



# 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

---

- For all non-academic questions, please submit a query: [www.hyperiondev.com/support](http://www.hyperiondev.com/support)
- We would love your **feedback** on lectures: [Feedback on Lectures](#)

## 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** midway and throughout the 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](#)

# 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:



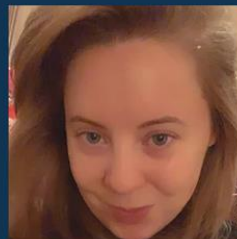
Ian Wyles  
Designated Safeguarding  
Lead



Simone Botes



Rafiq Manan



Charlotte Witcher



Nurhaan Snyman



Ronald Munodawafa



Tevin Pitts

Scan to report a  
safeguarding concern



or email the Designated  
Safeguarding Lead:  
Ian Wyles

[safeguarding@hyperiondev.com](mailto:safeguarding@hyperiondev.com)



# Skills Bootcamp Progression Overview

To be eligible for a certificate of completion, students must fulfil three specific criteria. These criteria ensure a high standard of achievement and alignment with the requirements for the successful completion of a Skills Bootcamp.

## ✓ Criterion 1 - Meeting Initial Requirements

Criterion 1 involves specific achievements **within the first two weeks** of the program. To meet this criterion, students need to:

- Attend a minimum of 7-8 hours per week of guided learning (lectures, workshops, or mentor calls) within the initial two-week period, for a total minimum of **15 guided learning hours** (GLH), by no later than **15 September 2024**.
- Successfully complete the Initial Assessment by the end of the first 14 days, by no later than **15 September 2024**.



# Skills Bootcamp Progression Overview

## ✓ Criterion 2 - Demonstrating Mid-Course Progress



Criterion 2 involves demonstrating meaningful progress through the successful completion of tasks **within the first half** of the bootcamp.

To meet this criterion, students should:

- Complete **42 guided learning hours** and the first half of the assigned tasks by the end of week 7, no later than **20 October 2024**.







# Skills Bootcamp Progression Overview

## ✓ Criterion 3 - Demonstrating Post-Course Progress

Criterion 3 involves showcasing students' progress after completing the course. To meet this criterion, students should:

- Complete all mandatory tasks before the bootcamp's end date. This includes any necessary resubmissions, no later than 22 December 2024.
- Achieve at least 84 guided learning hours by the end of the bootcamp, 22 December 2024.



**SKILLS  
FOR LIFE**

**SKILLS BOOTCAMPS**



Department  
for Education

# CoGrammar Exception-Handling



# Poll

## 1. What is an exception in programming?

- A. A type of function
- B. A special type of variable
- C. An error or unexpected event that occurs during program execution
- D. A method of input/output in programming

# Poll

## 2. What happens if an exception is not handled in Python?

- A. The program continues running as if nothing happened
- B. The program terminates and displays an error message
- C. The exception is automatically handled by Python
- D. The program restarts from the beginning

# Learning 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.
- Explain the role of the finally block in the try-except statement.
- Implement finally blocks to manage clean-up tasks such as closing files or releasing resources.

# Learning Outcomes

- 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



# 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.

$$1 + 1 = 3$$

# 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





# What are Exceptions?

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

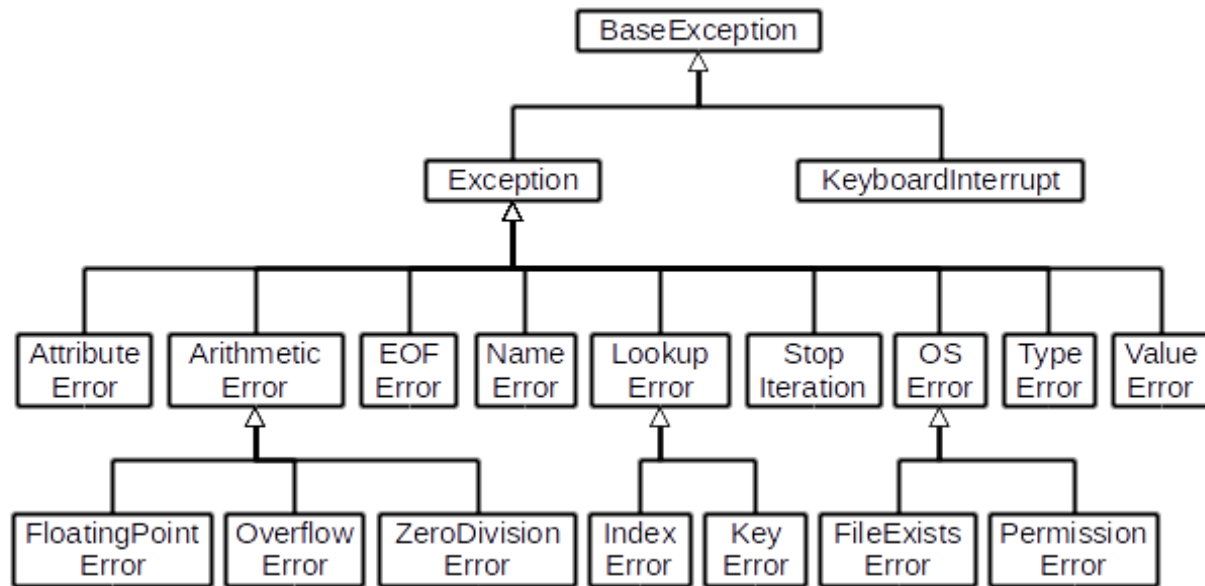
EXCEPTIONS



# 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.

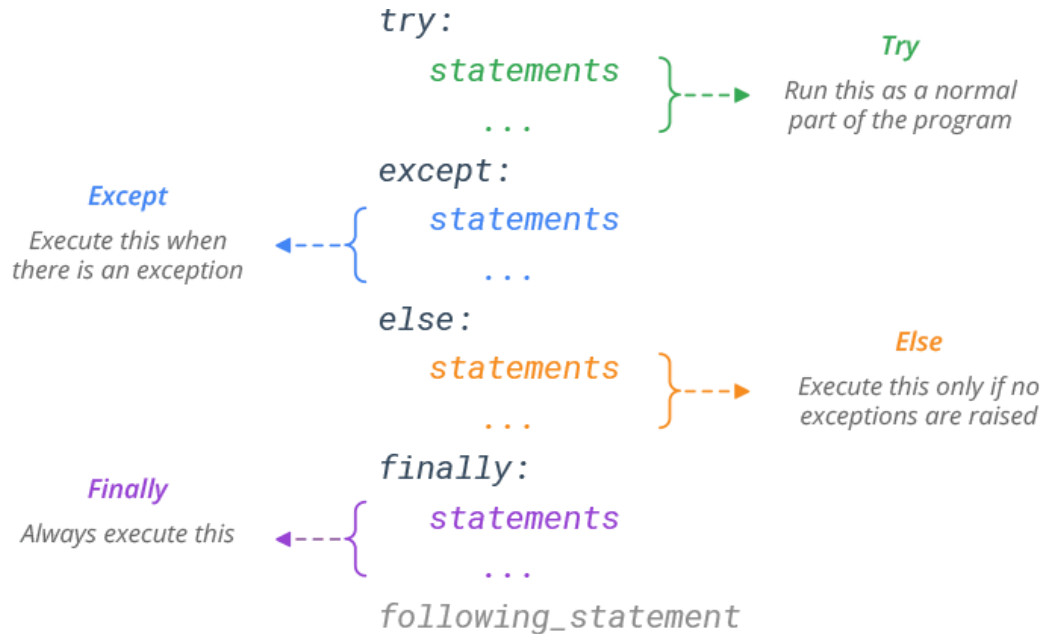
# Built-in Exceptions



# Dealing with Exceptions

- As seen in the previous slide, 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



# 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

- There will be occasions where you *would like more freedom* in your program to raise a custom exception whenever a certain condition is met.
- In Python we can do this by using the “raise” keyword and adding a custom message to the exception.
- The raise statement allows you to handle exceptional conditions in your program *explicitly*, providing better control over error handling and making your code more robust and predictable.

# 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.

```
num = int(input("Please enter a value greater than 10: "))

if num < 10:
    raise Exception(f"Your value is less than 10, being {num}.")
```

# 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}")
```

# Terminology

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.

**Let's take a short  
break**

**CoGrammar**





**Demo Time!**



# Poll

**1. What is the correct syntax for exception handling in Python?**

- A. try-finally-except
- B. try-catch-finally
- C. try-except-else-finally
- D. try-else-finally

# Poll

## 2. What is the purpose of the else clause in Python exception handling?

- A. It runs if an exception is raised
- B. It runs if no exceptions are raised
- C. It runs after the finally block
- D. It runs if the exception type matches the one specified in the except block

# 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

# Poll

**4. Which exception is raised when dividing by zero in Python?**

- A. ValueError
- B. ZeroDivisionError
- C. ArithmeticError
- D. TypeError

# Poll

**5. Which of the following exceptions cannot be caught using an except block in Python?**

- A. KeyboardInterrupt
- B. ZeroDivisionError
- C. IndexError
- D. ValueError

# Conclusion and Recap

- It may be tempting to wrap all code in a try-except block. However, you want to handle different errors differently.
- Don't try to use try-except blocks to avoid writing code that properly validates inputs.
- The correct usage for try except should only be for “exceptional” cases. Eg: The potential of Division by 0.
- Raise Exceptions When Necessary: If your code encounters an exceptional condition that it cannot handle, consider raising an exception using the raise statement.



# 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.

# Tasks To Complete

- T08 - IO Operations

# Questions and Answers



# Thank you for attending



Department  
for Education

CoGrammar

