Additional Important Topics for Python Fundamentals for Data Science

There are a few additional fundamental concepts that are particularly important for Data Science and Machine Learning that we might want to learn.

These topics will help us better understand how Python is used in data science workflows.

1. List Comprehensions:

List Comprehensions: A concise way to create lists.

Useful for data preprocessing and transformation.

In [1]:
```
# Example: List Comprehension
# Create a list of squares

numbers = [1, 2, 3, 4, 5]
squares = [num ** 2 for num in numbers]
print("Squares:", squares)

# Filter even numbers
evens = [num for num in numbers if num % 2 == 0]
print("Even Numbers:", evens)
```

```
Squares: [1, 4, 9, 16, 25]
Even Numbers: [2, 4]
```

Real-life Use Case:

Transforming or filtering datasets (e.g., extracting specific columns or rows).

2. Error Handling (try, except):

Error Handling: Prevents programs from crashing due to errors.

Essential for handling missing or invalid data.

In [2]:
```
# Example: Error Handling
try:
    num = int(input("Enter a number: "))
    print("You entered:", num)
except ValueError:
    print("Invalid input! Please enter a valid number.")
```

```
Enter a number: rossi
Invalid input! Please enter a valid number.
```

Real-life Use Case:

Handling user input errors or missing data in datasets.

3. Working with Files:

File Handling: Reading from and writing to files.

Data scientists often work with CSV, JSON, or text files.

In [3]:
```
# Example: File Handling
# Writing to a file
with open("example.txt", "w") as file:
    file.write("Hello, World!")

# Reading from a file
with open("example.txt", "r") as file:
    content = file.read()
    print("File Content:", content)
```

```
File Content: Hello, World!
```

Real-life Use Case:

Loading datasets from files or saving results to files.

4. Lambda Functions:

Lambda Functions: Small, anonymous functions defined with the lambda keyword.

Useful for quick, one-time operations.

In [4]:
```
# Example: Lambda Function
# Double a number

double = lambda x: x * 2
print("Double of 5:", double(5))

# Use with map() to apply a function to a list
numbers = [1, 2, 3, 4, 5]
doubled_numbers = list(map(lambda x: x * 2, numbers))
print("Doubled Numbers:", doubled_numbers)
```

```
Double of 5: 10
Doubled Numbers: [2, 4, 6, 8, 10]
```

Real-life Use Case:

Applying quick transformations to data (e.g., scaling values).

*5. Working with Libraries:**

Introduce the concept of importing libraries and using them.

This will prepare students for working with libraries like Pandas and NumPy.

In [5]:
```
# Example: Importing Libraries
import math

# Calculate square root
print("Square root of 16:", math.sqrt(16))

# Generate random numbers
import random
print("Random number between 1 and 10:", random.randint(1, 10))
```

```
Square root of 16: 4.0
Random number between 1 and 10: 3
```

Real-life Use Case:

Using libraries for mathematical operations, data manipulation, and visualization.

6. Introduction to Data Structures for Data Science:

Data Structures: Lists, dictionaries, and sets are commonly used in data science.

Introduce nested lists and dictionaries for handling structured data.

In [6]:
```
# Example: Nested Lists (2D List)
matrix = [
    [1, 2, 3],
    [4, 5, 6],
    [7, 8, 9]
]
print("Element at row 2, column 3:", matrix[1][2])

# Example: Nested Dictionary
student = {
    "name": "Alice",
    "grades": {
        "math": 90,
        "science": 85,
        "history": 88
    }
}
print("Science grade:", student["grades"]["science"])
```

```
Element at row 2, column 3: 6
Science grade: 85
```

Real-life Use Case:

Representing tabular data or hierarchical data.

7. Introduction to Iterators and Generators:

Iterators: Objects that allow traversal through all elements of a collection.

Generators: Functions that yield items one at a time, saving memory.

In [7]:
```
# Example: Generator Function
def generate_numbers(n):
    for i in range(n):
        yield i  # Yields one value at a time

# Using the generator
for num in generate_numbers(5):
    print("Generated Number:", num)
```

```
Generated Number: 0
Generated Number: 1
Generated Number: 2
Generated Number: 3
Generated Number: 4
```

Real-life Use Case:

Processing large datasets without loading everything into memory.

8. Introduction to Object-Oriented Programming (OOP):

OOP Basics: Classes and objects.

Useful for organizing code and creating reusable components.

In [8]:
```
# Example: Class and Object
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age

    def greet(self):
        print(f"Hello, my name is {self.name} and I am {self.age} years old.")

# Creating an object
person = Person("Alice", 25)
person.greet()
```

```
Hello, my name is Alice and I am 25 years old.
```

Real-life Use Case:

Building custom data structures or models in machine learning.

9. Practice Exercises:

- Use list comprehensions to filter out negative numbers from a list.
- Write a function that reads a file and counts the number of lines.
- Create a generator function to yield Fibonacci numbers up to a given limit.
- Use a lambda function to sort a list of tuples by the second element.