G1962 - Programación

Grado en Ingeniería Civil

Examen Avanzado

Instrucciones

- Crear un único fichero mediante JupyterLab con la extensión '. ipynb '.
- Cada ejercicio debe realizarse en una celda por separado dentro del Notebook de Jupyter a excepción de los ejercicios enlazados. Cada ejercicio deberá ser documentado de manera precisa.
- Al finalizar, subir el fichero al apartado Examen Avanzado alojado en Moodle.
- Se debe entregar esta hoja de examen como acreditación de asistencia al examen.
- Se dispone de un total de 2 horas de examen, tras las cuales la entrega en Moodle se cerrará, por lo que se aconseja entregar unos minutos antes ya que cualquier entrega fuera de ese periodo será descartada .

Nombre y apellidos:

Ejercicio 1 (2.5p):

Dado el siguiente código en Python, responder a las siguientes preguntas:

```
class ParametrosInvalidosError(Exception):
    pass

def presion_hidrostatica(p, h):
    '''
    Entradas: p (densidad del líquido en kg/m3)
              h (altura del nivel del líquido en m)
    Retorna: La presión hidrostatica (Pa)
    '''
    if h < 0 or p <= 0:
        raise ParametrosInvalidosError("La densidad del líquido y
        la altura deben ser valores positivos.")

    p_h = p * (9.8) * h
    return p_h

try:
    presion = presion_hidrostatica(1000, 3)
    print(f"La presión hidrostática es de {presion:.2f} Pa")
except Exception as e:
    print("Ocurrió un error del tipo (" + str(type(e)) +": " + str(e))
```

- (0.5p) ¿Cuál es el propósito de la función presion_hidrostatica?. Explica brevemente la implementación propuesta y que estructuras frente a errores se han utilizado.

- (0.75p) ¿Qué condiciones pueden generar errores en la ejecución de la función a parte de los ya previstos? y ¿Qué excepción llevan asociados en Python dichos errores?

- (0.5p) ¿Para qué sirve la clase ParametrosInvalidosError y que relación de dependencia tiene con la clase Exception?

- (0.75p) ¿Qué estructuras adicionales de prevención de errores incluirías para evitar los errores de ejecución antes identificados?

Ejercicio 2 (1.5p):

Para la gestión de parques eólicos gestionados por una empresa de energía renovable, se desea implementar una clase llamada **Aerogenerador** con su método **constructor asociado (0,5p)** que reciba como argumentos el *nombre, capacidad, coste de mantenimiento, latitud y longitud*, y los almacene en atributos de la clase. Esta clase deberá **sobrescribir el método especial para la suma de dos objetos de la clase Aerogenerador (__add__(self, ...))** de forma que devuelva la suma de las capacidades de los aerogeneradores al utilizar el operador **+** entre dos objetos de la clase previamente definida.

Ejercicio 3 (5p):

Implementar una clase llamada **ParqueEolico** que tenga como atributo un diccionario inicialmente vacío llamado *aerogeneradores* donde las claves serán los nombres de los aerogeneradores y los valores los objetos completos. También necesitamos implementar su método **constructor (0.5p)** que recibe como argumento el *nombre, operador y provincia* del parque y lo almacena en atributos de la clase. Por otro lado, también queremos implementar los siguientes métodos:

- **(0.5p) agregaAerogenerador(self, aerogenerador)**: el método simplemente añade el aerogenerador introducido como argumento al diccionario de aerogeneradores que la clase tiene como atributo, utilizando como clave el propio nombre del aerogenerador introducido.
- **(1p) eliminaAerogenerador(self, nombre_aerogenerador)**: mediante el **uso exclusivo de manejo de excepciones** el método intentará eliminar del diccionario (**del(valor a eliminar)**) el aerogenerador asociado al nombre que se introduce como argumento. En caso de que dicho nombre no exista, deberá retornar un mensaje indicando *Error: el aerogenerador que se desea eliminar no existe.*
- **(0.5p) sumaCostesIterativo(self)**: implementación iterativa de un método que retorne, a partir de los aerogeneradores que forman el parque eólico, el coste total de mantenimiento del parque como la suma de los costes individuales.
- **(0.5p) sumaCapacidadlRecursivo(self, ...)**: implementación recursiva de un método que retorne, a partir de los aerogeneradores que forman el parque eólico, la capacidad total del parque como la suma de las capacidades individuales.
- **(1p) representaAerogeneradores2D(self)**: este método representará, mediante un gráfico de puntos utilizando la librería MATPLOTLIB(*scatter(x,y,z)*), la ubicación, en términos de latitud (y) y longitud (x), y la capacidad (z) de los diferentes aerogeneradores que conforman el parque eólico.
- **(1p) informeCostes(self, nombreFichero)**: este método guardará en un fichero de texto de tipo .txt con el nombre especificado en el argumento *nombreFichero* parte de los datos almacenados, indicados con « » en la clase ParqueEolico, siguiendo el formato especificado a continuación:

```
Parque: << nombre >>
Provincia: << provincia >>
Operador: << operador >>
----------
Cantidad de aerogeneradores: << número de aerogeneradores >>
Coste de mantenimiento del parque: << coste total de mantenimiento >>
Potencia total del parque: << potencia total del parque >>
```

Ejercicio 4 (1p):

Una vez creadas las clases anteriores, se quiere probar el correcto funcionamiento de los diferentes métodos implementados a través de los siguientes datos referentes al **Parque Eólico Alto Palencia II (Castellón)** operado por **Acciona Energía**:

Cuadro 1: Lista de Aerogeneradores

Nombre	Capacidad (MW)	Coste (M€)	Latitud	Longitud
Aerogenerador 1	2	1.3	39.97	-0.66
Aerogenerador 2	3	2.1	39.96	-0.66
Aerogenerador 3	2.2	1.7	39.96	-0.65
Aerogenerador 4	3.1	2.5	39.97	-0.66
Aerogenerador 5	1.7	1.2	39.98	-0.65