

1. Resumen de funciones predefinidas de Haskell

1. `x + y` es la suma de `x` e `y`.
2. `x - y` es la resta de `x` e `y`.
3. `x / y` es el cociente de `x` entre `y`.
4. `x ^ y` es `x` elevado a `y`.
5. `x == y` se verifica si `x` es igual a `y`.
6. `x /= y` se verifica si `x` es distinto de `y`.
7. `x < y` se verifica si `x` es menor que `y`.
8. `x <= y` se verifica si `x` es menor o igual que `y`.
9. `x > y` se verifica si `x` es mayor que `y`.
10. `x >= y` se verifica si `x` es mayor o igual que `y`.
11. `x && y` es la conjunción de `x` e `y`.
12. `x || y` es la disyunción de `x` e `y`.
13. `x:ys` es la lista obtenida añadiendo `x` al principio de `ys`.
14. `xs ++ ys` es la concatenación de `xs` e `ys`.
15. `xs !! n` es el elemento `n`-ésimo de `xs`.
16. `f . g` es la composición de `f` y `g`.
17. `abs x` es el valor absoluto de `x`.
18. `and xs` es la conjunción de la lista de booleanos `xs`.
19. `ceiling x` es el menor entero no menor que `x`.
20. `chr n` es el carácter cuyo código ASCII es `n`.
21. `concat xss` es la concatenación de la lista de listas `xss`.
22. `const x y` es `x`.
23. `curry f` es la versión curryficada de la función `f`.
24. `div x y` es la división entera de `x` entre `y`.
25. `drop n xs` borra los `n` primeros elementos de `xs`.
26. `dropWhile p xs` borra el mayor prefijo de `xs` cuyos elementos satisfacen el predicado `p`.
27. `elem x ys` se verifica si `x` pertenece a `ys`.
28. `even x` se verifica si `x` es par.
29. `filter p xs` es la lista de elementos de la lista `xs` que verifican el predicado `p`.
30. `flip f x y` es `f y x`.
31. `floor x` es el mayor entero no mayor que `x`.
32. `foldl f e xs` pliega `xs` de izquierda a derecha usando el operador `f` y el valor inicial `e`.
33. `foldr f e xs` pliega `xs` de derecha a izquierda usando el operador `f` y el valor inicial `e`.
34. `fromIntegral x` transforma el número entero `x` al tipo numérico correspondiente.
35. `fst p` es el primer elemento del par `p`.
36. `gcd x y` es el máximo común divisor de `x` e `y`.

- 37. `head xs` es el primer elemento de la lista `xs`.
- 38. `init xs` es la lista obtenida eliminando el último elemento de `xs`.
- 39. `iterate f x` es la lista `[x, f(x), f(f(x)), ...]`.
- 40. `last xs` es el último elemento de la lista `xs`.
- 41. `length xs` es el número de elementos de la lista `xs`.
- 42. `map f xs` es la lista obtenida aplicado `f` a cada elemento de `xs`.
- 43. `max x y` es el máximo de `x` e `y`.
- 44. `maximum xs` es el máximo elemento de la lista `xs`.
- 45. `min x y` es el mínimo de `x` e `y`.
- 46. `minimum xs` es el mínimo elemento de la lista `xs`.
- 47. `mod x y` es el resto de `x` entre `y`.
- 48. `not x` es la negación lógica del booleano `x`.
- 49. `notElem x ys` se verifica si `x` no pertenece a `ys`.
- 50. `null xs` se verifica si `xs` es la lista vacía.
- 51. `odd x` se verifica si `x` es impar.
- 52. `or xs` es la disyunción de la lista de booleanos `xs`.
- 53. `ord c` es el código ASCII del carácter `c`.
- 54. `product xs` es el producto de la lista de números `xs`.
- 55. `read c` es la expresión representada por la cadena `c`.
- 56. `rem x y` es el resto de `x` entre `y`.
- 57. `repeat x` es la lista infinita `[x, x, x, ...]`.
- 58. `replicate n x` es la lista formada por `n` veces el elemento `x`.
- 59. `reverse xs` es la inversa de la lista `xs`.
- 60. `round x` es el redondeo de `x` al entero más cercano.
- 61. `scanr f e xs` es la lista de los resultados de plegar `xs` por la derecha con `f` y `e`.
- 62. `show x` es la representación de `x` como cadena.
- 63. `signum x` es 1 si `x` es positivo, 0 si `x` es cero y -1 si `x` es negativo.
- 64. `snd p` es el segundo elemento del par `p`.
- 65. `splitAt n xs` es `(take n xs, drop n xs)`.
- 66. `sqrt x` es la raíz cuadrada de `x`.
- 67. `sum xs` es la suma de la lista numérica `xs`.
- 68. `tail xs` es la lista obtenida eliminando el primer elemento de `xs`.
- 69. `take n xs` es la lista de los `n` primeros elementos de `xs`.
- 70. `takeWhile p xs` es el mayor prefijo de `xs` cuyos elementos satisfacen el predicado `p`.
- 71. `uncurry f` es la versión cartesiana de la función `f`.
- 72. `until p f x` aplica `f` a `x` hasta que se verifique `p`.
- 73. `zip xs ys` es la lista de pares formado por los correspondientes elementos de `xs` e `ys`.
- 74. `zipWith f xs ys` se obtiene aplicando `f` a los correspondientes elementos de `xs` e `ys`.