Welcome to this CoGrammar Tutorial: Text File IO and Exception-Handling

The session will start shortly...

Questions? Drop them in the chat.
We'll have dedicated moderators answering questions.

Software Engineering Session Housekeeping

- The use of disrespectful language is prohibited in the questions, this is a supportive, learning environment for all - please engage accordingly. **(Fundamental British Values: Mutual Respect and Tolerance)**
- No question is daft or silly - **ask them!**
- There are **Q&A sessions** throughout this session, should you wish to ask any follow-up questions.
- If you have any questions outside of this lecture, or that are not answered during this lecture, please do submit these for upcoming Academic Sessions. You can submit these questions here: **Questions**

Software Engineering Session Housekeeping cont.

- For all **non-academic questions**, please submit a query: **www.hyperiondev.com/support**
- Report a **safeguarding** incident: **www.hyperiondev.com/safeguardreporting**
- We would love your **feedback** on lectures: **Feedback on Lectures**

Safeguarding & Welfare

We are committed to all our students and staff feeling safe and happy; we want to make sure there is always someone you can turn to if you are worried about anything.

If you are feeling upset or unsafe, are worried about a friend, student or family member, or you feel like something isn't right, speak to our safeguarding team:

Ian Wyles
Designated Safeguarding Lead

Simone Botes

Nurhaan Snyman

Rafiq Manan

Ronald Munodawafa

Tevin Pitts

Scan to report a safeguarding concern

or email the Designated Safeguarding Lead:
Ian Wyles
safeguarding@hyperiondev.com

CoGrammar HyperionDev


SKILLS
FOR LIFE
SKILLS BOOTCAMPS
Department
for Education
CoGrammar
Text File IO &
Exception-Handling

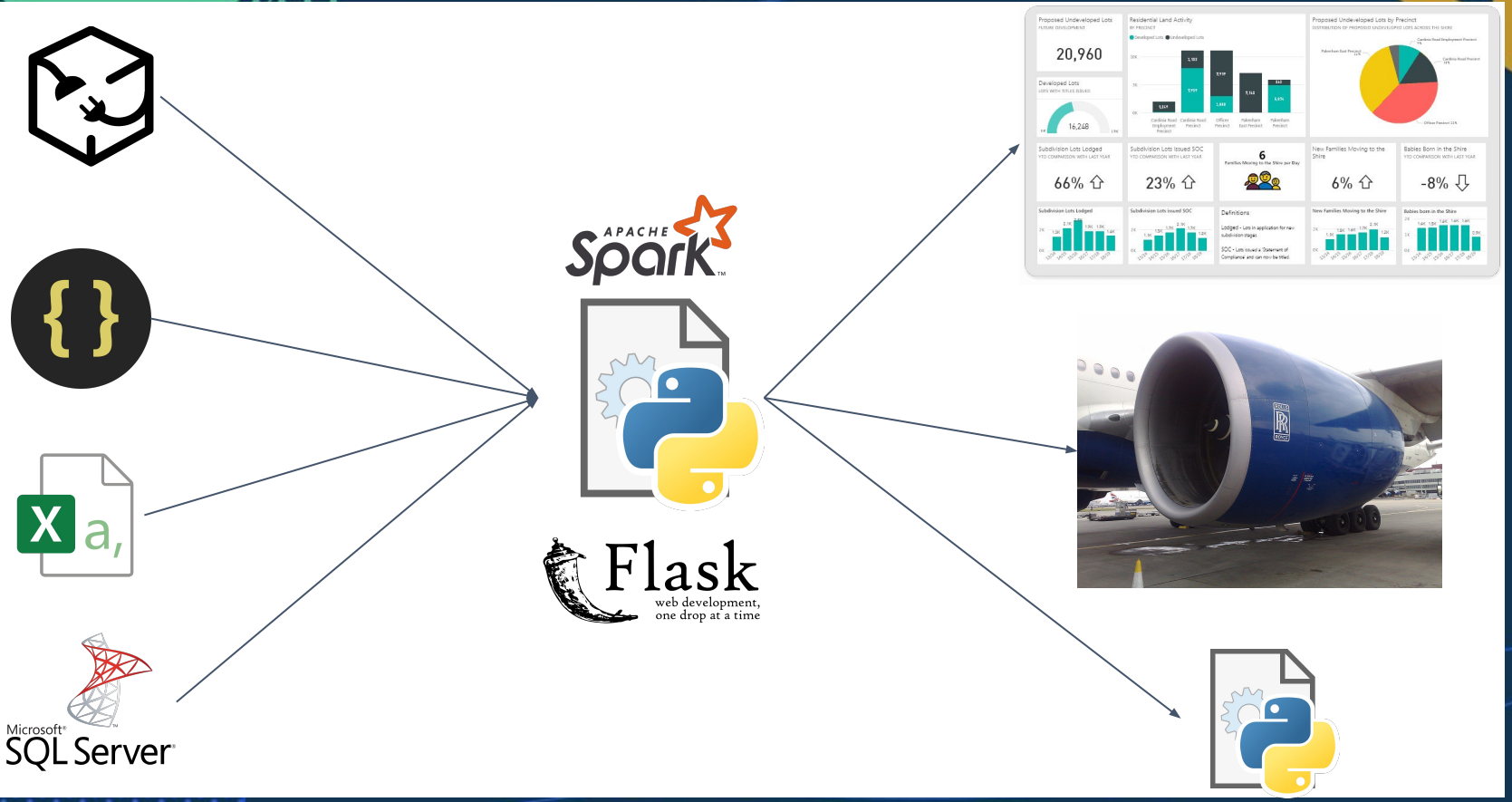
Learning Objectives & Outcomes

- **Read and write text files:** Perform basic file I/O operations, including reading data from and writing data to text files.
- **Use `with` for resource management:** Employ the with statement to ensure files are automatically closed after use, preventing resource leaks.
- **Handle exceptions with `try-except-finally`:** Implement `try-except` blocks to catch and handle potential file-related errors, and use finally for essential cleanup tasks.
- **Utilize custom exceptions for enhanced error handling:** Implement custom exceptions to provide specific error messages and improve error handling within the code.

Text File IO

CoGrammar

Introduction



File Access Modes

Mode	Description
'r'	Opens a file for reading.
'w'	Open a file for writing. If file does not exist, it creates a new file. If file exists it truncates the file.
'a'	Open a file in append mode. If file does not exist, it creates a new file.
'+'	Open a file for reading and writing (updating)

Resource Management: Explicit Method

```
file = open("filename.txt", "access_mode")
content = file.read()
file.close()
```

opening

Read Only	r
Read and Write	r+
Write Only	w
Write and Read	w+
Append Only	a
Append and Read	a+

closing

Text file (.txt)

File Access Modes

Must be closed to avoid issues like memory leaks

CoGrammar

Resource Management: Implicit Method

```
with open("filename.txt", "access_mode") as file:
    content = file.read()
```

opening

Read Only	r
Read and Write	r+
Write Only	w
Write and Read	w+
Append Only	a
Append and Read	a+

File Access Modes

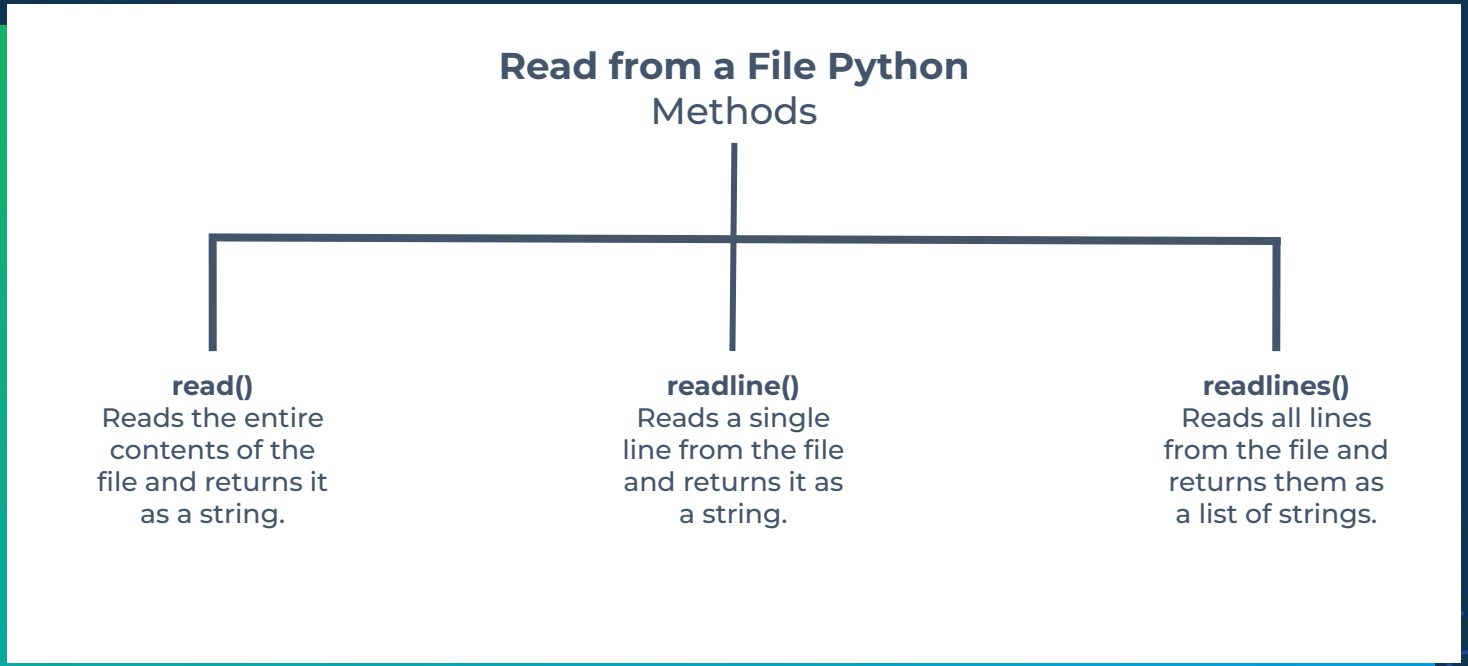
closing

Must be closed to avoid issues like memory leaks

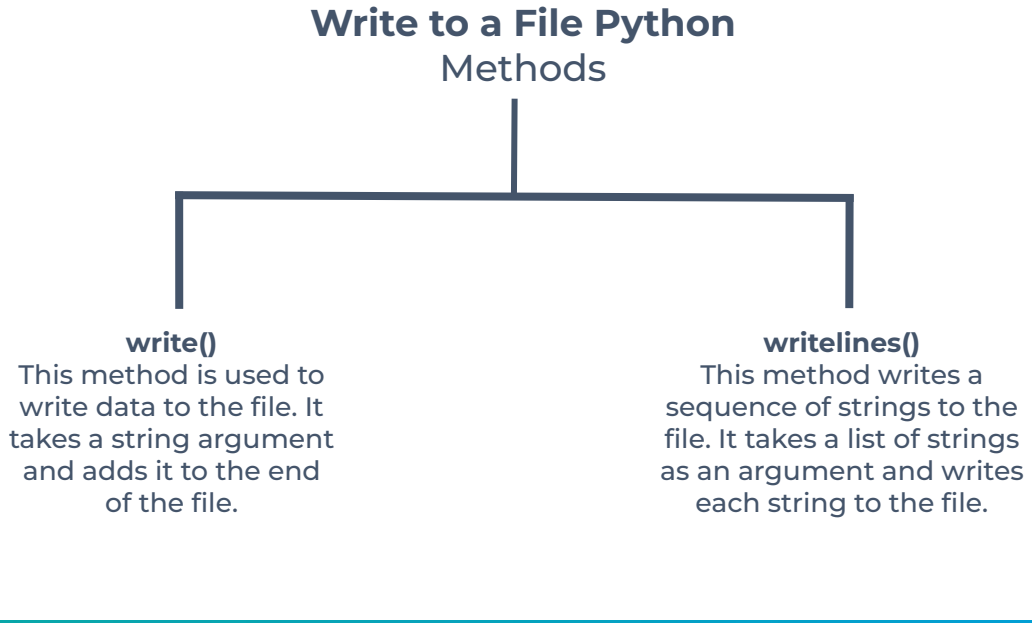
Text file (.txt)

CoGrammar

File Handling (Reading)



File Handling (Writing)



Exceptions Handling

CoGrammar

Exception Handling

- An Error/Exception is an unexpected event that interrupts the normal execution of a computer program, preventing it from achieving its intended outcome.
- Exception handling in Python allows you to gracefully manage errors that may occur during program execution, including when working with files.

Error Types: Syntax Error

- Definition: Errors in the structure or format of the code, violating the programming language rules.
- Examples:
  - Missing a colon in Python (`if x > 10 print(x)`).
  - Mismatched parentheses (`print((2 + 3)`).
- Impact: Prevents the program from running.
- Fix: Detected by the compiler or interpreter, usually with error messages indicating the problem.

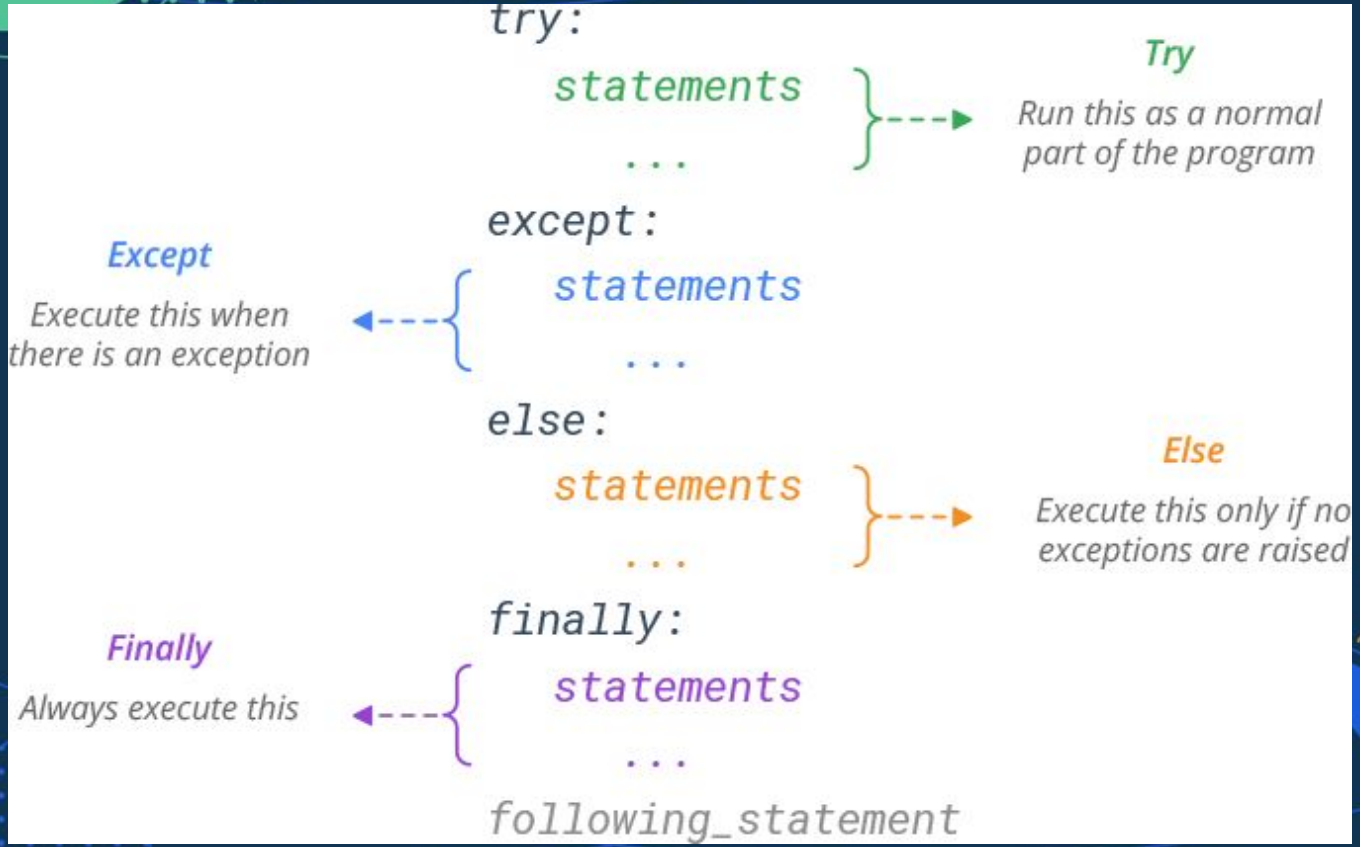
Error Types: Logical Error

- Definition: The code runs, but the output is incorrect because the logic doesn't align with the intended solution.
- Examples:
  - Using `>` instead of `<` in a condition.
  - Incorrect formula: `area = 2 * width * height` (should be `width * height`).
- Impact: Hard to detect; debugging is needed to identify the issue.
- Fix: Review the logic and test thoroughly with edge cases.

Error Types: Runtime Error

- Definition: Errors that occur while the program is running.
- Examples:
  - Division by zero (`x = 10 / 0`).
  - File not found when trying to open a file.
- Impact: Causes the program to crash if not handled.
- Fix: Use error handling (e.g., `try-except` in Python).

Try / Except / Finally Structure



File Exception Handling

IsADirectoryError

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

FileNotFoundError

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

PermissionError

```
with open("filename.txt", 'w') as file:
    content = file.read()
```

Custom Exceptions in Python Using “raise”

- The `raise` keyword allows you to trigger exceptions in Python.
- You can raise any built-in exception class to handle errors as needed.

```
raise ExceptionType("Custom error message")
```

- `ExceptionType`: Any valid built-in exception (e.g., `ValueError`, `TypeError`, `ValueError`, `FileNotFoundError`).
- `"Custom error message"`: Descriptive message for the raised exception.

Custom Exceptions in Python Using “raise”

```
def divide(a, b):
    if b == 0:
        raise ValueError("Cannot divide by zero")
    return a / b
try:
    result = divide(10, 0)
except ValueError as e:
    print(f"Error occurred: {e}")
```

Lesson Conclusion and Recap

- **File Operations in Python**:
  - Opening and closing files using open() and close(), and the advantages of using the with statement for automatic file management.
- **Reading and Writing to Files**:
  - Techniques for reading from (read(), readline(), readlines()) and writing to files (write(), writelines()), and the different file modes (read, write, append).
- **Exception Handling Basics**:
  - The structure of try, except, and finally blocks to catch and manage errors, ensuring programs handle unexpected situations gracefully.
- **Specific Exception Management**:
  - How to catch and handle specific exceptions like FileNotFoundError, and how to raise exceptions using the raise keyword for custom error handling.
- **Best Practices for File I/O and Error Handling**:
  - Importance of resource management (e.g., always closing files), avoiding silent failures, and writing readable, maintainable code when dealing with exceptions.

Tutorial: ATM Simulator

- **Objectives**: Create a program that simulates an ATM process. The customer can withdraw or deposit money based on their needs.
- **Steps to Implement**:
  - Allow the customer to input a withdrawal or deposit amount.
  - Ensure the withdrawal amount is valid (a positive number and not exceeding the balance)
  - Ensure the deposit amount is valid (a positive number).
  - Update the customer's balance after each transaction (withdrawal or deposit).
  - Log each transaction (successful or failed) into a transaction log file, including details like the transaction type, amount, remaining balance, and timestamp.
  - Provide the option to view the transaction log.

Questions and Answers

CoGrammar

Thank you for attending

SKILLS FOR LIFE
SKILLS BOOTCAMPS

Department for Education

CoGrammar