Welcome to this CoGrammar lecture: 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
Exception-Handling

Poll

How confident are you about the course so far?

https://forms.gle/C7CgzDJyYXa2DVtX8

Poll

What does the following code snippet do?

```
with open('log.txt', 'a') as file:
    file.write('New entry added.\n')
```

1. Appends the text `New entry added.` to the file without overwriting its content.
2. Overwrites the file content with `New entry added.`
3. Raises an error because `a` mode is not suitable for writing.

Poll

Which of the following is the best way to handle exceptions while opening a file?

b

```
file = open('data.txt', 'r')
print('File not found!')
```

c

```
if open('data.txt', 'r'):
    print('File exists!')
```

a

```
try:
    file = open('data.txt', 'r')
except IOError:
    print('File not found!')
```

CoGrammar

Learning Objectives & Outcomes

- Describe the purpose of exception handling and how it improves program robustness.
- Write try-except blocks to catch and handle exceptions.
- Differentiate between various built-in exceptions and handle them appropriately.
- Implement finally blocks to manage clean-up tasks such as closing files or releasing resources.
- Discuss the need for custom exceptions to handle specific error conditions in programs.
- Use custom exceptions to provide more informative error messages and improve error handling in code.

Dealing With Errors

CoGrammar

We all make mistakes :-)

- No programmer is perfect, and we're going to make a lot of mistakes on our journey – and that is perfectly okay!
- What separates the good programmers from the rest is the ability to find and debug errors that they encounter.

Defensive Programming

- Programmers anticipate errors:
  - User errors
  - Environment errors
  - Logical errors
- Code is written to ensure that errors don't crash the code base.
- Two ways - if statements and try-except blocks.

Error Types: Syntax Errors

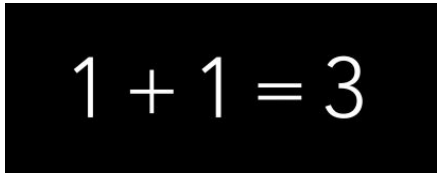
- Syntax errors are some of the easiest errors to fix... usually.
- Mainly caused by typos in code or Python specific keywords that were misspelled or rules that were not followed.
- When incorrect syntax is detected, Python will stop running and display an error message.

Syntax Error Example

```
print("Who let the dogs out ?"
"(" was not closed Pylance
SyntaxError: '(' was never closed Flake8(E999)
```

Error Types: Logical Errors

- Logical errors occur when your program is running, but the output you are receiving is not what you are expecting.
- The code could be typed incorrectly, or perhaps an important line has been omitted, or the instructions given to the program have been coded in the wrong order.



Error Types: Runtime Errors

- Runtime errors occur during the execution of a program, and they typically result from issues that manifest when the program is running rather than during the compilation or interpretation phase.
- Runtime errors are often detected when the program is running and can lead to the termination of the program if not handled properly.

```
    print(100/0)
ZeroDivisionError: division by zero
```

Exception Handling

CoGrammar

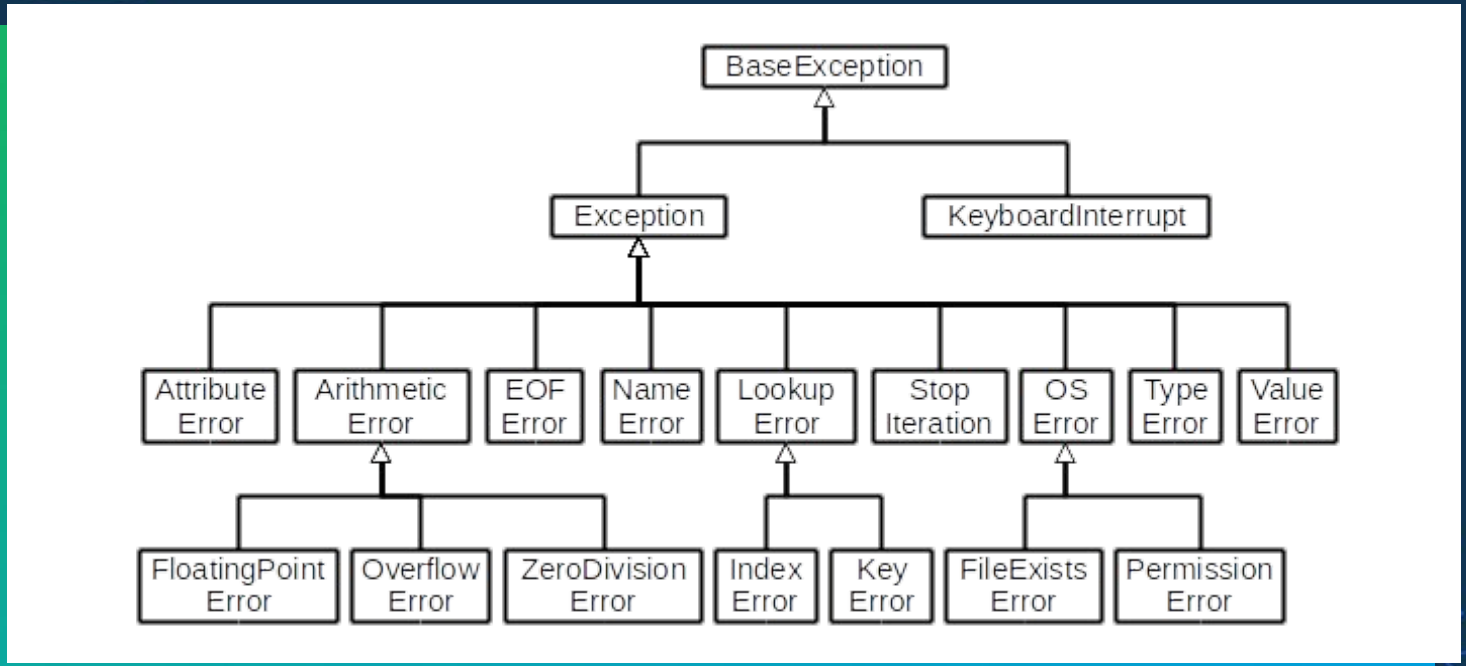
What are Exceptions?

- An exception is an event that occurs during the execution of a program, disrupting the normal flow of its initial instructions.

Why Handle Exceptions?

- Avoid program crashes: Unhandled exceptions will stop the program's execution and display an error.
- Improve user experience: Handling exceptions gracefully ensures your program runs smoothly, even when errors occur.

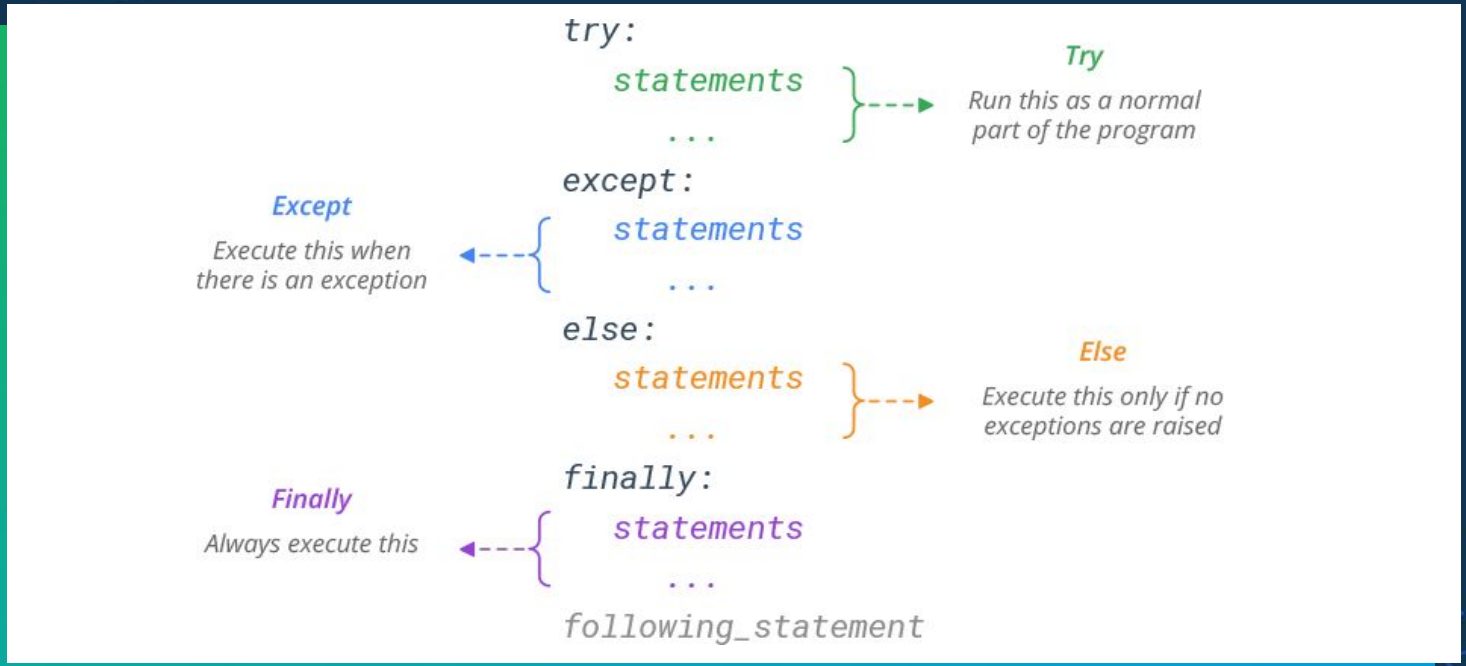
Built-in Exceptions



Dealing with Exceptions

- Exception handling is not meant to deal with a keyboard interrupt error caused by the user entering Ctrl + ‘C’ on their keyboard.
- Exception handling deals with the mentioned exceptions through a coding structure consisting of blocks of code divided into try, except, else and finally with the option of multiple except blocks and the else and finally blocks being optional.

Basic Exception Handling Syntax



CoGrammar

Handling a Built-in Exception

- Using try-except-else-finally, adheres to the standard of handling exceptions and helps maintain order in the program by handling unexpected behaviour.

```
try:
    num = int(input("Enter a number: "))
    result = 10 / num
except ZeroDivisionError:
    print("You can't divide by zero!")
except ValueError:
    print("Invalid input! Please enter a number.")
else:
    print(f"Result is: {result}")
finally:
    print("This will run no matter what.")
```

Resource Management

- Managing Clean-Up Tasks: Use the finally block to ensure clean-up actions are always performed, even if an error occurs.
- Example: Closing a file or releasing a database connection.

```
try:
    file = open("data.txt", "r")
except FileNotFoundError:
    print("File not found.")
finally:
    file.close()
```

Custom Exceptions with raise

- Use the `raise` keyword to trigger an exception with a custom message when a condition is met.
- Improves error handling and makes code more robust.

```
if condition:
    raise Exception("Custom error message")
```

Using a Custom Exception

- We're prompting the user to enter a value > 10. If the user enters a number that does not meet that condition, an exception is raised with a custom error message.

```
age = int(input("Enter your age: "))
if age < 0:
    raise ValueError("Age cannot be negative!")
```

Adding Context to Exceptions

- You can include additional information when raising exceptions by passing arguments to the exception constructor. This can be useful for providing context about the error:

```
def validate_input(value):
    if not isinstance(value, int):
        raise ValueError("Input must be an integer")

try:
    validate_input("hello")
except ValueError as e:
    print(f"Error: {e}")
```

Terminologies

KEYWORD	DESCRIPTION
try	The keyword used to start a try block.
except	The keyword used to catch an exception.
else	An optional clause that is executed if no exception is raised in the try block.
finally	An optional clause that is always executed, regardless of whether an exception is raised or not.
raise	The keyword used to manually raise an exception.
as	A keyword used to assign the exception object to a variable for further analysis.

Poll

1. **What is the purpose of the `else` block in a try statement?**

   A. It executes if an exception occurs in the try block.

   B. It executes only when the try block executes successfully without any exceptions.

   C. It is always executed after the try block, regardless of exceptions.

Poll

2. What will happen if a `finally` block is present but an exception occurs and is not caught?

```
try:
    print(10 / 0)
finally:
    print("This always runs.")
```

A. The program terminates immediately without running the finally block.
B. The finally block runs, then the program raises an exception.
C. The finally block runs, and the exception is ignored.

Poll

3. **What does the finally block do in Python exception handling?**

   A. Executes only if an exception is raised

   B. Executes only if no exception is raised

   C. Executes regardless of whether an exception was raised or not

   D. Skips execution if no exception is raised

Conclusion and Recap

- Use exception handling to ensure your program doesn't crash unexpectedly.
- try and except blocks allow you to catch and respond to specific errors.
- raise allows you to manually raise exceptions for custom error handling.

Learner Challenge

Build a Robust Division Program

- *Write a Python program that*:
  - Prompts the user to input two numbers.
  - Attempts to divide the first number by the second.
  - Handles the following exceptions: ZeroDivisionError - Occurs when the second number is zero; ValueError - Occurs if the user enters a non-numeric value.
  - If the division is successful, print the result.
  - Add an else block that runs if no exceptions occur, displaying a success message.
  - Add a finally block, printing a message that the program has completed, regardless of whether an exception occurred or not.

Questions and Answers

CoGrammar

Thank you for attending

SKILLS FOR LIFE
SKILLS BOOTCAMPS

Department for Education

CoGrammar