[https://tio.run/#haskell](https://tio.run/" \l "haskell)

square (x) = x \* x

main = print (square 42)

lst = [2, 3, 5, 7, 11]

total = sum (map (square) lst)

main = print total

Sumar elementos de una lista

sumarLst :: [Integer] -> Integer

sumarLst [] = 0

sumarLst (a:as) = a + sumarLst as

main = print (sumarLst [1,2,3])

sumatoria:: Integer ->Integer

sumatoria 0 = 0

sumatoria n = n + sumatoria (n-1)

main = print (sumatoria 10)

fibonacci :: Integer -> Integer

fibonacci 0 = 0

fibonacci 1 = 1

fibonacci n = fibonacci (n-1) + fibonacci(n-2)

main = print (fibonacci 7)

hailstone :: Integer -> Integer

hailstone n

| n `mod` 2 == 0 = n `div` 2

| otherwise = 3 \* n + 1

main = print (hailstone 3)

factorial :: Integer -> Integer

factorial 0 = 1

factorial n = n \* factorial (n-1)

main = print (factorial 5)

misterio :: [Integer] -> Integer

misterio [] = 0

misterio (x:xs) = 1 + misterio xs

main = print (misterio [1, 2, 3, 4, 5])

misterioDos :: [Integer] -> [Integer]

misterioDos [] = []

misterioDos [x] = [x \* x]

misterioDos (x:xs) = (x \* x) : (misterioDos xs)

main = print (misterioDos [1, 2, 3, 4, 5])

contar :: Integer -> Integer

contar x = case (x `mod` 2) of

0 -> 1

\_ -> 0

contarPares :: [Integer] -> Integer

contarPares [] = 0

contarPares (x:xs) = (contar x) + contarPares xs

main = print (contarPares [1,2,3,4,5,6])

quitaTres :: [a] -> [a]

quitaTres (\_:\_:\_:xs) = xs

quitaTres \_ = []

main = print(quitaTres [1,2,3,4,5])

queue :: a -> [a] -> [a]

queue x xs = xs++[x]

dequeue :: [a] -> a

dequeue(x:\_) = x

push :: a -> [a] -> [a]

push x xs = (x:xs)

pop :: [a] -> a

pop (x:\_) = x

Dado dos números enteros A y B, implemente una función que retorne la división entera de ambos  por el método de las restas sucesivas

division :: Integer -> Integer -> Integer

division x y

| x < y = 0

| otherwise = 1 + (division (x-y) y)

main = print (division 10 2)

Dado dos números enteros A y B encuentre el MCD (máximo  común divisor) entre ambos.

hallarMcd :: Integer -> Integer -> Integer -> Integer

hallarMcd a b c

| c < 1 = 0

| (a `mod` c == 0) && (b `mod` c == 0) = c

| otherwise = hallarMcd a b (c-1)

mcd :: Integer -> Integer -> Integer

mcd x y

| x > y = hallarMcd x y y

| otherwise = hallarMcd x y x

main = print (mcd 2 8)

Escribir una función para hallar la potencia de un número.

potencia :: Integer -> Integer ->Integer

potencia x y

| y < 1 = 1

| otherwise = x \* potencia x (y-1)

main = print(potencia 2 4)

Definir una función menor que devuelve el menor de sus dos argumentos enteros.

menor :: Integer -> Integer -> Integer

menor x y = case (x > y) of

True -> y

False -> x

main = print(menor 22 25)

Definir una función maxDeTres que devuelve el máximo de sus argumentos enteros

maxDeTres :: Integer -> Integer -> Integer -> Integer

maxDeTres x y z

| (x > y) && (x > z) = x

| (y > x) && (y > z) = y

| (z > x) && (z > y) = z

| z > x = z

| x > z = x

| y > x = y

| otherwise = x

main = print(maxDeTres 7 6 6)

Implemente una función recursiva para calcular el factorial de un número

factorial :: Integer -> Integer

factorial x = case (x < 1) of

True -> 1

False -> x \* factorial (x-1)

main = print(factorial 5)