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 6, Sesión 1 (Sesión 11)

Breve Repaso a Listas y For en Python

Introducción y Contexto

Motivación: NumPy

Primeros Pasos con NumPy

Manipulación de Arrays y Operaciones

Operaciones Numéricas y Práctica

Conclusiones y Próximos Pasos

Breve Repaso a Listas y For en Python

- Una **lista** es una colección ordenada y mutable de elementos.
- Se definen con corchetes, por ejemplo: `mi_lista = [1, 2, 3]`.
- Podemos mezclarlas con distintos tipos de datos (enteros, cadenas, etc.).
- Las listas se pueden indexar y "slicear" como en `mi_lista[0]`, `mi_lista[1:3]`, etc.

- El bucle **for** se utiliza para iterar sobre una secuencia (lista, cadena, etc.).
- Ejemplo básico:

```
numeros = [10, 20, 30]
for num in numeros:
    print(num)
```

- En cada iteración, **num** toma un valor de la lista.

Breve Repaso a Listas y For en Python ∋ Ejemplo: Suma de Elementos de una Lista

```python
numeros = [1, 2, 3, 4, 5]
suma = 0

for n in numeros:
    suma += n

print("La suma es:", suma)
```

- Con `for` iteramos directamente sobre cada elemento.
- Acumulamos la suma en `suma`.
- Resultado: `La suma es: 15`.

Introducción y Contexto

- **Semana 5** incluyó un repaso integral y la **Solemne I**.
- Ahora retomamos contenidos para profundizar nuestras habilidades de programación.
- **Objetivo de esta sesión**: Introducir la librería **NumPy** para manejo eficiente de arreglos y cálculos numéricos.

- **Comprender** los conceptos básicos de NumPy: arreglos (arrays), vectorización, broadcasting.
- **Practicar** la creación y manipulación de arrays en Google Colab.
- **Aplicar** estas herramientas a ejemplos numéricos y pequeños problemas de Física/Astronomía.
- **Fomentar** la colaboración y el trabajo práctico en laboratorio.

Motivación: NumPy

Motivación: NumPy ∋ ¿Qué es NumPy?

- **NumPy** = **Num**erical **Py**thon.
- Ofrece:
  - Estructura central: **ndarray** (arreglo multidimensional).
  - Funciones matemáticas optimizadas para operar sobre estos arreglos.
  - Alto rendimiento (implementaciones en C bajo la interfaz de Python).
- **Base de muchas librerías** científicas (p.e. SciPy, Pandas, etc.).

Motivación: NumPy ∋ Ventajas frente a Listas Nativas de Python

- **Vectorización**: escribir operaciones como `arr * 2` en lugar de bucles manuales.
- **Rendimiento**: implementaciones de bajo nivel (C/Fortran) para cálculos numéricos.
- **Herramientas avanzadas**: slicing sofisticado, broadcasting, manipulación de ejes (dimensiones).
- **Compatibilidad** con otras librerías (Matplotlib, SciPy, etc.).

Primeros Pasos con NumPy

- En muchos entornos (Colab, Anaconda), NumPy ya está incluido.
- Si fuera necesario:

  `pip install numpy`
- **Importación típica**:

```
1  import numpy as np
```

- A partir de ahí, usamos `np.array`, `np.mean`, etc.

```python
import numpy as np

# Convertir lista de Python a array NumPy
lista = [1, 2, 3, 4]
arr = np.array(lista)
print("Array:", arr)
print("Tipo:", type(arr))  # <class 'numpy.ndarray'>

# Array de ceros
zeros = np.zeros(5)
print("Zeros:", zeros)

# Array de unos
unos = np.ones((2,3))  # 2 filas x 3 columnas
print("Unos:\n", unos)

# Rango de valores
rango = np.arange(0, 10, 2)  # [0, 2, 4, 6, 8]
```

```python
arr = np.array([[1, 2, 3],
                [4, 5, 6]])
print("Shape:", arr.shape)   # (2, 3)
print("Dim:", arr.ndim)      # 2
print("Tipo de datos:", arr.dtype)  # int64 (en sistemas Linux)
print("Total de elementos:", arr.size) # 6
```

- `shape` = tupla de dimensiones.
- `dtype` = tipo de dato (`int32, float64`, etc.).
- `ndim` = número de ejes.

Manipulación de Arrays y Operaciones

```python
arr = np.array([10, 20, 30, 40, 50])
print(arr[0])      # 10
print(arr[1:3])    # [20 30]

mat = np.array([[1, 2, 3],
                [4, 5, 6],
                [7, 8, 9]])
# Acceder a fila 1, columna 2
print(mat[1, 2])   # 6

# Slicing de filas y columnas
submat = mat[0:2, 1:3]  # Filas [0..1], Cols [1..2]
print(submat)
```

- **Ojo**: en NumPy, slicing crea **vistas**, no copias (con implicaciones en la modificación de datos).

Manipulación de Arrays y Operaciones ∋ Operaciones Elemento a Elemento (Vectorización)

```
x = np.array([1, 2, 3])
y = np.array([4, 5, 6])

print("Suma:", x + y)   # [5  7  9]
print("Producto:", x * y)  # [4 10 18]
print("Potencia:", x**2)   # [1 4 9]
```

- **Sin** bucles explícitos.
- El núcleo de NumPy optimiza estas operaciones.

- Mecanismo que NumPy utiliza para manejar operaciones entre arrays de diferentes formas.
- **Ejemplo**:

```
a = np.array([1, 2, 3])
b = 2
# b se "expande" para coincidir con la forma de a
res = a * b  # [2 4 6]
```

- También puede ocurrir con arreglos 2D y 1D, etc.

```
mat = np.array([[1,2,3],
                [4,5,6],
                [7,8,9]])
fila = np.array([10, 20, 30])

# Broadcasting: se suma fila a cada fila de la matriz
res = mat + fila
print(res)
# [[11 22 33]
#  [14 25 36]
#  [17 28 39]]
```

- NumPy reconoce que `fila` es de 1D y la ”extiende” sobre las filas.

Operaciones Numéricas y Práctica

```python
arr = np.array([2, 4, 6, 8])

print("Suma total:", np.sum(arr))          # 20
print("Mínimo:", np.min(arr))              # 2
print("Máximo:", np.max(arr))              # 8
print("Promedio:", np.mean(arr))           # 5.0
print("Desv. Estándar:", np.std(arr))      # ~2.236
```

- También hay `np.median(arr)`, `np.var(arr)` y muchas más.
- Para arreglos 2D, se puede especificar `axis=0` o `axis=1`.

Ejercicio 1: Primer Contacto con NumPy

Enunciado

- Crear un arreglo 1D con los números del 1 al 10.
- Imprimir su **media**, **suma** y **producto** de todos los elementos.
- Reemplazar los valores ≤ 5 por 0, y los restantes por 1 (opcional: slicing lógico).
- Mostrar el arreglo resultante.

Sugerencia: Usa **`np.arange`** y **operaciones de comparación** (`arr <= 5`).

Operaciones Numéricas y Práctica ∋ Ejercicio 2: Matriz y Broadcasting

Enunciado

- Crear una matriz 3x3 con $\{[1, 2, 3], [4, 5, 6], [7, 8, 9]\}$.
- Definir un **vector** $[10, 20, 30]$.
- Sumar dicho vector a cada fila de la matriz (broadcasting).
- Mostrar el resultado.

Objetivo: Practicar la manipulación 2D y la "extensión" de forma con un vector 1D.

Enunciado

- Usar `np.arange` para generar un rango de **n** valores.
- Calcular una aproximación de π usando la serie de Leibniz:

$$\pi \approx 4 \sum_{k=0}^{n} \frac{(-1)^k}{2k+1}$$

- Comparar el valor obtenido para distintos **n**.

Sugerencia: Emplear **vectorización** para computar $\frac{(-1)^k}{2k+1}$.

- Dividir la clase en **parejas o tríos**.
- Resolver los ejercicios en un **notebook de Colab**.
- Explorar ejemplos adicionales:
  - Generar **matrices aleatorias** (`np.random.rand()`) y calcular su determinante (**opcional, con `np.linalg.det`**).
  - Practicar slicing y reemplazo de submatrices.

- ¿Problemas de instalación o importación en Colab?
- ¿Dificultad para entender la forma (`shape`) de un arreglo?
- ¿Alguna confusión con slicing, vistas y copias?

Comparte tus dudas o experiencias, discutimos en conjunto.

```python
import numpy as np

arr = np.arange(1, 11)  # [1..10]
print("Array:", arr)

print("Media:", np.mean(arr))   # 5.5
print("Suma:", np.sum(arr))     # 55
print("Producto:", np.prod(arr))# 3628800

# Reemplazo valores <=5 con 0, y >5 con 1
arr_mod = arr.copy()
arr_mod[arr_mod <= 5] = 0
arr_mod[arr_mod > 5] = 1
print("Array modificado:", arr_mod)
```

```python
import numpy as np

mat = np.array([[1,2,3],
                [4,5,6],
                [7,8,9]])
fila = np.array([10, 20, 30])

resultado = mat + fila  # Broadcasting
print("Matriz resultante:\n", resultado)
# [[11 22 33]
#  [14 25 36]
#  [17 28 39]]
```

Operaciones Numéricas y Práctica ∋ Ejemplo: Solución Ejercicio 3 (Leibniz π)

```python
import numpy as np

n = 100000  # término superior
k = np.arange(n+1)  # [0..n]

# (-1)^k = np.power(-1, k)
# denominador = (2*k + 1)
terminos = ((-1)**k) / (2*k + 1)
aprox_pi = 4 * np.sum(terminos)
print("Aproximación de pi con n =", n, ":", aprox_pi)

# Comparar con valor real
import math
print("Diferencia:", abs(math.pi - aprox_pi))
```

- **Verificación**: ¿Coincidió con tus resultados o surgieron discrepancias?
- **Rendimiento**: usar vectorización en lugar de bucles manuales.
- **Optimización**: manipulación y slicing evitan copias innecesarias.

Conclusiones y Próximos Pasos

- Entendimos la importancia de **NumPy** para cálculos numéricos y arreglos multidimensionales.
- Practicamos **creación**, **slicing**, **operaciones vectorizadas** y **broadcasting**.
- Vimos ejemplos de aplicación en problemas matemáticos y de simulación.
- Próximamente, profundizaremos en **manipulación avanzada** de datos y su visualización (Matplotlib, etc.).

¡Gracias por su atención!

- En la **próxima sesión** (Semana 6, Sesión 2) exploraremos más operaciones avanzadas con NumPy (reshaping, funciones linalg, etc.) y su integración con **Matplotlib**.
- ¡Sigan practicando con los ejercicios y datos reales para asimilar mejor NumPy!