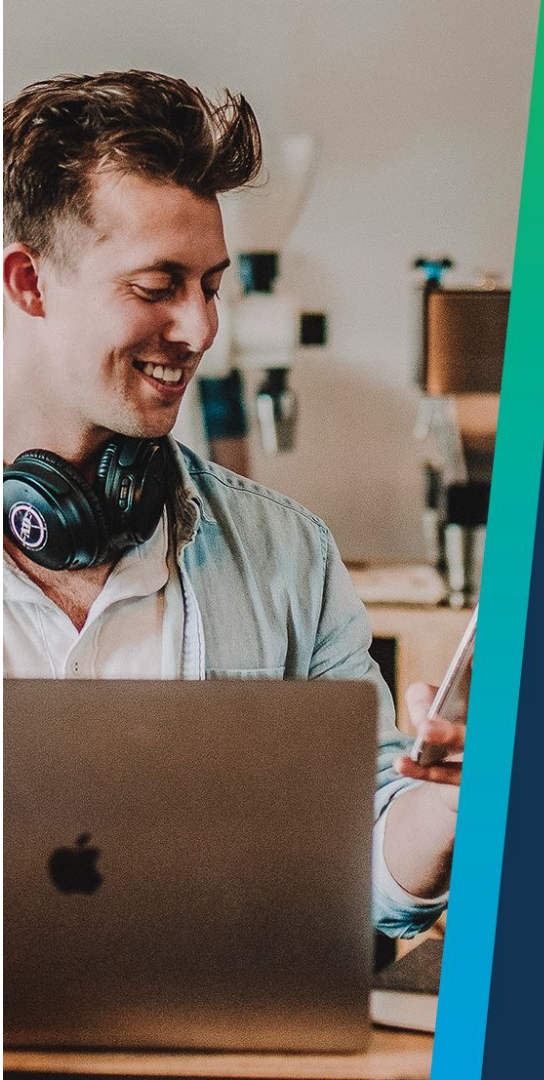


Software Engineering Bootcamp

Hyperiondev

Debugging

Lecture - Housekeeping

- The use of disrespectful language is prohibited in the questions, this is a supportive, learning environment for all - please engage accordingly.
- No question is daft or silly - ask them!
- There are Q/A sessions at the end of the session, should you wish to ask any follow-up questions.
- For all non-academic questions, please submit a query: www.hyperiondev.com/support
- Report a safeguarding incident: http://hyperiondev.com/safeguardreporting

Objectives

1. Define **debugging** and explain its importance in the software development process.
2. Identify common types of errors (**syntax**, **runtime**, **logical**) in Python code and **recognize** when each type occurs.
3. Use basic **debugging tools** in Python, such as print() statements to trace and fix errors.
4. Debug Python programs in an Integrated Development Environment **(IDE)** like Visual Studio Code by setting breakpoints and stepping through code.
5. Apply **best practices** for debugging to efficiently find and fix bugs in their Python code.

Poll

What will be the output of the following code snippet if output.txt does not exist initially?

```
with open("output.txt", 'w') as file:
    file.write("Python is great!")
with open("output.txt", 'w') as file:
    content = file.read()
print(content)
```

- The file will be empty
- An error will occur
- output.txt will not be created
- Python is great!

Poll

What will be the output of the following code snippet?

```
try:
    number = int("not_a_number")
except ValueError:
    print("ValueError occured")
except TypeError:
    print("TypeError occured")
except Exception as e:
    print(f"An error occured: {e}")
```

- TypeError occurred
- An error occurred: invalid literal for int() with base 10: 'not_a_number'
- The code will crash with an error
- ValueError occurred

Uncovering the Bugs in Your Code

- Have you ever wondered why we use the term "**bug**" for errors in our code?
- Is there a connection between **actual in**sects and the **problems in our software**?
- How does understanding the origin of this term help us become **better debuggers**?

Uncovering the Bugs in Your Code

9/9

0800 Arctan started
1000 " stopped - arctan ✓
13" u/c (032) MP - MC
(033) PRO 2
correct
{ 1.2700 9.037 847 025
9.037 846 ?95 correct
~~1.5821449000~~ ~~2.13047641 5 (-3)~~ 4.615925059(-2)
2.13047641 5
2.13067641 5
Relays 6-2 in 033 failed special speed test
In relay . 11.00 test.
Relays changed
1100 Started Cosine Tape (Sine check)
1525 Started Multy Adder Test.
1545 Relay #70 Panel F
(moth) in relay.

First actual case of bug being found.
~~1545~~ 1630 Arctangent started.
1700 closed down.

The First "Computer Bug"

- September 9, 1947: A **moth** was found trapped in a relay of the **Harvard Mark II computer**.
- The term "**bug**" had been used in engineering before, but this incident **popularized** it in computing.
- **Grace Hopper**'s team extracted the **moth** and taped it into the logbook, noting: "**First actual case of bug being found.**"

Introduction to Debugging

Definition of Debugging

- Debugging is the process of finding and fixing bugs in software code. These bugs may manifest as **syntax**, **runtime**, or **logical errors**, which can cause the program to **crash** or **produce incorrect results**.

Importance of Debugging

- **Code Correctness**: Ensures that the program functions as intended.
- **Reliability**: Reduces the risk of errors affecting future code changes.
- **Efficiency**: Saves time in the long run by identifying problems early.

Types of Bugs

- **Syntax Errors**: Violations of the language's grammar rules (e.g., missing a colon in an if statement).
- **Runtime Errors**: Errors that occur during program execution (e.g., dividing by zero).
- **Logical Errors**: Errors in the logic of the program that produce incorrect results (e.g., incorrect loop conditions).

Basic Debugging Techniques

print() Debugging Statements

- Insert **print()** statements to display **values of variables** at specific points in your code.
- Monitor the **output** to track the program's execution and identify **unexpected results**.
- Useful for checking variable values, **function inputs**/**outputs**, and control flow.

Rubber Duck Debugging

Credit: Alina Dumitru, April, 2022

Hyperiondev

Rubber Duck Debugging

- Explain your code to a **rubber duck** (or any inanimate object) to clarify your thinking.
- This technique forces you to **articulate** your code logic **clearly**, often leading to identifying the issue.
- It's a simple yet **effective** way to overcome mental blocks.

Code Review

- **Share** your code with **peers** for feedback and potential bug identification.
- **Different perspectives** can uncover overlooked issues.
- **Collaborate** to improve **code quality** and **maintainability**.

Python Debugging Tools

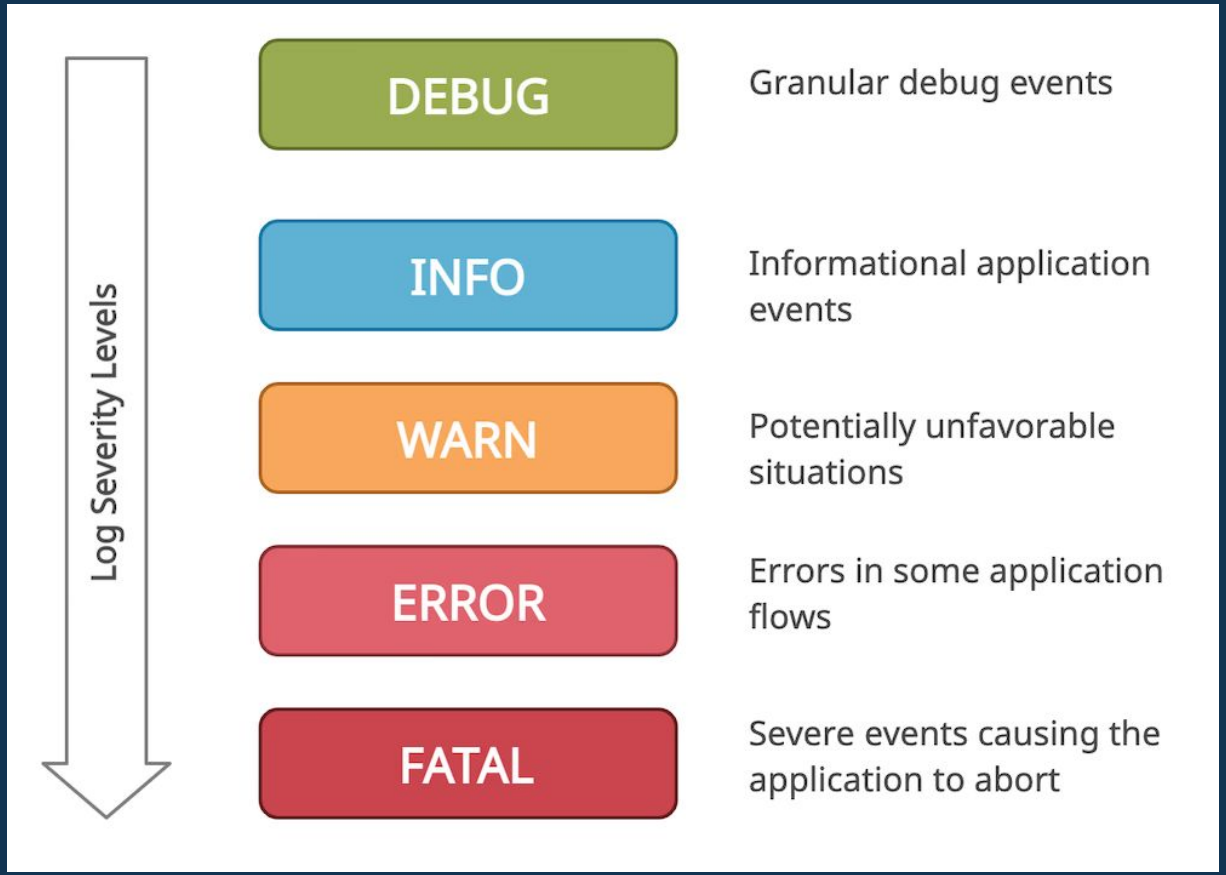
Python Debugger (pdb)

- What is **pdb**?
  - The Python Debugger (pdb) allows for **interactive** debugging within the **terminal**.
- Basic Commands:
  - **break** - Set a breakpoint to pause the program.
  - **continue** - Resume the program after hitting a breakpoint.
  - **step** - Step **into** function calls, executing the program one line at a time.
  - **next** - Execute the next line without stepping into functions.
  - **list** - Display source code around the current line.
  - **print** - Print variable values during debugging.

Logging

- **Logging**: **Record** program **events** and **messages** for **debugging** and monitoring.
- **Log levels**: Control the amount of detail captured (**DEBUG**, **INFO**, **WARNING**, **ERROR**, **CRITICAL**).
- **Benefits**:
  - Identify issues
  - Track program behavior
  - Analyze performance
- **Python logging module**:
  - Basic configuration and usage
  - Example:

Logging


Log Severity Levels
DEBUG
Granular debug events
INFO
Informational application events
WARN
Potentially unfavorable situations
ERROR
Errors in some application flows
FATAL
Severe events causing the application to abort

Logging

```python
import logging

logging.basicConfig(level=logging.DEBUG)
logger = logging.getLogger(__name__)

logger.debug("This is a debug message")
logger.info("Informational message")
logger.warning("Something unexpected happened")
logger.error("An error occurred")
logger.critical("Critical error!")
```

IDE Debuggers

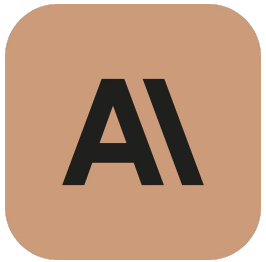
- **Setup**: Walk through setting up Python debugging in VS Code.
- **Features**:
  - **Breakpoints**: Pauses program execution at specific lines.
  - **Step Through Code**: Execute the code one line at a time to monitor program flow.
  - **Debug Console**: Allows for variable inspection and command execution during a paused state.

Using LLMs for Debugging

- What are **LLMs**?
  - Large Language Models like **ChatGPT**, **Claude**, and others.
  - AI models trained to **understand** and **generate** human-like text, including code.
- How They Help:
  - **Error Explanation**: Clarify error messages and suggest fixes.
  - **Code Review:** Analyze code snippets for potential issues.
  - **Debugging Assistance**: Provide step-by-step guidance for debugging.

Using LLMs for Debugging

- **Best Practices**:
  - Describe the problem **clearly**.
  - Provide **context** (error messages, code snippets).
  - **Verify** suggested solutions before applying.



perplexity

Gemini

Common Python Bugs and Debugging Best Practices

Common Bugs

- **Off-by-One Errors**: Common in loops when the range isn't set correctly.
- **Indentation Errors**: Python relies on indentation for code blocks, making incorrect indentation a common source of bugs.
- **Data Type Issues**: Using the wrong data type can lead to unexpected behavior (e.g., concatenating a string and an integer).
- **NameErrors**: Referencing variables before they are assigned.

Best Practices

- **Incremental Coding**: Write small pieces of code and test them before moving on to the next section.
- **Read Error Messages**: Python error messages provide valuable information, including line numbers and error types.
- **Use Debugging Tools**: Start with simple print statements, but leverage advanced tools like pdb or IDE debuggers for complex issues.
- **Keep Code Organized**: Well-structured and readable code is easier to debug.

Hyperiondev

Q & A Section

Please use this time to ask any questions relating to the topic explained, should you have any

Lesson Conclusion and Recap

Recap the key concepts and techniques covered during the lesson.

- **Understanding Bugs and Errors**:
  - Recognising different types of bugs, including syntax errors, runtime errors, and logical errors, and knowing when and why they occur.
- **Using print() for Debugging**:
  - Leveraging print() statements to monitor variable values and trace program flow as a simple and quick debugging method for basic issues.
- **IDE Debugging Features:**
  - Using debugging tools in an Integrated Development Environment (IDE) like Visual Studio Code, including breakpoints, step-through execution, and the debug console for more efficient debugging.
- **Common Python Bugs**:
  - Identifying common issues in Python code, such as off-by-one errors, incorrect indentation, and data type mismatches, and knowing how to approach fixing them.
- **Debugging Best Practices**:
  - Applying best practices such as coding incrementally, thoroughly reading error messages, and maintaining well-organised code to minimise errors and simplify the debugging process.

Hyperiondev

Thank you for joining us

Take regular breaks.
Stay hydrated.
Avoid prolonged screen time.
Remember to have fun :)

Some useful links

https://medium.com/javarevisited/debugging-tips-and-tricks-a-comprehensive-guide-8d84a58ca9f2

https://dev.to/surajondev/debugging-techniques-every-developer-should-know-85f

https://wearebrain.com/blog/10-effective-debugging-techniques-for-developers/

https://education.nationalgeographic.org/resource/worlds-first-computer-bug/