**Instituto Superior del Milagro**

**Tecnicatura Superior en Análisis de Sistemas**

**Profesora: Ramiro, Medina**

**PROGRAMACIÓN III**

**PROGRAMACION ORIENTADA A OBJETOS**

Texto

Descripción generada automáticamente

**Alumna:**

**Castillo Sonia Verónica**

**Ejercicio 1:** Crear una clase llamada Persona con los atributos: nombre, edad y DNI. Diseñar los siguientes métodos para la clase:

● Un constructor, donde los datos pueden estar vacíos;

● Mostrar los datos de las personas;

● Determinar si es mayor de edad.

class Persona:

    def \_\_init\_\_(self, nombre="", edad=0, dni=""):

        self.nombre = nombre

        self.edad = edad

        self.dni = dni

    def mostrar\_datos(self):

        print(f"Nombre: {self.nombre}")

        print(f"Edad: {self.edad}")

        print(f"DNI: {self.dni}")

    def es\_mayor\_de\_edad(self):

        return self.edad >= 18

if \_\_name\_\_ == "\_\_main\_\_":

    persona1 = Persona("Juan", 20, "12345678A")

    persona1.mostrar\_datos()

    if persona1.es\_mayor\_de\_edad():

        print("Es mayor de edad.")

    else:

        print("No es mayor de edad.")

**PS C:\Users\VERONICA CASTILLO> & "C:/Users/VERONICA CASTILLO/AppData/Local/Programs/Python/Python312/python.exe" "c:/Users/VERONICA CASTILLO/Downloads/ejercicio 1.py"**

**Nombre: Juan**

**Edad: 20**

**DNI: 12345678A**

**Es mayor de edad.**

**PS C:\Users\VERONICA CASTILLO>**

**Ejercicio 2:** Crear una clase llamada Cuenta que tendrá los atributos: titular (que es una persona) y cantidad (puede tener decimales). El titular será obligatorio y la cantidad es opcional. Construye los siguientes métodos para la clase:

● Un constructor;

● El atributo cuenta no se puede modificar directamente, sólo ingresando o retirando dinero;

● mostrar(): Muestra los datos de la cuenta;

● ingresar(cantidad): se ingresa una cantidad a la cuenta, si la cantidad introducida es negativa, no se hará nada;

● retirar(cantidad): se retira una cantidad a la cuenta. La cuenta puede estar en números rojos.

class Persona:

    def \_\_init\_\_(self, nombre="", edad=0, dni=""):

        self.nombre = nombre

        self.edad = edad

        self.dni = dni

    def mostrar\_datos(self):

        print(f"Nombre: {self.nombre}")

        print(f"Edad: {self.edad}")

        print(f"DNI: {self.dni}")

    def es\_mayor\_de\_edad(self):

        return self.edad >= 18

class Cuenta:

    def \_\_init\_\_(self, titular, cantidad=0.0):

        if not isinstance(titular, Persona):

            raise ValueError("El titular debe ser una instancia de la clase Persona.")

        self.\_\_titular = titular

        self.\_\_cantidad = float(cantidad)

    def mostrar(self):

        print("Datos de la cuenta:")

        self.\_\_titular.mostrar\_datos()

        print(f"Cantidad: {self.\_\_cantidad}")

    def ingresar(self, cantidad):

        if cantidad > 0:

            self.\_\_cantidad += cantidad

        else:

            print("La cantidad a ingresar debe ser positiva.")

    def retirar(self, cantidad):

        self.\_\_cantidad -= cantidad

    def cantidad(self):

        return self.\_\_cantidad

    def cantidad(self, value):

        print("No se puede modificar la cantidad directamente. Use los métodos ingresar o retirar.")

    def titular(self):

        return self.\_\_titular

    def titular(self, value):

        print("No se puede modificar el titular directamente.")

if \_\_name\_\_ == "\_\_main\_\_":

    titular = Persona("Juan", 30, "12345678A")

    cuenta = Cuenta(titular, 100.0)

    cuenta.mostrar()

    cuenta.ingresar(50)

    cuenta.mostrar()

    cuenta.ingresar(-20)

    cuenta.retirar(30)

    cuenta.mostrar()

    cuenta.cantidad = 500

**Datos de la cuenta:**

**Nombre: Juan**

**Edad: 30**

**DNI: 12345678A**

**Cantidad: 150.0**

**La cantidad a ingresar debe ser positiva.**

**Datos de la cuenta:**

**Nombre: Juan**

**Edad: 30**

**DNI: 12345678A**

**Cantidad: 120.0**

**PS C:\Users\VERONICA CASTILLO>**

**Ejercicio 3:** Diseñar una clase Cuentas Bancarias que tiene como atributo una Lista de cuentas. Añadir las siguientes funcionalidades:

● Crear constructor;

● Saldo\_deudor: Muestra el saldo deudor total de todas las cuentas con saldo negativo;

● Mostrar todas las cuentas con saldo negativo con el valor adeudado.

**Ejercicio 4:** Escribir una clase en python que obtenga todas las posibles sublistas únicas de una lista de números enteros distintos. No tener en cuenta una lista vacía.

● Entrada: [4, 5, 6]

● Salida: [[6], [5], [5, 6], [4], [4, 6], [4, 5], [4, 5, 6]]

class SublistasUnicas:

    def \_\_init\_\_(self, lista):

        self.lista = lista

        self.sublistas = []

    def obtener\_sublistas(self):

        self.sublistas = []

        self.\_generar\_sublistas([], 0)

        return self.sublistas

    def \_generar\_sublistas(self, actual, indice):

        if actual:

            self.sublistas.append(actual)

        for i in range(indice, len(self.lista)):

            self.\_generar\_sublistas(actual + [self.lista[i]], i + 1)

if \_\_name\_\_ == "\_\_main\_\_":

    lista = [4, 5, 6]

    generador\_sublistas = SublistasUnicas(lista)

    resultado = generador\_sublistas.obtener\_sublistas()

    print(resultado)

**PS C:\Users\VERONICA CASTILLO> & "C:/Users/VERONICA CASTILLO/AppData/Local/Programs/Python/Python312/python.exe" "c:/Users/VERONICA CASTILLO/Downloads/ejercicio 4.py"**

**[[4], [4, 5], [4, 5, 6], [4, 6], [5], [5, 6], [6]]**

**PS C:\Users\VERONICA CASTILLO>**

**Ejercicio 5:** Escribir una clase en python que encuentre un par de elementos de una matriz dada cuya suma es igual a un número determinado. Se debe indicar la posición de los números que cumplen la condición.

● Entrada: números = [10,20,10,40,50,60,70], objetivo=50

● Salida: 3, 4

class BuscadorPares:

    def encontrar\_pares(self, numeros, objetivo):

        mapa\_indices = {}

        for i, num in enumerate(numeros):

            complemento = objetivo - num

            if complemento in mapa\_indices:

                return mapa\_indices[complemento], i

            mapa\_indices[num] = i

        return None

if \_\_name\_\_ == "\_\_main\_\_":

    numeros = [10, 20, 10, 40, 50, 60, 70]

    objetivo = 50

    buscador = BuscadorPares()

    resultado = buscador.encontrar\_pares(numeros, objetivo)

    if resultado:

        print(f"Posiciones de los números que suman {objetivo}: {resultado[0]}, {resultado[1]}")

    else:

        print("No se encontró ningún par de números que sumen el objetivo.")

**PS C:\Users\VERONICA CASTILLO> & "C:/Users/VERONICA CASTILLO/AppData/Local/Programs/Python/Python312/python.exe" "c:/Users/VERONICA CASTILLO/Downloads/ejercicio 5.py"**

**Posiciones de los números que suman 50: 2, 3**

**PS C:\Users\VERONICA CASTILLO>**

**Ejercicio 6:** Crear una clase llamada Rectangulo con dos puntos (inicial y final) que formarán la diagonal del rectángulo (los puntos se formarán con una lista de 2 elementos).

● Añade un método constructor para crear ambos puntos fácilmente, si no se envían se crearán dos puntos en el origen por defecto.

● Añade al rectángulo un método llamado base que muestre la base.

● Añade al rectángulo un método llamado altura que muestre la altura.

● Añade al rectángulo un método llamado área que muestre el área.

class Punto:

    def \_\_init\_\_(self, x=0, y=0):

        self.x = x

        self.y = y

class Rectangulo:

    def \_\_init\_\_(self, punto1=None, punto2=None):

        if punto1 is None:

            punto1 = Punto(0, 0)

        if punto2 is None:

            punto2 = Punto(1, 1)

        self.punto1 = punto1

        self.punto2 = punto2

    def base(self):

        return abs(self.punto2.x - self.punto1.x)

    def altura(self):

        return abs(self.punto2.y - self.punto1.y)

    def area(self):

        return self.base() \* self.altura()

if \_\_name\_\_ == "\_\_main\_\_":

    punto\_inicio = Punto(1, 1)

    punto\_fin = Punto(4, 5)

    rectangulo1 = Rectangulo(punto\_inicio, punto\_fin)

    print(f"Base del rectángulo: {rectangulo1.base()}")

    print(f"Altura del rectángulo: {rectangulo1.altura()}")

    print(f"Área del rectángulo: {rectangulo1.area()}")

    rectangulo2 = Rectangulo()

    print(f"\nBase del rectángulo por defecto: {rectangulo2.base()}")

    print(f"Altura del rectángulo por defecto: {rectangulo2.altura()}")

    print(f"Área del rectángulo por defecto: {rectangulo2.area()}")

**PS C:\Users\VERONICA CASTILLO> & "C:/Users/VERONICA CASTILLO/AppData/Local/Programs/Python/Python312/python.exe" "c:/Users/VERONICA CASTILLO/Downloads/ejercicio 6.py"**

**Base del rectángulo: 3**

**Altura del rectángulo: 4**

**Área del rectángulo: 12**

**Base del rectángulo por defecto: 1**

**Altura del rectángulo por defecto: 1**

**Área del rectángulo por defecto: 1**

**PS C:\Users\VERONICA CASTILLO>**

**Ejercicio 7:** Crear una clase llamada Punto con 2 coordenadas (atributos) X e Y:

● Añade un método constructor para crear puntos fácilmente. Si no se recibe una coordenada, su valor será cero.

● Sobreescribe el método string, para que al imprimir por pantalla un punto aparezca en formato (X,Y)

● Añade un método llamado cuadrante que indique a qué cuadrante pertenece el punto, teniendo en cuenta que si X == 0 e Y != 0 se sitúa sobre el eje Y, si X != 0 e Y == 0 se sitúa sobre el eje X y si X == 0 e Y == 0 está sobre el origen.

● Añade un método llamado vector, que tome otro punto y calcule el vector (se representaría como la diferencia entre las coordenadas del segundo punto respecto al primero) resultante entre los dos puntos.

import math

class Punto:

    def \_\_init\_\_(self, x=0, y=0):

        self.x = x

        self.y = y

    def \_\_str\_\_(self):

        return f"({self.x},{self.y})"

    def cuadrante(self):

        if self.x > 0 and self.y > 0:

            return "Cuadrante I"

        elif self.x < 0 and self.y > 0:

            return "Cuadrante II"

        elif self.x < 0 and self.y < 0:

            return "Cuadrante III"

        elif self.x > 0 and self.y < 0:

            return "Cuadrante IV"

        elif self.x == 0 and self.y != 0:

            return "Sobre el eje Y"

        elif self.x != 0 and self.y == 0:

            return "Sobre el eje X"

        else:

            return "Está sobre el origen"

    def vector(self, otro\_punto):

        dx = otro\_punto.x - self.x

        dy = otro\_punto.y - self.y

        return (dx, dy)

if \_\_name\_\_ == "\_\_main\_\_":

    punto1 = Punto(3, 4)

    punto2 = Punto(-2, 1)

    punto3 = Punto(0, 0)

    punto4 = Punto(0, 5)

    print(f"Punto 1: {punto1}")

    print(f"Punto 2: {punto2}")

    print(f"Punto 3: {punto3}")

    print(f"Punto 4: {punto4}")

    print(f"\nPunto 1 está en: {punto1.cuadrante()}")

    print(f"Punto 2 está en: {punto2.cuadrante()}")

    print(f"Punto 3 está en: {punto3.cuadrante()}")

    print(f"Punto 4 está en: {punto4.cuadrante()}")

    vector\_resultante = punto1.vector(punto2)

    print(f"\nVector resultante entre Punto 1 y Punto 2: {vector\_resultante}")

**PS C:\Users\VERONICA CASTILLO> & "C:/Users/VERONICA CASTILLO/AppData/Local/Programs/Python/Python312/python.exe" "c:/Users/VERONICA CASTILLO/Downloads/ejercicio 7.py"**

**Punto 1: (3,4)**

**Punto 2: (-2,1)**

**Punto 3: (0,0)**

**Punto 4: (0,5)**

**Punto 1 está en: Cuadrante I**

**Punto 2 está en: Cuadrante II**

**Punto 3 está en: Está sobre el origen**

**Punto 4 está en: Sobre el eje Y**

**Vector resultante entre Punto 1 y Punto 2: (-5, -3)**

**PS C:\Users\VERONICA CASTILLO>**

**Ejercicio 8:** Crear una clase Fecha con atributos para el día, el mes y el año de la fecha. Incluye, al menos, los siguientes métodos:

● Constructor predeterminado con el 1-1-1900 como fecha por defecto.

● año\_bisiesto(): indicará si el año de la fecha es bisiesto o no.

● dias\_mes(int): devolverá el número de días del mes que se le indique (para el año de la fecha).

● validar(): comprobará si la fecha es correcta (entre el 1-1-1900 y el 31-12- 2050); si el día no es correcto, lo pondrá a 1; si el mes no es correcto, lo pondrá a 1; y si el año no es correcto, lo pondrá a 1900. Será un método auxiliar (privado).

● Definir métodos getter y setter.

● hoy(): devuelve la fecha actual.

● corta(): mostrará la fecha en formato corto (02-09-2003).

● larga(): mostrará la fecha en formato largo, empezando por el día de la semana (martes 2 de septiembre de 2003). los días que se indiquen desde el 1-1-1900.

import datetime

class Fecha:

    def \_\_init\_\_(self, dia=1, mes=1, anio=1900):

        self.\_\_dia = dia

        self.\_\_mes = mes

        self.\_\_anio = anio

        self.validar()

    def \_\_str\_\_(self):

        return f"{self.\_\_dia:02}-{self.\_\_mes:02}-{self.\_\_anio}"

    def año\_bisiesto(self):

        if (self.\_\_anio % 4 == 0 and self.\_\_anio % 100 != 0) or (self.\_\_anio % 400 == 0):

            return True

        else:

            return False

    def dias\_mes(self, mes=None):

        if mes is None:

            mes = self.\_\_mes

        if mes == 2:

            if self.año\_bisiesto():

                return 29

            else:

                return 28

        elif mes in [4, 6, 9, 11]:

            return 30

        else:

            return 31

    def validar(self):

        # Ajustar año

        if self.\_\_anio < 1900 or self.\_\_anio > 2050:

            self.\_\_anio = 1900

        # Ajustar mes

        if self.\_\_mes < 1 or self.\_\_mes > 12:

            self.\_\_mes = 1

        # Ajustar día

        if self.\_\_dia < 1 or self.\_\_dia > self.dias\_mes():

            self.\_\_dia = 1

    def hoy(self):

        hoy = datetime.date.today()

        return Fecha(hoy.day, hoy.month, hoy.year)

    def corta(self):

        return f"{self.\_\_dia:02}-{self.\_\_mes:02}-{self.\_\_anio}"

    def larga(self):

        nombre\_meses = [

            "enero", "febrero", "marzo", "abril", "mayo", "junio",

            "julio", "agosto", "septiembre", "octubre", "noviembre", "diciembre"

        ]

        fecha\_actual = datetime.date(self.\_\_anio, self.\_\_mes, self.\_\_dia)

        dia\_semana = fecha\_actual.strftime("%A")

        return f"{dia\_semana} {self.\_\_dia} de {nombre\_meses[self.\_\_mes - 1]} de {self.\_\_anio}"

    def get\_dia(self):

        return self.\_\_dia

    def set\_dia(self, dia):

        self.\_\_dia = dia

        self.validar()

    def get\_mes(self):

        return self.\_\_mes

    def set\_mes(self, mes):

        self.\_\_mes = mes

        self.validar()

    def get\_anio(self):

        return self.\_\_anio

    def set\_anio(self, anio):

        self.\_\_anio = anio

        self.validar()

if \_\_name\_\_ == "\_\_main\_\_":

    fecha1 = Fecha(31, 12, 2023)

    print(f"Fecha 1 en formato largo: {fecha1.larga()}")

    fecha1.set\_dia(15)

    fecha1.set\_mes(2)

    print(f"Fecha 1 en formato corto después de cambios: {fecha1.corta()}")

    fecha\_actual = Fecha().hoy()

    print(f"Fecha actual en formato largo: {fecha\_actual.larga()}")

**PS C:\Users\VERONICA CASTILLO> & "C:/Users/VERONICA CASTILLO/AppData/Local/Programs/Python/Python312/python.exe" "c:/Users/VERONICA CASTILLO/Downloads/ejercicio 8.py"**

**Fecha 1 en formato largo: Sunday 31 de diciembre de 2023**

**Fecha 1 en formato corto después de cambios: 15-02-2023**

**Fecha actual en formato largo: Thursday 4 de julio de 2024**

**PS C:\Users\VERONICA CASTILLO>**

**MÉTODOS DE CLASES Y MÉTODOS ESPECIALES.OBJETOS DENTRO DE OBJETOS.**

**Ejercicio 9:** Crear una clase de números enteros y redefinir las funciones elementales (+ , - , \*, /) usando los métodos denominados mágicos.

● Sobrecargar el operador "/": si el divisor es 0 devuelve como resultado 0 y un msj de error.

class NumeroEntero:

    def \_\_init\_\_(self, numero):

        self.numero = numero

    def \_\_str\_\_(self):

        return f"NumeroEntero({self.numero})"

    def \_\_add\_\_(self, otro):

        return self.numero + otro.numero

    def \_\_sub\_\_(self, otro):

        return self.numero - otro.numero

    def \_\_mul\_\_(self, otro):

        return self.numero \* otro.numero

    def \_\_truediv\_\_(self, otro):

        if otro.numero == 0:

            print("Error: División por cero.")

            return 0

        else:

            return self.numero / otro.numero

if \_\_name\_\_ == "\_\_main\_\_":

    num1 = NumeroEntero(10)

    num2 = NumeroEntero(5)

    resultado\_suma = num1 + num2

    print(f"Suma: {resultado\_suma}")

    resultado\_resta = num1 - num2

    print(f"Resta: {resultado\_resta}")

    resultado\_mult = num1 \* num2

    print(f"Multiplicación: {resultado\_mult}")

    resultado\_div = num1 / num2

    print(f"División: {resultado\_div}")

    num3 = NumeroEntero(0)

    resultado\_div\_cero = num1 / num3

    print(f"División por cero: {resultado\_div\_cero}")

**PS C:\Users\VERONICA CASTILLO> & "C:/Users/VERONICA CASTILLO/AppData/Local/Programs/Python/Python312/python.exe" "c:/Users/VERONICA CASTILLO/Downloads/ejercicio 9.py"**

**Suma: 15**

**Resta: 5**

**Multiplicación: 50**

**División: 2.0**

**Error: División por cero.**

**División por cero: 0**

**PS C:\Users\VERONICA CASTILLO>**

**Ejercicio 10:** Usando la clase Personas del ejercicio nro 1 del práctico anterior, crear 2 clases, Alumnos y Profesores. Asignar notas a los alumnos de acuerdo a las siguientes materias (lengua, matemática, historia, geografía) y una comisión (A, B, C) de acuerdo al promedio de sus notas (A si es >=8, B si es >=6 y <B, C si es <6) A los profesores asignar como máximo 1 materia.

Los datos personales de alumnos y profesores no pueden ser nulos

● Asignar comisión al alumno. No se puede tener acceso de forma directa al atributo comisión. Si el alumno es mayor de 17 asignar a una comisión especial N ● Mostrar todas las notas identificando a qué materia pertenece

● Los profesores pueden evaluar y asignar notas a los alumnos.

class Persona:

    def \_\_init\_\_(self, nombre="", edad=0, dni=""):

        self.nombre = nombre

        self.edad = edad

        self.dni = dni

    def mostrar\_datos(self):

        print(f"Nombre: {self.nombre}")

        print(f"Edad: {self.edad}")

        print(f"DNI: {self.dni}")

    def es\_mayor\_de\_edad(self):

        return self.edad >= 18

class Alumno(Persona):

    def \_\_init\_\_(self, nombre="", edad=0, dni="", comision=""):

        super().\_\_init\_\_(nombre, edad, dni)

        self.\_\_comision = comision

        self.\_\_notas = {

            'lengua': None,

            'matemática': None,

            'historia': None,

            'geografía': None

        }

    def asignar\_comision(self):

        if self.edad > 17:

            self.\_\_comision = 'N'

    def asignar\_nota(self, materia, nota):

        if materia in self.\_\_notas:

            self.\_\_notas[materia] = nota

    def mostrar\_notas(self):

        for materia, nota in self.\_\_notas.items():

            print(f"{materia}: {nota}")

    def comision(self):

        return self.\_\_comision

class Profesor(Persona):

    def \_\_init\_\_(self, nombre="", edad=0, dni="", materia=""):

        super().\_\_init\_\_(nombre, edad, dni)

        self.\_\_materia = materia

    def evaluar(self, alumno, materia, nota):

        alumno.asignar\_nota(materia, nota)

    def asignar\_materia(self, materia):

        self.\_\_materia = materia

    def mostrar\_materia\_asignada(self):

        print(f"Profesor {self.nombre} asignado a la materia: {self.\_\_materia}")

alumno1 = Alumno("Juan", 18, "12345678A")

alumno1.asignar\_comision()

alumno1.mostrar\_datos()

print(f"Comisión: {alumno1.comision}")

alumno1.asignar\_nota('lengua', 8)

alumno1.asignar\_nota('matemática', 7)

alumno1.mostrar\_notas()

profesor1 = Profesor("María", 35, "87654321B")

profesor1.asignar\_materia("lengua")

profesor1.mostrar\_materia\_asignada()

profesor1.evaluar(alumno1, 'lengua', 9)

alumno1.mostrar\_notas()

**PS C:\Users\VERONICA CASTILLO> & "C:/Users/VERONICA CASTILLO/AppData/Local/Programs/Python/Python312/python.exe" "c:/Users/VERONICA CASTILLO/Downloads/ejercicio 10.py"**

**Nombre: Juan**

**Edad: 18**

**DNI: 12345678A**

**Comisión: <bound method Alumno.comision of <\_\_main\_\_.Alumno object at 0x000001F903EF86E0>>**

**lengua: 8**

**matemática: 7**

**historia: None**

**geografía: None**

**Profesor María asignado a la materia: lengua**

**lengua: 9**

**matemática: 7**

**historia: None**

**geografía: None**

**PS C:\Users\VERONICA CASTILLO> & "C:/Users/VERONICA CASTILLO/AppData/Local/Programs/Python/Python312/python.exe" "c:/Users/VERONICA CASTILLO/Downloads/ejercicio 1.py"**

**Ejercicio 11:** Crear una clase llamada Rectangulo con dos puntos (inicial y final) que formarán la diagonal del rectángulo (los puntos se formarán con una lista de 2 elementos). Usar la clase punto como superclase.

● Añadir un método constructor para crear ambos puntos fácilmente, si no se envían se crearán dos puntos en el origen por defecto.

● Añadir al rectángulo un método llamado base que muestre la base.

● Añadir al rectángulo un método llamado altura que muestre la altura.

● Añadir al rectángulo un método llamado área que muestre el área.

● Dibujar el rectángulo, usar "matplotlib".

class Punto:

    def \_\_init\_\_(self, x=0, y=0):

        self.x = x

        self.y = y

    def \_\_str\_\_(self):

        return f"({self.x},{self.y})"

import matplotlib.pyplot as plt

import matplotlib.patches as patches

class Rectangulo(Punto):

    def \_\_init\_\_(self, punto\_inicial=None, punto\_final=None):

        if punto\_inicial is None:

            punto\_inicial = Punto(0, 0)

        if punto\_final is None:

            punto\_final = Punto(1, 1)

        super().\_\_init\_\_(punto\_inicial.x, punto\_inicial.y)

        self.punto\_final = punto\_final

    def base(self):

        return abs(self.punto\_final.x - self.x)

    def altura(self):

        return abs(self.punto\_final.y - self.y)

    def area(self):

        return self.base() \* self.altura()

    def dibujar(self):

        fig, ax = plt.subplots()

        base = self.base()

        altura = self.altura()

        rect = patches.Rectangle((self.x, self.y), base, altura, linewidth=1, edgecolor='r', facecolor='none')

        ax.add\_patch(rect)

        ax.set\_xlim(min(self.x, self.punto\_final.x) - 1, max(self.x, self.punto\_final.x) + 1)

        ax.set\_ylim(min(self.y, self.punto\_final.y) - 1, max(self.y, self.punto\_final.y) + 1)

        plt.gca().set\_aspect('equal', adjustable='box')

        plt.show()

punto1 = Punto(1, 1)

punto2 = Punto(4, 5)

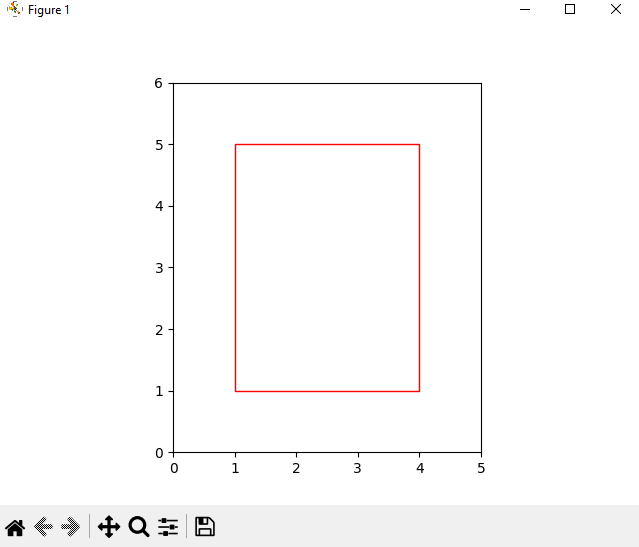
rectangulo = Rectangulo(punto1, punto2)

print(f"Base: {rectangulo.base()}")

print(f"Altura: {rectangulo.altura()}")

print(f"Área: {rectangulo.area()}")

rectangulo.dibujar()



**PS C:\Users\VERONICA CASTILLO> & "C:/Users/VERONICA CASTILLO/AppData/Local/Programs/Python/Python312/python.exe" "c:/Users/VERONICA CASTILLO/Downloads/ejercicio 11.py"**

**Base: 3**

**Altura: 4**

**Área: 12**

**PS C:\Users\VERONICA CASTILLO>**

**Ejercicio 12:** Escribir una clase con el nombre Monedas. Esta debe contener valores en varias monedas, por ejemplo, "EUR", "USD", "ARG", "YEN". Una instancia debe contener la cantidad y la unidad monetaria. La clase, que va a diseñar debe tener los siguientes métodos:

● Crear constructor

● Sumar distintas monedas

● Sumar una moneda con otro valor distinto

● Redefinir función \_str\_()

class Monedas:

    def \_\_init\_\_(self, cantidad, unidad):

        self.cantidad = cantidad

        self.unidad = unidad

    def sumar\_distintas\_monedas(self, otra\_moneda):

        if self.unidad != otra\_moneda.unidad:

            raise ValueError("Las monedas deben ser del mismo tipo para sumarse.")

        self.cantidad += otra\_moneda.cantidad

    def sumar\_moneda\_valor\_distinto(self, valor, otra\_unidad):

        if self.unidad != otra\_unidad:

            raise ValueError("Las monedas deben ser del mismo tipo para sumarse.")

        self.cantidad += valor

    def \_\_str\_\_(self):

        return f"{self.cantidad} {self.unidad}"

if \_\_name\_\_ == "\_\_main\_\_":

    moneda1 = Monedas(100, "EUR")

    moneda2 = Monedas(50, "EUR")

    moneda3 = Monedas(500, "USD")

    try:

        moneda1.sumar\_distintas\_monedas(moneda3)

    except ValueError as e:

        print(f"Error al sumar distintas monedas: {e}")

    moneda2.sumar\_moneda\_valor\_distinto(20, "EUR")

    print(moneda1)

    print(moneda2)

    print(moneda3)

**PS C:\Users\VERONICA CASTILLO> & "C:/Users/VERONICA CASTILLO/AppData/Local/Programs/Python/Python312/python.exe" "c:/Users/VERONICA CASTILLO/Downloads/ejercicio 12.py"**

**Error al sumar distintas monedas: Las monedas deben ser del mismo tipo para sumarse.**

**100 EUR**

**70 EUR**

**500 USD**

**PS C:\Users\VERONICA CASTILLO>**

**Ejercicio 13:** Crear una clase con los datos de una cuenta de correo electrónico. Cada dato de la cuenta de correo (id, dominio, y password) debe digitarse por separado. Al mostrar la cuenta los datos de ID y dominio deben aparecer unidos. Por ejemplo: juan@gmail.com. Debe validarse el password, solicitando que sea digitado dos veces.

● Crear constructor usando input() y getpass()

● El password no se puede editar directamente, se debe crear una función específica para tal operación

● Mostrar dirección de mail.

from getpass import getpass

class CorreoElectronico:

    def \_\_init\_\_(self):

        self.id = input("Ingrese el ID de su correo electrónico: ")

        self.dominio = input("Ingrese el dominio de su correo electrónico: ")

        self.\_password = self.\_validar\_password()

    def \_validar\_password(self):

        while True:

            password1 = getpass("Ingrese su contraseña: ")

            password2 = getpass("Confirme su contraseña: ")

            if password1 == password2:

                return password1

            else:

                print("Las contraseñas no coinciden. Inténtelo de nuevo.")

    def cambiar\_password(self):

        new\_password = self.\_validar\_password()

        self.\_password = new\_password

        print("Contraseña cambiada exitosamente.")

    def mostrar\_direccion(self):

        return f"{self.id}@{self.dominio}"

if \_\_name\_\_ == "\_\_main\_\_":

    cuenta = CorreoElectronico()

    print("Dirección de correo electrónico:", cuenta.mostrar\_direccion())

    cuenta.cambiar\_password()

**PS C:\Users\VERONICA CASTILLO> & "C:/Users/VERONICA CASTILLO/AppData/Local/Programs/Python/Python312/python.exe" "c:/Users/VERONICA CASTILLO/Downloads/ejercicio 13.py"**

**Ingrese el ID de su correo electrónico: mr.argentina2586@hotmail.com**

**Ingrese el dominio de su correo electrónico: hotmail.com**

**Ingrese su contraseña:**

**FICHEROS DE TEXTO Y OBJETOS. FICHEROS CSV Y FICHEROS JSON.**

**Ejercicio 14:** Crear las siguientes funciones (el archivo estará en tu google drive):

● Crear archivo de texto

● Abrir y mostrar contenido de archivo de texto

● Guardar datos en archivo de texto.

def crear\_archivo(nombre\_archivo):

    try:

        with open(nombre\_archivo, 'w') as archivo:

            print(f"Archivo '{nombre\_archivo}' creado exitosamente.")

    except IOError as e:

        print(f"No se pudo crear el archivo '{nombre\_archivo}': {e}")

def abrir\_mostrar\_contenido(nombre\_archivo):

    try:

        with open(nombre\_archivo, 'r') as archivo:

            contenido = archivo.read()

            print(f"Contenido de '{nombre\_archivo}':")

            print(contenido)

    except IOError as e:

        print(f"No se pudo abrir el archivo '{nombre\_archivo}': {e}")

def guardar\_datos(nombre\_archivo, datos):

    try:

        with open(nombre\_archivo, 'w') as archivo:

            for dato in datos:

                archivo.write(f"{dato}\n")

        print(f"Datos guardados en '{nombre\_archivo}' correctamente.")

    except IOError as e:

        print(f"No se pudo guardar datos en '{nombre\_archivo}': {e}")

if \_\_name\_\_ == "\_\_main\_\_":

    nombre\_archivo = "mi\_archivo.txt"

    crear\_archivo(nombre\_archivo)

    datos = ["Dato 1", "Dato 2", "Dato 3"]

    guardar\_datos(nombre\_archivo, datos)

    abrir\_mostrar\_contenido(nombre\_archivo)

**PS C:\Users\VERONICA CASTILLO> & "C:/Users/VERONICA CASTILLO/AppData/Local/Programs/Python/Python312/python.exe" "c:/Users/VERONICA CASTILLO/Downloads/ejercicio 14.py"**

**Archivo 'mi\_archivo.txt' creado exitosamente.**

**Datos guardados en 'mi\_archivo.txt' correctamente.**

**Contenido de 'mi\_archivo.txt':**

**Dato 1**

**Dato 2**

**Dato 3**

**PS C:\Users\VERONICA CASTILLO>**

**Ejercicio 15:** Crear un programa que lea una lista de personas, cuyos datos son DNI, Apellido y Nombre contenidos en un archivo csv, los liste en orden alfabético y muestre el resultado del ordenamiento.

import pandas as pd

def ordenar\_personas\_por\_apellido(nombre\_archivo):

    try:

        df = pd.read\_csv(nombre\_archivo)

        df\_sorted = df.sort\_values(by=['Apellido', 'Nombre'])

        print("Personas ordenadas por Apellido y Nombre:")

        print(df\_sorted)

    except FileNotFoundError:

        print(f"El archivo '{nombre\_archivo}' no existe.")

    except Exception as e:

        print(f"Error al procesar el archivo '{nombre\_archivo}': {e}")

if \_\_name\_\_ == "\_\_main\_\_":

    nombre\_archivo = "personas.csv"  # Reemplaza con el nombre de tu archivo CSV

    ordenar\_personas\_por\_apellido(nombre\_archivo)

**Ejercicio 16:** Escribir un programa que acceda a un archivo de texto de google drive y muestre por pantalla el número de palabras que contiene.

def contar\_palabras\_en\_archivo(nombre\_archivo):

    try:

        with open(nombre\_archivo, 'r', encoding='utf-8') as archivo:

            contenido = archivo.read()

            palabras = contenido.split()

            num\_palabras = len(palabras)

            print(f"El archivo '{nombre\_archivo}' contiene {num\_palabras} palabras.")

    except FileNotFoundError:

        print(f"No se encontró el archivo '{nombre\_archivo}'.")

    except Exception as e:

        print(f"Error al procesar el archivo '{nombre\_archivo}': {e}")

if \_\_name\_\_ == "\_\_main\_\_":

    nombre\_archivo = "archivo.txt"

    contar\_palabras\_en\_archivo(nombre\_archivo)

**Ejercicio 17:** Escribir un programa que lea los datos de un fichero de texto, y transforme cada fila en un diccionario usando sus campos. Al contenido del diccionario añadirlo a una lista llamada personas. Luego hacer una función que muestre el contenido. El fichero de texto se denominará personas.csv y tendrá el siguiente contenido en texto plano (créalo previamente):

● 1,Carlos,Pérez,05/01/1989

● 2,Manuel,Heredia,26/12/1973

● 3,Rosa,Campos,12/06/1961

● 4,David;García,25/07/2006 Los campos del diccionario seguirán el orden: id, nombre, apellido y nacimiento.

def leer\_archivo\_personas(nombre\_archivo):

    personas = []

    try:

        with open(nombre\_archivo, 'r', encoding='utf-8') as archivo:

            for linea in archivo:

                campos = linea.strip().split(',')

                persona = {

                    'id': campos[0],

                    'nombre': campos[1],

                    'apellido': campos[2],

                    'nacimiento': campos[3]

                }

                personas.append(persona)

        return personas

    except FileNotFoundError:

        print(f"No se encontró el archivo '{nombre\_archivo}'.")

        return []

    except Exception as e:

        print(f"Error al procesar el archivo '{nombre\_archivo}': {e}")

        return []

def mostrar\_personas(personas):

    if personas:

        print("Contenido de la lista 'personas':")

        for persona in personas:

            print(persona)

    else:

        print("La lista 'personas' está vacía.")

if \_\_name\_\_ == "\_\_main\_\_":

    nombre\_archivo = ""

    lista\_personas = leer\_archivo\_personas(nombre\_archivo)

    mostrar\_personas(lista\_personas)

**Ejercicio 18:** A partir del contenido de una página generar un archivo que guarde la información extraída. La página contiene una lista de los 40 temas principales de música. Link: <https://los40.com/los40/2021/02/18/musica/1613476104_405901.html>

**Ejercicio 19:** A partir de varios objetos de tipo diccionario genere un archivo JSON. En los objetos de tipo diccionario, deberá agregar los datos de tres clientes entre los que se encuentra el nombre, apellido, la edad y la cantidad gastada por cada uno.

**Ejercicio 20:** Crear una función que convierta una cadena de texto JSON en un diccionario. Usaremos la página http://ip-api.com que devuelve datos de una IP pública en formato json. La ip que consultaremos será una de la UNSa. IP 170.210.200.2 y url <http://ip-api.com/json/170.210.200.2>

**Ejercicio 21:** Con el archivo Entidades.json que contiene un conjunto de clases y sus propiedades realizar una función que lea el archivo decodificándolo. Ayuda:

● Primero debemos abrir el archivo json, indicando donde se encuentra a la función open(), en forma de solo lectura.

● Luego debemos leer el contenido con la función read(), y lo guardaremos en una variable contenido.

● Por último es necesario descodificar el JSON para que podamos utilizarlo posteriormente dentro de nuestra aplicación. Por lo que llamamos al paquete JSON para ejecutar la función loads() sobre el contenido leído previamente. De esa forma tendremos el contenido de JSON en memoria y podremos trabajar con él.