GUSTAVO CORONEL
DESARROLLA SOFTWARE

CIENCIA DE DATOS: TALLER DE FUNDAMENTOS DE MACHINE LEARNING CON PYTHON

Módulo 03
COMPUTACIÓN NUMÉRICA CON NUMPY

Dr. Eric Gustavo Coronel Castillo
Docente UNI
gcoronel@uni.edu.pe

ÍNDICE

1 PRESENTACIÓN 5

2 OBJETIVO 6

2.1 OBJETIVO GENERAL 6

2.2 OBJETIVOS ESPECÍFICOS 6

3 LIBRERÍAS: INSTALACIÓN E IMPORTACIÓN 7

3.1 INTRODUCCIÓN A NUMPY 7

3.1.1 ¿Qué es NumPy? 7

3.1.2 Importancia en Ciencia de Datos 7

3.2 INSTALACIÓN DE NUMPY 8

3.2.1 Verificación si NumPy Existente 8

3.2.2 Verificación de Python y Gestores de Paquetes 10

3.2.3 Instalación con pip 10

3.2.4 Instalación con conda 10

3.2.5 Buenas Prácticas: Uso de Entornos Virtuales 11

3.2.6 Verificación de la Instalación 12

3.3 IMPORTACIÓN DE NUMPY 13

3.3.1 Convención Estándar de Importación 13

3.3.2 Convenciones de Alias en el Ecosistema de Ciencia de Datos 14

3.4 JUPYTER NOTEBOOK: ENTORNO INTERACTIVO PARA CIENCIA DE DATOS 15

3.4.1 ¿Qué es Jupyter Notebook? 15

3.4.2 Instalación de Jupyter Notebook 15

3.4.3 Iniciando Jupyter Notebook 17

3.4.4 Opciones mas seguras para su instalación 17

3.4.5 Atajos de teclado útiles en Jupyter: 18

3.5 EJEMPLOS PRÁCTICOS ILUSTRATIVOS 18

3.5.1 Ejemplo 1: Notebook de Verificación Completa 18

3.6 EJERCICIOS PROPUESTOS 20
3.6.1 Ejercicio 1: Configuración Completa del Entorno 20
3.6.2 Ejercicio 2: Primer Notebook de NumPy 20
3.6.3 Ejercicio 3: Diagnóstico de Entorno 20
4 ARREGLOS CON NUMPY: ARRAY & NDARRAY 21
4.1 INTRODUCCIÓN A LOS ARREGLOS DE NUMPY 21
4.1.1 ¿Qué es un ndarray? 21
4.1.2 Diferencia entre array y ndarray 21
4.2 CREACIÓN DE ARREGLOS CON NUMPY 22
4.2.1 Creación con numpy.array() 22
4.2.2 Funciones de Creación de Arreglos 25
4.3 ATRIBUTOS FUNDAMENTALES DEL NDARRAY 29
4.3.1 ndarray.shape - Forma del Arreglo 29
4.3.2 ndarray.ndim - Número de Dimensiones 29
4.3.3 ndarray.size - Número Total de Elementos 30
4.3.4 ndarray.dtype - Tipo de Dato 30
4.3.5 ndarray.itemsize - Tamaño de Cada Elemento 30
4.3.6 ndarray.nbytes - Bytes Totales 31
4.3.7 Tabla Resumen de Atributos 31
4.4 EJEMPLO INTEGRADOR COMPLETO 31
4.5 EJERCICIOS PROPUESTOS 33
4.5.1 Ejercicio 1: Creación de Arreglos Básicos 33
4.5.2 Ejercicio 2: Funciones de Creación 33
4.5.3 Ejercicio 3: Análisis de Atributos 34
4.5.4 Ejercicio 4: Conversión de Tipos 34
4.5.5 Ejercicio 5: Proyecto Integrador 34
5 7. REFERENCIAS 35

1 Presentación

En este módulo se introduce el uso de NumPy (Numerical Python) como base de la computación numérica en Python y como pilar del ecosistema de ciencia de datos. NumPy permite trabajar con arreglos multidimensionales (ndarray) y realizar operaciones vectorizadas de forma eficiente, lo que resulta clave cuando se procesan datos numéricos en análisis, visualización y aprendizaje automático.

A lo largo del módulo, el estudiante aprenderá a instalar e importar NumPy siguiendo convenciones del entorno profesional, y a configurar un entorno de trabajo práctico con Jupyter Notebook para ejecutar y documentar análisis de manera interactiva. Luego, se desarrollan los fundamentos de los arreglos en NumPy: creación de arreglos con numpy.array() y funciones como zeros(), ones(), full(), arange() y linspace(), así como la comprensión de atributos esenciales como shape, ndim, size, dtype y el consumo de memoria (itemsize, nbytes).

Finalmente, el módulo integra estos conceptos mediante ejemplos guiados y ejercicios propuestos orientados a consolidar habilidades operativas y criterios básicos de buenas prácticas. El propósito es que el estudiante adquiera una base sólida para trabajar con datos numéricos de forma ordenada y eficiente, preparando el terreno para etapas posteriores del curso, como análisis de datos y construcción de modelos de machine learning.

2 Objetivo

2.1 Objetivo general

2.2 Objetivos específicos

3 Librerías: instalación e importación

3.1 Introducción a NumPy

3.1.1 ¿Qué es NumPy?

NumPy (Numerical Python) es la biblioteca fundamental para la computación científica en Python (NumPy, 2025). Proporciona soporte para arreglos multidimensionales de gran tamaño y una amplia colección de funciones matemáticas de alto nivel para operar con estos arreglos de manera eficiente.

Según la documentación oficial de NumPy (2025), esta biblioteca es esencial porque ofrece:

- **ndarray:** Un objeto de arreglo multidimensional eficiente y versátil
- **Operaciones vectorizadas:** Capacidad para realizar operaciones matemáticas sobre arreglos completos sin necesidad de bucles explícitos
- **Rendimiento optimizado:** Implementación en C y Fortran que hace que las operaciones sean hasta 50 veces más rápidas que las listas nativas de Python
- **Interoperabilidad:** Base para el ecosistema científico de Python, incluyendo Pandas, SciPy, Matplotlib y scikit-learn

3.1.2 Importancia en Ciencia de Datos

NumPy es el pilar sobre el cual se construye gran parte del ecosistema de ciencia de datos en Python. Su eficiencia en el manejo de datos numéricos lo convierte en una herramienta indispensable para:

- Análisis de grandes volúmenes de datos
- Álgebra lineal y operaciones matriciales
- Procesamiento de señales e imágenes
- Simulaciones científicas y computación numérica
- Preparación de datos para modelos de aprendizaje automático

3.2 Instalación de NumPy

Antes de utilizar NumPy en cualquier proyecto de ciencia de datos, es necesario instalarlo en el entorno de Python. La instalación varía según el gestor de paquetes que se utilice.

3.2.1 Verificación si NumPy Existente

Existen varios métodos para verificar si `NumPy` ya está instalado en su sistema. A continuación, se presentan las opciones más comunes y confiables.

Método 1: Verificación con pip

Verificación desde la línea de comandos:

```
pip show numpy
```

Salida esperada si está instalado:

```
Name: numpy
Version: 2.4.2
Summary: Fundamental package for array computing in Python
Location: C:\Users\Usuario\anaconda3\lib\site-packages
Requires: ...
```

Salida si NO está instalado:

```
WARNING: Package(s) not found: numpy
```

Método 2: Verificación directa desde Python

Este método intenta importar NumPy y muestra su versión:

```
python -c "import numpy; print(numpy.__version__)"
```

Salida esperada si está instalado:

```
2.4.2
```

Salida si NO está instalado:

```
ModuleNotFoundError: No module named 'numpy'
```

Método 3: Verificación con conda

Si utiliza Anaconda, puede verificar con conda:

```
conda list numpy
```

Salida esperada si está instalado:

```
# packages in environment at C:\Users\Usuario\anaconda3:
#
# Name                    Version                   Build  Channel
numpy                     2.4.2           py311h5bd46f5_0
```

Método 4: Script de diagnóstico completo (Recomendado)

Para una verificación exhaustiva, puede crear un script de diagnóstico:

```
# verificar_numpy.py

import sys

def verificar_numpy_instalado():
    try:
        import numpy as np
        print(f"✓ NumPy YA está instalado")
        print(f"  Versión: {np.__version__}")
        print(f"  Ubicación: {np.__file__}")
        return True
    except ImportError:
        print("✗ NumPy NO está instalado")
        print("  Proceda con la instalación")
        return False

if __name__ == "__main__":
    verificar_numpy_instalado()
```

3.2.2 Verificación de Python y Gestores de Paquetes

Antes de instalar NumPy, es importante verificar que Python esté correctamente instalado. Para ello, ejecute el siguiente comando en la terminal o línea de comandos:

```
python --version
```

o, en algunos sistemas:

```
python3 --version
```

Debería ver una salida similar a: `Python 3.13.9`

Para verificar la versión de conda ejecute el siguiente comando:

```
conda --version
```

Debería tener una salida similar a: `conda 25.11.0`

3.2.3 Instalación con pip

El gestor de paquetes estándar de Python es `pip`. Según la documentación oficial de NumPy (2025), la forma más directa de instalar `NumPy` es mediante el siguiente comando:

```
pip install numpy
```

Instalación de una versión específica:

Si necesita instalar una versión particular de `NumPy` (por ejemplo, para compatibilidad con otros proyectos), puede especificarla explícitamente:

```
pip install numpy==2.4.2
```

Actualización de NumPy:

Para actualizar `NumPy` a la versión más reciente:

```
pip install --upgrade numpy
```

3.2.4 Instalación con conda

`conda` es el gestor de paquetes incluido en la distribución Anaconda, ampliamente utilizada en ciencia de datos. Según la documentación de NumPy (2025), conda ofrece ventajas significativas sobre pip:

- **Multiplataforma y multilenguaje:** conda puede instalar Python mismo y bibliotecas no-Python (compiladores, CUDA, HDF5)
- **Gestión de dependencias robusta:** maneja dependencias complejas de manera más eficiente

- **Entornos aislados:** solución integrada para crear y gestionar entornos virtuales

Instalación básica con conda:

```
conda install numpy
```

Instalación desde conda-forge:

El canal conda-forge suele ofrecer versiones más actualizadas:

```
conda install -c conda-forge numpy
```

3.2.5 Buenas Prácticas: Uso de Entornos Virtuales

Es altamente recomendable instalar NumPy dentro de un entorno virtual para evitar conflictos de dependencias entre proyectos. Esto es especialmente importante en ciencia de datos, donde diferentes proyectos pueden requerir versiones distintas de las mismas bibliotecas.

Opción 1: Command Prompt

```
REM 1. Crear el entorno virtual
python -m venv curso70160

REM 2. Activar el entorno virtual
curso70160\Scripts\activate.bat

REM 3. Verificar que el entorno está activo
where python

REM 4. Instalar NumPy
pip install numpy

REM 5. Verificar la instalación
python -c "import numpy; print(f'NumPy {numpy.__version__} instalado
correctamente')"

REM 6. Cuando termines, desactivar el entorno
Deactivate
```

Opción 2: Anaconda Prompt

```
REM 1. Consultar los entornos actuales
conda env list

REM 2. También puedes usar
onda info --envs

REM 3. Crear el entorno con conda.
REM 3. Más robusto para ciencia de datos.
conda create -n curso70160 python=3.11

REM 4. Activar el entorno
conda activate curso70160

REM 5. Verificar la instala NumPy
conda list numpy

REM 6. Instalar NumPy
conda install numpy

REM 7. Desactivar entorno
conda deactivate

REM 7. Eliminar entorno
conda env remove --name curso70160
```

Nota importante: Se recomienda especificar la versión de Python (python=3.11) al crear el entorno para garantizar compatibilidad y reproducibilidad. Python 3.11 ofrece un excelente balance entre modernidad, estabilidad y compatibilidad con el ecosistema de ciencia de datos.

3.2.6 Verificación de la Instalación

Una vez completada la instalación, es importante verificar que NumPy se instaló correctamente y conocer qué versión está disponible. Ejecute el siguiente comando:

```
python -c "import numpy; print(numpy.__version__)"
```

La salida debería mostrar la versión instalada, por ejemplo: 2.4.2

Al momento de escribir esta separata (febrero de 2026), la versión estable más reciente de NumPy es la 2.4.2, lanzada el 20 de diciembre de 2025 (NumPy, 2025).

3.3 Importación de NumPy

Una vez instalado NumPy, el siguiente paso es importarlo en los scripts o notebooks de Python. La forma en que se importa una biblioteca puede afectar significativamente la legibilidad y mantenibilidad del código.

3.3.1 Convención Estándar de Importación

La documentación oficial de NumPy (2025) establece claramente la convención estándar para importar la biblioteca:

```
import numpy as np
```

Esta convención es prácticamente universal en la comunidad de Python y ciencia de datos. El uso del alias np permite:

- **Concisión:** Reduce la cantidad de código a escribir `np.array()` en lugar de `numpy.array()`
- **Legibilidad:** Facilita identificar que se están usando funciones de NumPy
- **Consistencia:** Permite que el código sea comprensible para cualquier desarrollador familiarizado con Python científico
- **Compatibilidad:** La mayoría de ejemplos en documentación y tutoriales usan esta convención

3.3.2 Convenciones de Alias en el Ecosistema de Ciencia de Datos

El ecosistema de ciencia de datos en Python ha establecido convenciones de alias que son reconocidas internacionalmente. Estas son las más comunes:

Librería	Alias Estándar	Área de Aplicación
numpy	np	Computación numérica
pandas	pd	Análisis de datos
matplotlib.pyplot	plt	Visualización
seaborn	sns	Visualización estadística
tensorflow	tf	Aprendizaje automático

Importante: Respetar estas convenciones facilita la colaboración y el mantenimiento del código. Cualquier científico de datos que lea su código esperará ver np para NumPy, no otro alias.

3.4 Jupyter Notebook: Entorno Interactivo para Ciencia de Datos

Jupyter Notebook es una aplicación web de código abierto que permite crear y compartir documentos que contienen código ejecutable, ecuaciones, visualizaciones y texto narrativo. Es la herramienta estándar para el trabajo interactivo en ciencia de datos y se integra perfectamente con NumPy y todo el ecosistema científico de Python.

3.4.1 ¿Qué es Jupyter Notebook?

Jupyter Notebook (anteriormente conocido como IPython Notebook) es un entorno de computación interactiva que permite combinar código, resultados, gráficos y documentación en un solo documento llamado notebook. Su nombre proviene de tres lenguajes de programación: Julia, Python y R.

Ventajas de Jupyter Notebook para Ciencia de Datos:

- **Ejecución interactiva:** Ejecute código celda por celda y vea resultados inmediatos
- **Visualización integrada:** Los gráficos se muestran directamente en el notebook
- **Documentación narrativa:** Combine código con explicaciones en Markdown
- **Reproducibilidad:** Comparta análisis completos con código y resultados
- **Exploración iterativa:** Ideal para análisis exploratorio de datos (EDA)

3.4.2 Instalación de Jupyter Notebook

La instalación de Jupyter Notebook varía según si está utilizando Anaconda o una instalación estándar de Python. A continuación, se presentan ambas opciones para Windows.

Opción 1: Con Anaconda

Si tiene Anaconda instalado, Jupyter Notebook ya está incluido en la instalación base. Sin embargo, es recomendable instalarlo también en su entorno específico:

```
REM 1. Activar su entorno (ejemplo: curso70160)
conda activate curso70160

REM 2. Instalar Jupyter Notebook en el entorno
conda install jupyter
```

```
REM 3. Verificar la instalación
jupyter --version

REM 4. Iniciar Jupyter Notebook
jupyter notebook
```

Opción 2: Con pip (Python estándar)

Si está utilizando un entorno virtual creado con venv:

```
REM 1. Activar el entorno virtual
mi_proyecto\Scripts\activate.bat

REM 2. Instalar Jupyter
pip install jupyter

REM 3. Iniciar Jupyter Notebook
jupyter notebook
```

Opción 3: Instalación completa del stack de ciencia de datos

Para instalar Jupyter junto con todas las bibliotecas necesarias para el curso:

```
conda activate curso70160
conda install jupyter numpy pandas matplotlib seaborn
```

3.4.3 Iniciando Jupyter Notebook

Una vez instalado Jupyter, puede iniciarlo de diferentes maneras según su entorno de trabajo.

Método 1: Desde la línea de comandos (Recomendado)

```
REM 1. Activar el entorno
conda activate curso70160

REM 2. Navegar a la carpeta de su proyecto
cd C:\Users\Usuario\MisProyectos\CursoPython

REM 3. Iniciar Jupyter Notebook
jupyter notebook
```

Esto abrirá automáticamente Jupyter en su navegador web predeterminado, generalmente en `http://localhost:8888`.

Método 2: Desde Anaconda Navigator (Interfaz gráfica)

1. Abra Anaconda Navigator desde el menú de inicio
2. En el menú desplegable de entornos, seleccione su entorno (curso70160)
3. Haga clic en el botón Launch debajo del ícono de Jupyter Notebook

3.4.4 Opciones mas seguras para su instalación

Si ya tienes el entorno activo

```
conda install -c conda-forge python=3.11 jupyter
```

¿Por qué esta es la opción más segura?

1. **Canal conda-forge:** Este repositorio comunitario suele tener versiones más actualizadas y mejor testeadas para evitar el error de "Solving environment" que tuviste antes.
2. **Especificación de versión:** Al incluir `python=3.11`, fuerzas a Conda a resolver todas las dependencias de Jupyter específicamente para esa versión, ignorando cualquier intento de usar la 3.14.

crear un entorno limpio desde cero

Es la forma más fiable de garantizar que todo funcione:

```
conda create -n mi_entorno_jupyter python=3.11 jupyter -c conda-forge
```

Verificación

Una vez termine la instalación, puedes verificar que Jupyter use la versión correcta ejecutando:

```
jupyter notebook
```

```
jupyter lab
```

3.4.5 Atajos de teclado útiles en Jupyter:

- **Shift + Enter**: Ejecutar celda actual y avanzar a la siguiente
- **Ctrl + Enter**: Ejecutar celda actual sin avanzar
- **A**: Insertar celda arriba (en modo comando)
- **B**: Insertar celda abajo (en modo comando)
- **M**: Convertir celda a Markdown
- **Y**: Convertir celda a código
- **DD**: Eliminar celda (presionar D dos veces)

3.5 Ejemplos Prácticos Ilustrativos

A continuación, se presentan ejemplos prácticos que demuestran la correcta instalación e importación de NumPy, así como operaciones básicas para verificar su funcionamiento en un entorno de Jupyter Notebook.

3.5.1 Ejemplo 1: Notebook de Verificación Completa

Este notebook verifica la instalación de NumPy y realiza operaciones básicas. Cree un nuevo notebook llamado `verificacion_entorno.ipynb` y ejecute las siguientes celdas.

Celda 1 - Verificación del entorno

```
import sys
import numpy as np

print("=" * 60)
print("VERIFICACIÓN DEL ENTORNO DE TRABAJO")
print("=" * 60)
print(f"\nPython: {sys.version}")
print(f"Ubicación: {sys.executable}")
print(f"\nNumPy: {np.__version__}")
print(f"Ubicación: {np.__file__}")
```

Celda 2 - Crear arreglo y operaciones

```
# Crear un arreglo simple
arr = np.array([1, 2, 3, 4, 5])

print("\nARREGLO CREADO:")
print(f"Contenido: {arr}")
print(f"Tipo: {type(arr)}")
print(f"Forma: {arr.shape}")
print(f"Tipo de datos: {arr.dtype}")

print("\nOPERACIONES BÁSICAS:")
print(f"Suma: {np.sum(arr)}")
print(f"Media: {np.mean(arr)}")
print(f"Desviación estándar: {np.std(arr)}")
print(f"Mínimo: {np.min(arr)}")
print(f"Máximo: {np.max(arr)}")
```

3.6 Ejercicios Propuestos

Los siguientes ejercicios están diseñados para consolidar los conceptos de verificación, instalación, importación de NumPy y uso de Jupyter Notebook.

3.6.1 Ejercicio 1: Configuración Completa del Entorno

Instrucciones:

1. Verifique si NumPy está instalado usando al menos dos métodos diferentes
2. Cree un entorno virtual llamado entorno_datascience con Python 3.11
3. Active el entorno e instale NumPy y Jupyter
4. Verifique las versiones instaladas
5. Inicie Jupyter Notebook y verifique que usa el entorno correcto

3.6.2 Ejercicio 2: Primer Notebook de NumPy

Instrucciones:

Cree un notebook llamado `intro_numpy.ipynb` con las siguientes celdas:

6. Una celda Markdown con el título del notebook y su nombre
7. Una celda de código que importe NumPy y muestre su versión
8. Una celda que cree un arreglo con números del 1 al 10
9. Una celda que calcule y muestre: suma, media, mediana, desviación estándar
10. Una celda Markdown que explique los resultados obtenidos

3.6.3 Ejercicio 3: Diagnóstico de Entorno

Instrucciones:

Cree un script Python (diagnostico.py) que:

11. Verifique si NumPy está instalado
12. Si está instalado, muestre versión y ubicación
13. Si NO está instalado, muestre instrucciones de instalación
14. Verifique la versión de Python en uso
15. Muestre un mensaje indicando si el entorno está listo para el curso

4 ARREGLOS CON NUMPY: array & ndarray

4.1 Introducción a los Arreglos de NumPy

El objeto fundamental de NumPy es el ndarray (N-dimensional array), también conocido simplemente como array. Según la documentación oficial de NumPy (2025), un ndarray es un contenedor multidimensional de elementos del mismo tipo y tamaño, optimizado para operaciones numéricas de alto rendimiento.

Definición formal: Un ndarray representa un arreglo multidimensional homogéneo de elementos de tamaño fijo. El número de dimensiones y elementos en un arreglo está definido por su forma (shape), que es una tupla de N enteros no negativos que especifican el tamaño de cada dimensión (NumPy, 2025).

4.1.1 ¿Qué es un ndarray?

La clase numpy.ndarray es el objeto más importante definido en NumPy. Según la documentación oficial (NumPy, 2025), tiene las siguientes características esenciales:

- **Homogeneidad:** Todos los elementos deben ser del mismo tipo de dato
- **Multidimensionalidad:** Puede tener 0, 1, 2, 3 o más dimensiones
- **Tamaño fijo:** Generalmente, el tamaño es fijo al momento de la creación
- **Memoria contigua:** Los elementos se almacenan en un bloque continuo de memoria
- **Eficiencia:** Operaciones optimizadas implementadas en C y Fortran

4.1.2 Diferencia entre array y ndarray

Es importante aclarar la nomenclatura. Según la documentación de NumPy (2025):

- **numpy.ndarray**: Es la clase que define el objeto de arreglo N-dimensional
- **numpy.array()**: Es una función para crear instancias de ndarray
- **array**: Es un alias común para referirse a objetos ndarray

> ***Nota importante:*** `numpy.array` *NO es lo mismo que* `array.array` *de la biblioteca estándar de Python, que solo maneja arreglos unidimensionales y ofrece menos funcionalidad (NumPy, 2025).*

4.2 Creación de Arreglos con NumPy

Según la documentación oficial (NumPy, 2025), los arreglos deben construirse usando funciones como `array()`, `zeros()` o `empty()`. Aunque técnicamente es posible crear arreglos usando directamente el constructor ndarray(), este es un método de bajo nivel y no se recomienda para uso general.

4.2.1 Creación con numpy.array()

La función `numpy.array()` es el método más común y versátil para crear arreglos. Convierte secuencias de Python (listas, tuplas) en objetos `ndarray` (NumPy, 2025).

Sintaxis básica:

```
numpy.array(object, dtype=None, copy=True, order='K',
            subok=False, ndmin=0)
```

Parámetros principales:

- **object:** Secuencia, arreglo o cualquier objeto que exponga la interfaz de arreglo
- **dtype:** Tipo de dato deseado (int, float, etc.). Si no se especifica, NumPy lo infiere
- **copy:** Si True (predeterminado), copia los datos
- **ndmin:** Número mínimo de dimensiones del arreglo resultante

4.2.1.1 Arreglos 0-D (Escalares)

Un arreglo 0-dimensional contiene un solo elemento escalar:

```
import numpy as np

# Crear un escalar
escalar = np.array(42)
print(escalar)
print(f"Dimensiones: {escalar.ndim}")
print(f"Forma: {escalar.shape}")
```

Salida:

```
42
Dimensiones: 0
Forma: ()
```

4.2.1.2 Arreglos 1-D (Unidimensionales)

Los arreglos 1-D son los más comunes y básicos. Contienen una secuencia de elementos:

```
# Desde una lista
arr1d = np.array([1, 2, 3, 4, 5])
print(arr1d)
print(f"Tipo: {type(arr1d)}")
print(f"Dimensiones: {arr1d.ndim}")
print(f"Forma: {arr1d.shape}")
print(f"Tamaño: {arr1d.size}")
```

Salida:

```
[1 2 3 4 5]
Tipo: <class 'numpy.ndarray'>
Dimensiones: 1
Forma: (5,)
Tamaño: 5
```

Desde una tupla:

```
# NumPy acepta tanto listas como tuplas
arr_tupla = np.array((10, 20, 30, 40))
print(arr_tupla)
```

Salida:

```
[10 20 30 40]
```

4.2.1.3 Arreglos 2-D (Bidimensionales)

Los arreglos 2-D representan matrices. Se crean usando listas anidadas:

```
# Crear una matriz 3x3
arr2d = np.array([[1, 2, 3],
                  [4, 5, 6],
                  [7, 8, 9]])

print(arr2d)
print(f"Dimensiones: {arr2d.ndim}")
print(f"Forma: {arr2d.shape}")
print(f"Tamaño total: {arr2d.size}")
```

Salida:

```
[[1 2 3]
 [4 5 6]
 [7 8 9]]
Dimensiones: 2
Forma: (3, 3)
Tamaño total: 9
```

4.2.1.4 Arreglos 3-D y Superiores

NumPy soporta arreglos de N dimensiones. Cada nivel adicional de anidamiento agrega una dimensión:

```
# Arreglo 3-D (2x2x2)
arr3d = np.array([[[1, 2], [3, 4]],
                  [[5, 6], [7, 8]]])

print(arr3d)
print(f"Dimensiones: {arr3d.ndim}")
print(f"Forma: {arr3d.shape}")
```

Salida:

```
[[[1 2]
  [3 4]]

 [[5 6]
  [7 8]]]
Dimensiones: 3
Forma: (2, 2, 2)
```

4.2.1.5 Especificación del Tipo de Dato (dtype)

Según NumPy (2025), el parámetro `dtype` permite especificar explícitamente el tipo de dato de los elementos. Si no se especifica, NumPy infiere el tipo automáticamente.

```
# NumPy infiere int64
arr_int = np.array([1, 2, 3])
print(f"Tipo inferido: {arr_int.dtype}")

# NumPy infiere float64
arr_float = np.array([1.0, 2.0, 3.0])
print(f"Tipo inferido: {arr_float.dtype}")

# Especificar dtype explícitamente
arr_especifico = np.array([1, 2, 3], dtype=np.float32)
print(f"Tipo especificado: {arr_especifico.dtype}")
```

Salida:

```
Tipo inferido: int64
Tipo inferido: float64
Tipo especificado: float32
```

4.2.2 Funciones de Creación de Arreglos

NumPy proporciona múltiples funciones especializadas para crear arreglos con valores predefinidos (NumPy, 2025).

4.2.2.1 numpy.zeros()

Crea un arreglo lleno de ceros:

```
# Arreglo 1-D de 5 ceros
zeros_1d = np.zeros(5)
print(zeros_1d)

# Matriz 3x4 de ceros
zeros_2d = np.zeros((3, 4))
print(zeros_2d)
```

Salida:

```
[0. 0. 0. 0. 0.]

[[0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]]
```

4.2.2.2 numpy.ones()

Crea un arreglo lleno de unos:

```
# Matriz 2x3 de unos
ones_2d = np.ones((2, 3))
print(ones_2d)
```

Salida:

```
[[1. 1. 1.]
 [1. 1. 1.]]
```

4.2.2.3 numpy.full()

Crea un arreglo lleno de un valor constante específico:

```
# Matriz 2x2 llena de 7
full_arr = np.full((2, 2), 7)
print(full_arr)
```

Salida:

```
[[7 7]
 [7 7]]
```

4.2.2.4 numpy.empty()

Crea un arreglo sin inicializar (contenido aleatorio basado en el estado de la memoria). Según NumPy (2025), se usa por velocidad cuando los valores se llenarán posteriormente:

```
# Arreglo vacío de 3 elementos
empty_arr = np.empty(3)
print(empty_arr)  # Valores aleatorios
```

4.2.2.5 numpy.eye() - Matriz Identidad

Crea una matriz identidad (unos en la diagonal, ceros en el resto):

```
# Matriz identidad 3x3
identidad = np.eye(3)
print(identidad)
```

Salida:

```
[[1. 0. 0.]
 [0. 1. 0.]
 [0. 0. 1.]]
```

4.2.2.6 numpy.arange() - Secuencias Numéricas

Crea arreglos con valores espaciados regularmente. Según NumPy (2025), la mejor práctica es usar valores enteros para inicio, fin y paso.

```
# Del 0 al 9
arange1 = np.arange(10)
print(arange1)

# Del 2 al 9
arange2 = np.arange(2, 10)
print(arange2)

# Del 0 al 20 de 2 en 2
arange3 = np.arange(0, 21, 2)
print(arange3)
```

Salida:

```
[0 1 2 3 4 5 6 7 8 9]
[2 3 4 5 6 7 8 9]
[ 0  2  4  6  8 10 12 14 16 18 20]
```

4.2.2.7 numpy.linspace() - Espaciado Lineal

Crea arreglos con un número específico de elementos espaciados uniformemente entre dos valores:

```
# 5 valores entre 0 y 10 (incluye ambos extremos)
linspace_arr = np.linspace(0, 10, num=5)
print(linspace_arr)
```

Salida:

```
[ 0.   2.5  5.   7.5 10. ]
```

4.3 Atributos Fundamentales del ndarray

Según la documentación oficial (NumPy, 2025), los atributos de un ndarray reflejan información intrínseca del arreglo. A continuación se desarrollan los mas utilizados.

4.3.1 ndarray.shape - Forma del Arreglo

Tupla que indica el tamaño del arreglo en cada dimensión:

```
arr = np.array([[1, 2, 3, 4],
                [5, 6, 7, 8],
                [9, 10, 11, 12]])

print(f"Forma: {arr.shape}")
# Forma: (3, 4)  -> 3 filas, 4 columnas
```

4.3.2 ndarray.ndim - Número de Dimensiones

Número de ejes (dimensiones) del arreglo:

```
arr_1d = np.array([1, 2, 3])
arr_2d = np.array([[1, 2], [3, 4]])
arr_3d = np.array([[[1, 2]], [[3, 4]]])

print(f"1-D: {arr_1d.ndim} dimensiones")
print(f"2-D: {arr_2d.ndim} dimensiones")
print(f"3-D: {arr_3d.ndim} dimensiones")
```

Salida:

```
1-D: 1 dimensiones
2-D: 2 dimensiones
3-D: 3 dimensiones
```

4.3.3 ndarray.size - Número Total de Elementos

Cantidad total de elementos en el arreglo:

```
arr = np.array([[1, 2, 3], [4, 5, 6]])
print(f"Forma: {arr.shape}")
print(f"Tamaño: {arr.size}")
# size = 2 × 3 = 6 elementos
```

4.3.4 ndarray.dtype - Tipo de Dato

Objeto que describe el tipo de los elementos:

```
arr_int = np.array([1, 2, 3])
arr_float = np.array([1.5, 2.5, 3.5])

print(f"Tipo de arr_int: {arr_int.dtype}")
print(f"Tipo de arr_float: {arr_float.dtype}")
```

Salida:

```
Tipo de arr_int: int64
Tipo de arr_float: float64
```

4.3.5 ndarray.itemsize - Tamaño de Cada Elemento

Tamaño en bytes de cada elemento del arreglo:

```
arr_int32 = np.array([1, 2, 3], dtype=np.int32)
arr_float64 = np.array([1.0, 2.0], dtype=np.float64)

print(f"int32: {arr_int32.itemsize} bytes")
print(f"float64: {arr_float64.itemsize} bytes")
```

Salida:

```
int32: 4 bytes  (32 bits / 8 = 4 bytes)
float64: 8 bytes  (64 bits / 8 = 8 bytes)
```

4.3.6 ndarray.nbytes - Bytes Totales

Memoria total consumida por los elementos del arreglo:

```
arr = np.array([[1, 2, 3], [4, 5, 6]], dtype=np.int32)
print(f"Tamaño: {arr.size} elementos")
print(f"Item size: {arr.itemsize} bytes")
print(f"Total bytes: {arr.nbytes} bytes")
# nbytes = size × itemsize = 6 × 4 = 24 bytes
```

Salida:

```
Tamaño: 6 elementos
Item size: 4 bytes
Total bytes: 24 bytes
```

4.3.7 Tabla Resumen de Atributos

Atributo	Descripción	Ejemplo
`.shape`	Tupla con dimensiones	`(3, 4)`
`.ndim`	Número de dimensiones	`2`
`.size`	Número total de elementos	`12`
`.dtype`	Tipo de dato	`int64`
`.itemsize`	Bytes por elemento	`8`
`.nbytes`	Bytes totales	`96`

4.4 Ejemplo Integrador Completo

El siguiente ejemplo en Jupyter Notebook consolida todos los conceptos aprendidos:

```
import numpy as np

# Crear diferentes tipos de arreglos
print("=" * 60)
print("EJEMPLO INTEGRADOR: CREACIÓN Y EXPLORACIÓN DE ARREGLOS")
print("=" * 60)
```

```
# 1. Arreglo 1-D desde lista
arr1d = np.array([10, 20, 30, 40, 50])
print("\n1. ARREGLO 1-D:")
print(arr1d)
print(f"   Tipo: {type(arr1d)}")
print(f"   Shape: {arr1d.shape}")
print(f"   Ndim: {arr1d.ndim}")
print(f"   Size: {arr1d.size}")
print(f"   Dtype: {arr1d.dtype}")

# 2. Matriz 2-D
matriz = np.array([[1, 2, 3, 4],
                   [5, 6, 7, 8],
                   [9, 10, 11, 12]])
print("\n2. MATRIZ 2-D:")
print(matriz)
print(f"   Shape: {matriz.shape} (3 filas, 4 columnas)")
print(f"   Ndim: {matriz.ndim}")
print(f"   Size: {matriz.size}")

# 3. Arreglos con funciones especializadas
print("\n3. ARREGLOS CON FUNCIONES ESPECIALIZADAS:")

zeros = np.zeros((2, 3))
print("   Zeros (2x3):")
print(zeros)

ones = np.ones((2, 2), dtype=np.int32)
print("\n   Ones (2x2, int32):")
print(ones)

identidad = np.eye(3)
print("\n   Identidad (3x3):")
print(identidad)

rango = np.arange(0, 20, 3)
print("\n   Arange (0 a 20, paso 3):")
print(rango)
```

```
lineal = np.linspace(0, 1, 6)
print("\n   Linspace (0 a 1, 6 valores):")
print(lineal)

# 4. Comparación de memoria
arr_int32 = np.array([1, 2, 3, 4, 5], dtype=np.int32)
arr_int64 = np.array([1, 2, 3, 4, 5], dtype=np.int64)

print("\n4. COMPARACIÓN DE MEMORIA:")
print(f"   int32: {arr_int32.itemsize} bytes/elemento, "
      f"{arr_int32.nbytes} bytes total")
print(f"   int64: {arr_int64.itemsize} bytes/elemento, "
      f"{arr_int64.nbytes} bytes total")
```

4.5 Ejercicios Propuestos

Los siguientes ejercicios están diseñados para consolidar el conocimiento sobre creación y manipulación de arreglos NumPy. Realícelos en Jupyter Notebook.

4.5.1 Ejercicio 1: Creación de Arreglos Básicos

1. Cree un arreglo 1-D con los números del 1 al 10
2. Cree una matriz 4x5 con números del 1 al 20
3. Cree un arreglo 3-D de forma (2, 3, 4) con valores de su elección
4. Muestre el shape, ndim, size y dtype de cada uno

4.5.2 Ejercicio 2: Funciones de Creación

5. Cree una matriz 5x5 de ceros con dtype float32
6. Cree una matriz 3x3 de unos con dtype int16
7. Cree una matriz identidad de 6x6
8. Cree un arreglo con números del 10 al 100 de 5 en 5 usando arange
9. Cree un arreglo de 15 valores equiespaciados entre 0 y 50 usando linspace

4.5.3 Ejercicio 3: Análisis de Atributos

Dado el siguiente arreglo:

```
arr = np.array([[[1, 2], [3, 4], [5, 6]],
                [[7, 8], [9, 10], [11, 12]]])
```

Determine:

10. ¿Cuántas dimensiones tiene?
11. ¿Cuál es su forma?
12. ¿Cuántos elementos contiene en total?
13. ¿Cuántos bytes ocupa en memoria?
14. Si cambia el dtype a float64, ¿cuántos bytes ocuparía?

4.5.4 Ejercicio 4: Conversión de Tipos

15. Cree un arreglo con valores [1.7, 2.3, 3.9, 4.1, 5.6] de tipo float64
16. Conviértalo a int32 (observe qué sucede con los decimales)
17. Compare el espacio en memoria ocupado en ambos casos

4.5.5 Ejercicio 5: Proyecto Integrador

Cree un notebook que simule datos de temperatura de una semana:

1. Cree una matriz 7x3 (7 días, 3 mediciones diarias) con temperaturas aleatorias entre 15 y 30 grados usando np.random.uniform(15, 30, (7, 3))
2. Muestre todos los atributos del arreglo (shape, ndim, size, dtype, etc.)
3. Cree etiquetas para los días usando un arreglo 1-D: ['Lun', 'Mar', 'Mié', 'Jue', 'Vie', 'Sáb', 'Dom']
4. Documente su código con celdas Markdown explicando cada paso

5 7. Referencias

Jupyter Project. (2025). *Project Jupyter*. https://jupyter.org/

NumPy. (2025). *Array creation*. NumPy Documentation. https://numpy.org/doc/stable/user/basics.creation.html

NumPy. (2025). *Installing NumPy*. NumPy Documentation. https://numpy.org/install/

NumPy. (2025). *News*. NumPy Official Website. https://numpy.org/news/

NumPy. (2025). *NumPy documentation*. NumPy Developers. . (2025). NumPy v2.4 Manual. . https://numpy.org/doc/stable/

NumPy. (2025). *NumPy quickstart*. NumPy Documentation. https://numpy.org/doc/stable/user/quickstart.html

NumPy. (2025). *numpy.array*. NumPy v2.4 Manual. https://numpy.org/doc/stable/reference/generated/numpy.array.html

NumPy. (2025). *numpy.ndarray*. NumPy v2.4 Manual. . (2025). . . https://numpy.org/doc/stable/reference/generated/numpy.ndarray.html

NumPy. (2025). *NumPy: the absolute basics for beginners*. NumPy Documentation. https://numpy.org/doc/stable/user/absolute_beginners.html

NumPy. (2025). *The N-dimensional array (ndarray)*. NumPy v2.4 Manual. https://numpy.org/doc/stable/reference/arrays.ndarray.html