cuadernos es menor que 4*

entonces calcular el total siendo el precio $60 por cuaderno;

si no, calcular el total siendo el precio $54 por cuaderno."

Acercándonos un poco a nuestro lenguaje, la definición tendrá la siguiente estructura:

```
(define (precio x) ...)
```

donde debemos completar los puntos suspensivos por una expresión que calcule:

- `(* 60 x)`, si x es menor a `4`
- `(* 54 x)` si x es mayor o igual a `4`

Para lograrlo, usaremos una expresión `if`. Una expresión de este tipo se construye escribiendo `if` seguido de tres expresiones, y encerrando todo entre paréntesis (obviamente, pues estamos construyendo una expresión).

En nuestro caso, escribiremos:

```
(if (< x 4) (* 60 x) (* 54 x))
```

Al evaluar esta expresión, se procede de la siguiente forma:

1. Se evalúa la primer expresión. El resultado de esta **debe** ser un booleano.
2. Si el resultado de la primer expresión es `#true`, entonces la segunda expresión es evaluada; caso contrario, se evalúa la tercera.

Completemos la definición:

```
(define
  (precio x)
  (if (< x 4)  (* 60 x)  (* 54 x)))
```

Siguiendo las reglas vistas más arriba, podemos evaluar la expresión `(precio 3)`:

```
(precio 3)

== definición de la función precio, reemplazando x por 3

(if (< 3 4) (* 60 3) (* 54 3))

== 3 es menor que 4, y por lo tanto (< 3 4) evalúa a #true

(if #true (* 60 3) (* 54 3))

== Se elige la segunda expresión del if

(* 60 3)

== definición de *

180
```

Antes de continuar, experimente con otro valor (quizás mayor a 4) en el área de interacción y usando el evaluador *paso a paso*.

1.2 Leyes de reducción

La descripción informal que hicimos de cómo se evalúa una expresión `if` puede precisarse explicitando sus *leyes de reducción*. Estas nos indican cómo se evalúa una expresión en nuestro lenguaje.

Por ejemplo, la siguiente definición

```
(define (f x) e)
```

agrega a nuestro sistema la siguiente ley (o regla) de reducción:

```
(f a)
== definición de f (ley 1)
e[a/x]
```

donde `e [a/x]` significa "*reemplazar en `e` todas las ocurrencias de `x` por la expresión `a`*". Esta ley nos la explican cuando aprendemos funciones en los cursos de matemática.

En el caso de las expresiones `if`, tenemos dos leyes de reducción, de acuerdo a lo explicado en la subsección anterior:

```
(if #true a b)
== definición de if (ley 1)
a
```

```
(if #false a b)
== definición de if (ley 2)
b
```

Observemos entonces que, cuando nos enfrentamos a una expresión de la forma `(if c a b)`, necesitamos evaluar primero la expresión `c` para poder determinar cuál de las dos leyes aplicar. Una vez que la evaluamos, si el resultado es un booleano, entonces las leyes nos indican cuál de las dos expresiones debemos seguir evaluando para obtener el valor final de la expresión.

> Pruebe evaluar la expresión
>
> ```
> (if (+ 1 1) "A" "B")
> ```

Observemos que si queremos evaluar una expresión `if` donde la primera expresión no reduce a un booleano, no es posible aplicar ninguna de las leyes, lo cual resulta en un error.

1.3 Ejercicios

1. Diremos que una imagen es *angosta* si su alto es mayor o igual a su ancho. Caso contrario diremos que es *ancha*. Defina una función que tome como

argumento una imagen, y la clasifique en `"Angosta"` o `"Ancha"`.

2. Modifique el ejercicio anterior para que ahora clasifique las imágenes en `"Angosta"`, `"Ancha"` o `"Cuadrada"` (si su alto es igual a su ancho).

3. Defina una función que dado tres números que representan la amplitud de los tres ángulos interiores de un triánguno, devuelva un string que nos indique si es equilátero, isósceles o escaleno.

4. Modifique el ejercicio anterior para que nos aparezca un mensaje de error en caso que los valores recibidos como argumento no correspondan a los ángulos de un triángulo. **Ayuda:** La suma de los ángulos interiores de un triángulo es 180°.

5. La contracción en el consumo ha obligado a *El lápiz curioso* a extender sus ofertas. Al ingresar a la librería, se lee el siguiente cartel:

   Lápices, llevando 5 o más: 15% de descuento.

   Cuadernos, llevando 4 o más: 10% de descuento.

   Agregue al área de definiciones las siguientes líneas:

   ```
   (define PC 60)
   (define PL 8)
   ```

   Estos valores representan el precio unitario de los cuadernos y lápices respectivamente. Defina una función que, dados dos valores `c` y `l`, devuelva el monto a pagar si se compran `c` cuadernos y `l` lápices. Sería bueno que en la definición aparezcan las constantes `PL` y `PC`. Su función será más fácil de modificar si hay cambios de precios.

6. Luego de un tiempo, las cosas siguen difíciles para *El lápiz curioso*. Además del cartel del ejercicio anterior, se lee el siguiente:

   Llevando 10 o más unidades, 18% de descuento.

   Obviamente, si en una compra aplica más de una oferta, al cliente se le hace aquel descuento que más lo beneficie.

   Considere nuevamente la consigna del ejercicio anterior, pero modifique su función (o mejor, defina una nueva) para contemplar la nueva oferta.

7. Redefina, usando `if`, la función `pitagorica?` de la Práctica 0.

8. Modifique el ejercicio anterior para que se genere un `string` en lugar de un booleano. En esta versión, esperamos que `(pitagorica? 3 4 5)` evalúe a `"Los números 3, 4 y 5 forman una terna pitagórica."` **Ayuda:** La función `number->string` puede resultar útil.

9. > Esta función puede ayudarlx a hacerse famosx resolviendo la Conjetura

de Collatz.

Defina una función `collatz` que reciba un número natural `n` y devuelva `n/2` si `n` es par, o `3*n+1` si `n` es impar.

2 Banderas

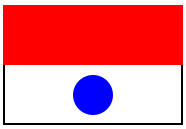
En esta sección comenzaremos a discutir algunas buenas prácticas de programación, que serán motivadas por los siguientes ejercicios, en los cuales tenemos que generar imágenes que representan banderas.

1. Primero, nos pondremos de acuerdo en algunas definiciones comunes. La idea es que las banderas sean imágenes de 90 pixeles de ancho por 60 de alto. Segundo, todas las banderas las dibujaremos sobre un marco del tamaño indicado. Este marco lo provee la función `empty-scene`. A modo de ejemplo, si uno escribe el siguiente código,

> Recuerde que se encuentran disponibles las funciones: `rectangle`, `triangle`, `rotate`, `place-image`, etc., importando el módulo image desde el menú *Lenguaje -> Seleccionar paquete de enseñanza*.

```
(define ejemplo (place-image (rectangle 90 30 "solid" "red")
                             45 15
                             (place-image (circle 10 "solid" "blue")
                                          45 45
                                          (empty-scene 90 60))))
```

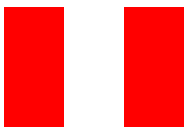
`ejemplo` será:



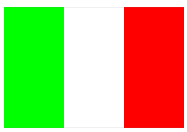
Antes de continuar con los ejercicios, asegúrese de entender el ejemplo. Realice modificaciones y vea qué resultados obtiene.

Ahora sí, continúe con los ejercicios, dibujando en cada caso lo pedido.

a. Perú:



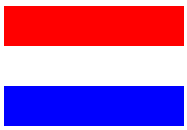
b. Italia:



c. Defina una única función que permita, de acuerdo al valor de sus parámetros, crear tanto la bandera de Perú, como la de Italia, como cualquier bandera que esté formada por tres bandas verticales de igual ancho.

d. Alemania:

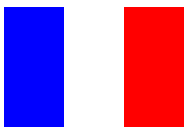
e. Holanda:



f. Defina una única función que permita, de acuerdo al valor de sus parámetros, crear tanto la bandera de Alemania, como la de Holanda, como cualquier otra bandera formada por tres bandas horizontales de igual alto.

g. Defina una única función que dados tres colores y un sentido (vertical/horizontal), permita definir las banderas anteriores.

h. Utilice dicha función para generar la bandera de Francia:



i.

> Como vimos en este ejercicio y en el 5 de la sección 1.3, el uso de constantes tiene ciertos beneficios:
>
> i. Simplifica el código. El código es más entendible si tenemos una constante (con un nombre significativo) en lugar de un valor puntual (un número, un string, etc).
> ii. Es más robusto a cambios. Si usamos constantes, sólo tenemos que cambiar el valor en cuestión una vez (en la definición de la constante) y no en todas las apariciones del valor.
>
> Por ello, el uso de constantes, cuando sea apropiado, constituye una buena práctica de programación que debemos seguir.

Utilizar dicha función para ayudarse a definir las banderas de Sudán, Argentina y Camerún.

j. La siguiente es más desafiante, Brasil:

k. Modifique el código para que las banderas se vean en el doble de su tamaño. ¿Debe modificar muchos sectores del código? Modifique las definiciones adecuadamente, utilizando constantes globales, para que los cambios a futuro sean más simples.