PW SKILLS

Python Modules, Packages, and File Handling

Lesson Plan

1. Introduction

In Python, modules, packages, and file handling are fundamental concepts that help in organizing code, reusing components, and managing data. This document provides an overview of these concepts, including practical examples to illustrate their usage.

2. Python Modules

What is a Module?
A module in Python is a file containing Python code. Modules can define functions, classes, and variables, and can include runnable code. The primary purpose of a module is to allow code reuse and organization.

Creating and Using Modules
To create a module, simply write Python code and save it with a .py extension. For example, you can create a file named mymodule.py with the following content:

Example: Simple Module
mymodule.py

```python
# mymodule.py

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

def add(a, b):
    return a + b
```

To use this module in another script, you can import it using the import statement:
main.py

```python
# main.py

import mymodule

print(mymodule.greet("Alice"))
print(mymodule.add(5, 3))
```

Output:

```
Hello, Alice!
8
```

3. Python Packages

What is a Package?
A package in Python is a collection of modules organized in directories. Each directory contains a special file named __init__.py, which makes Python treat the directory as a package. Packages help in organizing large codebases into manageable modules.

Creating and Using Packages
To create a package, follow these steps:

1. Create a directory for the package.
2. Inside this directory, create the __init__.py file and other module files.

Example: Simple Package
Directory Structure:

```
mypackage/
    __init__.py
    module1.py
    module2.py
```

mypackage/module1.py

```
# module1.py

def multiply(x, y):
    return x * y
```

mypackage/module2.py

```
# module2.py

def subtract(x, y):
    return x - y
```

mypackage/init.py

```
# __init__.py

from .module1 import multiply
from .module2 import subtract
```

To use this package in a script:
main.py

```
# main.py

from mypackage import multiply, subtract

print(multiply(4, 5))
print(subtract(10, 3))
```

Output:
20
7

4. File Handling in Python

Opening and Closing Files
In Python, files are handled using built-in functions. The open() function is used to open a file, and close() is used to close it.

Example: File Handling Operations
Writing to a File

```
# write_file.py

with open('example.txt', 'w') as file:
    file.write("Hello, World!\n")
    file.write("Welcome to Python file handling.")
```

Reading from a File

```
# read_file.py

with open('example.txt', 'r') as file:
    content = file.read()
    print(content)
```

Output:

```
Hello, World!
Welcome to Python file handling.
```

Reading from and Writing to Files
Python provides various methods for reading and writing files, including read(), readline(), readlines(), and write().

Example: File Handling Operations
Appending to a File

```
# append_file.py

with open('example.txt', 'a') as file:
file.write("\nAppended text.")
```

```
# read_lines.py

with open('example.txt', 'r') as file:
    for line in file:
        print(line.strip())
```

Output:

```
Hello, World!
Welcome to Python file handling.
Appended text.
```

Handling File Exceptions
File operations may fail for various reasons (e.g., file not found). Use try...except blocks to handle exceptions gracefully.

Example: File Handling with Exception

```
# safe_file_operations.py

try:
    with open('nonexistent_file.txt', 'r') as file:
        content = file.read()
except FileNotFoundError:
    print("The file does not exist.")
```

Output:
The file does not exist.

5. Conclusion

Understanding Python modules, packages, and file handling is essential for writing efficient and organized code. Modules and packages help in managing and reusing code, while file handling allows you to interact with the file system for reading and writing data. Mastery of these concepts enhances your ability to develop robust and maintainable Python applications.