**Day 1 – Inheritance and Private Variables**

*Inheritance in Python*

Definition: Mechanism allowing a class to inherit attributes and methods from another class.

Example:

```python
class Animal:
    def speak(self):
        print("Animal speaks")

class Dog(Animal):
    def bark(self):
        print("Dog barks")
```

- `Dog` inherits from `Animal`.
- Instance of `Dog` can use both `speak` and `bark` methods.

*Multiple Inheritance in Python*

Definition: A class can inherit from more than one class.

Example:

```python
class A:
    def method_a(self):
        print("Method A")

class B:
    def method_b(self):
        print("Method B")
```

```
class C(A, B):
    def method_c(self):
        print("Method C")
```

- `C` inherits from both `A` and `B`.
- Instance of `C` can use methods from `A`, `B`, and `C`.

*Private Variables in Python*

Definition: Variables with double underscores (`__`) are considered private.

Example:

```python
class MyClass:
    def __init__(self):
        self.__private_variable = 42

    def get_private_variable(self):
        return self.__private_variable
```

- `__private_variable` is private.
- Accessed using a method (`get_private_variable`).
- Direct access from outside the class raises `AttributeError`.

**Day 2 – Iterators, Generators…**

*Iterators in Python*

- Definition: Objects implementing `__iter__` and `__next__` methods for iteration.
- Example:

```python
class MyIterator:
    def __init__(self, data):
        self.data = data
        self.index = 0

    def __iter__(self):
        return self

    def __next__(self):
        if self.index < len(self.data):
            result = self.data[self.index]
            self.index += 1
            return result
        else:
            raise StopIteration
```

*Generators in Python*

- Definition: Functions using `yield` to lazily produce a sequence of values.
- Example:

```python
def my_generator():
    yield 1
    yield 2
    yield 3
```

*Generator Expressions in Python*

- Definition: Concise syntax for creating generators.
- Example:

  ```python
  my_generator = (x for x in range(3))
  ```

*Operating System Interface in Python*

- Module: `os`
- Example:

  ```python
  import os

  print(os.getcwd())
  print(os.listdir('/path/to/directory'))
  ```

*Command Line Arguments in Python*

- Module: `sys` or `argparse`
- Example (`sys`):

  ```python
  import sys

  arguments = sys.argv
  ```

*Error Output Redirection and Program Termination*

- Module: `sys`
- Example:

```python
import sys

try:
    Code that may raise an exception
except ValueError as e:
    print(f"Error: {e}", file=sys.stderr)
    sys.exit(1)
```

*String Pattern Matching in Python*

- Module: `re` (regular expressions)
- Example:
  ```python
  import re

  pattern = re.compile(r'\d+')
  result = pattern.findall('The price is $10 and the quantity is 5')
  ```

*Internet Access in Python*

- Modules: `urllib`, `requests`
- Example (`requests`):
  ```python
  import requests

  response = requests.get('https://www.example.com')
  print(response.text)
  ```

**Day 3 – Dates and Times, Data Compression, Output Formatting…**

*Dates and Times in Python*

- Modules: `datetime`, `time`
- Example:

```python
from datetime import datetime


current_time = datetime.now()
```

*Data Compression in Python*

- Modules: `zipfile`, `gzip`, `tarfile`
- Example (`gzip`):

```python
import gzip


with open('file.txt', 'rb') as f_in:
    with gzip.open('file.txt.gz', 'wb') as f_out:
        f_out.writelines(f_in)
```

*Performance Measurement in Python*

- Module: `timeit`
- Example:

```python
import timeit


code_to_measure = """
```

*Code to measure performance*

```
"""
duration = timeit.timeit(code_to_measure, number=1000)
```

*Quality Control in Python*

- Tools: `pylint`, `flake8`, `black`
- Example (`pylint`):

```bash
pylint your_file.py
```

*Output Formatting in Python*

- Methods: `format()`, f-strings (Python 3.6+)
- Example (`format()`):

```python
name = "John"
age = 25
print("Name: {}, Age: {}".format(name, age))
```

*Templating in Python*

- Modules: `string.Template`, `Jinja2`
- Example (`string.Template`):

```python
from string import Template

template = Template("Hello, $name!")
result = template.substitute(name="Alice")
```

**Day 4 – Logging, Managing packages with pip, and Floating point Arithmetic**

*Logging in Python*

- Module: `logging`
- Example:

  ```python
  import logging

  logging.basicConfig(level=logging.INFO)
  logging.info("This is an info message")
  logging.error("This is an error message")
  ```

*Virtual Environments in Python*

- Purpose: Isolate project dependencies to avoid conflicts.
- Benefits: Dependency management, version control.

*Creating Virtual Environments*

- Module: `venv` (Python 3.3+), `virtualenv` (third-party)
- Commands:
  - Using `venv`:

    ```bash
    python -m venv myenv
    ```

  - Using `virtualenv`:

    ```bash
    virtualenv myenv
    ```

*Managing Packages with `pip`*

- Purpose: Install, upgrade, and uninstall Python packages.
- Commands:

- Install a package:

  ```bash
  pip install package_name
  ```

- Upgrade a package:

  ```bash
  pip install --upgrade package_name
  ```

- Uninstall a package:

  ```bash
  pip uninstall package_name
  ```

*Floating Point Arithmetic in Python*

- Issue: Precision limitations due to the binary representation of floating-point numbers.
- Example:

  ```python
  result = 0.1 + 0.2
  print(result)  # Output: 0.30000000000000004
  ```