Module 5 & 6

Created by	Karan Maurya
Status	Not started

Topics

Python Cheatsheet: Module 5 + Module 6

(Advance + Intermediate Friendly — Structured and Practical)

Module 5: Exception Handling

5.1 What are Exceptions?

- **Exceptions** are runtime errors that can disrupt normal program flow.

- Python encourages **handling errors gracefully.**

Examples of common exceptions:

- `ZeroDivisionError`
- `TypeError`
- `ValueError`
- `FileNotFoundError`
- `KeyError`
- `IndexError`

5.2 Try-Except Block

Basic Syntax:

```
try:
    risky_code()
except SomeException:
    handle_error()
```

Example:

```
try:
    result = 10 / 0
except ZeroDivisionError:
    print("You can't divide by zero!")
```

5.3 Catching Multiple Exceptions

```
try:
    num = int(input())
    result = 10 / num
except (ValueError, ZeroDivisionError) as e:
    print(f"Error occurred: {e}")
```

5.4 Else and Finally Blocks

Block	Purpose
else	Executes **if no exception occurs**
finally	Executes **always**, used for cleanup (e.g., closing files)

```
try:
    x = int(input())
except ValueError:
    print("Invalid input!")
else:
    print("Input is valid.")
finally:
    print("Program execution completed.")
```

5.5 Raising Exceptions

Manually trigger exceptions:

```
def set_age(age):
    if age <= 0:
        raise ValueError("Age must be positive!")
    return age
```

5.6 Creating Custom Exceptions

Define your own exception class:

```
class NegativeAgeError(Exception):
    pass

def check_age(age):
    if age < 0:
        raise NegativeAgeError("Age can't be negative!")
```

> Advanced Tip: Always inherit custom exceptions from Exception, not from BaseException.

Module 6: File Handling

6.1 File Operations

Mode	Purpose
`'r'`	Read (default), error if file missing
`'w'`	Write (overwrite), create if missing
`'a'`	Append to file
`'x'`	Exclusive creation, error if file exists
`'b'`	Binary mode
`'t'`	Text mode (default)

6.2 Opening and Reading Files

```
f = open('data.txt', 'r')
content = f.read()
print(content)
f.close()
```

Read Line-by-Line:

```
f = open('data.txt', 'r')
for line in f:
    print(line.strip())
f.close()
```

> Always close the file or use a context manager!

6.3 Writing to Files

```
with open('output.txt', 'w') as f:
    f.write("Hello, World!\n")
```

Append Mode:

```
with open('output.txt', 'a') as f:
    f.write("Appended Line\n")
```

6.4 Working with Binary Files

```
with open('image.png', 'rb') as img_file:
    data = img_file.read()

with open('copy.png', 'wb') as copy_file:
    copy_file.write(data)
```

6.5 Useful File Methods

Method	Purpose
`read(size)`	Read `size` bytes
`readline()`	Read one line
`readlines()`	List of lines
`write(str)`	Write a string
`writelines(list)`	Write a list of strings
`seek(offset)`	Move cursor to position
`tell()`	Get current cursor position
`flush()`	Force writing to file

6.6 Exception Handling with Files

Robust File Handling:

```
try:
    with open('data.txt', 'r') as file:
        print(file.read())
except FileNotFoundError:
    print("File does not exist.")
except IOError:
    print("Input/Output error.")
```

Summary for Module 5 + 6

Topic	Focus	Special Notes
Exception Handling	Manage unexpected errors	Use `try-except-finally`, custom exceptions
File Handling	Read, write, manage files	Always prefer `with open()` for safety

---

Bonus: Practical Expert-Level Insights

- Always **log exceptions** instead of printing them during production development.
- Always **use context managers** for file operations (`with open()`).
- Always **validate user inputs** before file operations to avoid unnecessary exceptions.

This concludes Module 5 + 6 cheatsheet!

(Crisp, detailed, structured — ready for deep learning and quick revision.)

Would you like me to immediately continue and create the cheatsheet for **Module 7 + 8** too in the same professional style?

Shall I proceed?

(We're almost building an entire *high-quality Python Book Cheatsheet* here!)