▸ Práctica 1, 2da parte

Práctica 1, 2da parte
1 Condicionales múltiples
1.1 Calculando con expresiones cond
2 Predicados

Práctica 1, 2da parte

Programación I

1 Condicionales múltiples

En la Práctica 1, primera parte vimos cómo formar sentencias condicionales usando `if`.

Una desventaja de las expresiones que utilizan `if` se observa cuando uno debe elegir entre más de dos posibilidades. En estas situaciones, es habitual encontrar expresiones condicionales unas dentro de otras, y puede resultar más difícil entender el comportamiento del programa.

Un ejemplo clásico es la función *signo*, que recibe un número y nos devuelve `-1`, `0` o `1` de acuerdo a si su argumento es negativo, cero o positivo respectivamente. Podemos definir esta función usando `if`:

```
(define (sgn1 x) (if (< x 0) -1 (if (= x 0) 0 1)))
```

La definición de `sgn1` verifica primero si su argumento es negativo, y en ese caso, devuelve `-1`. En otro caso, compara por igualdad a `x` con `0`, devolviendo `0` si resultan iguales, y `1` en caso contrario.

Una forma alternativa de definir este tipo de funciones la provee la expresión `cond`. Las expresiones de esta clase tienen la siguiente forma:

```
(cond [Condición-1  Resultado-1]
      [Condición-2  Resultado-2]
      ....
      [Condición-n  Resultado-n])
```

Es decir, la expresión condicional está formada por una lista de *parejas*. Cada pareja va encerrada entre `[]`. La primera parte de cada pareja (*Condición-X*) es una expresión que debe evaluar a un valor de tipo booleano. La segunda parte de la pareja se conoce como el *resultado* de la condición correspondiente, y puede evaluar un valor de cualquier tipo de dato.

¿Cómo procede *DrRacket* para evaluar una expresión `cond`? Primero se evalúa la primer condición (*Condición-1*). Si esta reduce a `#true`, entonces el resultado de toda la expresión es el que se obtiene de evaluar *Resultado-1*. Caso contrario, *DrRacket* descarta la primer pareja de la expresión condicional, y procede con la segunda pareja del mismo modo que con la primera. Si todas las condiciones evalúan a `#false`, entonces se produce un error.

Es decir, tenemos dos **leyes de reducción** que permiten evaluar expresiones condicionales. La primera nos dice qué hacer cuando la primer condición es verdadera:

```
(cond [#true       Resultado-1]
      [Condición-2  Resultado-2]
      ...
      [Condición-N  Resultado-N])

== definición de cond (ley 1)

Resultado-1
```

La segunda nos indica cómo reducir la expresión si la primer condición es falsa:

```
(cond [#false      Resultado-1]
      [Condición-2  Resultado-2]
      ...
      [Condición-N  Resultado-N])

== definición de cond (ley 2)

(cond [Condición-2  Resultado-2]
      ...
       [Condición-N  Resultado-N])
```

Volviendo a la función signo, podemos definir una nueva versión:

```
 (define (sgn2 x) (cond [(< x 0) -1]
                        [(= x 0) 0]
                        [(> x 0) 1]))
```

Observemos que es más sencillo entender qué condiciones se asocian a cada posible resultado.

1.1 Calculando con expresiones cond

Calculemos el valor de la expresión `(sgn2 0)`:

```
(sgn2 0)

== definición de la función sgn2, reemplazando x por 0

(cond [(< 0 0) -1]
      [(= 0 0) 0]
      [(> 0 0) 1])

== 0 no es menor a 0, y por lo tanto (< 0 0) evalúa a #false

(cond [#false -1]
      [(= 0 0) 0]
      [(> 0 0) 1])

== definición de cond, ley 2

(cond [(= 0 0) 0]
      [(> 0 0) 1])

== 0 es igual a 0, y por lo tanto (= 0 0) evalúa a #true

(cond [#true 0]
      [(> 0 0) 1])

== definición de cond, ley 1

0
```

Como la segunda parte de la pareja elegida ya es una constante, no hay más evaluaciones para realizar.

Ejercicio 1. Calcule el resultado de las expresiones `(sgn2 (- 2 3))` y `(sgn2 6)`. Verifique sus procedimientos en el evaluador paso a paso de *DrRacket* .

Ejercicio 2. Rehaga, utilizando expresiones `cond`, los ejercicios 2, 4, 6 y 7 de la Práctica 1, 1ra parte - sección 1.3.

Ejercicio 3. Calcule el resultado de las expresiones `(pitagorica? 3 5 6)` y `(pitagorica? 3 5 4)`. **Atención:** El resultado de la última expresión debe ser `#true`.

Ejercicio 4. Hemos decidido hilar más fino en la clasificación de imágenes. Ahora diremos que una imagen es `"Muy gorda"` si su ancho es más que el doble que su alto. Del mismo modo, diremos que `"Muy flaca"` si su alto es más que el doble que su ancho. Defina una función, utilizando una expresión `cond`, que clasifique imágenes en alguna de las categorías `"Muy Gorda"`, `"Gorda"`, `"Cuadrada"`, `"Flaca"`, `"Muy flaca"`.

Ayuda: Recuerde que el orden en que uno coloca las *parejas* es relevante ¿Qué condiciones le conviene poner primero?

Piense en resolver el mismo problema utilizando sólo expresiones `if`.

Ejercicio 5. Como parte de una aplicación para observar el clima, se pide clasificar una temperatura de la siguiente forma:

- `"No Olvidar Bufanda (NOB)"` (menos de 0 grados);
- `"Frío (F)"` (entre 0 y 15 grados);
- `"Agradable (A)"` (entre 15 y 25 grados);
- `"Realmente Caluroso (RC)"` (más de 25 grados);

Considere la siguiente función:

```
(define (clasificar t) (cond [(< t 0) "No Olvidar Bufanda (NOB)"]
                             [(and (> t 0) (< t 15)) "Frío (F)"]
                             [(and (> t 15) (< t 25)) "Agradable (A)"]
                             [(> t 25) "Realmente Caluroso (RC)"]))
```

Evalúe cada expresión de la forma indicada:

- `(clasificar -3)`, en el área de interacción
- `(clasificar 12)`, con lápiz y papel
- `(clasificar 28)`, con el evaluador paso a paso de *DrRacket* .

¿Qué sucede con la expresión `(clasificar 15)`? Tómese un minuto para entender cuál es el problema ¿Para qué valores sucede lo mismo? Redefina la función para todas las temperaturas queden clasificadas (decida usted en qué categoría quedan los valores no contemplados previamente).

2 Predicados

Hasta ahora hemos trabajado con varias clases de datos: números, cadenas, booleanos, imágenes. Sin embargo, cuando definimos funciones no tenemos en cuenta qué clase de valores éstas pueden recibir, pudiendo caer en errores como el siguiente:

```
> (define A "23")
> (define B 84)
> (sgn2 B)
1
> (sgn2 A)
<: contract violation
  expected: real?
  given: "23"
  argument position: 1st
  other arguments...:
   0
```

Los *predicados* son funciones que devuelven un valor de verdad. *DrRacket* nos provee este tipo de funciones para establecer si un valor pertenece a una determinada clase. Por ejemplo:

```
> (string? "Hola")
#t
> (string? 23)
#f
> (number? -4)
#t
> (number? (sqrt 2))
#t
> (number? "23")
#f
```

```
> (boolean? 0)
#f
> (string? (number->string 23))
#t
> (number? (sgn2 (if #t 3 "Chau")))
#t
```

El uso de estas funciones permite prevenir errores como el presentado más arriba y son de gran utilidad a la hora de definir otras funciones que deben comportarse distinto de acuerdo a la clase de sus argumentos. Veamos un caso concreto.

Ejemplo 1. Decidimos extender la función `signo` para que pueda recibir como argumento `strings`.

Procedemos como sigue:

```
(define (sgn3 x) (cond [(number? x) (sgn2 x)]
                       [(string? x) (sgn2 (string->number x))]))
```

Si el argumento de la función es un número, entonces utiliza la función `sgn2`, que tenía justamente este propósito. Si el argumento es un `string`, entonces lo convertimos a número y luego aplicamos la función `sgn2`.

Evaluando:

```
> (sgn3 34.3)
1
> (sgn3 "-345.23")
-1
> (sgn3 "345a")
<: contract violation
  expected: real?
  given: #f
  argument position: 1st
  other arguments...:
   0
```

Obviamente, la última expresión nos da un error, pues `number->string` no puede convertir `"345a"` a un número.

Ejercicio 6. Muchas veces se identifica a los valores booleanos con los números `0` (para `#false`) y `1` (para `#true`). Con esta identificación en mente, extienda la función `signo` para que pueda procesar booleanos.

Ejercicio 7. Las imágenes `"Flacas"` son negativas, mientras que las `"Gordas"` son positivas. Obviamente, las `"Cuadradas"` equivalen al 0 de los números. Extienda la función `signo` para soportar imágenes.

Ejercicio 8. Modifique la función definida en el ejecicio anterior para que, en caso de no recibir un número, booleano, imagen o string nos muestre el siguiente mensaje `"Clase no soportada por la función."`.

Ayuda: Quizás sea interesante mirar la documentación de las expresiones `cond`, en particular lo referente a las cláusulas `else`.

Ejercicio 9. Como última extensión, modifique la función para que imprima `"La cadena no se puede convertir a un número"`, en caso que se procese un `string` para el que la función `number->string` no devuelva un número.