

# Curso especialización Python

Examen Segundo Trimestre

Curso 2025-2026  
Fecha: 12/02/2026

---

## Sistema de Análisis de Datos Meteorológicos

Trabajas para MeteoData Solutions, una empresa de análisis climático que necesita automatizar el procesamiento de datos históricos de temperatura y conectarse a APIs meteorológicas para generar informes diarios. El sistema debe:

- Cargar configuraciones sensibles desde variables de entorno-Procesar archivos CSV con datos históricos.
- Consultar APIs externas (OpenWeatherMap) para datos en tiempo real.
- Generar informes en formato JSON con nombres timestamped.
- Validar todos los datos con Pydantic.
- Usar programación orientada a objetos con métodos de clase e instancia.

### Ejercicio 1 (2 puntos)

Crea el archivo modelos.py con la siguiente clase:

#### Clase RegistroTemperatura

- Atributos:
  - fecha: date - Fecha del registro.
  - temperatura\_max: float - Temperatura máxima (entre -100 y 60).
  - temperatura\_min: float - Temperatura mínima (entre -100 y 60).
  - ciudad: str.

#### Método de clase:

- desde\_dict(cls, datos: dict)

Modifica los datos que le vendrán de temperaturas.csv. Cambia los datos del diccionario y el return hace la unión de lo nuevo y viejo.

- Comprobar si fecha es "str", en cuyo caso la convierte a formato ISO (método fromisoformat).
- Convertir temperatura máxima y mínima a float
- Retornar el diccionario convertido con return cls(\*\*datos)

#### Método especial:

Al imprimir el objeto de la clase RegistroTemperatura, mostrará el siguiente formato:

[fecha] Ciudad: temperatura\_min°C - temperatura\_max°C

Ejemplo: "[2025-01-15] Madrid: 3.2°C - 12.5°C"

### Ejercicio 2 (3 puntos)

Crea el archivo procesador.py con la **Clase ProcesadorTemperaturas**:

#### Atributos de Clase:

total\_procesados: int = 0 - Contador global de registros procesados por TODAS las instancias

#### Atributos de instancia

ruta\_datos: Path - Ruta al archivo CSV

registros: List[RegistroTemperatura] - Lista de registros cargados (inicialmente vacía)

Crea el constructor teniendo en cuenta estos atributos

#### Método de instancia:

### **cargar\_desde\_csv(self) -> None**

Abre el archivo CSV, debe comprobar si el archivo existe antes de abrirlo, si no, excepción FileNotFoundError indicando que no se encontró el archivo X.

Por cada fila:

- Convierte el cada fila en RegistroTemperatura usando RegistroTemperatura.desde\_dict() y lo añades al atributo de instancia "registros".
- Incrementa el atributo de clase total\_procesados.

Al finalizar de leer todas las filas, mostrar el mensaje:

"Cargados X registros desde NOMBRE\_FICHERO"

Maneja ValidationError de Pydantic por cada fila. Debes importar: from pydantic import ValidationError. Debe mostrar "Error en fila X del CSV, esta fila será omitida", por lo que no se acaba el programa, solo informa que se salta esa línea.

### **calcular\_estadisticas(self) -> dict**

Calcula las estadísticas de los registros de la clase (el atributo de instancia).

Retorna un diccionario con:

```
{  
    "temperatura_maxima_absoluta": float, #2 decimales  
    "temperatura_minima_absoluta": float, #2 decimales  
    "temperatura_media": float, #2 decimales  
    "total_registros": int,  
    "periodo": {  
        "desde": str, # Primera fecha en formato "DD de MMMM de YYYY en español"  
        "hasta": str # Última fecha en formato "DD de MMMM de YYYY en español"  
    }  
}
```

Si no hay registros, retorna un diccionario como el anterior, pero con valores None.

Para obtener la temperatura máxima de los registros, al ser una Lista, aconsejo usar una función lambda con "max".

Para obtener la fecha desde y hasta, usa list comprehension para crear una lista de fechas y coge la máxima y mínima.

### Método de clase

```
def mostrar_total_procesados(cls) -> None:
```

Muestra el mensaje: "Total de registros procesados por todas las instancias: X"

### **Ejercicio 3 (3 puntos)**

Crea el archivo api\_client.py con la clase ClienteClimaAPI:

#### **Atributos cogidos de .env:**

api\_key. La key de la api cogida de .env.

url\_base. URL base de la API cogida de .env.

ciudad\_defecto. Madrid si no está definida en el .env.

unidades. metric si no está definida en el .env.

Si no existe la API Key en el .env que lance la excepción ValueError con el mensaje:

"Variable de entorno API\_KEY no encontrada. Configúrala en el archivo .env"

Mostrar el mensaje al finalizar todo correctamente en el constructor:

“Cliente API inicializado

- Ciudad por defecto: X
- Unidades: X”

#### Métodos de instancia:

def obtener\_clima\_actual(self, ciudad: Optional[str] = None) -> Optional[Dict]:

Si no se especifica ciudad, usa el atributo “ciudad\_defecto”

Construye los parámetros de la petición:

```
params = {
    'q': ciudad,
    'appid': LA API KEY,
    'units': UNIDAD DE MEDIDA,
    'lang': 'es' # Para descripciones en español
}
```

Hace petición GET usando los parámetros y tiempo máximo de respuesta de 10 segundos.

Verifica errores HTTP.

Convierte la respuesta a JSON.

Extrae y retorna un diccionario limpio:

```
{
    "ciudad": str,      # response['name']
    "temperatura": float, # response['main']['temp']
    "sensacion_termica": float, # response['main']['feels_like']
    "descripcion": str,    # response['weather'][0]['description']
    "humedad": int,       # response['main']['humidity']
    "timestamp": str      # datetime.now().isoformat()
}
```

#### **Ejercicio 4 (2 puntos)**

Crea el archivo generador\_informes.py con la clase GeneradorInformes:

#### Método estático:

formatear\_fecha\_espanol(fecha: date) -> str

Recibe una fecha como parámetro de entrada y retorna dicha fecha en un determinado formato.

- Configura locale español.
- Formatea la fecha como: "lunes, 15 de enero de 2025"

#### Método de instancia:

generar\_informe\_completo(self, procesador: ProcesadorTemperaturas, datos\_api: dict) -> Path

Los parámetros de entrada son:

- Estadísticas del procesador: procesador.calcular\_estadisticas()
- Datos actuales de la API: datos\_api

Formato del nombre del archivo: informe\_YYYYMMDD\_HHMMSS.json

Ejemplo: informe\_20250215\_143052.json

Si no existe, generar el directorio reportes y dentro mete el archivo.

Obtén las estadísticas de ProcesadorTemperaturas que meterás en "estadísticas\_históricas" y datos\_api lo metes en "clima\_actual". "fecha\_generacion\_legible" usa el método de instancia de esta clase. Construir estructura del informe:

```
{  
    "fecha_generacion": "2025-02-15T14:30:52.123456", # ISO format  
    "fecha_generacion_legible": "sábado, 15 de febrero de 2025",  
    "estadisticas_historicas": {  
        "temperatura_maxima_absoluta": 15.2,  
        "temperatura_minima_absoluta": 2.8,  
        "temperatura_media": 8.5,  
        "total_registros": 7,  
        "periodo": {  
            "desde": "15 de enero de 2025",  
            "hasta": "21 de enero de 2025"  
        }  
    },  
    "clima_actual": {  
        "ciudad": "Madrid",  
        "temperatura": 12.3,  
        "sensacion_termica": 10.8,  
        "descripcion": "nubes dispersas",  
        "humedad": 65,  
        "timestamp": "2025-02-15T14:30:00"  
    }  
}
```

Guarda el json y devuelve la ruta del informe generado