Primera unidad

Calcula el doble del valor 'a', pero usa una expresión 'min x z' que no está definida.

```
dobleMe a = a + (min x z)
```

Calcula una operación entre el sucesor de 'b', la división entera de 'a' por 'b' y resta 'b'.

```
funcion1 a b = ((succ b) + div a b) - b
```

Realiza una operación con 'b', 'funcion1' usando el sucesor de 'b' y 'funcion1 3 4'.

```
funcion2 a b = b + funcion1 a (succ b) * succ (funcion1 3 4)
```

Verifica si 'b' es igual a 10 o si 'a' es igual a 5.

```
funcion4 a b = (10*1 == b) || (5*1 == a)
```

Calcula el cuadrado de 'a'.

```
cuadro a = a * a
```

Define una lista con los números del 1 al 5.

```
lista2 = [1,2,3,4,5]
```

Devuelve una lista con operaciones entre 'a' y 'b'.

```
funcion a b = [a,b,a+b,b+2,a+2,b-a,a*2,b+1]
```

Genera una lista combinando valores y operaciones entre 'a' y 'b'.

```
funcion5 a b = [a,b] ++ [a+b,a+a] ++ [b+a,a*2,b*4,min a b]
```

Devuelve una lista con el mínimo, máximo, sucesor y operaciones cuadráticas.

```
fun a b = [min a b, max a b, succ b] ++ [a^2, b^2, succ (min a (b+1)^2)]
```

Calcula el sucesor del mínimo entre 'a' y el cuadrado de 'b'.

```
funi a b = succ (min a (b^2))
```

-- Define una lista tridimensional con números.

```
cubi = [[[2,3,4,5], [3,4,5,90], [4,7,89,9]]]
```

-- Devuelve el mayor entre 'a' y 'b'.

```
mayor a b = if (a > b) then a else b
```

Segunda unidad

Calcula el doble del valor 'a', pero usa una expresión 'min x z' que no está definida.

```
dobleMe a = a + (min x z)
```

Calcula una operación entre el sucesor de 'b', la división entera de 'a' por 'b' y resta 'b'.

```
funcion1 a b = ((succ b) + div a b) - b
```

Realiza una operación con 'b', 'funcion1' usando el sucesor de 'b' y 'funcion1 3 4'.

```
funcion2 a b = b + funcion1 a (succ b) * succ (funcion1 3 4)
```

Verifica si 'b' es igual a 10 o si 'a' es igual a 5.

```
funcion4 a b = (10*1 == b) || (5*1 == a)
```

Calcula el cuadrado de 'a'.

```
cuadro a = a * a
```

Define una lista con los números del 1 al 5.

```
lista2 = [1,2,3,4,5]
```

Devuelve una lista con operaciones entre 'a' y 'b'.

```
funcion a b = [a,b,a+b,b+2,a+2,b-a,a*2,b+1]
```

Genera una lista combinando valores y operaciones entre 'a' y 'b'.

```
funcion5 a b = [a,b] ++ [a+b,a+a] ++ [b+a,a*2,b*4,min a b]
```

Devuelve una lista con el mínimo, máximo, sucesor y operaciones cua dráticas.

```
fun a b = [min a b, max a b, succ b] ++ [a^2, b^2, succ (min a (b+1)^2)]
```

Calcula el sucesor del mínimo entre 'a' y el cuadrado de 'b'.

```
funi a b = succ (min a (b^2))
```

Define una lista tridimensional con números.

```
cubi = [[[2,3,4,5], [3,4,5,90], [4,7,89,9]]]
```

Devuelve el mayor entre 'a' y 'b'.

```
mayor a b = if (a > b) then a else b
```

Calcula la suma del cuadrado de 'a' y el cubo de 'b'.

```
nuevaFuncion a b = (a^2) + (b^3)
```

Tercera unidad

Convierte una lista de mayúsculas a minúsculas.

```
funMay xs = [toLower x | x <- xs]
```

Convierte una lista de minúsculas a mayúsculas.

```
funMin2 xs = [toUpper x | x <- xs]
```

Convierte letras minúsculas a mayúsculas, espacios a guiones bajos y el resto a '#'.

```
funnt cadena = [if elem c ['a'..'z'] then toUpper c else if c == ' ' then '_' else '#' | c <- cadena]
```

Calcula el discriminante de una ecuación cuadrática.

```
dis a b c = (b^2 - 4 * a * c)
```

Calcula la primera raíz de una ecuación cuadrática.

```
x1 a b c = (-b + sqrt(dis a b c)) / (2 * a)
```

Calcula la segunda raíz de una ecuación cuadrática.

```
x2 a b c = (-b - sqrt(dis a b c)) / (2 * a)
```

Resuelve la fórmula general para ecuaciones cuadráticas y da las raíces si el discriminante es positivo.

```
forGeneral a b c = if dis a b c > 0
    then "x1 = " ++ show (x1 a b c) ++ ", x2 = " ++ show (x2 a b c)
    else "no se puede hacer"
```

Resuelve la fórmula general usando guardas.

```
forGeneral2 a b c
  | dis a b c > 0 = show (x1 a b c) ++ show (x2 a b c)
  | otherwise = "no se puede"
```

Calcula una versión alternativa del discriminante.

```
pppp a b c = (b^2 - a * c)
```

Suma dos elementos de una tupla.

```
funTuplas (a, b) = a + b
```

Suma tres elementos de una tupla.

```
funTriplas (a, b, c) = c + a + b
```

Filtra y suma las segundas componentes de tuplas donde la segunda componente sea mayor a 2.

```
funTuLI xs = [a + b | (a, b) <- xs, b > 2]
```

Aplica la fórmula general a una lista de tuplas con coeficientes de ecuaciones cuadráticas.

```
funPF xs = [forGeneral a b c | (a, b, c) <- xs]
```

Calcula la distancia entre dos puntos en un plano.

```
distancia (x1, y1) (x2, y2) = sqrt ((x2 - x1)^2 + (y2 - y1)^2)
```

Filtra las tuplas donde el primer elemento sea mayor a 'n'.

```
filtrarPorPrimerElemento n xs = [(a, b) | (a, b) <- xs, a > n]
```

Filtra y devuelve las letras mayúsculas de una cadena.

```
funFMnn cadena = [c | c <- cadena, c `elem` ['A'..'Z']]
```

Convierte minúsculas a mayúsculas, espacios a guiones bajos y el resto a '+'.

```
funcioMinMay cadena = [if c `elem` ['a'..'z'] then toUpper c else if c == ' ' then '_' else '+' | c <- cadena]
```

Calcula la suma de las longitudes de tres cadenas en cada tupla.

```
funcioEx xs = sum [length a + length b + length c | (a, b, c) <- xs]
```

Convierte minúsculas a mayúsculas, espacios a guiones bajos y el resto a '#'.

```
funpra cadena = [if c `elem` ['a'..'z'] then toUpper c else if c == ' ' then '_' else '#' | c <- cadena]
```

Calcula la suma de las longitudes de tres cadenas en cada tupla.

```
funpraT xs = sum [length a + length b + length c | (a, b, c) <- xs]
```

Calcula la suma de los primeros elementos de una lista de tuplas.

```
funpraTu xs = sum [x | (x, _) <- xs]
```

Genera pares (número, etiqueta) para números divisibles por 3 o 5.

```
bizzBuzz xs = [(x, if x `mod` 15 == 0 then "bizzbuzz" else if x `mod` 3 == 0 then "bizz" else if x `mod` 5 == 0 then "Buzz" else "") | x <- xs, x `mod` 3 == 0 || x `mod` 5 == 0]
```

Calcula el índice de masa corporal (IMC) dado el peso y la altura.

```
imc2 p h = p / (h * h)
```

Evalúa el IMC y devuelve un comentario en función del resultado.

```
bmiTell a b
  | imc2 a b <= 18.5 = "Tienes infrapeso ¿Eres emo?"
  | imc2 a b <= 25.0 = "Supuestamente eres normal... Espero que seas feo."
  | imc2 a b <= 30.0 = "¡Estás gordo! Pierde algo de peso gordito."
  | otherwise = "¡Enhorabuena, eres una ballena!"
```

Devuelve un nombre según un carácter específico.

```
charName 'a' = "Albert"
charName 'b' = "Broseph"
charName 'c' = "Cecil"
charName x = "hola"
```

Devuelve el primer elemento de una tupla de tres elementos.

```
first (x, g, f) = x
```

Suma los primeros tres elementos de una lista.

```
list (a:b:x:xs) = a + b + x
```

Suma el quinto elemento de una lista más 1.

```
list2 (_:_:_:_:x:xs) = x + 1
```

Cuarta unidad

Determina si un número es divisible por 3, 5 o ambos y devuelve un par (número, "resultado")

```
funbiz x
    | mod x 3 == 0 && mod x 5 == 0 = (x, "BizzBuzz")
    | mod x 3 == 0 = (x, "Bizz")
    | mod x 5 == 0 = (x, "Buzz")
    | otherwise = (x, "")
```

Aplica la función funbiz a cada elemento de una lista

```
funn xs = [funbiz x | x <- xs]
```

Calcula el factorial de un número

```
factorial 0 = 1
factorial 1 = 1
factorial x = x * factorial (x - 1)
```

Genera una lista de pares (número, factorial) para cada elemento de una lista

```
factoria1 xs = [(x, factorial x) | x <- xs]
```

Calcula el número de Fibonacci de un número dado

```
fibonacci 0 = 0
```

```
fibonacci 1 = 1
fibonacci x = fibonacci (x - 1) + fibonacci (x - 2)

Genera una lista de números de Fibonacci hasta el índice dado
fibo x = [fibonacci y | y <- [0 .. x]]

Encuentra el máximo elemento de una lista
maximum' [] = error "Esta vacía la lista, intenta de nuevo"
maximum' [x] = x
maximum' (x:xs)
    | x > maxTail = x
    | otherwise   = maxTail
    where maxTail = maximum' xs

Encuentra el mínimo elemento de una lista
minimum' [] = error "Máximo de una lista vacía"
minimum' [x] = x
minimum' (x:xs)
    | x < minTail = x
    | otherwise   = minTail
    where minTail = minimum' xs

Genera una lista de números decrecientes hasta 0
repeat' 0 = [0, 1]
repeat' x = x : repeat' (x - 1)

Determina la suma de los elementos de una lista
sumlista [] = 0
sumlista [x] = x
```

```
sumlista (x:xs) = x + sumlista xs

Determina el producto de los elementos de una lista
productolista [] = 0
productolista [x] = x
productolista (x:xs) = x * productolista xs

Encuentra la longitud de una lista sin usar length
lengthlista [] = 0
lengthlista [x] = 1
lengthlista (x:xs) = 1 + lengthlista xs

Determina si un elemento está en una lista
buscar x [] = False
buscar x (y:xs)
    | x == y = True
    | otherwise = buscar x xs

Reversa una lista
reversa [] = []
reversa (x:xs) = reversa xs ++ [x]

Determina si una lista es un palíndromo
palindromo :: Eq a => [a] -> String
palindromo xs
    | xs == reversa xs = "Es palindromo"
    | otherwise = "No es Palindromo"
```

Duplica cada elemento de una lista

```
listados [] = []
listados (x:xs) = x : x : listados xs
```

Elimina todas las apariciones de un elemento dado en una lista

```
eliminarList _ [] = []
eliminarList y (x:xs)
    | y == x     = eliminarList y xs
    | otherwise = x : eliminarList y xs
```

Genera una lista de enteros desde 1 hasta n

```
listaEnteros 0 = []
listaEnteros n = listaEnteros (n - 1) ++ [n]
```

Genera una lista alternada entre dos números

```
listaDos 0 0 = []
listaDos a b = listaDos (a + 1) (b - 1) ++ [a] ++ [b]
```

Quita las vocales de un texto

```
quitarvocales [] = []
quitarvocales (x:xs)
    | x `elem` ['a', 'e', 'i', 'o', 'u'] = quitarvocales xs
    | otherwise = x : quitarvocales xs
```

Dada una lista de listas, suma los elementos de cada lista interna

```
sumarListas [] = []
sumarListas (xs:xss) = sum xs : sumarListas xss
```

```
Combina todas las listas internas en una sola lista
soloLista [] = []
soloLista (xs:xss) = xs ++ soloLista xss

Quita todas las listas vacías de una lista de listas
eliminarVacias [] = []
eliminarVacias (xs:xss)
    | null xs = eliminarVacias xss
    | otherwise = xs : eliminarVacias xss

Cuenta el número total de listas internas
contarlistas [] = 0
contarlistas (_:xss) = 1 + contarlistas xss

Cuenta el número total de elementos en todas las listas internas
contarlistas2 [] = 0
contarlistas2 (xs:xss) = length xs + contarlistas2 xss

Invierte cada lista interna en una lista de listas
invertirCadaLista [] = []
invertirCadaLista (x:xs) = invertir x : invertirCadaLista xs

Inserta un valor al inicio de cada lista interna y las concatena
concatenarConValor [] _ = []
concatenarConValor (x:xs) val = (val : x) ++ concatenarConValor xs val
```