Data Analysis with Python

1. What is Data Analysis?

Data analysis is the process of examining, cleaning, transforming, and interpreting data to uncover useful information, draw conclusions, and support decision-making.

2. Various Tools Used to Analyze Data

Data analysis involves multiple tools categorized into:

- **Programming Languages:** Python, R, SQL, Julia.
- **Spreadsheet Software:** Microsoft Excel, Google Sheets.
- **Data Visualization Tools:** Tableau, Power BI, Matplotlib, Seaborn.
- **Databases:** MySQL, PostgreSQL, MongoDB.

Each tool is suited for specific tasks based on data size, type, and required outcomes.

3. Why Python?

Python is a popular programming language for data analysis because:

1. Ease of Use: Python has a simple and intuitive syntax, making it beginner-friendly.
2. Extensive Libraries: Includes powerful libraries like Pandas (data manipulation and analysis), NumPy (numerical computations), Matplotlib/Seaborn (data visualization), and Scikit-learn (machine learning).
3. Flexibility: Can handle tasks from small-scale data processing to building complex machine learning models.
4. Open Source: Free and has a large community for support.
5. Integration: Easily integrates with databases, web services, artificial intelligence and other programming languages.

4. Introduction to Variables

In Python, a variable is used to store data values. It acts as a container for information or a placeholder.

Key Point About Variables:

- The = symbol is used to assign a value to a variable.

Examples of Creating Variables:

- Storing numbers: age = 25, salary = 50000.50.
- Storing text: name = "Alice", city = "New York".
- Storing Boolean values: is_employed = True, has_license = False.
- Storing multiple values in a list: scores = [85, 90, 78, 92].

Good Practices:

- Use meaningful variable names (e.g., age, salary, name).
- Follow Python naming conventions (e.g., snake_case for variable names).
- Avoid using reserved keywords (e.g., for, while, if).

5. Data Types in Python

Python supports various data types to store different kinds of information. Below are some commonly used data types:

- **Integers (int):** Whole numbers, e.g., 7, -3.
- **Float (float):** Decimal numbers, e.g., 2.5, 0.01.
- **String (str):** Text data, e.g., "Hello world!", "Gideon".
- **Boolean (bool):** Represents True or False, e.g., True, False.

  Each data type serves a specific purpose and can be used in different scenarios.

Python Arithmetic Operators

Python provides several arithmetic operators to perform basic mathematical operations. Below is a list of the operators and examples for each:

1. **Addition (+)**
   - Adds two numbers.
   - **Example:**
   - result = 5 + 3 # Output: 8
2. **Subtraction (-)**
   - Subtracts the second number from the first.
   - **Example:**
   - result = 10 - 4 # Output: 6
3. **Multiplication (*)**
   - Multiplies two numbers.
   - **Example:**
   - result = 6 * 7 # Output: 42
4. **Division (/)**
   - Divides the first number by the second, returning a float.
   - **Example:**
   - result = 15 / 3 # Output: 5.0
5. **Modulus (%)**
   - Returns the remainder of a division.
   - **Example:**
   - result = 10 % 3 # Output: 1
6. **Exponentiation (**)**
   - Raises the first number to the power of the second.
   - **Example:**
   - result = 2 ** 3 # Output: 8
7. **Floor Division (//)**
   - Divides the first number by the second and truncates the result to an integer.
   - **Example:**
   - result = 15 // 4 # Output: 3

Summary Table

Operator	Description	Example	Output
+	Addition	5 + 3	8
-	Subtraction	10 - 4	6
*	Multiplication	6 * 7	42
/	Division	15 / 3	5.0
%	Modulus	10 % 3	1
**	Exponentiation	2 ** 3	8
//	Floor Division	15 // 4	3

These operators are foundational for performing mathematical operations in Python and are often used in data analysis and other computational tasks.

Python Conditional Statements

Conditional statements allow programs to make decisions based on certain conditions. Python provides three primary conditional statements: **if**, **elif**, and **else**.

1. if Statement

The **if** statement is used to test a condition. If the condition is True, the code block inside the **if** statement is executed.

Syntax:

```
if condition:
    # Code to execute if the condition is True
```

Example:

```
age = 18
if age >= 18:
    print("You are eligible to vote.")
```

2. else Statement

The **else** statement provides an alternative set of instructions if the condition in the **if** statement is False.

Syntax:

```
if condition:
    # Code to execute if the condition is True
else:
    # Code to execute if the condition is False
```

Example:

```
age = 16
if age >= 18:
    print("You are eligible to vote.")
else:
    print("You are not eligible to vote.")
```

3. elif Statement

The **elif** statement stands for "else if" and is used to check multiple conditions. If the **if** condition is False, Python checks the **elif** conditions in sequence.

Syntax:

```
if condition1:
    # Code to execute if condition1 is True
elif condition2:
    # Code to execute if condition2 is True
else:
    # Code to execute if all conditions are False
```

Example:

```
marks = 75
if marks >= 90:
   print("Grade: A")
elif marks >= 75:
   print("Grade: B")
elif marks >= 50:
   print("Grade: C")
else:
   print("Grade: F")
```

Key Points to Remember

- The **if** statement is evaluated first. If it is `True`, the **elif** and **else** statements are skipped.
- You can use multiple **elif** statements but only one **else** statement.
- Indentation is crucial in Python; all code blocks under **if**, **elif**, or **else** must be indented.