1. Considérense las siguientes definiciones de funciones:

   `f x y z = x + y - z`   `g x y = f x x y`   `g' x = f x x`   `g'' = f x`

   Entonces:

   ○ `g`, `g'` y `g''` son equivalentes

   ⊗ `g` y `g'` son equivalentes, pero la definición de `g''` contiene un error

   ○ Las dos anteriores son falsas.

2. Considérense las siguientes definiciones de funciones:

   `f x y z = x + y - z`   `g x y = f x y y`   `g' x = f x`   `g'' x = f x y`

   Entonces:

   ○ `g`, `g'` y `g''` son equivalentes

   ○ `g` y `g'` son equivalentes, pero la definición de `g''` contiene un error

   ⊗ Las dos anteriores son falsas.

3. Considérense las siguientes definiciones de funciones:

   `f x y z = x + y - z`   `g x y z = f (x+1) y z`   `g' x y = f (x+1) y`   `g'' x = f (x+1)`

   ⊗ `g`, `g'` y `g''` son equivalentes

   ○ `g` y `g'` son equivalentes entre sí, pero no equivalentes a `g''`

   ○ La definición de `g''` contiene un error

4. El valor de la expresión `let x=1:3:x in take 3 x` es:

   ⊗ [1,3,1]

   ○ $\perp$, porque la evaluación no termina

   ○ Hay un error sintáctico porque `x` no puede aparecer en el lado derecho de `let x= ...`

5. El valor de la expresión `let x=1:3:x in head (x ++ x)` es:

   ⊗ 1

   ○ $\perp$, porque la evaluación no termina

   ○ Hay un error sintáctico porque `x` no puede aparecer en el lado derecho de `let x= ...`

6. El valor de la expresión `let x= reverse (x++[1]) in head x` es:

   ○ 1   ⊗ La evaluación no termina   ○ La expresión es incorrecta, porque `x` no puede aparecer a la derecha de `let x = ...`

7. El valor de la expresión `let x=x++x in head (2:x)` es:

   ⊗ 2   ○ $\perp$, porque la evaluación no termina   ○ La expresión está mal construida o mal tipada

8. El valor de la expresión `let x=x++[1] in last x` es:

   ○ 1   ⊗ $\perp$, porque la evaluación no termina   ○ La expresión está mal construida o mal tipada

9. Considére la evaluación de las expresiones `let x=2:filter (/=2) x in head x`
   `let x=2:filter (/=2) x in head (tail x)`

   ○ Ninguna de las dos termina

   ○ Una de las dos termina

   ○ Las dos terminan

10. El valor de la expresión `let x=x++[1] in x!!1` es:

    ○ 1   ⊗ $\perp$, porque la evaluación no termina   ○ La expresión está mal construida o mal tipada

11. El valor de la expresión `let x=[1]++x in head (tail x)` es:
    ⊗ 1
    ◯ La evaluación no termina
    ◯ La expresión está mal construida o mal tipada

12. El valor de la expresión `let x=1:y in head x` es:
    ◯ 1 ◯ $\perp$, porque la evaluación no termina ⊗ La expresión está mal construida o mal tipada

13. Considérense las expresiones siguientes:
    $e_1 \equiv$ `(let x=5 in x+x) + 3` $e_2 \equiv$ `let x=2 in let y=x+x in y*y`
    $e_3 \equiv$ `let x=2 in let y=x+x in y*y*x` $e_4 \equiv$ `let y=x+x in let x=2 in y*y*x`
    $e_5 \equiv$ `let y=(let x=2 in x+x) in y*y` $e_6 \equiv$ `let y=(let x=2 in x+x) in y*y*x`
    La evaluación de esas expresiones dará error:
    ◯ Exactamente en tres de ellas
    ⊗ Exactamente en dos de ellas
    ◯ Todas están correctamente formadas

14. Dadas las expresiones: `length [1..]` y `let x=length [1..] in fst (1,x)`
    ⊗ La evaluación de la segunda termina pero la de la primera no
    ◯ La evaluación de ninguna de las dos termina
    ◯ Las dos anteriores son falsas.

15. Dadas las expresiones: `[True,[]]` `True:[]` `[True]:[]` `[[True],[]]`
    ⊗ Exactamente una de las expresiones está mal tipada
    ◯ Exactamente dos de las expresiones están mal tipadas
    ◯ Las dos anteriores son falsas.

16. Sean las cuatro expresiones: `[True:[]]` `[]:[True]` `[True]:[]` `[[True],[]]`
    ◯ Dos de ellas están mal tipadas
    ⊗ Dos de ellas son sintácticmente equivalentes y una está mal tipada
    ◯ Las dos anteriores son falsas.

Nota: en las preguntas siguientes, suponemos que los numerales `1,2,...` tienen el tipo concreto `Int`.

17. Dadas las expresiones: `0:[1]` `0:[1]:[2]` `0:[1,2]` `[0,1]:[[2]]` `([]:[],2)`
    ⊗ Exactamente una de las expresiones está mal tipada
    ◯ Exactamente dos de las expresiones están mal tipadas
    ◯ Las dos anteriores son falsas.

18. Dadas las expresiones: `[]:[1]` `[1:[2]]:[]` `[1:[2]]:[[]]` `[1,1]:(2:[])` `(1:[]):[]`
    ◯ Exactamente tres de las expresiones están mal tipadas
    ⊗ Exactamente dos de las expresiones están mal tipadas
    ◯ Las dos anteriores son falsas.

19. Dadas las expresiones: `0:[1]` `0:[1]:[2]` `0:[1,2]` `[0,1]:[2]` `(1:[]):[]`
    ◯ Exactamente tres de las expresiones están mal tipadas
    ⊗ Exactamente dos de las expresiones están mal tipadas
    ◯ Las dos anteriores son falsas.

20. Dadas las expresiones: `[1]:[]` `[[]]:[]` `[]:[]` `(1:2):[]` `1:(2:[])`
⊗ Exactamente una de ellas está mal tipada
◯ Exactamente dos de ellas están mal tipadas
◯ Las dos anteriores son falsas.

21. Dadas las expresiones: `[0]:[1]` `[]:[[]]:[]` `[0]:[[]]:[]` `[0]:[[1,2]]` `([[]]:[],[1])`
◯ Exactamente una de las expresiones está mal tipada
⊗ Exactamente dos de las expresiones están mal tipadas
◯ Exactamente tres de las expresiones están mal tipadas

22. Dadas las expresiones: `[1]:[]` `[1]:[[]]:[2]` `[1]:[[[]]]` `0:1:2` `(0:[1],2)`
◯ Exactamente una de las expresiones está mal tipada
◯ Exactamente dos de las expresiones están mal tipadas
⊗ Las dos anteriores son falsas.

23. ¿Cuál de las siguientes expresiones es sintácticamente equivalente a `[[[0],[],[2,2]]]`?
⊗ `((0:[]):[[],2:2:[]]):[]`
◯ `[0]:[]:[2,2]:[]:[]`
◯ Ninguna de las anteriores, porque de hecho la expresión está mal tipada

24. ¿Cuál de las siguientes expresiones es sintácticamente equivalente a `[[1,[]],[3,4]]`?
◯ `((1:[]):[[],3:4:[]]):[]`
◯ `[1]:[]:[3,4]:[]:[]`
⊗ Ninguna de las anteriores, porque de hecho la expresión está mal tipada

25. ¿Cuál de las siguientes expresiones es sintácticamente equivalente a `[[3,4],[]]`?
◯ `3:4:([],[])`
⊗ `(3:4:[]):[]:[]`
◯ Ninguna de las anteriores, porque de hecho la expresión está mal tipada

26. ¿Cuál de las siguientes expresiones es sintácticamente equivalente a `(1:[]):(1:2:[]):[]`?
⊗ `[[1],[1,2]]`
◯ `[[1],[1,2],[]]`
◯ Ninguna de las anteriores, porque de hecho la expresión está mal tipada

27. ¿Cúantas de las siguientes expresiones son sintácticamente equivalentes a `[[1,2],[]]`?
`[1:2:[]]` `1:2:[[]]` `[1,2]:[[]]` `[1:[2],[]]`
⊗ Exactamente dos ◯ Exactamente tres ◯ Exactamente cuatro

28. Considérense las definiciones

```
f x = 1 + f (x+1)
g x = if x >= 1 then 1 else 0
h x = 3
```

y las expresiones $e \equiv$ `g (f 1)` y $e' \equiv$ `h (g (f 1))`. Entonces:
◯ Ni la evaluación de e ni la de e' terminan, tanto al usar evaluación impaciente como perezosa.
◯ Ni la evaluación de e ni la de e' terminan al usar evaluación impaciente, pero sí lo hacen (ambas) al usar evaluación perezosa.
⊗ Las dos anteriores son falsas.

29. Sea e una expresión. ¿Cuál de las siguientes situaciones es posible?
   - ○ Al evaluar e por evaluación impaciente se obtiene el valor 3, y por evaluación perezosa el valor 2
   - ○ Al evaluar e por evaluación impaciente resulta el valor 3, y por evaluación perezosa el cómputo no termina
   - ⊗ Al evaluar e por evaluación perezosa resulta el valor 3, y por evaluación impaciente el cómputo no termina

30. Sea e una expresión. ¿Cuántas de las siguientes situaciones son posibles al evaluar e?
   - • Por evaluación impaciente se obtiene el valor 3, y por evaluación perezosa el valor 2.
   - • Se obtiene el valor 3 tanto por evaluación impaciente como por evaluación perezosa.
   - • Por evaluación perezosa el cómputo no termina, y por evaluación impaciente el cómputo termina.
   - • Por evaluación perezosa el cómputo termina, y por evaluación impaciente el cómputo no termina.
   - • El cómputo no termina ni por evaluación impaciente ni por evaluación perezosa.
   - ○ Exactamente dos
   - ⊗ Exactamente tres
   - ○ Las dos anteriores son falsas.

31. Sean las funciones `f` y `g` definidas por `f x = f x` y por `g f x = if x==0 then 0 else f (x-1)`.
   Entonces, al evaluar la expresión `g f 1`:
   - ○ Con evaluación perezosa se obtiene el valor 0, y con evaluación impaciente el cómputo no termina.
   - ○ Tanto con evaluación perezosa como con evaluación impaciente se obtiene el valor 0.
   - ⊗ Tanto con evaluación perezosa como con evaluación impaciente el cómputo no termina.

32. Sean las funciones `f` y `g` definidas por `f g = g (f g)` y por `g f x = if x==0 then 0 else f (x-1)`.
   Entonces, al evaluar la expresión `f g 1`:
   - ⊗ Con evaluación perezosa se obtiene el valor 0, y con evaluación impaciente el cómputo no termina.
   - ○ Tanto con evaluación perezosa como con evaluación impaciente se obtiene el valor 0.
   - ○ Tanto con evaluación perezosa como con evaluación impaciente el cómputo no termina.

33. Sean las funciones `f` y `g` definidas por `f g = g (f g)` y por `g f x = if x==[] then 0 else 1 + f (tail x)`.
   Entonces, al evaluar la expresión `f g [2]`:
   - ○ Con evaluación perezosa se obtiene el valor 2, y con evaluación impaciente el cómputo no termina.
   - ⊗ Con evaluación perezosa se obtiene el valor 1.
   - ○ Tanto con evaluación perezosa como con evaluación impaciente el cómputo no termina.

34. Supongamos que `1::Int` , `(+)::Int->Int->Int`, y considérese la función `f` definida por las dos reglas siguientes:
   ```
   f True x y = (x,y)
   f False y x = (y,x+1)
   ```
   - ⊗ El tipo que se infiere para `f` es `Bool -> a -> Int -> (a,Int)`
   - ○ El tipo que se infiere para `f` es `Bool -> Int -> Int -> (Int,Int)`
   - ○ `f` está mal tipada.

35. Considérese la función definida por `f x y = y x x` . El tipo de `f` es:
   - ⊗ `a -> (a -> a -> b) -> b`
   - ○ `a -> (a -> a -> a) -> a`
   - ○ No está bien tipada

36. Sea `f` definida por `f g x = x (g True) g`. El tipo de `f` es:
   - ⊗ `(Bool->a) -> (a -> (Bool->a) -> b) -> b`
   - ○ `(Bool->a) -> (a -> (Bool->a) -> a) -> a`
   - ○ Está mal tipada

37. Considérese la función definida por `f x y = x (y x)` . El tipo de `f` es:
   ⊗ `(a -> b) -> ((a -> b) -> a) -> b`
   ◯ `(a -> b -> a) -> (b -> a) -> a`
   ◯ No está bien tipada

38. Considérese la función definida por `f g = g (f g)` . El tipo de `f` es:
   ◯ `a -> a -> a`
   ⊗ `(a -> a) -> a`
   ◯ No está bien tipada

39. Sea `f` definida por `f g x = x (x g)`. El tipo de `f` es:
   ◯ `a -> (a -> b) -> b` ⊗ `a -> (a -> a) -> a` ◯ Está mal tipada

40. Sea `f` definida por `f x g = x (x (g True))`. El tipo de `f` es:
   ◯ `(a -> b) -> (Bool -> a) -> b`
   ⊗ `(a -> a) -> (Bool -> a) -> a`
   ◯ Ninguno, `f` está mal tipada

41. Sea `f` definida por `f x y = (x y).(x y)`. El tipo de `f` es:
   ◯ `(a -> a) -> (a -> a)`
   ◯ `(a -> b) -> (a -> b) -> (a -> b)`
   ⊗ `(a -> b -> b) -> a -> b -> b`

42. Sea `f` definida por `f x y = x (x y)`. El tipo de `f` es:
   ⊗ `(a -> a) -> (a -> a)`
   ◯ `(a -> b) -> a -> b`
   ◯ `a -> b -> a`

43. Considérense las expresiones de tipo (solo difieren en los paréntesis): $\tau_1 =$ `(a -> a) -> (a -> a) -> (a -> a)`
   $\tau_2 =$ `(a -> a) -> (a -> a -> a -> a)`
   $\tau_3 =$ `(a -> a) -> (a -> a) -> a -> a`
   ◯ $\tau_1 \equiv \tau_2 \not\equiv \tau_3$
   ⊗ $\tau_1 \equiv \tau_3 \not\equiv \tau_2$
   ◯ $\tau_1 \equiv \tau_2 \equiv \tau_3$

44. Considérense las expresiones de tipo (solo difieren en los paréntesis):
   $\tau_1 =$ `((b -> a) -> a) -> ((a -> b) -> (b -> b))`
   $\tau_2 =$ `(b -> (a -> a)) -> (a -> b) -> b -> b`
   $\tau_3 =$ `(b -> a -> a) -> (a -> b -> b -> b)`
   Entonces:
   ◯ $\tau_1 \equiv \tau_2 \not\equiv \tau_3$
   ◯ $\tau_1 \equiv \tau_2 \equiv \tau_3$
   ⊗ $\tau_1 \not\equiv \tau_2 \not\equiv \tau_3 \not\equiv \tau_1$

45. Considérense las expresiones de tipo (que solo difieren en los paréntesis): $\tau_1 =$ `(a -> (b -> a) -> a) -> b -> b`
   $\tau_2 =$ `(a -> ((b -> a) -> a)) -> (b -> b)`
   $\tau_3 =$ `a -> b -> a -> a -> b -> b`
   ⊗ $\tau_1 \equiv \tau_2 \not\equiv \tau_3$

- ○ $\tau_1 \not\equiv \tau_2 \not\equiv \tau_3 \not\equiv \tau_1$
- ○ $\tau_1 \equiv \tau_3 \not\equiv \tau_2$

46. Considérense las expresiones de tipo (solo difieren en los paréntesis): $\tau_1 =$ `(a -> b -> a) -> (a -> b -> b)`
$\tau_2 =$ `(a -> b -> a) -> a -> (b -> b)`
$\tau_3 =$ `(a -> (b -> a)) -> a -> b -> b`
    - ○ $\tau_1 \equiv \tau_2 \not\equiv \tau_3$
    - ○ $\tau_1 \equiv \tau_3 \not\equiv \tau_2$
    - ⊗ $\tau_1 \equiv \tau_2 \equiv \tau_3$

47. Considérense las expresiones de tipo (solo difieren en los paréntesis): $\tau_1 =$ `a -> (b -> a -> a) -> b -> b`
$\tau_2 =$ `(a -> ((b -> a) -> a)) -> b -> b`
$\tau_3 =$ `a -> (b -> (a -> a)) -> (b -> b)`
    - ○ $\tau_1 \equiv \tau_2 \not\equiv \tau_3$
    - ⊗ $\tau_1 \equiv \tau_3 \not\equiv \tau_2$
    - ○ $\tau_1 \equiv \tau_2 \equiv \tau_3$

48. Considérense las expresiones de tipo (solo difieren en los paréntesis): $\tau_1 =$ `(a -> b -> a -> a) -> b -> b`
$\tau_2 =$ `a -> b -> a -> a -> b -> b`
$\tau_3 =$ `(a -> b -> (a -> a)) -> (b -> b)`
    - ○ $\tau_1 \equiv \tau_2 \not\equiv \tau_3$
    - ⊗ $\tau_1 \equiv \tau_3 \not\equiv \tau_2$
    - ○ $\tau_1 \equiv \tau_2 \equiv \tau_3$

49. Considérense las expresiones de tipo (que solo difieren en los paréntesis): $\tau_1 =$ `a -> a -> (a -> a) -> (b -> b)`
$\tau_2 =$ `a -> (a -> ((a -> a) -> b -> b))`
$\tau_3 =$ `(a -> a) -> (a -> a) -> b -> b`
    - ○ $\tau_1 \equiv \tau_2 \equiv \tau_3$
    - ○ $\tau_1 \not\equiv \tau_2 \not\equiv \tau_3 \not\equiv \tau_1$
    - ⊗ $\tau_1 \equiv \tau_2 \not\equiv \tau_3$

50. Considérense las expresiones de tipo (que solo difieren en los paréntesis): $\tau_1 =$ `a -> ((a -> a) -> a)`
$\tau_2 =$ `a -> (a -> a) -> a`
$\tau_3 =$ `a -> a -> a -> a`
    - ⊗ $\tau_1 \equiv \tau_2 \not\equiv \tau_3$
    - ○ $\tau_1 \equiv \tau_3 \not\equiv \tau_2$
    - ○ $\tau_1 \not\equiv \tau_2 \not\equiv \tau_3 \not\equiv \tau_1$

51. Considérense las expresiones (que solo difieren en los paréntesis): $e_1 =$ `f (f x (y^2)) y`
$e_2 =$ `f ((f x) ((^) y 2)) y`
$e_3 =$ `f (f x (y^) 2) y`
    - ○ $e_1 \equiv e_2 \equiv e_3$
    - ⊗ $e_1 \equiv e_2 \not\equiv e_3$
    - ○ $e_1 \not\equiv e_2 \not\equiv e_3$

52. Considérense las expresiones (solo difieren en los paréntesis):
$e_1 =$ `f (z (y x)) ((z 0) x)`
$e_2 =$ `(f (z (y x))) (z 0 x)`
$e_3 =$ `f z (y x) (z 0 x)`

Entonces:
○ $e_1 \equiv e_2 \equiv e_3$
⊗ $e_1 \equiv e_2 \not\equiv e_3$
○ $e_1 \not\equiv e_2 \not\equiv e_3 \not\equiv e_1$

53. Considérense las expresiones (que solo difieren en los paréntesis): $e_1 =$ `f x (g x,y/2)`
$e_2 =$ `f x (g x) (y/2)`
$e_3 =$ `(f x) (g x,(/y) 2)`
○ $e_1 \equiv e_2 \equiv e_3$
⊗ $e_1 \not\equiv e_2 \not\equiv e_3 \not\equiv e_1$
○ $e_1 \equiv e_3 \not\equiv e_2$

54. Considérense las expresiones (que solo difieren en los paréntesis): $e_1 =$ `f x (g (x+1) y)`
$e_2 =$ `(f x) (g (((+) x) 1) y)`
$e_3 =$ `(f x) (g (x+1)) y`
⊗ $e_1 \equiv e_2 \not\equiv e_3$
○ $e_1 \not\equiv e_2 \equiv e_3$
○ $e_1 \equiv e_2 \equiv e_3$

55. Considérense las expresiones (que solo difieren en los paréntesis): $e_1 =$ `f x (g x,y+1)`
$e_2 =$ `f x (g x) (y+1)`
$e_3 =$ `(f x) (g x,(+) y 1)`
○ $e_1 \equiv e_2 \equiv e_3$
○ $e_1 \not\equiv e_2 \not\equiv e_3$
⊗ $e_1 \equiv e_3 \not\equiv e_2$

56. Considérense las expresiones (que solo difieren en los paréntesis): $e_1 =$ `f x g (x+1) y`
$e_2 =$ `(f x) (g (x+1) y)`
$e_3 =$ `(f x) g ((+) x 1) y`
○ $e_1 \equiv e_2 \not\equiv e_3$
⊗ $e_1 \equiv e_3 \not\equiv e_2$
○ Las dos anteriores son falsas.

57. Considérense las expresiones (que solo difieren en los paréntesis): $e_1 =$ `f x 1 (x + y)`
$e_2 =$ `(f x 1) (x + y)`
$e_3 =$ `f x 1 ((+) x y)`
○ $e_1 \not\equiv e_2 \not\equiv e_3 \not\equiv e_1$
○ $e_1 \equiv e_3 \not\equiv e_2$
⊗ $e_1 \equiv e_2 \equiv e_3$

58. Suponiendo la declaración `infixr 9 !` , considérense las expresiones (que solo difieren en los paréntesis):
$e_1 =$ `((! g) f) ! ((h !) i) ! j`
$e_2 =$ `((!) (f ! g)) ((h ! i) ! j)`
$e_3 =$ `(!) ((!) f g) ((!) ((!) h i) j)`
⊗ $e_1 \equiv e_2 \equiv e_3$
○ $e_1 \equiv e_2 \not\equiv e_3$
○ $e_1 \not\equiv e_2 \equiv e_3$

59. Considérense las expresiones de tipo (solo difieren en los paréntesis): $\tau_1 =$ `(b -> a -> a) -> (a -> b) -> b`
$\tau_2 =$ `(b -> (a -> a)) -> ((a -> b) -> b)`
$\tau_3 =$ `(b -> a -> a) -> a -> b -> b`

- ◯ $\tau_1 \equiv \tau_2 \equiv \tau_3$
- ◯ $\tau_1 \equiv \tau_3 \not\equiv \tau_2$
- ⊗ $\tau_1 \equiv \tau_2 \not\equiv \tau_3$

60. Considérense las expresiones de tipo (solo difieren en los paréntesis): $\tau_1 =$ `a -> b -> (c -> d)`
$\tau_2 =$ `a -> (b -> c -> d)`
$\tau_3 =$ `a -> b -> c -> d`

⊗ $\tau_1 \equiv \tau_2 \equiv \tau_3$ ◯ $\tau_1 \equiv \tau_3 \not\equiv \tau_2$ ◯ $\tau_1 \not\equiv \tau_2 \not\equiv \tau_3$

61. Considérense las expresiones (solo difieren en los paréntesis): $e_1 =$ `(f ((x y) y)) (f 0)`
$e_2 =$ `f (x y y) (f 0)`
$e_3 =$ `f (x y y) f 0`

◯ $e_1 \equiv e_2 \equiv e_3$ ◯ $e_1 \not\equiv e_2 \equiv e_3$ ⊗ $e_1 \equiv e_2 \not\equiv e_3$

62. Considérense las expresiones: $e_1 =$ `f x (y-1):z`
$e_2 =$ `((:)  f) x ((-) y 1) z`
$e_3 =$ `(:  z) ((f x) ((-) y 1))`

- ◯ $e_1 \equiv e_2 \equiv e_3$
- ◯ $e_1 \not\equiv e_2 \not\equiv e_3 \not\equiv e_1$
- ⊗ $e_1 \equiv e_3 \not\equiv e_2$

63. Considérese la función `f` definida como `f xs = foldr g [] xs where f x y = y++[x]`. Entonces:
- ⊗ `f xs` computa la inversa de `xs`
- ◯ `f xs` computa la propia lista `xs`
- ◯ `f` está mal tipada

64. Considérense las funciones
```
f xs  = foldr g [] xs where g x y = x:filter (/= x) y
f' xs = foldl g [] xs where g y x = x:filter (/= x) y
```
(y ojo al orden de argumentos en `g`*).* Entonces:
- ◯ `f xs` y `f' xs` coinciden, para cualquier lista finita `xs`.
- ◯ Los elementos de `f xs` y `f' xs` coinciden, quizás en otro orden, para cualquier lista finita `xs`.
- ◯ Una de las dos está mal tipada.

65. Considérese la función `f` definida como `f xs = foldl g [] xs where g y x = y++[x]`. Entonces:
- ◯ `f xs` computa la inversa de `xs`
- ⊗ `f xs` computa la propia lista `xs`
- ◯ `f` está mal tipada

66. Considérense las funciones:
```
f x y    = [2*i|i <- [1..x], i>y]
f' x y   = filter (> y) (map (2 *) [1..x])
f'' x y  = map (2 *) (filter (> y) [1..x])
```
- ◯ `f`, `f'` y `f''` computan lo mismo
- ◯ `f` y `f'` computan lo mismo, pero `f''` no
- ⊗ `f` y `f''` computan lo mismo, pero `f'` no

67. La evaluación de `foldl (\e x -> x:x:e) [] [1,2,3]` produce como resultado

◯ `[1,1,2,2,3,3]`
⊗ `[3,3,2,2,1,1]`
◯ `[3,2,1,3,2,1]`

68. La evaluación de `map (zipWith (-) [3,2,1]) [[1,2,3],[4,5,6]] !! 1 !! 0` produce como resultado:

◯ `[-1]`
⊗ `-1`
◯ Un error, porque esa expresión está mal tipada

69. La evaluación de `foldl (\x y -> y-x ) 0 [1,2,3]` produce como resultado

◯ -4
◯ -6
⊗ 2

70. La evaluación de `[j|j <- [i-1..i+1],i <- [1..5], i > 1]` produce como resultado

◯ 2 ◯ 3 ⊗ Un error

71. La evaluación de `length [i|i <- [1..5], j <- [1..i],i+j<5]` produce como resultado

◯ 3
⊗ 4
◯ 5

72. La evaluación de `length [i+j|i <- [1..5], i > 2, j <- [3..i]]` produce como resultado

◯ 0
⊗ 6
◯ Da un error en tiempo de ejecución

73. La evaluación de `length [i+j|i <- [1..5], i > 2, j <- [i-1,i]]` produce como resultado

◯ 9
⊗ 6
◯ Da un error en tiempo de ejecución

74. La evaluación de `[i+j|i <- [1..5], i > 2, j <- [i-1,i]] !! 2` produce como resultado

◯ 9
⊗ 7
◯ Da un error en tiempo de ejecución

75. La evaluación de `length [i+j|i <- [1..5], i > 2, j <- [i-1,i], j < i]` produce como resultado

◯ 6
⊗ 3
◯ 0

76. La evaluación de `foldr (\x y -> x-y ) 0 [1,2,3]` produce como resultado

◯ -4
◯ -6
⊗ 2

77. La evaluación de `foldr (\x y -> x/y ) 1 [8,4,2]` produce como resultado
   ⊗ 4
   ◯ 2
   ◯ 1

78. La evaluación de `foldl (\x y -> y-x ) 1 [1,2,3]` produce como resultado
   ◯ -1 ⊗ 1 ◯ 0

79. La evaluación de `take 2 [take j [i..2*i] | i <- [1..10], i > 2, j <- [i-1..i+1]]` produce el resultado
   ◯ `[3,4]`
   ⊗ `[[3,4],[3,4,5]]`
   ◯ No produce ningún valor, porque la expresión está mal tipada

80. La evaluación de `length [take j [1..i] | i <- [1..5], i > 2, j <- [i-1,i]]` produce como resultado
   ◯ 3
   ⊗ 6
   ◯ No produce ningún valor, porque la expresión está mal tipada

81. La evaluación de `map (zip [1..4]) [2,5..12]` produce el resultado
   ◯ `[(1,11),(2,11),(3,11),(4,11)]`
   ◯ `[(4,11)]`
   ⊗ No produce ningún valor, porque la expresión está mal tipada

82. La evaluación de `map length [take 2 ys | x <- [1..3], ys <- iterate (+ 3) x]` produce como resultado

   ◯ [2,2,2]
   ◯ [2]
   ⊗ Un error, porque la expresión está mal tipada

83. La evaluación de `(head.tail) (map ((take 3).(iterate (+ 2))) [1,3..10])` produce como resultado
   ◯ `[4,6,8,11,14]`
   ◯ `[1,4,7]`
   ⊗ `[3,5,7]`

84. La evaluación de `take 3 (zipWith (+) [x+y|x<-[1..4],y<-[x,x+2,x+4]] (iterate (*2) 1))` produce como resultado
   ◯ `[3,5,7]`
   ⊗ `[3,6,10]`
   ◯ []

85. La evaluación de `(head.tail) (map ((take 2).(iterate (+ 3))) [1,3..10])` produce como resultado
   ◯ `[4,6,8,11,14]`
   ◯ `[1,4]`
   ⊗ `[3,6]`

86. Sea $\bigoplus$`::Int -> Int -> Int` una función semánticamente asociativa, n el valor de `foldl` $\bigoplus$ `0 [1,2,3]` y m el valor de
   `foldr` $\bigoplus$ `0 [1,2,3]`. Entonces
   ◯ Es seguro que $n = m$

◯ Es seguro que $n \neq m$

⊗ Las dos anteriores son falsas.

87. Sea n el valor de `foldl (\x y -> y) 0 [1,2,3]` y m el valor de `foldr (\x y -> y) 0 [1,2,3]`. Entonces

⊗ $n > m$

◯ $n < m$

◯ $n = m$

88. La evaluación de `[j|i <- [1..5], i > 1, j <- [i-1..i+1]] !! i` produce como resultado

◯ 2 ◯ 3 ⊗ Un error, porque hay una variable fuera de ámbito

89. La función `f` definida por las siguientes ecuaciones:

```
f     x  False  = True
f  False     y  = True
f   True  True  = False
```

⊗ Es estricta en el segundo argumento pero no en el primero

◯ No es estricta en ninguno de sus argumentos

◯ Es estricta en sus dos argumentos

90. Sean las funciones

```
f x y = if x > 0 then 1 else y
g x = if x > 0 then 1 else 0
h x y = (g.(f x)) y
```

⊗ La función `h` es estricta en el primer argumento pero no en el segundo

◯ La función `h` no es estricta en ninguno de sus argumentos

◯ La función `h` es estricta en sus dos argumentos

91. Considérese el programa

```
f y 0 = g y
f 0 x = h x (x*x)
g x = if x > 0 then 1 else 0
h x y = if x > y then 1 else 0
```

◯ La función `f` es estricta en el segundo argumento pero no en el primero

◯ La función `f` no es estricta en ninguno de sus argumentos

⊗ La función `f` es estricta en sus dos argumentos

92. Considérese el programa

```
f 0 y = g y
f x 0 = h x (x*x)
g x = if x > 0 then 1 else 0
h x y = if x > y then 1 else 0
```

◯ La función `f` es estricta en el primer argumento pero no en el segundo

◯ La función `f` no es estricta en ninguno de sus argumentos

⊗ La función `f` es estricta en sus dos argumentos

93. Considérese el programa

```
f 0 y = g y
f x y = h y
g x = if x > 0 then 1 else 0
h x = 0
```

◯ La función `f` no es estricta en ninguno de sus argumentos
◯ La función `f` es estricta en sus dos argumentos
⊗ Las dos anteriores son falsas.

94. La función `f` definida por las siguientes ecuaciones:

```
f  False      y  = True
f      x  False  = True
f   True   True  = False
```

◯ No es estricta en ninguno de sus argumentos
⊗ Es estricta en el primer argumento pero no en el segundo
◯ Las dos anteriores son falsas.

95. Sea la función `f` definida por las siguientes ecuaciones:

```
f      x  False  = True
f  False      y  = True
f   True   True  = False
```

y considérense las siguientes afirmaciones: *(a)* $[\![$`f e` $\perp]\!] = \perp$, para toda expresión `e`
(b) $[\![$`f` $\perp$ `e`$]\!] = \perp$, para toda expresión `e`

◯ *(a)* y *(b)* son ciertas
⊗ *(a)* es cierta y *(b)* es falsa
◯ Las dos anteriores son falsas.

96. Considérese el programa

```
f 0 y = g y        g x = if x > 0 then 1 else 0
f x y = h y        h x = 0
mal = head []
```

◯ Existen e, e' tales que ni la evaluación de `(f` e `mal)` ni la de `(f mal` e'`)` dan error
◯ Para todo e, e' `f` e `mal` y `f mal` e' dan error
⊗ Las dos anteriores son falsas.

97. Sea `f` definida por las siguientes ecuaciones: y sea `mal = head []`.

```
f      x  False  = True
f  False      y  = True
f      _      _  = False
```

¿Cuál de las siguientes afirmaciones es cierta?
◯ `f mal` e da error de ejecución, para cualquier expresión e
⊗ `f` e `mal` da error de ejecución, para cualquier expresión e
◯ `f mal True` se evalúa a `False`

98. Sea `f` definida por las siguientes ecuaciones: y sea `mal = head []`.

```
f      x  False  = True
f  False      y  = True
f   True   True  = False
```

¿Cuál de las siguientes afirmaciones es cierta?
◯ La evaluación de `f mal` e da error, para cualquier expresión e
⊗ La evaluación de `f` e `mal` da error, para cualquier expresión e
◯ `f mal True` se evalúa a `False`

99. Sea `f` definida por las siguientes ecuaciones:

```
f  False      y  = y
f      x  False  = True
f   True   True  = False
```

y sea `mal = head []`. ¿Cuál de las siguientes afirmaciones es **falsa**?
◯ La evaluación de `f mal` e da error, para cualquier expresión e
◯ La evaluación de `f` e `mal` da error, para cualquier expresión e cuya evaluaci
'on termine
⊗ Las dos anteriores son falsas.

100. Sean las definiciones `f g h x y = g (h x (g y))` `f' g h x = g.(h x).g` `f'' g h = g.(\x -> h x).g`
◯ `f`, `f'` y `f''` son extensionalmente equivalentes
⊗ `f` y `f'` son extensionalmente equivalentes, pero `f''` no
◯ `f` y `f''` son extensionalmente equivalentes, pero `f'` no

101. Sean las definiciones `f g h x = g (h (g x))` `f' g h x = g.h.(g x)` `f'' g h = g.h.g`
◯ `f`, `f'` y `f''` son extensionalmente equivalentes
◯ `f` y `f'` son extensionalmente equivalentes, pero `f''` no
⊗ `f` y `f''` son extensionalmente equivalentes, pero `f'` no

102. Considérense las siguientes definiciones de funciones:

`f1 x y z = x .  y`   `f2 x y z = (x .  y) z`   `f3 x y z = x (y z)`   `f4 x y = x .  y`

◯ `f1, f2, f3, f4` computan la misma función
◯ `f1, f2, f3` computan la misma función, pero `f4` no
⊗ `f2, f3, f4` computan la misma función, pero `f1` no

103. ¿Cuál de los siguientes tipos para `f` hacen que la expresión `(curry f 0).(|| True)` esté bien tipada?
⊗ `f::(Int,Bool) -> Int`
◯ `f::  Int -> Bool -> Int`
◯ Esa expresión está mal tipada, sea cual sea el tipo de `f`

104. Sea `f` de tipo τ `->` τ, y `unaLista` de tipo `[`τ`]`. El tipo de la expresión `map (take 2) (map (iterate f) unaLista)` es:
◯ `[`τ`]`
⊗ `[[`τ`]]`
◯ Esa expresión está mal tipada

105. Sea `f` de tipo τ `->` τ, y `unaLista` de tipo `[`τ`]`. El tipo de la expresión `map (iterate f) (map (take 2) unaLista)` es:
◯ `[`τ`]`
⊗ `[[`τ`]]`, si es que τ es de la forma `[`τ'`]`
◯ Esa expresión está en cualquier caso mal tipada

106. La evaluación de `foldr (\x e -> x:[1..length e]) [0] [1,2,3]` produce como resultado
◯ `[1,2,3,0]`
◯ `[3,1,2,3]`
⊗ `[1,1,2,3]`

107. La evaluación de `foldr (\x y -> x y) 1 [\x -> x*x,\x -> x-1,(+ 3)]` produce como resultado
⊗ `9`
◯ `[1,0,4]`
◯ Una lista de funciones

108. La reducción de la expresión `(\x y -> (\z -> y (z+2)) (y x)) 3 (\x -> x+1)` producirá el resultado
- ◯ 8
- ⊗ 7
- ◯ 6

109. La reducción de la expresion `(\x y -> x (x y)) (\x -> x + 3) 4` producirá el resultado
- ◯ 7
- ⊗ 10
- ◯ Las dos anteriores son falsas.

110. La reducción de la expresion `(\x y -> x (x y)) (\x -> x + y) 4` producirá el resultado
- ◯ 8
- ◯ 12
- ⊗ Las dos anteriores son falsas.

111. La evaluación de `map (!! 2) (map (iterate (\x -> 2*x)) [0..3])` produce el resultado
- ◯ 2
- ⊗ `[0,4,8,12]`
- ◯ No produce ningún valor, porque la expresión está mal tipada

112. La evaluación de `map fst [(zip [0..i] [1..j]) !! i | i <- [1..3], j <- [1..(i+1)], j > i]` produce el resultado
- ◯ `(0,1)`
- ⊗ `[1,2,3]`
- ◯ No produce ningún valor, porque la expresión está mal tipada

113. La reducción de la expresión `(\x y -> (\u -> x (u+1)) (x y)) (\x -> x+2) 3` producirá el resultado
- ⊗ 8
- ◯ 7
- ◯ 6

114. La reducción de la expresión `(\x y -> x (y x))(\x -> x+1) (\x -> x 1)` producirá el resultado
- ◯ 1
- ◯ 2
- ⊗ 3

115. La reducción de la expresión `(\x y -> (\u -> x (u+1)) (x y)) (\x -> x+1) 3` producirá el resultado
- ◯ 5
- ⊗ 6
- ◯ 7

116. La reducción de la expresion `(\x y -> y (y x)) ((\x -> x+1) 0) (\x -> x+1)` producirá el resultado
◯ 1 ◯ 2 ⊗ 3

117. La evaluación de `foldr (\x y -> y+(x!!0)) 2 [[1,2],[3,4],[5,6]]` produce como resultado
- ⊗ 11
- ◯ `[2,3,4,5,6,7]`
- ◯ Nada porque la expresión está mal tipada

118. La evaluación de `foldr (\x y -> y+(x!!1)) 2 [[1,2],[3,4],[5,6]]` produce como resultado
◯ `[3,4,5,6,7,8]`
⊗ `14`
◯ Nada porque la expresión está mal tipada

119. ¿Cuál de las siguientes definiciones es equivalente a `f n m = [x*y | x <- [1..n], y <- [x..m]]`?
⊗ `f n m = concat (map f [1..n]) where f x = map (\y -> x*y) [x..m]`
◯ `f n m = concat (map f [x..m]) where f y = map (\y -> x*y) [1..n]`
◯ `f n m = zipWith (*) [1..n] [x..m]`

120. ¿Cuál de las siguientes definiciones es equivalente a `f n m = [x*n | x <- [1..n], x > m]`?
⊗ `f n m = map (\x -> x*n) (filter (> m) [1..n])`
◯ `f n m = concat (map (\x -> x*n) (filter (> m) [1..n]))`
◯ `f n m = filter (> m) (map (\x -> x*n) [1..n])`

121. ¿Cuál de los siguientes tipos para `f` hace que la expresión `(|| True).(uncurry f)` esté bien tipada?
◯ `f::(Int,Int)-> Bool`
⊗ `f::  Int -> Int -> Bool`
◯ Esa expresión está mal tipada, sea cual sea el tipo de `f`

122. ¿Cuál de los siguientes tipos para la expresión e hace que la expresión `zipWith filter [(> 0),(< 0)]` e esté bien tipada?
◯ `[Int]`
⊗ `[[Int]]`
◯ `[(Int,Int)]`

123. ¿Cuál de los siguientes tipos para la expresión e hace que la expresión `zipWith filter` e `[[1..4],[-2..3]]` esté bien tipada?
◯ `Int -> Bool`
⊗ `[Int -> Bool]`
◯ Las dos anteriores son falsas.

124. Sea `f::  Int -> Int -> Int` una función conmutativa, y considérense las igualdades siguientes:
(i) `foldr f 0 [3] = foldl f 0 [3]`

125. (ii) `foldr f 0 [3,5] = foldl f 0 [3,5]`. Entonces:
◯ Tanto (i) como (ii) son con seguridad ciertas.
⊗ Sólo (i) es con seguridad cierta.
◯ Las dos anteriores son falsas.

126. Considérense las definiciones:

```
f x y  | x == 0      = 1        .
       | x < y       = g (x-y)
       | otherwise   = 2
g x    | x >= 0      = x
```

¿Qué afirmación es correcta?
◯ $f\ x\ y$ tiene un valor definido para valores cualesquiera de x e y
⊗ $f\ x\ y$ tiene un valor definido $\Leftrightarrow x = 0 \vee x \geq y$
◯ La definición es errónea y será rechazada por el sistema

127. La reducción de la expresion `(\x y -> x + (\x -> y+x) y) ((\x -> x+1) 5) 2` producirá el resultado

⊗ 10
◯ 14
◯ Depende del orden en que se efectúe

128. La reducción de la expresion `(\x y -> x (y x)) (\x -> x + 2) (\x -> x 3)` producirá el resultado

◯ 5
◯ 3
⊗ 7

129. Considérense las igualdades

`(take m).(take n) = take (min n m)` `(drop m).(drop n) = drop (n+m)` `(take m).(drop n) = (drop n).(take (m+n))`

donde `m,n` son ≥ 0. Se tiene:

⊗ Las tres son correctas
◯ Solo dos son correctas
◯ Solo una es correcta

130. ¿Cuántas de las siguientes igualdades son correctas?

`map id = id`   `map f (xs ++ ys) = (map f xs) ++ (map f ys)`   `map (f.g) xs = (map f .  map g) xs`

Nota: en la primera de ellas, se entiende que las funciones de ambos lados se aplican solo a listas

⊗ Las tres
◯ Solo dos
◯ Solo una

131. ¿Cuál de las siguientes afirmaciones es cierta?

⊗ Las listas intensionales pueden eliminarse usando `map, filter, concat`, y recíprocamente
◯ Las listas intensionales pueden eliminarse usando `map, filter, concat`, pero el recíproco no es cierto
◯ Las dos anteriores son falsas.

132. Sea `f` una función definida previamente, y considérense las definiciones

```
g   x y   =  let z = f x y in (z,x*y,z+1)
g'  x y   =  (f x y,x*y,f x y + 1)
g'' x y   =  h x y (f x y)
h   x y z =  (z,x*y,z+1)
```

¿Qué afirmación es correcta?

⊗ `g, g'` y `g''` computan los mismos valores, `g` y `g''` tienen una eficiencia similar, pero `g'` es menos eficiente
◯ `g, g'` y `g''` computan los mismos valores, `g` y `g''` tienen una eficiencia similar, pero `g'` es más eficiente
◯ No es verdad que `g, g'` y `g''` computen los mismos valores

133. Sea `h::Int -> Int` una función costosa de calcular, y sean `f, f', f''` definidas por

```
f x y =  (x+u,y+u)        f' x y = (x+(h x) , y+(h x))   f'' x y = g x y (h x)
         where u = h x                                   g x y z = (x+z,y+z)
```

◯ Tanto `f'` como `f''` son extensionalmente equivalentes a `f` y comparables en eficiencia con ella.
⊗ Tanto `f'` como `f''` son extensionalmente equivalentes a `f`, pero `f'` es menos eficiente.
◯ Las dos anteriores son falsas.

134. Considérense las expresiones siguientes:

$e_1 \equiv$ `(let x=5 in x+x) + 5` $e_2 \equiv$ `let x=2 in let y=x+x in y*x`

$e_3 \equiv$ `let x=2 in let y=x+x in x` $e_4 \equiv$ `let x=y in let x=2 in y*y*x`

$e_5 \equiv$ `let y=(let x=2 in x+x) in y*y` $e_6 \equiv$ `let y=(let x=2 in x+x) in y*y*x`

La evaluación de esas expresiones dará error por problemas de ámbito de alguna variable:

○ Exactamente en una de ellas ⊗ Exactamente en dos de ellas ○ Exactamente en tres de ellas

135. Considérense las expresiones siguientes:

$e_1 \equiv$ `(let x=5 in x+x) + x` $e_2 \equiv$ `let x=2 in let y=x+x in y*y*x`

$e_3 \equiv$ `let y=x+x in let x=2 in y*y*x` $e_4 \equiv$ `let {y=x+x;x=2} in y*y*x`

$e_5 \equiv$ `let y=(let x=2 in x+x) in y*y*x` $e_6 \equiv$ `let y=(let x=2 in 3) in y*y`

La evaluación de esas expresiones dará error por problemas de ámbito de alguna variable:

○ Exactamente en dos de ellas

⊗ Exactamente en tres de ellas

○ Exactamente en cuatro de ellas

136. Considérense las expresiones siguientes:

$e_1 \equiv$ `(let x=5 in x+x) + 5` $e_2 \equiv$ `let x=2 in let y=x+x in y*x`

$e_3 \equiv$ `let x=2 in let y=x+x in x` $e_4 \equiv$ `let x=y in let x=2 in y*y*x`

$e_5 \equiv$ `let y=(let x=x in x+x) in y*y` $e_6 \equiv$ `let y=(let x=2 in x+x) in y*y*x`

De ellas, son sintácticamente erróneas por problemas de ámbito de alguna variable:

○ Exactamente una de ellas ⊗ Exactamente dos de ellas ○ Exactamente tres de ellas

137. Considérense las expresiones siguientes:

$e_1 \equiv$ `(let x=5 in x+x) + (let x=3 in 2*x)` $e_2 \equiv$ `let y=x+x in let x=2 in y*y*x`

$e_3 \equiv$ `let x=2 in let y=x+x in y*y*x` $e_4 \equiv$ `let {y=x+x;x=2} in y*y*x`

$e_5 \equiv$ `[i | i<-[1..j],j<-[0..100],mod j 3 == 0]` $e_6 \equiv$ `[i | j<-[0..100],i<-[1..j],mod j 3 == 0]`

La evaluación de esas expresiones dará error por problemas de ámbito de variables:

○ Exactamente en una de ellas

⊗ Exactamente en dos de ellas

○ Exactamente en tres de ellas