📝 Take-Away Assignment: Modules & Error Handling in Python

Modules

1. What is a module in Python, and why is it important in software development?

A **module** in Python is simply a file that contains Python code such as functions, classes, or variables. It allows you to logically organize code into reusable components.

🔹 **Importance**:

- Encourages **reusability** (use the same code in multiple programs).
- Makes code **modular and maintainable** (easy to debug).
- Promotes **collaboration** (different developers can work on different modules).

✅ **Example**:
Suppose we create a file `math_utils.py`:

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

def multiply(a, b):
    return a * b
```

We can import and use it in another program:

```
import math_utils
print(math_utils.add(3, 4))      # Output: 7
print(math_utils.multiply(2, 5))  # Output: 10
```

---

2. Built-in module vs User-defined module

- **Built-in module**: Comes pre-installed with Python.
- **User-defined module**: Created by the programmer.

✅ **Examples**:

- Built-in modules:
  - `math` → mathematical operations (`math.sqrt(16)` → 4.0)
  - `random` → random numbers (`random.randint(1, 10)`)
- User-defined modules:
  - `student.py` with student functions.

- `bank.py` with deposit/withdraw logic.

---

3. Importing modules (different ways)

1. **`import module`** → imports the whole module.

```
import math
print(math.sqrt(25))   # 5.0
```

2. **`from module import function`** → imports a specific function/class.

```
from math import sqrt
print(sqrt(25))   # 5.0
```

3. **`import module as alias`** → shorter name for convenience.

```
import math as m
print(m.sqrt(25))   # 5.0
```

---

4. Advantages of using modules in large programs

- **Reusability** – no need to rewrite code.
- **Maintainability** – easier to debug and update.
- **Collaboration** – team members can work on separate modules.
- **Organization** – keeps code structured and clean.

---

5. How does Python locate a module? (PYTHONPATH)

When you import a module, Python searches for it in the following order:

1. The **current working directory**.
2. Built-in Python libraries.
3. Directories listed in the **PYTHONPATH environment variable**.
4. Installed packages (`site-packages`).

✅ Example: If you do:

```
import mymodule
```

Python will first look in the same folder, then in system paths.

Error Handling

6. What is an exception in Python? How does it differ from a syntax error?

- **Exception**: An error that occurs **while the program is running** (runtime error).
- **Syntax error**: Mistake in code structure that prevents execution (compile-time error).

✅ Example:

```
# Syntax Error:
print("Hello"   # Missing parenthesis

# Exception:
x = 10 / 0      # ZeroDivisionError
```

---

7. `try, except, else, finally`

- **try** → block of code to test.
- **except** → block of code to handle the error.
- **else** → runs if no exception occurs.
- **finally** → always runs (cleanup).

✅ Example:

```
try:
    num = int(input("Enter a number: "))
    print(10 / num)
except ZeroDivisionError:
    print("Cannot divide by zero.")
except ValueError:
    print("Invalid input, please enter a number.")
else:
    print("Division successful.")
finally:
    print("End of program.")
```

---

8. Built-in vs User-defined Exceptions

- **Built-in Exceptions**: Already available in Python.
  - `ZeroDivisionError`: dividing by zero.
  - `ValueError`: wrong data type.
- **User-defined Exceptions**: Programmer creates custom exceptions.

✅ Example:

```
class NegativeNumberError(Exception):
    pass

num = -5
if num < 0:
    raise NegativeNumberError("Negative numbers not allowed")
```

---

9. Why is error handling important?

It ensures that programs **don’t crash unexpectedly** and can **gracefully handle issues**.

✅ **Examples**:

1. **Banking app**: Preventing crashes when a user tries to withdraw more money than available.
2. **E-commerce site**: Handling failed payments without breaking the whole checkout process.

---

10. Exception Chaining in Python

Exception chaining occurs when handling one exception leads to another exception. Python allows linking them together using `raise ... from ...`.

✅ Example:

```
try:
    int("abc")    # Raises ValueError
except ValueError as e:
    raise RuntimeError("Conversion failed") from e
```

This shows both the **original error (ValueError)** and the **new error (RuntimeError)**.

Applied / Reflection

11. Banking System Example

If I were creating a **banking system**:

- I’d create **modules** like:
  - `accounts.py` (manage balances)
  - `transactions.py` (deposits, withdrawals)
  - `security.py` (authentication).
- I’d use **error handling** for:
  - Invalid login attempts.
  - Transactions with insufficient funds.
  - Network errors when connecting to online banking services.

---

12. Consequences of not using error handling in critical systems

- **Healthcare**: If a hospital system crashes due to an error, it may delay treatment or lose patient data.
- **Aviation**: Flight control software crashing mid-flight can cause disasters.
- **Finance**: Unhandled errors could result in wrong balances or loss of millions.

👉 Lack of error handling can lead to **loss of data, financial damage, security breaches, or even human lives**.