🧮 NUMPY AND PANDAS: A DETAILED BEGINNER'S GUIDE

Welcome to this comprehensive guide on **NumPy** and **Pandas**! These are two must-know Python libraries for anyone working with data. Whether you're a beginner or looking to deepen your understanding, this guide covers every key concept with detailed explanations, examples, and tips—all in simple English. Let's make learning fun and easy! 🚀

TABLE OF CONTENTS

🧮 PART 1: NUMPY – THE NUMBER CRUNCHER

🌟 WHAT IS NUMPY?

NumPy (short for Numerical Python) is a library that helps you work with numbers efficiently. It's the foundation for many data-related tasks in Python because it handles **arrays**—special lists or grids of numbers—super fast.

Why it's awesome: NumPy is written in C (a fast programming language), so it's much quicker than regular Python lists for math operations.

Who uses it: Data scientists, engineers, researchers, and even game developers!

🎯 KEY CONCEPTS IN NUMPY

Let's break down every important idea in NumPy with examples.

1. 📊 ARRAYS – THE BUILDING BLOCKS

An **array** is like a list, but more powerful. It can be 1D (a row of numbers), 2D (a table), or even higher dimensions (like a cube of numbers). All items in an array must be the same type (e.g., all numbers).

Arrays are the core of NumPy. Create them from lists or use built-in functions.

```
import numpy as np

# From a list
arr1 = np.array([1, 2, 3, 4, 5])
print(arr1)  # [1 2 3 4 5]

# 2D array
arr2 = np.array([[1, 2, 3], [4, 5, 6]])
print(arr2)  # [[1 2 3]
```

```
             #  [4 5 6]]

# Zeros array
zeros = np.zeros((2, 3))
print(zeros)  # [[0. 0. 0.]
              #  [0. 0. 0.]]
```

ARRAY OPERATIONS

Perform math on entire arrays effortlessly.

```
# Add 5 to all elements
print(arr1 + 5)  # [6 7 8 9 10]

# Element-wise multiplication
print(arr1 * arr1)  # [ 1  4  9 16 25]
```

INDEXING AND SLICING

Access specific elements or sections of an array.

```
# First element
print(arr1[0])  # 1

# Slice from index 1 to 3
print(arr1[1:4])  # [2 3 4]

# 2D array: row 0, column 1
print(arr2[0, 1])  # 2
```

BROADCASTING

Apply operations across arrays of different sizes.

```
# Add a scalar to a 2D array
print(arr2 + 5)  # [[6 7 8]
                 #  [9 10 11]]
```

```
# Add a 1D array to a 2D array
print(arr2 + np.array([1, 2, 3]))  # [[2 4 6]
                                   #  [5 7 9]]
```

ARRAY ATTRIBUTES

Get details about your array.

```
arr = np.array([[1, 2, 3], [4, 5, 6]])
print(arr.shape)  # (2, 3) - rows, columns
print(arr.size)   # 6 - total elements
print(arr.dtype)  # int64 - data type
```

UNIVERSAL FUNCTIONS (UFUNCS)

Fast, element-wise math operations.

```
arr = np.array([1, 2, 3])
print(np.sqrt(arr))  # [1. 1.41421356 1.73205081]
print(np.exp(arr))   # [2.71828183 7.3890561
20.08553692]
```

LINEAR ALGEBRA

Handle matrix operations like multiplication or inverses.

```
a = np.array([[1, 2], [3, 4]])
b = np.array([[5, 6], [7, 8]])
print(np.dot(a, b))  # [[19 22]
                     #  [43 50]]
print(a.T)           # [[1 3]
                     #  [2 4]]
```

RANDOM NUMBER GENERATION

Create random data for testing or simulations.

```
# Uniform distribution (0 to 1)
print(np.random.rand(3))  # e.g., [0.5488135  0.71518937
0.60276338]

# Normal distribution
print(np.random.randn(3))  # e.g., [-0.234  1.567 -0.892]
```

ADVANCED INDEXING

Select elements using conditions or index arrays.

```
arr = np.array([1, 2, 3, 4, 5])
mask = arr > 3  # Boolean mask
print(arr[mask])  # [4 5]

# Fancy indexing
print(arr[[0, 2, 4]])  # [1 3 5]
```


🐼 PART 2: PANDAS – THE DATA ORGANIZER

🌟 WHAT IS PANDAS?

Pandas is a library for managing and analyzing data in tables. It's built on NumPy and introduces two key structures: **Series** (1D) and **DataFrames** (2D).

Why it's great: It's like Excel in Python—perfect for organizing, cleaning, and exploring data.

🎯 KEY CONCEPTS IN PANDAS

Let's dive into every major Pandas feature.

1. 📋 SERIES – LABELED LISTS

A **Series** is a 1D array with labels (indices) for each item.

SERIES AND DATAFRAMES

- **Series**: A labeled column.
- **DataFrame**: A table of Series.

```
import pandas as pd

# Series
s = pd.Series([10, 20, 30], index=['Apples', 'Bananas',
'Oranges'])
print(s)  # Apples     10
          # Bananas    20
          # Oranges    30

# DataFrame
df = pd.DataFrame({'Name': ['Alice', 'Bob'], 'Age': [25,
30]})
print(df)  #     Name  Age
           # 0  Alice   25
           # 1    Bob   30
```

READING AND WRITING DATA

Load and save data from/to files.

```
# Read CSV
df = pd.read_csv('data.csv')

# Write to CSV
df.to_csv('output.csv', index=False)
```

DATA SELECTION AND FILTERING

Extract specific data easily.

```
# Select column
print(df['Name'])  # 0    Alice
                   # 1      Bob
```

```
# Filter rows
print(df[df['Age'] > 25])  #   Name  Age
                           # 1  Bob   30
```

DATA CLEANING

Fix missing values, duplicates, or errors.

```
# Fill NaN with 0
df.fillna(0)

# Drop duplicates
df.drop_duplicates()

# Replace values
df['Age'].replace(25, 26)
```

GROUPING AND AGGREGATION

Summarize data by groups.

```
# Group by 'City' and calculate mean 'Sales'
df.groupby('City')['Sales'].mean()
```

MERGING AND JOINING

Combine datasets.

```
df1 = pd.DataFrame({'ID': [1, 2], 'Name': ['Alice',
'Bob']})
df2 = pd.DataFrame({'ID': [1, 2], 'Score': [85, 90]})
merged = pd.merge(df1, df2, on='ID')
print(merged)  #    ID   Name  Score
               # 0   1  Alice     85
               # 1   2    Bob     90
```

TIME SERIES HANDLING

Manage date-based data (e.g., stock prices).

```
# Parse dates
df['Date'] = pd.to_datetime(df['Date'])

# Set index
df.set_index('Date', inplace=True)

# Resample to monthly sums
monthly = df.resample('M').sum()
```

PIVOT TABLES

Reshape data for summaries (e.g., sales reports).

```
pivot = df.pivot_table(values='Sales', index='City',
columns='Product', aggfunc='sum')
```

HANDLING LARGE DATASETS

Process big files efficiently.

```
# Chunking
for chunk in pd.read_csv('large_file.csv',
chunksize=1000):
    process(chunk)

# Optimize types
df['Sales'] = df['Sales'].astype('float32')
```

CUSTOM FUNCTIONS WITH APPLY()

Apply your own logic to data.

```
# Discount prices
df['Discounted'] = df['Price'].apply(lambda x: x * 0.9)
```

VISUALIZATION INTEGRATION

Plot data directly or with libraries.

```
# Built-in plot
df.plot(x='Date', y='Sales')

# Seaborn plot
import seaborn as sns
sns.barplot(x='City', y='Sales', data=df)
```

NUMPY VS PANDAS: WHEN TO USE WHICH

- NumPy:
  - For **numerical computations**.
  - Homogeneous data (same type).
  - Faster for math operations.
- Pandas:
  - For **data analysis**.
  - Heterogeneous data (mixed types).
  - Easier for cleaning and exploration.

COMMON MISTAKES AND HOW TO AVOID THEM

NUMPY

1. **Forgetting Import**: Use `import numpy as np`.
2. **Shape Mismatch**: Check `arr.shape` before operations.
3. **NaN Issues**: Use `np.isnan()` to handle missing values.

PANDAS

1. **Index Errors**: Set with `df.set_index('column')`.
2. **loc vs iloc**: `loc` for labels, `iloc` for positions.
3. **Type Issues**: Check `df.dtypes` and convert (e.g., `astype`).

TIPS AND BEST PRACTICES

- **NumPy**: Use vectorized operations, not loops.
- **Pandas**: Always inspect data with `df.head()` or `df.info()`.

CHALLENGES FOR PRACTICE

NUMPY

- Create a 3x3 array of random integers (1-10).
- Compute row means.
- Replace values > 5 with 0.

PANDAS

- Load a CSV.
- Find top 5 rows by a column value.
- Add a squared column.

RESOURCES FOR FURTHER LEARNING

- NumPy Docs
- Pandas Docs
- Python for Data Analysis by Wes McKinney