SUMMARY - DAY 6

NAME : TEJASWINI GOKANAKONDA

ROLL No. : DE142

DATE : 12 - 11 -2024

Mapping Function

- Apply a function to each item in an iterable (e.g., `map()`).
- **Example**:

```
result = list(map(lambda x: x * 2, [1, 2, 3]))  # Output:
[2, 4, 6]
```

String Functions

- Functions for string manipulation (e.g., `str.upper()`, `str.split()`).
- **Example**:

```
text = "hello world"
print(text.upper())  # Output: "HELLO WORLD"
```

Number Functions

- Functions for number operations (e.g., `abs()`, `round()`, `max()`).
- **Example**:

```
print(round(4.567, 2))  # Output: 4.57
```

Date and Time Functions

- Use `datetime` module to work with dates and times.

- **Example**:

```
from datetime import datetime
now = datetime.now()
print(now.strftime("%Y-%m-%d %H:%M:%S"))  # Current date a
nd time
```

Python Functions

- Define reusable code blocks using `def`.
- **Example**:

```
def greet(name):
    return f"Hello, {name}!"
```

Default Argument Values

- Set default values for function arguments.
- **Example**:

```
def greet(name="User"):
    print(f"Hello, {name}!")
greet()  # Output: Hello, User!
```

Keyword Arguments

- Use keyword arguments to call a function with named arguments.
- **Example**:

```
def display_info(name, age):
    print(f"Name: {name}, Age: {age}")
display_info(age=30, name="Alice")
```

Special Parameters

- Use `/` and `*` to define positional-only or keyword-only arguments.
- **Example**:

```
def func(a, b, /, c, *, d):
    print(a, b, c, d)
```

Arbitrary Argument Lists

- Use `*args` for any number of positional arguments, `**kwargs` for keyword arguments.
- **Example**:

```
def sum_all(*args):
    return sum(args)
print(sum_all(1, 2, 3))  # Output: 6
```

Read and Write CSV File

- Use `csv` or `pandas` for CSV operations.
- **Example** (using pandas):

```
import pandas as pd
df = pd.read_csv("input.csv")
df.to_csv("output.csv", index=False)
```

Lambda Expressions

- Anonymous functions using `lambda` keyword.
- **Example**:

```
square = lambda x: x * x
print(square(5))  # Output: 25
```

OOPS (Object-Oriented Programming)

- Organize code with classes and objects, focusing on encapsulation, inheritance, and polymorphism.
- **Example**:

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

Class and Object

- Classes define objects with attributes and methods.
- **Example**:

```
class Dog:
    def bark(self):
        print("Woof!")
my_dog = Dog()
my_dog.bark()
```

Access Specifiers

- Control attribute/method visibility: public (`x`), protected (`_x`), private (`__x`).
- **Example**:

```
class Test:
    def __init__(self):
        self._protected = "protected"
        self.__private = "private"
```

Constructor

- Initialize object attributes with `__init__`.
- **Example**:

```
class Car:
    def __init__(self, model):
        self.model = model
```

Inheritance

- Create a new class inheriting attributes/methods from another class.
- **Example**:

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

Polymorphism

- Different classes can use the same method names with unique implementations.
- **Example**:

```
class Cat:
    def sound(self):
        print("Meow")
class Dog:
    def sound(self):
        print("Bark")
```

Method Overriding

- Redefine a method in a subclass.
- **Example**:

```
class Animal:
    def sound(self):
        print("Generic sound")
class Dog(Animal):
    def sound(self):
        print("Woof!")
```

File Handling

- Use `open()` for reading/writing files.
- **Example**:

```
with open("file.txt", "w") as file:
    file.write("Hello, world!")
```

Exception Handling

- Use `try`, `except`, and `finally` to handle exceptions.
- **Example**:

```
try:
    x = 10 / 0
except ZeroDivisionError:
    print("Cannot divide by zero!")
```

Python Modules

- Organize reusable code into `.py` files.
- **Example**:

```
import math
print(math.sqrt(16))
```

User-defined Modules

- Create custom modules by saving functions in a `.py` file.
- **Example** (file: `mymodule.py`):

```
def greet(name):
    return f"Hello, {name}!"
# Usage:
# from mymodule import greet
```

Executing Modules as Scripts

- Use `if __name__ == "__main__":` to run code only when module is executed directly.
- **Example**:

```
if __name__ == "__main__":
    print("Running as script")
```

Standard Modules

- Built-in modules in Python (e.g., `os`, `sys`, `math`).
- **Example**:

```
import os
print(os.getcwd())  # Current working directory
```

Packages

- A collection of modules organized in directories with `__init__.py`.
- **Example**:

```
# package structure:
# mypackage/
```

```
# ├── __init__.py
# ├── module1.py
```

Importing * From a Package

- Import all modules/functions from a package.
- **Example**:

```
from mypackage import *
```

Intra-package References

- Use relative imports within a package.
- **Example**:

```
# Inside `subpackage/module.py`
from ..module import my_function
```

Packages in Multiple Directories

- Use multiple paths in `sys.path` for accessing packages in different directories.
- **Example**:

```
import sys
sys.path.append('/path/to/package')
import package
```