🐍 Python Day 3 Notes - Exception Handling, Modules & File Handling

Theme: Making Python Programs Safer and More Powerful

📚 Table of Contents

---

1. Exception Handling

🔍 What is Exception Handling?

Sometimes your program crashes due to unexpected errors. Exception handling helps your program continue running even when errors occur.

Common Errors:

- **ZeroDivisionError:** Dividing by zero
- **TypeError:** Wrong data type
- **NameError:** Variable not defined
- **ValueError:** Wrong value

Basic try-except:

```python
try:
    # Risky code that might fail
    print(1 / 0)
except ZeroDivisionError:
    print("Cannot divide by zero!")
```

Output: `Cannot divide by zero!`

Multiple Exception Handling:

```python
age = '23'
try:
    print(vari)  # Variable not defined
    if age >= 18:  # String comparison error
        print('Can vote')
except TypeError:
    print('Wrong data type!')
except NameError:
    print('Variable not found!')
except:
    print('Something else went wrong!')
```

🔑 Key Point:

Always put specific exceptions first, then general ones.

---

2. Using Finally Block

🔍 What is Finally?

Code in `finally` block runs no matter what - whether error occurs or not.

Example:

```python
try:
    print(5/0)
except ZeroDivisionError:
    print("Cannot divide by zero!")
finally:
    print("This always runs!")
```

Output:

```
Cannot divide by zero!
This always runs!
```

💡 Real-life Use:

- Closing files
- Cleaning up resources

- Logging information

---

3. Practical Timer Example

Simple Timer with Exception Handling:

```python
def show_time():
    try:
        while True:
            import time
            print(time.asctime())
            time.sleep(1)
    except:
        print("Timer stopped!")

# show_time()  # Press Ctrl+C to stop
```

🔑 Key Point:

Use `try-except` to handle user interruptions gracefully.

---

4. Modules in Python

🔍 What are Modules?

Modules are Python files containing reusable functions and code.

Types of Modules:

Type	Examples	Purpose
Built-in	`random`, `math`, `time`	Come with Python
External	`qrcode`, `pygame`	Install using pip
User-Defined	Your own `.py` files	Custom functions

Using Built-in Modules:

```python
import random
print(random.randint(1, 100))  # Random number 1-100

import math
print(math.sqrt(16))  # Square root of 16
```

Creating Your Own Module:

File: `mymodule.py`

```python
def give_fibo(n):
    fibo = [0, 1]
    for i in range(n-2):
        fibo.append(fibo[-1] + fibo[-2])
    return fibo

def find_min(*numbers):
    return min(numbers)

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

Using Your Module:

```python
import mymodule as m

print(m.give_fibo(5))       # [0, 1, 1, 2, 3]
print(m.find_min(5, 2, 8))  # 2
print(m.greet("Rahul"))     # Hello, Rahul!
```

🔑 Key Point:

Modules help organize code and make it reusable.

---

5. File Handling

🔍 What is File Handling?

Reading from and writing to files on your computer.

File Types:

- **Text Files:** `.txt`, `.py`, `.csv`
- **Binary Files:** `.jpg`, `.mp3`, `.exe`

Basic File Operations:

1. `open()` - Open a file
2. `read()` - Read content

3. `write()` - Write content
4. `close()` - Close file

Reading a File:

```python
# Method 1: Read entire file
f = open('myfile.txt', 'r')
content = f.read()
print(content)
f.close()

# Method 2: Read first 50 characters
f = open('myfile.txt', 'r')
print(f.read(50))
f.close()
```

Reading Line by Line:

```python
f = open('myfile.txt', 'r')
for line in f:
    print(line.strip())  # strip() removes extra spaces
f.close()
```

Reading Specific Lines:

```python
f = open('myfile.txt', 'r')
lines = f.readlines()
print(lines[0])  # First line
print(lines[2])  # Third line
f.close()
```

Writing to a File:

```python
f = open('output.txt', 'w')
f.write("Hello, World!")
f.write("\nThis is line 2")
f.close()
```

🔑 Key Point:

Always close files after use with `f.close()`!

---

6. Practical Examples

Sound Box Function:

python

```python
def sound_box(amount, platform='Paytm'):
    try:
        from gtts import gTTS
        import pygame

        text = f"{platform} par {amount} rupay prapt hue."
        audio = gTTS(text, lang='hi')
        audio.save('payment.mp3')

        pygame.init()
        pygame.mixer.music.load('payment.mp3')
        pygame.mixer.music.play()

    except Exception as e:
        print(f"Error: {e}")

# sound_box(500, "Google Pay")
```

Colorful Turtle Pattern:

python

```python
import turtle
import random

turtle.speed(0)
turtle.bgcolor('black')
colors = ['red', 'yellow', 'green', 'blue', 'orange']

for i in range(120):
    turtle.color(random.choice(colors))
    turtle.circle(100)
    turtle.left(3)
```

Safe Division Calculator:

python

```
def safe_divide():
    try:
        a = float(input("Enter first number: "))
        b = float(input("Enter second number: "))
        result = a / b
        print(f"Result: {result}")
    except ZeroDivisionError:
        print("Cannot divide by zero!")
    except ValueError:
        print("Please enter valid numbers!")
    except Exception as e:
        print(f"Error: {e}")

safe_divide()
```

---

7. Mini Assignments

📝 Practice Problems:

1. **Create Your Own Module**
   - Make a file called `mathhelper.py`
   - Add functions for: prime check, factorial, fibonacci
   - Import and use it in another file
2. **Safe Calculator**
   - Create a calculator that handles division by zero
   - Handle invalid input gracefully
   - Use try-except blocks
3. **File Reader**
   - Write a program to read first 5 lines of any `.py` file
   - Handle file not found errors
   - Display line numbers
4. **Simple Quiz Game**
   - Store questions in a text file
   - Read questions and display to user
   - Keep score in another file
5. **Error Logger**
   - Create a program that logs errors to a file
   - Include timestamp and error type

- Use finally block for cleanup

---

8. Key Takeaways

🎯 Important Points:

1. **Exception Handling Protects Your Code**
   - Use `try-except` for risky operations
   - Handle specific errors first
   - Always have a general `except` as backup
2. **Modules Make Code Reusable**
   - Built-in modules add functionality
   - Create your own modules for custom functions
   - Use meaningful names for modules
3. **File Handling Enables Data Storage**
   - Read files to load data
   - Write files to save results
   - Always close files after use
4. **Error Prevention is Better Than Handling**
   - Check inputs before processing
   - Validate data types
   - Test with different scenarios

🔧 Best Practices:

- Always handle exceptions in user input
- Use meaningful error messages
- Close files properly
- Test your code with wrong inputs
- Keep modules small and focused

💡 Real-World Applications:

- **Banking Apps:** Handle invalid transactions
- **Games:** Save/load game progress
- **Websites:** Log errors for debugging
- **Data Analysis:** Read CSV files safely

📖 Study Plan

Today's Focus:

- Practice exception handling with real examples
- Create and use your own modules
- Work with file reading and writing

Tomorrow's Preview:

- Object-Oriented Programming (OOP)
- Classes and Objects
- Advanced data structures

🎯 Quick Review Questions:

1. What happens if you don't handle exceptions?
2. When would you use a `finally` block?
3. How do you create your own module?
4. What's the difference between `read()` and `readlines()`?
5. Why should you always close files?

Remember: Good programmers expect things to go wrong and prepare for it! 🛡️

Keep practicing and stay curious! 🚀