Ejercicios Adicionales de Práctica
Evaluación Paso a Paso

En los siguientes ejercicios debe **realizar, con papel y lápiz, las evaluaciones paso a paso** de las expresiones dadas. **No olvide justificar cada paso de reducción.**

Le sugerimos que luego de resolver los ejercicios **utilice el evaluador paso a paso de Racket para verificar** que los pasos de reducción que ha establecido, son los correctos.

Al final de este documento están las soluciones de algunos de los ejercicios. Le sugerimos que intente resolverlos antes de mirar la solución.

Ejercicio 1. Considere la función `xor`

```
(define (xor a b)
        (or (and a (not b))
            (and (not a) b)
         )
)
```

De la evaluación paso a paso de las siguientes expresiones.

1. `(or (even? 10) (xor #false #true) (>3 0))`
2. `(or #false (xor #false #true) (>3 0))`
3. `(and (odd? 3) (xor #true #true) (>5 10))`
4. `(and (xor #true #false) (boolean? #false) (string? "#true"))`

Ejercicio 2. Considere la función `f`.

```
(define (f x y)
  (if (and (< x y) (even? y))
      "correcto"
      (if (< x y) "incorrecto impar"
                  "incorrecto")))
```

De la evaluación paso a paso de las siguientes expresiones.

1. `(f 5 4)`
2. `(string? (f 6 12))`
3. `(f 3 11)`
4. `(or (= 2 2) (+ 1 1) (f 3 11))`

Ejercicio 3. Considere la función g.

```
(define (g x)
          (cond [ (not (string? x))  "No es un String"]
                [ (not (string-numeric? x)) "No son caracteres numéricos"]
                [else (string->number x)]
          )
)
```

De la evaluación paso a paso de las siguientes expresiones.

1. `(g "a")`
2. `(g (+ 1 3))`
3. `(number? (g "45"))`
4. `(and (boolean=? #true #false) (string? (g 12)))`

Soluciones

Ejercicio 1.2

```
(or #false (xor #false #true) (> 3 0))
==<por def. de xor>
(or #false
    (or (and #false (not #true))
        (and (not #false) #true))
    (> 3 0))
==<por def. de and, evaluación de cortocircuito>
(or #false
    (or #false
        (and (not #false) #true))
    (> 3 0))
==<def. de not>
(or #false
    (or #false
        (and #true #true))
    (> 3 0))
==<def. de and>
(or #false
    (or #false
        #true)
    (> 3 0))
```

```
==<def. de or >
(or #false
   #true
   (> 3 0))
==<def. de or, evaluación de cortocircuito >
#true
```

Ejercicio 2.1

```
(f 5 4)
==<def. de f>
(if (and (< 5 4) (even? 4))
    "correcto"
    (if (< 5 4) "incorrecto impar"
                "incorrecto")))
==<def. de '<' >
(if (and #false (even? 4))
    "correcto"
    (if (< 5 4) "incorrecto impar"
                "incorrecto")))
==<def. de and, evaluación de cortocircuito>
(if #false
    "correcto"
    (if (< 5 4) "incorrecto impar"
                "incorrecto")))
==<def. de if>
(if (< 5 4) "incorrecto impar"
            "incorrecto"))
==<def. de '<' >
(if #false "incorrecto impar"
            "incorrecto"))
==<def. de if>
"incorrecto"
```

Ejercicio 3.1

```
(g  "a")
==<def. de g >
(cond [ (not (string? "a"))  "No es un String"]
      [ (not (string-numeric? "a")) "No son caracteres numéricos"]
      [ else (string->number "a")]
)
==<def. de string?>
(cond [ (not #true)  "No es un String"]
      [ (not (string-numeric? "a")) "No son caracteres numéricos"]
      [ else (string->number "a")]
)
```

```
==<def. de not>
(cond [ #false  "No es un String"]
      [ (not (string-numeric? "a")) "No son caracteres numéricos"]
      [ else (string->number "a")]
)
==<def. de cond>
(cond [ (not (string-numeric? "a")) "No son caracteres numéricos"]
      [ else (string->number "a")]
)
==<def. de string-numeric?>
(cond [ (not #false) "No son caracteres numéricos"]
      [ else (string->number "a")]
)
==<def. de not>
(cond [ #true "No son caracteres numéricos"]
      [ else (string->number "a")]
)
==<def. cond>
"No son caracteres numéricos"
```