rectangulo.py

def area\_rectangulo(b,a):

    Esta linea define una función llamada area\_rectangulo que calcula el área de un rectángulo.

    La función toma dos parámetros: b que representa la base del rectángulo, y a que representa la altura del rectángulo.

    A continuación, se realizan comparaciones para asegurarse de que las variables

    if not isinstance(b,(int,float)):

        raise ValueError('Base debe ser numerico')     un número entero o de punto flotante

    if not isinstance(a,(int,float)):

        raise ValueError('Altura debe ser numerico')    es un número entero o de punto flotante

    if b < 0:

        raise ValueError('Base debe ser un numero positivo')  comprueba si b es menor que cero

    if a < 0:

        raise ValueError('Altura debe ser un numero')  comprueba si a es menor que cero

    return b\*a

    Si pasan todas las validaciones, la función calcula el área del rectángulo multiplicando la base por la altura y devuelve el resultado.

permutacion.py

def calcular\_perm(n, r):

    if not isinstance(n, int) or not isinstance(r, int):

        raise ValueError('Ambos valores deben ser enteros')

    if n < 0 or r < 0:

        raise ValueError('Ambos valores deben ser positivos')

    if r > n:

        raise ValueError('El valor de r no puede ser mayor que n')

    stas primeras líneas definen una función calcular permutaciones

   que toma dos variableds: n y r. Luego, se realizan comparaciones para

    segurarse de que n y r sean números enteros positivos y que r no sea mayor que n.

    resultado = 1       Aquí creamos una variable resultadoy la inicializamos en 1. Luego, utilizamos un

    for i in range(n, n - r, -1): para iterar desde n hasta n - r + 1, restando 1 en cada iteración. Dentro del bucle, multiplicamos

        resultado \*= i resultado por los valores de i en cada iteración. Esto nos da el resultado de todos los números desde n hasta n - r + 1.

    return resultado finalmente, se devuelve el valor de resultado.

Combinación.py

def calcular\_combi(z, r):

    if not isinstance(z, int) or not isinstance(r, int):

        raise ValueError('Ambos valores deben ser enteros')

    if z < 0 or r < 0:

        raise ValueError('Ambos valores deben ser positivos')

    if r > z:

        raise ValueError('El valor de r no puede ser mayor que z')

    Estas lineas definen una función llamada calcular\_combinaciones que llama dos variables como z y r.

    # Se realizan comparaciones para asegurarse de que z y r sean números enteros positivos y que r no sea mayor que z.

    resultado = 1          Aquí creamos una variable resultado y la inicializamos en 1. Luego, utilizamos un for para iterar desde z

    for i in range(z, z - r, -1): hasta z - r + 1, restando 1 en cada iteración. Dentro del bucle, multiplicamos resultadopor los valores de ien cada iteración.

        resultado \*= i   Esto nos da el resultado de todos los números desde n hasta z - r + 1.

    for i in range(1, r + 1): Después, utilizamos otro bucle for para iterar desde 1 hasta r + 1,

        resultado //= i

    incrementando en 1 en cada iteración. Dentro de este segundo bucle, dividimos resultado por los valores de ien cada iteración.

    Esto nos permite calcular el resultado de todos los números desde 1 hasta r.

    return resultado

    Finalmente, se devuelve el valor de resultado.

Per\_test.py

import unittest

from areas.permutacion import calcular\_perm

Método de prueba para verificar que calcular\_perm funcione correctamente con valores enteros

def test\_calcular\_perm\_enteros(self):

n = 5

r = 3

resultado = calcular\_perm(n, r)

Comprueba que el resultado sea igual a 60

self.assertEqual(resultado, 60)

Método de prueba para verificar que calcular\_perm genere una excepción ValueError con valores negativos

def test\_calcular\_perm\_valores\_negativos(self):

with self.assertRaises(ValueError) as exc:

calcular\_perm(-2, 3)

Comprueba que la excepción generada tenga el mensaje adecuado

self.assertEqual(str(exc.exception), 'Ambos valores deben ser positivos')

Método de prueba para verificar que calcular\_perm genere una excepción ValueError con r mayor que n

def test\_calcular\_perm\_r\_mayor\_que\_n(self):

with self.assertRaises(ValueError) as exc:

calcular\_perm(3, 5)

Comprueba que la excepción generada tenga el mensaje adecuado

self.assertEqual(str(exc.exception), 'El valor de r no puede ser mayor que n')

Método de prueba para verificar que calcular\_perm genere una excepción ValueError con valores no enteros

def test\_calcular\_perm\_no\_enteros(self):

with self.assertRaises(ValueError) as exc:

calcular\_perm(2, 'Hola')

Comprueba que la excepción generada tenga el mensaje adecuado

self.assertEqual(str(exc.exception), 'Ambos valores deben ser enteros')

Ejecuta las pruebas unitarias si este archivo es el programa principal

if \_\_name\_\_ == '\_\_main\_\_':

unittest.main()

Comb\_test.py

import unittest

from areas.combinacion import calcular\_combi

# Método de prueba para verificar que calcular\_combi funcione correctamente con valores enteros

def test\_calcular\_combi\_enteros(self):

z = 5

r = 3

resultado = calcular\_combi(z, r)

Comprueba que el resultado sea igual a 10

self.assertEqual(resultado, 10)

Método de prueba para verificar que calcular\_combi genere una excepción ValueError con valores negativos

def test\_calcular\_combi\_valores\_negativos(self):

with self.assertRaises(ValueError) as exc:

calcular\_combi(-2, 3)

Comprueba que la excepción generada tenga el mensaje adecuado

self.assertEqual(str(exc.exception), 'Ambos valores deben ser positivos')

Método de prueba para verificar que calcular\_combi genere una excepción ValueError con r mayor que z

def test\_calcular\_combi\_r\_mayor\_que\_z(self):

with self.assertRaises(ValueError) as exc:

calcular\_combi(3, 5)

Comprueba que la excepción generada tenga el mensaje adecuado

self.assertEqual(str(exc.exception), 'El valor de r no puede ser mayor que z')

Método de prueba para verificar que calcular\_combi genere una excepción ValueError con valores no enteros

def test\_calcular\_combi\_no\_enteros(self):

with self.assertRaises(ValueError) as exc:

calcular\_combi(2, 'Hola')

Comprueba que la excepción generada tenga el mensaje adecuado

self.assertEqual(str(exc.exception), 'Ambos valores deben ser enteros')

Ejecuta las pruebas unitarias si este archivo es el programa principal

if \_\_name\_\_ == '\_\_main\_\_':

unittest.main()

rec\_test.py

import unittest

from areas.rectangulo import area\_rectangulo

Método de prueba para verificar que area\_rectangulo funcione correctamente con valores enteros

def test\_area\_rectangulo\_enteros(self):

base = 2

altura = 4

area = area\_rectangulo(base, altura)

Comprueba que el área calculada sea igual a 8

self.assertEqual(area, 8)

Método de prueba para verificar que area\_rectangulo funcione correctamente con valores de punto flotante

def test\_area\_rectangulo\_floats(self):

base = 1

altura = 3.1416

area = area\_rectangulo(base, altura)

Comprueba que el área calculada sea igual a 3.1416

self.assertEqual(area, 3.1416)

Método de prueba para verificar que area\_rectangulo genere una excepción ValueError con base no numérica

def test\_area\_rectangulo\_base\_invalida(self):

base = 'Hola'

altura = 3

with self.assertRaises(ValueError) as exc:

area\_rectangulo(base, altura)

Comprueba que la excepción generada tenga el mensaje adecuado

self.assertEqual(str(exc.exception), 'Base debe ser numerico')

Método de prueba para verificar que area\_rectangulo genere una excepción ValueError con altura negativa

def test\_area\_rectangulo\_altura\_negativa(self):

base = 2

altura = -2

with self.assertRaises(ValueError) as exc:

area\_rectangulo(base, altura)

Comprueba que la excepción generada tenga el mensaje adecuado

self.assertEqual(str(exc.exception), 'Altura debe ser un numero positivo')

Ejecuta las pruebas unitarias si este archivo es el programa principal

if \_\_name\_\_ == '\_\_main\_\_':

unittest.main()