Programación para Física y Astronomía

Departamento de Física.

Corodinadora: C Loyola
Profesores C Femenías / F Bugini / D Basantes

Primer Semestre 2025

Universidad Andrés Bello
Departamento de Física y Astronomía

Universidad Andrés Bello®

python powered

Resumen - Semana 8, Sesión 1 (Sesión 15)

Introducción y Contexto

Fundamentos de POO

Análisis de Datos Rápido

Ejercicios Prácticos

Conclusiones y Próximos Pasos

Introducción y Contexto

- **Semana 7, Sesión 1 (Sesión 13)**:
  - Exploramos gráficos avanzados en Matplotlib (subplots múltiples, histogramas, 3D).
  - Tuvimos un vistazo opcional a pandas para el manejo de datos tabulares.
- **Semana 7, Sesión 2 (Sesión 14)**:
  - Realizamos un **problema evaluado** integrando NumPy y Matplotlib (matriz aleatoria, álgebra lineal, gráficas 3D e histograma).
  - Aclaramos dudas tras la entrega.
- **Objetivo de hoy**: Iniciar **programación orientada a objetos (POO)** en Python y repasar análisis básico de datos en NumPy/pandas.

- **Comprender** los fundamentos de la Programación Orientada a Objetos (POO) en Python:
  - Clases, objetos, atributos, métodos.
  - Encapsulamiento básico.
- **Aplicar** estos conceptos en un ejemplo sencillo (clase `Particle` o similar).
- **Refrescar** el uso de NumPy/pandas para análisis de datos básicos.
- **Ejercitar** con un problema que combine POO y datos.

Fundamentos de POO

- **Organización y modularidad**: agrupa datos y métodos lógicamente.
- **Reutilización de código**: herencia y polimorfismo (más adelante).
- **Facilita** el modelado de entidades del mundo real (partículas, sistemas, objetos astronómicos).
- Python lo hace **fácil** con la palabra clave `class`.

```python
class MiClase:
    """Docstring o descripción de la clase."""
    def __init__(self, valor):
        # Método constructor (inicializador)
        self.atributo = valor

    def mostrar_atributo(self):
        print("El atributo es:", self.atributo)

# Uso
obj = MiClase(10)
obj.mostrar_atributo()  # "El atributo es: 10"
```

- **self** refiere a la instancia en los métodos.
- `__init__` se llama al crear el objeto (**constructor**).

- **Atributos** (variables) caracterizan el estado de un objeto.
  - Definidos en `__init__` o fuera (variables de clase).
- **Métodos** (funciones) definen el comportamiento.
  - **Instancia**: necesitan `self`.
  - **Clase**: usan `@classmethod` y tienen `cls` en lugar de `self`.
  - **Estáticos**: usan `@staticmethod` y no reciben `self` ni `cls`.
- Encapsulamiento es **limitado** en Python (no hay private real), pero se puede usar guión bajo (`_atributo`) para indicar uso interno.

```python
import math

class Particle:
    def __init__(self, mass, x, y, vx, vy):
        self.mass = mass
        self.x = x
        self.y = y
        self.vx = vx
        self.vy = vy

    def kinetic_energy(self):
        """Retorna la energía cinética de la partícula."""
        v2 = self.vx**2 + self.vy**2
        return 0.5 * self.mass * v2

    def distance(self, other):
        """Distancia a otra partícula."""
        dx = self.x - other.x
        dy = self.y - other.y
        return math.sqrt(dx**2 + dy**2)
```

```python
p1 = Particle(mass=2.0, x=0, y=0, vx=3, vy=4)
p2 = Particle(mass=1.5, x=5, y=5, vx=0, vy=-2)

print("E_c de p1:", p1.kinetic_energy())  # 0.5*2*25 = 25
dist = p1.distance(p2)
print("Distancia p1 - p2:", dist)
```

- Se pueden crear múltiples partículas y realizar cálculos **similares** sin reescribir la lógica.
- Facilita la **extensión** futura (métodos de colisiones, etc.).

Análisis de Datos Rápido

```python
import pandas as pd

df = pd.read_csv("datos_simulacion.csv")
print(df.head())
print(df.describe())

# Seleccionar columna "VelX"
velx = df["VelX"]
# Calcular promedio de VelX
prom_velx = velx.mean()
print("Promedio de VelX:", prom_velx)
```

- **df.head()** muestra las primeras filas.
- **df.describe()** da estadísticas (count, mean, std, min, max, quartiles).

Análisis de Datos Rápido $\in$ Unir POO y Pandas

- **Ejemplo**:
  - Cargar datos de partículas (`mass, x, y, vx, vy`).
  - Crear objetos `Particle` a partir de cada fila del DataFrame.
  - Calcular energía cinética total, distancias, etc.
- Nos ayuda a **organizar** simulaciones más complejas.

Ejercicios Prácticos

Ejercicios Prácticos ∈ Ejercicio 1: Clase para Administrar Notas de Estudiantes

Enunciado

- Crear una clase **Student** con atributos: `name`, `grades` (lista o array).
- Método `average()` que retorne el promedio de `grades`.
- Método `show_info()` que imprima nombre y promedio.
- Instanciar varios objetos y mostrarlos en un **DataFrame de pandas** (opcional).

Sugerencia: Manejar notas como lista de floats y **numpy.mean** u operaciones directas.

Ejercicios Prácticos ∈ Ejercicio 2: Partículas desde CSV (pandas + POO)

Enunciado

- Suponer un archivo `particles.csv` con columnas: `mass, x, y, vx, vy`.
- Cargarlo con **pandas**.
- Por cada fila, crear un objeto `Particle`.
- Calcular la **energía cinética total** y la **posición promedio** $((\bar{x}, \bar{y}))$.
- (Opcional) Graficar las partículas en un scatter (**x** vs **y**) usando **Matplotlib**.

Objetivo: Combinar la **clase Particle** con datos reales/leídos de un CSV.

Ejercicios Prácticos ∈ Ejercicio 3: Matriz de Distancias con NumPy

Enunciado

- Teniendo **n** partículas (ya sea generadas o leídas):
- Crear una **matriz NxN** donde la entrada (i, j) sea la distancia entre partícula i y j.
- Usar preferentemente **NumPy** para vectorizar o, si no, hacerlo en un doble **for**.
- (Opcional) Mostrarla como un **mapa de calor** con `plt.imshow`.

Sugerencia: **p1.distance(p2)** es el método que definimos en la clase `Particle`.

Ejercicios Prácticos $\in$ Trabajo en Grupos

- Formar **parejas/tríos**.
- Seleccionar al menos **2 ejercicios** anteriores (o crear una fusión).
- Implementar soluciones en un **notebook** de Colab.
- Agregar **gráficas**, **comentarios** y **pruebas** de cada parte.
- Comparar resultados y dudas al final.

- **POO**: mantén las clases simples y bien comentadas.
- **pandas**: `df.iterrows()` o `df.itertuples()` puede ayudarte a iterar filas.
- **NumPy**: para la matriz de distancias, considera `np.zeros((n,n))` de base.
- **Visualización**: usa `plt.scatter(df['x'], df['y'])` si usas DataFrame.

Ejercicios Prácticos $\in$ Espacio para Dudas

- ¿Dificultades al instanciar objetos desde DataFrame?
- ¿Uso de **NumPy** en la creación de la matriz de distancias?
- ¿Cómo mostrar la información en un **DataFrame** de forma clara?
- **Preguntar abiertamente** si algo no está claro.

Conclusiones y Próximos Pasos

- Comparte **cómo implementaste** la clase y las funciones.
- Menciona si **pandas** facilitó la lectura de datos.
- Para la matriz de distancias, ¿usaste **doble for** o intentaste algo vectorizado?
- Resultados numéricos y/o visuales (**gráficas**) que surgieron.

- Iniciamos la **Programación Orientada a Objetos** (POO) en Python.
  - Clases, atributos, métodos, constructor.
  - Ejemplo con `Particle` y distancias.
- Reforzamos la **integración** con **NumPy** y **pandas**, importante para **análisis de datos** y simulaciones.
- Seguiremos expandiendo POO y su aplicación en futuros problemas más complejos.

Conclusiones y Próximos Pasos $\in$ Próximos Temas (Semana 8, Sesión 2)

- Continuar con **POO** (herencia, métodos especiales).
- Ejemplos avanzados: **clases para manejo estadístico**, o **modelos de partículas** con interacciones básicas.
- Se revisará la **retroalimentación** del problema evaluado de la semana pasada.

Recomendación: Repasar la sintaxis de `class` y crear ejemplos propios para consolidar POO.

- **Python Tutorial - Classes** (documentación oficial).
- **Pandas Documentation** para operaciones de DataFrame.
- **Real Python** - guías sobre OOP y análisis de datos en Python.
- **NumPy linalg** - recordatorio de funciones.

Gracias y hasta la próxima sesión

- Practiquen la creación de **clases** y la **lectura** de datos con **pandas**.
- ¡Nos vemos en la **Semana 8, Sesión 2** para profundizar POO!