TP de POO (borrador)

[PASO 1: INICIAR HERRAMIENTA GITHUB 1](#_Toc200710507)

[1.1: CREAR REPOSITORIO EN GITHUB 1](#_Toc200710508)

[1.2: Completar los datos del nuevo repositorio 2](#_Toc200710509)

[1.3: Subir tu código al repo vacío desde tu computadora 4](#_Toc200710510)

[PASO 2: armar la estructura básica del proyecto en VS Code 6](#_Toc200710511)

[Para guardar tus cambios y que se reflejen en el repositorio remote 8](#_Toc200710512)

[2.1: Preparar entorno y dependencias 8](#_Toc200710513)

[3.2: Crear módulos principales (`modelo\_orm.py`, `gestionar\_obras.py`) 10](#_Toc200710514)

[2.3: Importar y analizar el CSV 10](#_Toc200710515)

[PASO 3: Crear base de datos y mapear ORM 11](#_Toc200710516)

[3.1: Limpiar y normalizar datos 12](#_Toc200710517)

[3.2: Cargar datos limpios a la base de datos 12](#_Toc200710518)

[3.3: Crear métodos para manejar obras 12](#_Toc200710519)

[4: Mostrar indicadores 12](#_Toc200710520)

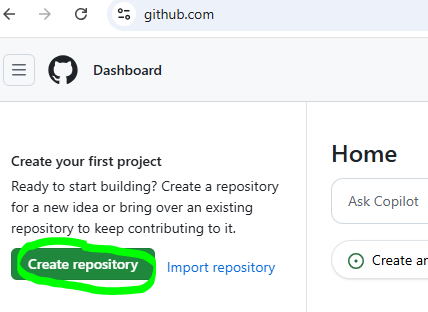
[Enunciado Trabajo Práctico Final Integrador 14](#_Toc200710521)

# PASO 1: INICIAR HERRAMIENTA GITHUB

## 1.1: CREAR REPOSITORIO EN GITHUB

**1: Entrar a GitHub y crear el repositorio**

1. [https://github.com](https://github.com/)
2. Iniciá sesión con cuenta (usuario y contraseña)
3. Clic en el botón verde arriba a la izquierda que dice **“New”** o **“+” → “New repository”**.



## 1.2: Completar los datos del nuevo repositorio

En la pantalla de creación:

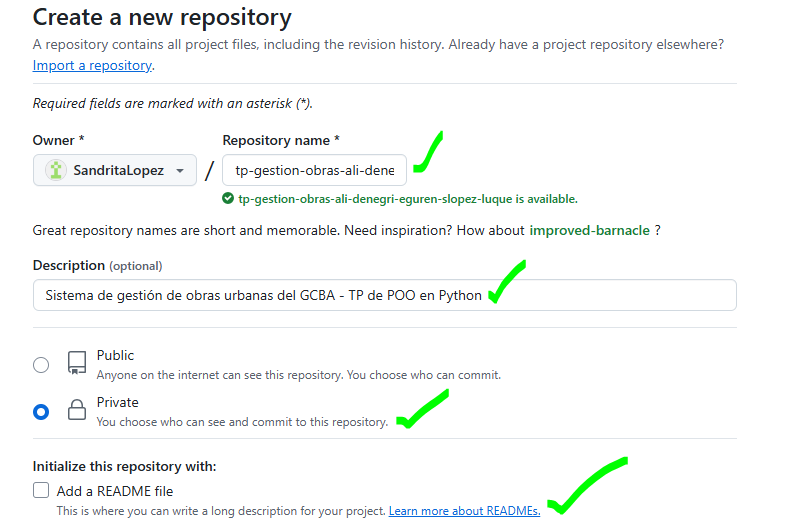
* **Repository name**: Nombre del proyecto, por ejemplo: gestion-obras-sanchez-gomez-lopez-diaz-perez

Ali Candela  
DeNegri Eduardo  
Luque Ana Rocio  
Eguren Leonel  
Lopez Sandra

tp-gestion-obras-ali-denegri-eguren-slopez-luque

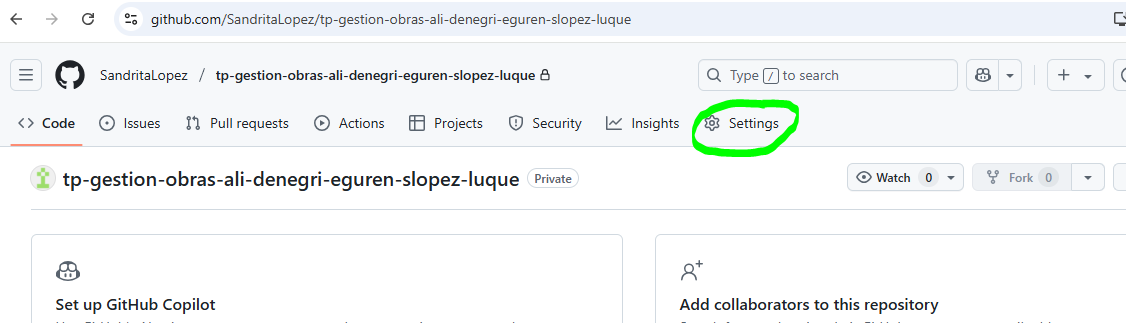
* **Description** (opcional, se puede cambiar): algo como " Sistema de gestión de obras urbanas del GCBA - TP de POO en Python"
* **Private**
* ☑️ **NO marcar** la opción que dice *Initialize this repository with a README*, porque lo vamos a clonar vacío.

Luego hacé clic en el botón **“Create repository”**.

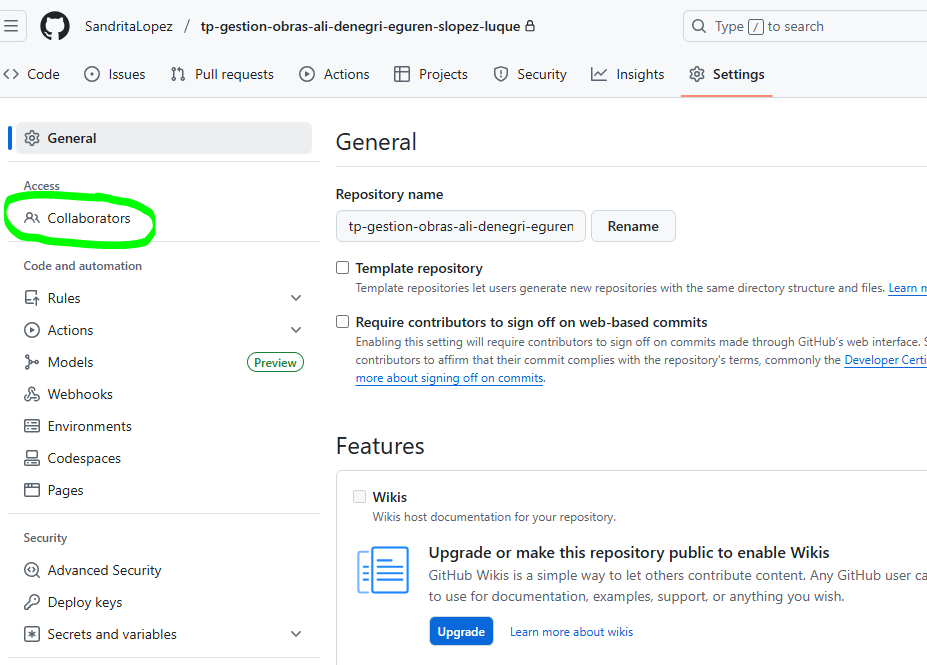


**Agregar colaboradores**

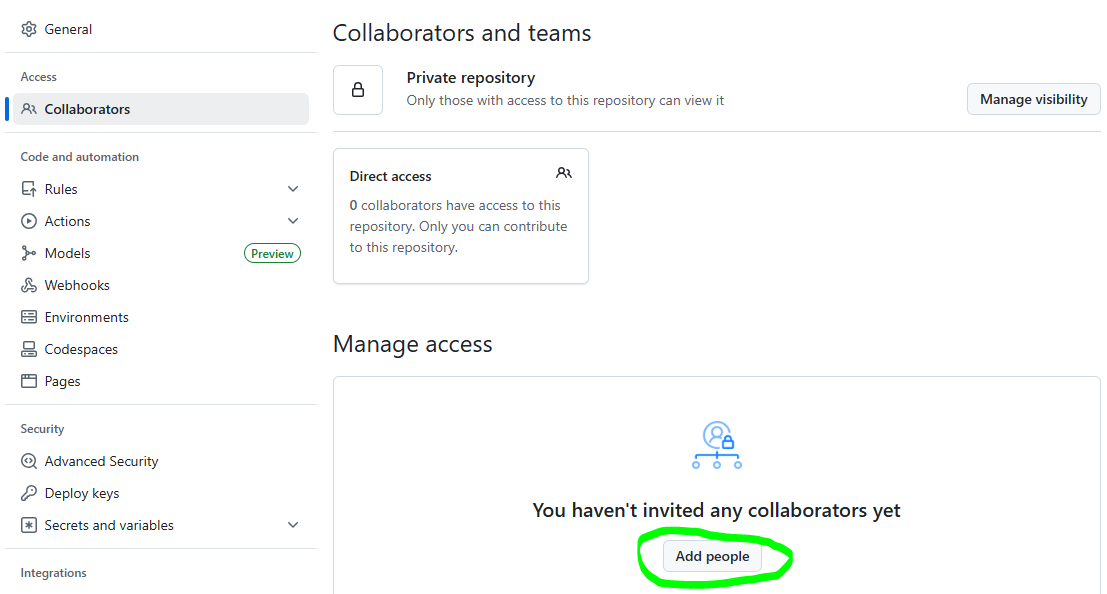
1. Ir al repositorio en GitHub.
2. Hacer clic en la pestaña **"Settings"** (arriba del todo).



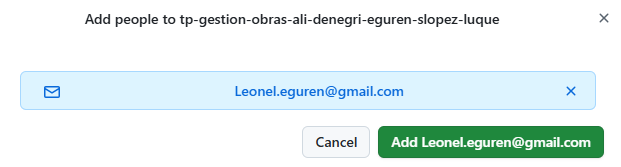
1. En el menú lateral: **"Collaborators"**.

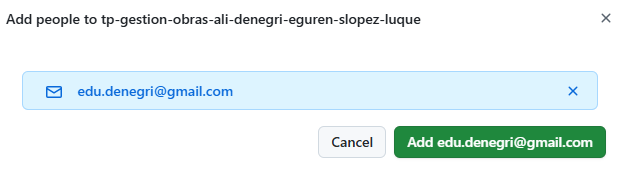


1. Clic en **"Add people"**.



1. Escribir los usuarios de GitHub de compañeros y agregalos.





Una vez acepten la invitación, van a poder:

* Clonar el repo
* Hacer cambios
* Hacer push y trabajar en equipo

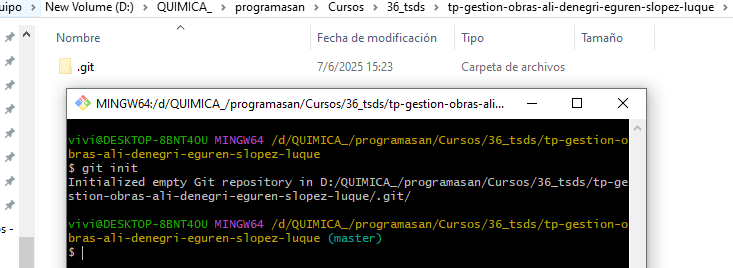
**(Ahora, ya teniendo una estructura armada en tu compu, la subimos al repositorio)**

## 1.3: Subir tu código al repo vacío desde tu computadora

D:\QUIMICA\_\programasan\Cursos\36\_tsds\tp-gestion-obras-ali-denegri-eguren-slopez-luque

1. **Abrir el proyecto en VS Code** (o la carpeta donde vas a trabajar).
2. Abri Git Bash (o la terminal que uses).
3. Inicializar git (si no lo hiciste aún):

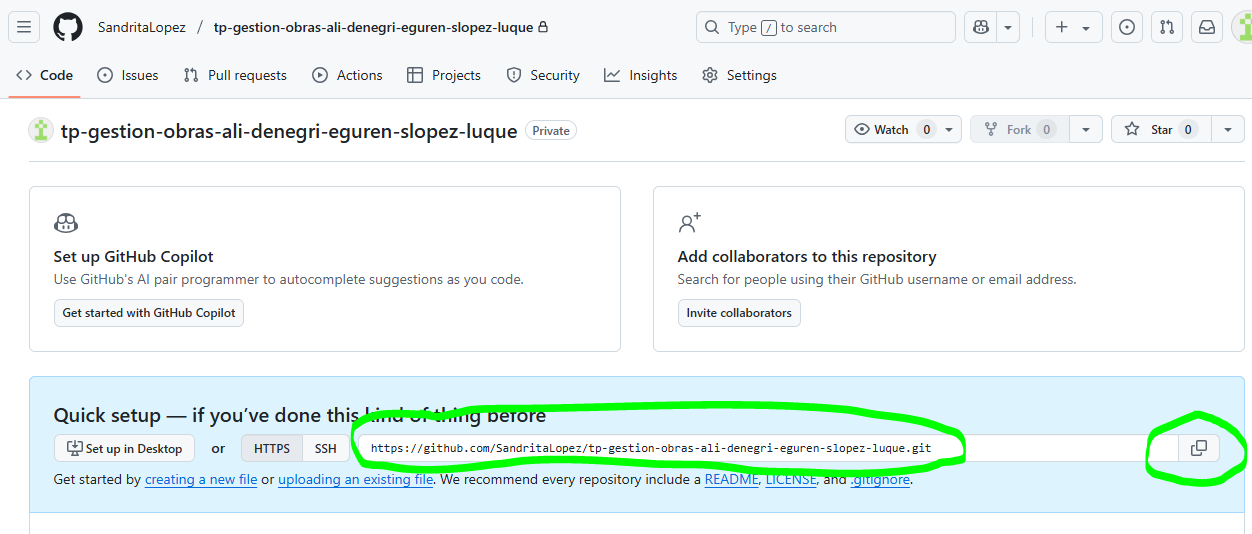
git init



1. Conectar repo local con el remoto (el que se creo en GitHub).

**URL .git de tu repo**

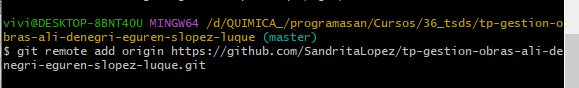
1. Ir a [https://github.com](https://github.com/) y abrír **tu repositorio** (el que creaste).
2. Hacer clic en el botón verde que dice **“Code”**.
3. Ahí esta una URL que termina en .git. Por ejemplo:
4. https://github.com/sandralopez/tp-gestion-obras-ali-denegri-eguren-slopez-luque.git
5. Copiar la URL.



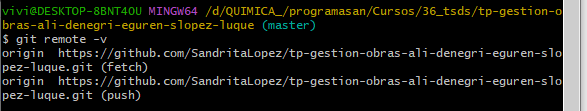
**Luego en la terminal (Git Bash):**

Pegar eso en el siguiente comando:

git remote add origin https://github.com/tu\_usuario/ tp-gestion-obras-ali-denegri-eguren-slopez-luque



git remote -v



1. Agregar los archivos al área de preparación (stage):

git add .

1. Hacer el primer commit:

git commit -m "Primer commit - estructura inicial"

1. Subir los cambios al repo remoto (GitHub):

git push -u origin main

**Nota:** Si la rama principal se llama master en lugar de main, usá master en el push.

**(Ahora se podrá clonar ese repositorio y empezar a trabajar.)**

# PASO 2: armar la estructura básica del proyecto en VS Code

1: Estructura inicial del proyecto

Qué: Crear carpetas y archivos ordenados (código, datos, docs).

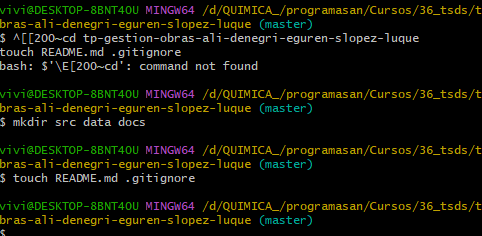
Por qué: Facilita organizar todo, encontrar rápido lo que se necesita y que el proyecto sea profesional y fácil de mantener.

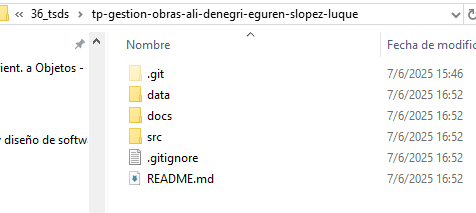
mkdir tp-gestion-obras-ali-denegri-eguren-slopez-luque

cd tp-gestion-obras-ali-denegri-eguren-slopez-luque

mkdir src data docs

touch README.md .gitignore





**Subcarpetas (esto lo puse yo, no lo vi que lo comentara el profe, pero creo que lo mejor es tener todo ordenado. Tal vez el profe lo explico pero yo no lo vi.):**

**mkdir src data docs**

**src/ → Source code**

**Qué va ahí:** Todo el código Python: clases, scripts, etc.  
**Por qué:** Es buena práctica separar el código del resto. Así no se mezcla con archivos de datos, pruebas, etc.

**data/ → Datos**

**Qué va ahí:** El archivo CSV original.  
**Por qué:** Para que los datos estén localizados en un solo lugar.

**docs/ → Documentación**

**Qué va ahí:** Diagramas, instrucciones, capturas, bocetos, cualquier archivo explicativo.  
**Por qué:** documentación aparte, ya está ordenada y no se mezcla con el código.

Para guardar tus cambios y que se reflejen en el repositorio remote

, hacés así:

1. git add . — agrega todos los archivos modificados para preparar el “commit”.
2. git commit -m "mensaje descriptivo" — guardás esos cambios localmente con un mensaje.
3. git push — subís esos cambios al repositorio remoto (GitHub).

git add .

git commit -m "readme"

git push

---

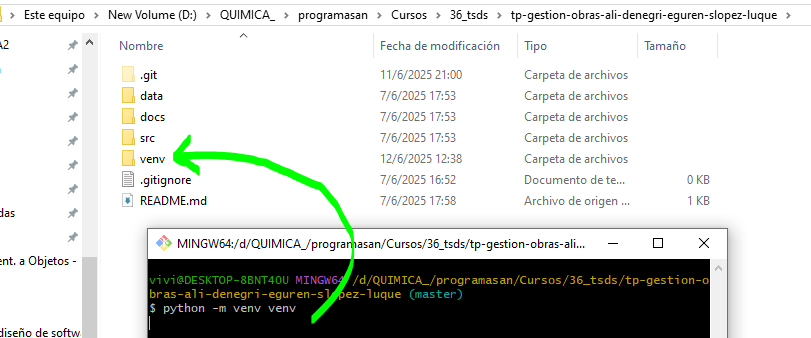
## 2.1: Preparar entorno y dependencias

Crear un entorno virtual y bajar las librerías necesarias (`pandas`, `numpy`, `peewee`).

Por qué: El entorno virtual aísla las librerías para que no interfieran con otros proyectos; las librerías te permiten manejar datos y la base de datos.

# crear entorno virtual

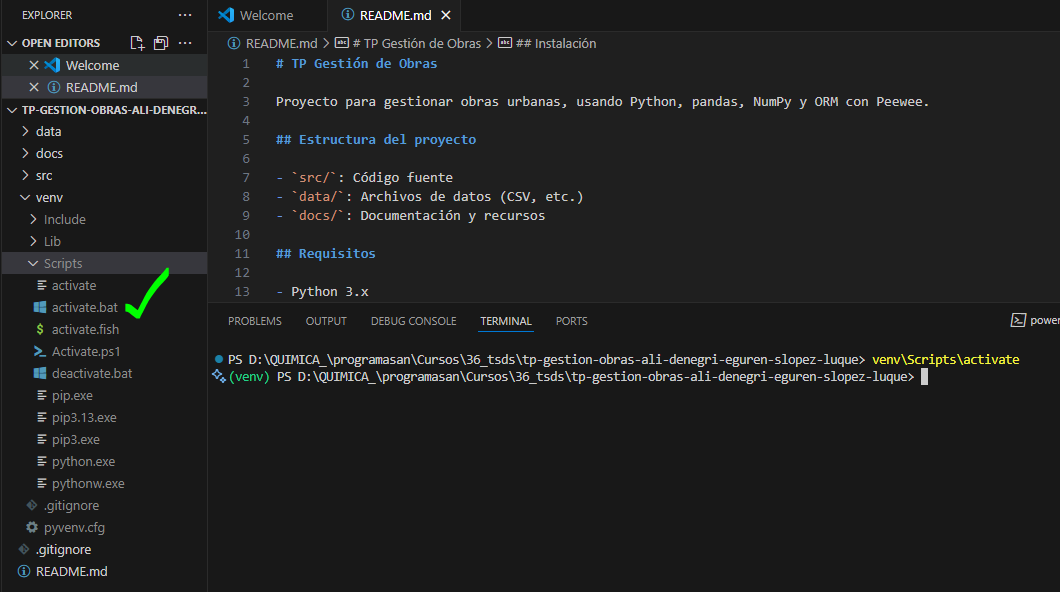
python -m venv venv



# Activar entorno:

# En Windows:

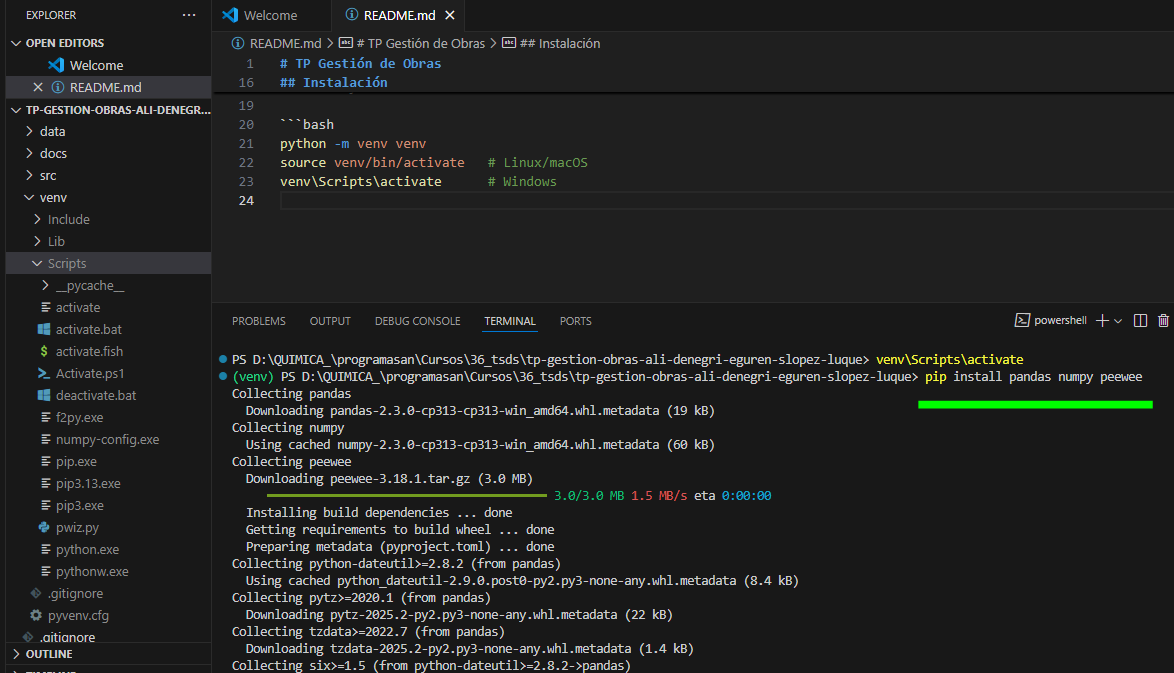
venv\Scripts\activate



# En Mac/Linux:

source venv/bin/activate

pip install pandas numpy peewee



---

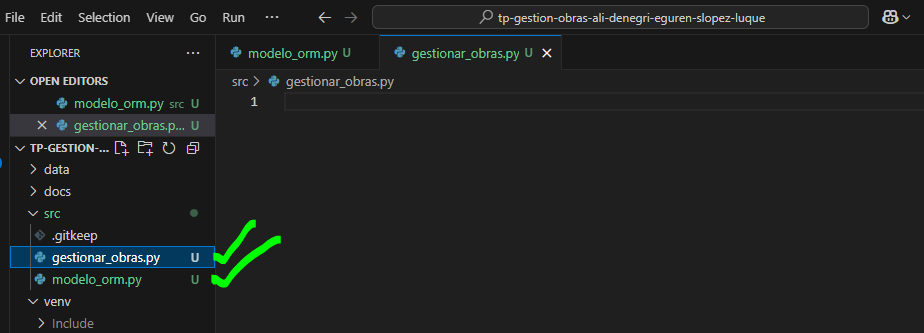
## 3.2: Crear módulos principales (`modelo\_orm.py`, `gestionar\_obras.py`)

Definir la estructura de las clases que manejarán la base de datos y la gestión del sistema.

Por qué: Tener un diseño claro y separado hace que el código sea más ordenado y fácil de ampliar.

cd src

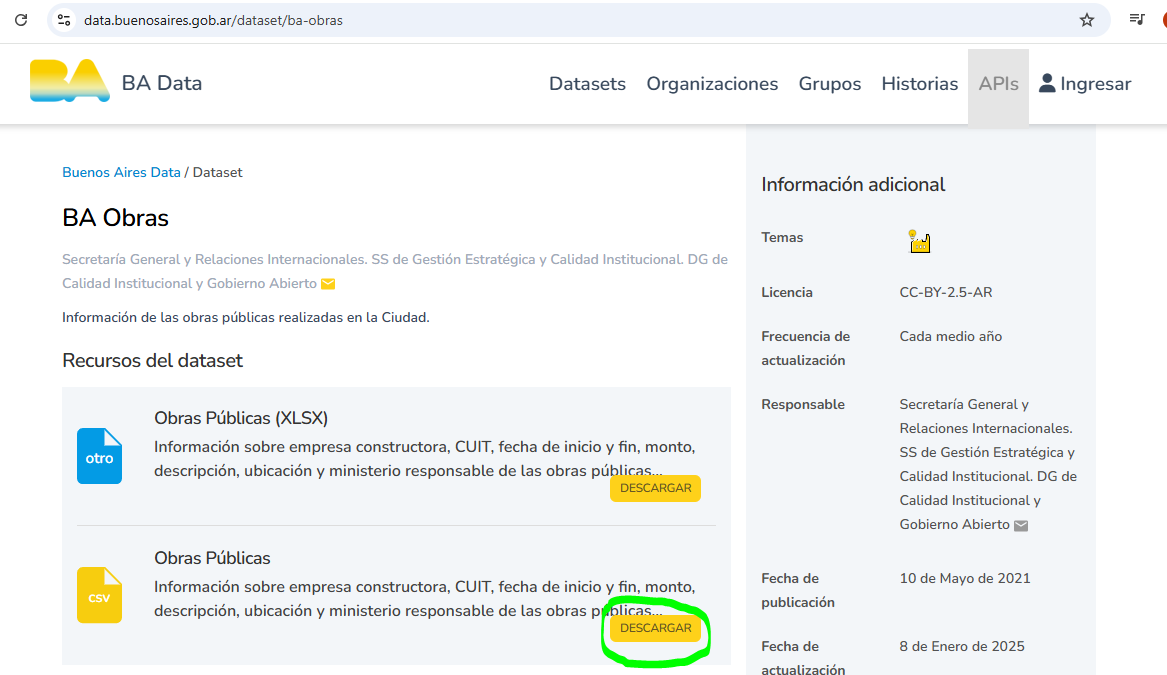
touch modelo\_orm.py gestionar\_obras.py



---

## 2.3: Importar y analizar el CSV

https://data.buenosaires.gob.ar/dataset/ba-obras



Qué: Cargar el archivo con pandas para ver qué datos hay, cuántas columnas, tipos, valores faltantes.

Por qué: Antes de usar los datos, tenés que conocerlos para saber cómo limpiarlos y usarlos.

(analisis\_dataset.py es un archivo auxiliar que me ayudó a verificar si el .csv se había cargado correctamente al proyecto. Le pedí a ChatGPT que me generara un pequeño script en Python usando pandas para poder inspeccionar los datos (ver columnas, tipos, valores faltantes, etc.). Solo lo usé para pruebas y exploración inicial del dataset.)

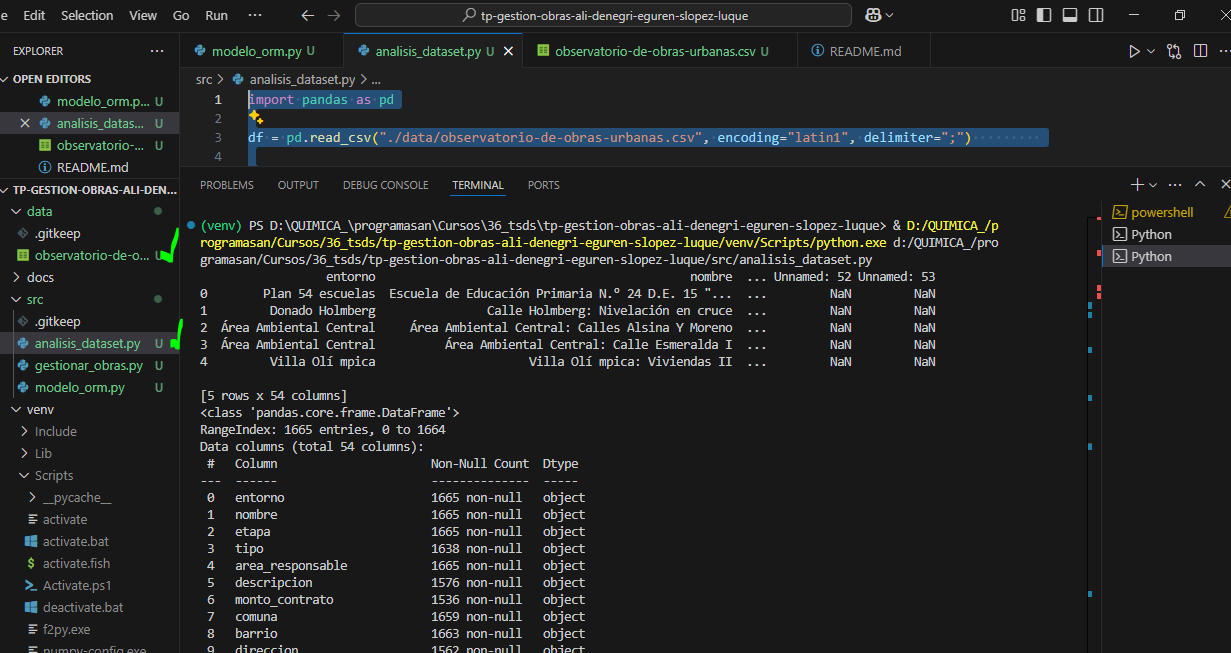
import pandas as pd

df = pd.read\_csv("./data/observatorio-de-obras-urbanas.csv", encoding="latin1", delimiter=";")

print(df.head())

print(df.info())

print(df.isnull().sum())



---

# PASO 3: Crear base de datos y mapear ORM

Qué: Conectar con SQLite y crear tablas que reflejen las clases que diseñaste.

Por qué: La base de datos es donde se va a guardar la información, y el ORM es la forma de mapear tus clases a esas tablas.

(Borrador…)

from peewee import SqliteDatabase

db = SqliteDatabase('obras\_urbanas.db')

db.connect()

db.create\_tables([Obra, OtraClase, ...])

db.close()

---

## 3.1: Limpiar y normalizar datos

Qué: Arreglar datos faltantes, formatos incorrectos o inconsistencias en el DataFrame.

Por qué: Datos sucios causan errores o resultados incorrectos, hay que dejarlos bien para que el sistema funcione bien.

---

## 3.2: Cargar datos limpios a la base de datos

Qué: Pasar los datos ya arreglados a la base de datos para que queden guardados.

Por qué: Para poder consultarlos y trabajar con ellos en el programa.

---

## 3.3: Crear métodos para manejar obras

Qué: Programar funciones para agregar nuevas obras y cambiar su estado (etapas).

Por qué: El sistema tiene que permitir crear y actualizar obras, no solo leer datos.

---

# 4: Mostrar indicadores

Qué: Hacer consultas para mostrar resúmenes, estadísticas y reportes pedidos.

Por qué: Para que el usuario pueda entender y tomar decisiones con la información.

**Instituto de Formación Técnica Superior Nro 18**

**CARRERA de TÉCNICO SUPERIOR en DESARROLLO DE SOFTWARE**

# Enunciado Trabajo Práctico Final Integrador

**Materia:** Desarrollo de Sistemas Orientados a Objetos

**Curso:** 1er Año

**Profesor:** Lic. Eduardo Iberti

**Ciclo Lectivo:** 1er Cuatrimestre 2025

**Fecha límite de entrega:** jueves 26 de Junio de 2025

**El trabajo se debe realizar en grupo de cinco alumnos.**

tp-gestion-obras-ali-denegri-eguren-slopez-luque

**Tema: Sistema de gestión de obras urbanas del GCBA. Desarrollo con POO, importación de datasets desde un archivo csv, limpieza y normalización de datos, y persistencia de objetos con ORM Peewee en una base de datos SQLite.**

Se requiere desarrollar un software en Python para gestionar las obras urbanas de la Ciudad de Buenos Aires, tomando como origen de datos un dataset público del gobierno de la ciudad y haciendo uso de la librería **“peewee”**. Para el manejo y operaciones con el dataset se debe utilizar la librería **“pandas”**. Y para el manejo y operaciones con arrays pueden utilizar la librería **“numpy”**.

**Requerimientos:**

1. En primer lugar, se debe **crear una carpeta para el proyecto solución del TP**, cuyo nombre debe contener el apellido de cada alumno integrante del equipo (cada apellido separado por guión).
2. Utilizar el **archivo csv “observatorio-de-obras-urbanas.csv”** que contiene los datos de las obras urbanas (más de 1600 registros). Lo pueden descargar desde:

<https://data.buenosaires.gob.ar/dataset/ba-obras>

[https://cdn.buenosaires.gob.ar/datosabiertos/datasets/secretaria-general-y-relacionesinternacionales/ba-obras/observatorio-de-obras-urbanas.csv.](https://cdn.buenosaires.gob.ar/datosabiertos/datasets/secretaria-general-y-relaciones-internacionales/ba-obras/observatorio-de-obras-urbanas.csv)

Ubicarlo en la carpeta del proyecto y analizar su estructura de datos **(ver estructura al pie del presente documento)**.

1. Crear el **módulo “modelo\_orm.py”** que contenga la definición de las clases y atributos que considere necesarios, siguiendo el modelo **ORM de Peewee** para poder persistir los datos importados del dataset en una base de datos relacional de tipo SQLite llamada **“obras\_urbanas.db”**, ubicada en la misma carpeta solución del proyecto. Aquí se debe incluir además la clase BaseModel heredando de peewee.Model.
2. Crear otro **módulo “gestionar\_obras.py”** que contenga la definición de la **clase abstracta “GestionarObra”** y los siguientes métodos de clase:
   1. **extraer\_datos()**, que debe incluir las sentencias necesarias para manipular el dataset a través de un objeto Dataframe del **módulo “pandas”**.
   2. **conectar\_db()**, que debe incluir las sentencias necesarias para realizar la conexión a la base de datos “obras\_urbanas.db”.
   3. **mapear\_orm()**, que debe incluir las sentencias necesarias para realizar la creación de la estructura de la base de datos (tablas y relaciones) utilizando el método de instancia “create\_tables(list)” del **módulo “peewee”**.
   4. **limpiar\_datos()**, que debe incluir las sentencias necesarias para realizar la “limpieza” de los datos nulos y no accesibles del Dataframe.
   5. **cargar\_datos()**, que debe incluir las sentencias necesarias para persistir los datos de las obras (ya transformados y “limpios”) que contiene el objeto Dataframe en la base de datos relacional SQLite. Para ello se debe utilizar el método de clase Model create() en cada una de las clase del modelo ORM definido.
   6. **nueva\_obra()**, que debe incluir las sentencias necesarias para crear nuevas instancias de Obra. Se deben considerar los siguientes requisitos:
      * Todos los valores requeridos para la creación de estas nuevas instancias deben ser ingresados por teclado.
      * Para los valores correspondientes a registros de tablas relacionadas (foreign key), el valor ingresado debe buscarse en la tabla correspondiente mediante sentencia de búsqueda ORM, para obtener la instancia relacionada, si el valor ingresado no existe en la tabla, se le debe informar al usuario y solicitarle un nuevo ingreso por teclado.
      * Para persistir en la BD los datos de la nueva instancia de Obra debe usarse el método save() de Model del **módulo “peewee”**.
      * Este método debe retornar la nueva instancia de obra.
   7. **obtener\_indicadores()**, que debe incluir las sentencias necesarias para obtener información de las obras existentes en la base de datos SQLite a través de sentencias ORM.
3. La **clase “Obra”**, que es una de las clases que debe formar parte del modelo ORM, debe incluir los siguientes métodos de instancia con el objetivo de definir las diferentes **etapas de avance de obra**:
   1. nuevo\_proyecto().
   2. iniciar\_contratacion().
   3. adjudicar\_obra().
   4. iniciar\_obra().
   5. actualizar\_porcentaje\_avance().
   6. incrementar\_plazo().
   7. incrementar\_mano\_obra().
   8. finalizar\_obra().
   9. rescindir\_obra().
4. Se deberán crear nuevas instancias de Obra (dos instancias como mínimo) invocando al método de clase “GestionarObra.nueva\_obra()”.
5. Cada una de **las nuevas obras deben pasar por todas las etapas definidas**, salvo **incrementar\_plazo()** e **incrementar\_mano\_obra()** que son opcionales. Para ello se debe invocar a los métodos de instancia de la clase Obra, siguiendo el orden de la declaración de las etapas (desde **nuevo\_proyecto()** hasta **finalizar\_obra()** ó **rescindir\_obra()**). Luego de cada cambio de estado del objeto Obra producto de una nueva etapa de avance de la obra, se deben persistir los nuevos valores usando el método save().
6. Para **iniciar un nuevo proyecto de obra se debe invocar al método nuevo\_proyecto()**. Aquí la etapa inicial de las nuevas instancias de Obra debe tener el valor “Proyecto” (si este valor no existe en la tabla “etapas” de la BD, se deberá crear la instancia y luego insertar el nuevo registro). **Los valores de los atributos tipo\_obra, area\_responsable y barrio deben ser alguno de los existentes en la base de datos**.
7. A continuación, se debe **iniciar la licitación/contratación de la obra**, para ello se debe invocar al método **iniciar\_contratacion()**, asignando el TipoContratacion (debe ser un valor existente en la BD) y el nro\_contratacion.
8. Para **adjudicar la obra a una empresa, se debe invocar al método adjudicar\_obra()** y asignarle la Empresa (debe ser una empresa existente en la BD) y el nro\_expediente.
9. Para indicar el **inicio de la obra, se debe invocar al método iniciar\_obra()**, y asignarle valores a los siguientes atributos: destacada, fecha\_inicio, fecha\_fin\_inicial, fuente\_financiamiento (debe ser un valor existente en la BD) y mano\_obra.
10. Para registrar **avances de la obra, se debe invocar al método actualizar\_porcentaje\_avance()** y actualizar el valor del atributo porcentaje\_avance.
11. Para **incrementar el plazo de la obra, se debe invocar al método incrementar\_plazo()** y actualizar el valor del atributo plazo\_meses. **(Esta acción es opcional, pero el método debe estar definido).**
12. Para **incrementar la cantidad de mano de obra, se debe invocar al método incrementar\_mano\_obra()** y actualizar el valor del atributo mano\_obra. **(Esta acción es opcional, pero el método debe estar definido)**.
13. Para **indicar la finalización de una obra, se debe invocar al método finalizar\_obra()** y actualizar el valor del atributo etapa a “Finalizada” y del atributo porcentaje\_avance a “100”.
14. Para **indicar la rescisión de una obra, se debe invocar al método rescindir\_obra()** y actualizar el valor del atributo etapa a “Rescindida”.
15. Para finalizar la ejecución del programa, se debe **invocar al método de clase GestionarObra.obtener\_indicadores()** para obtener y mostrar por consola la siguiente información:
    1. Listado de todas las áreas responsables.
    2. Listado de todos los tipos de obra.
    3. Cantidad de obras que se encuentran en cada etapa.
    4. Cantidad de obras y monto total de inversión por tipo de obra.
    5. Listado de todos los barrios pertenecientes a las comunas 1, 2 y 3.
    6. Cantidad de obras finalizadas en un plazo menor o igual a 24 meses.
    7. Monto total de inversión.

**Aclaraciones:**

1. En la **clase abstracta “GestionarObra”, todos sus métodos deben ser métodos de clase y sus atributos** (en caso que considere que deba existir alguno) **también serán atributos de clase.**
2. Incluir código Python para **manejar posibles excepciones** donde considere conveniente para atrapar aquellas que puedan llegar a generarse.

**Qué es la librería o módulo “Pandas”?**

Esta biblioteca de software de código abierto está diseñada específicamente para la manipulación y el análisis de datos en el lenguaje [Python.](https://datascientest.com/es/python-un-zoom-en-el-lenguaje-mas-popular) Es potente, flexible y fácil de usar.

Pandas se puede utilizar en el lenguaje Python para cargar, alinear, manipular o incluso fusionar datos.

Facilita el procesamiento de datos estructurados en forma de tablas, matrices o series temporales. También es compatible con otras bibliotecas de Python.

Pandas trabaja sobre “[DataFrames”](https://datascientest.com/quest-ce-quun-dataframe) (tablas de datos bidimensionales), donde cada columna contiene los valores de una variable y cada fila contiene un conjunto de valores de cada columna. Los datos almacenados en un DataFrame pueden ser números o caracteres.

Los Data Scientists y los programadores familiarizados con el lenguaje de programación R para cálculo estadístico utilizan DataFrames para almacenar datos en una cuadrícula muy sencilla de revisar. Por eso Pandas se utiliza mucho para Machine Learning.

Esta herramienta permite importar y exportar datos en distintos formatos, como CSV o JSON. [**https://datascientest.com/es/pandas-python**](https://datascientest.com/es/pandas-python) [**https://www.w3schools.com/python/pandas/default.asp**](https://www.w3schools.com/python/pandas/default.asp) [**https://pypi.org/project/pandas/**](https://pypi.org/project/pandas/)

**Qué es la librería o módulo “NumPy”?**

[NumPy](https://www.numpy.org/) es una librería de Python especializada en el cálculo numérico y el análisis de datos, especialmente para un gran volumen de datos.

Incorpora una nueva clase de objetos llamados **arrays** que permite representar colecciones de datos de un mismo tipo en varias dimensiones, y funciones muy eficientes para su manipulación.

La ventaja de Numpy frente a las listas predefinidas en Python es que el procesamiento de los arrays se realiza mucho más rápido (hasta 50 veces más) que las listas, lo cual la hace ideal para el procesamiento de vectores y matrices de grandes dimensiones. <https://aprendeconalf.es/docencia/python/manual/numpy/><https://facundoq.github.io/courses/images/res/03_numpy.html><https://joserzapata.github.io/courses/python-ciencia-datos/numpy/><https://geekflare.com/es/numpy-arrays/>

**Estructura del dataset “Obras Públicas”**

Información de las obras públicas realizadas en la Ciudad. Información sobre las empresas constructoras, , fecha de inicio y fin, monto, descripción, ubicación y ministerio responsable de las obras públicas realizadas por el Gobierno de la Ciudad.

La estructura del dataset (campos, tipo de dato y descripción) se puede consultar en el siguiente link:

[https://data.buenosaires.gob.ar/dataset/ba-obras/resource/b18dc277-ed29-4d2b-a5f5d8bd10751920](https://data.buenosaires.gob.ar/dataset/ba-obras/resource/b18dc277-ed29-4d2b-a5f5-d8bd10751920)