Comprehensive Python Programming Example with Detailed Documentation

1. File Handling: Create a Directory and Write Data to a File

In this section, we create a directory if it doesn't already exist, write data to a text file, and ensure everything is properly saved. This demonstrates basic file I/O operations in Python.

Code Sample:

```
import os

def create_directory_and_file(directory: str, filename: str, data: str):
    if not os.path.exists(directory):
        os.makedirs(directory)
        print(f"Directory '{directory}' created.")

    filepath = os.path.join(directory, filename)
    with open(filepath, 'w') as file:
        file.write(data)
    print(f"File '{filename}' written in '{directory}'.")
```

Explanation:
1. Directory Creation: The os.makedirs() method ensures the directory is created if it doesn't already exist. The code uses os.path.exists() to check for the directory’s existence before creating it.
2. Writing to File: The open() function is used to write to a file. By using the with statement, the file is automatically closed after writing, ensuring no resources are left open.

Expected Output:
When the code is executed, it will:
- Create the directory if it doesn't exist.
- Write the provided data to the file and print confirmation messages.

Example Execution:

```
Directory 'example_dir' created.
File 'example_file.txt' written in 'example_dir.'
```

2. Random String Generation

This function generates a random string of letters, which could be useful for generating random identifiers or passwords.

Code Sample:

```
import random
import string

def generate_random_string(length: int = 10) -> str:
    letters = string.ascii_letters
    return ''.join(random.choice(letters) for i in range(length))
```

Explanation:
- String Generation: We use string.ascii_letters, which contains all uppercase and lowercase letters, and random.choice() to select a letter randomly. The string is constructed using list comprehension.

Expected Output:
For a default length of 10, the function will generate a random string like 'aZkLmDnfWg'. The output will vary each time the function is called.
Example Execution:

```
print(generate_random_string())  # Output: aZkLmDnfWg
```

3. Class Definition with Inheritance

This section demonstrates Object-Oriented Programming (OOP) by defining a basic Animal class and a subclass Dog that inherits from Animal. The speak() method is overridden in the Dog class.

Code Sample:

```
class Animal:
    def __init__(self, name: str, species: str):
        self.name = name
        self.species = species

    def speak(self) -> str:
        return f"{self.name} says hello!"

class Dog(Animal):
    def __init__(self, name: str, breed: str):
        super().__init__(name, "Dog")
        self.breed = breed

    def speak(self) -> str:
        return f"{self.name} barks loudly!"
```

Explanation:
1. Animal Class: The constructor (__init__) initializes the name and species attributes, and the speak() method returns a general greeting.
2. Dog Class: Inherits from Animal and calls the parent constructor using super() to initialize common attributes. It overrides the speak() method to provide dog-specific functionality.

Expected Output:
Example Execution:

```
dog = Dog('Buddy', 'Golden Retriever')
print(dog.speak())  # Output: Buddy barks loudly!
cat = Animal('Whiskers', 'Cat')
print(cat.speak())  # Output: Whiskers says hello!
```

4. JSON Handling: Read and Write JSON Data

In this section, we demonstrate how to handle JSON data in Python, by writing it to a file and reading it back into Python objects.

Code Sample:

```
import json

def json_example():
    data = {
        "name": "John Doe",
        "age": 30,
        "email": "johndoe@example.com",
        "is_active": True,
        "friends": ["Alice", "Bob"]
    }

    json_data = json.dumps(data, indent=4)
    print(f"JSON String:\n{json_data}")

    with open("data.json", "w") as json_file:
        json.dump(data, json_file, indent=4)

    with open("data.json", "r") as json_file:
        loaded_data = json.load(json_file)
        print(f"Loaded Data from JSON:\n{loaded_data}")
```

Explanation:
1. JSON Serialization: json.dumps() converts a Python dictionary to a formatted JSON string.
2. JSON Deserialization: The json.load() method is used to read the JSON data from the file and convert it back into a Python dictionary.

Expected Output:
Example Execution:

```
JSON String:
{
    "name": "John Doe",
    "age": 30,
    "email": "johndoe@example.com",
    "is_active": true,
    "friends": ["Alice", "Bob"]
}
Loaded Data from JSON:
{'name': 'John Doe', 'age': 30, 'email': 'johndoe@example.com', 'is_active': True, 'friends': ['Alice', 'Bob']}
```

5. Recursion Example: Factorial Function

This section demonstrates recursion by implementing a function to compute the factorial of a number.

Code Sample:

```
def factorial(n: int) -> int:
    if n == 0:
        return 1
    else:
        return n * factorial(n - 1)
```

Explanation:
1. Recursion: The function calls itself with a decremented value of n until the base case (n == 0) is reached. This is a classic example of recursion.

Expected Output:
Example Execution:

```
print(factorial(5))  # Output: 120
```

6. Error Handling

In this example, we demonstrate how to catch and handle errors using the try, except, and finally blocks.

Code Sample:

```
def exception_handling_example():
    try:
        result = 10 / 0  # This will raise a ZeroDivisionError
    except ZeroDivisionError as e:
        print(f"Error: {e}")
    finally:
        print("Cleaning up resources...")
```

Explanation:
1. Error Handling: The try block contains code that may raise an exception. The except block

catches the exception, and the finally block ensures that certain code runs regardless of the outcome.

Expected Output:
Example Execution:

```
Error: division by zero
Cleaning up resources...
```