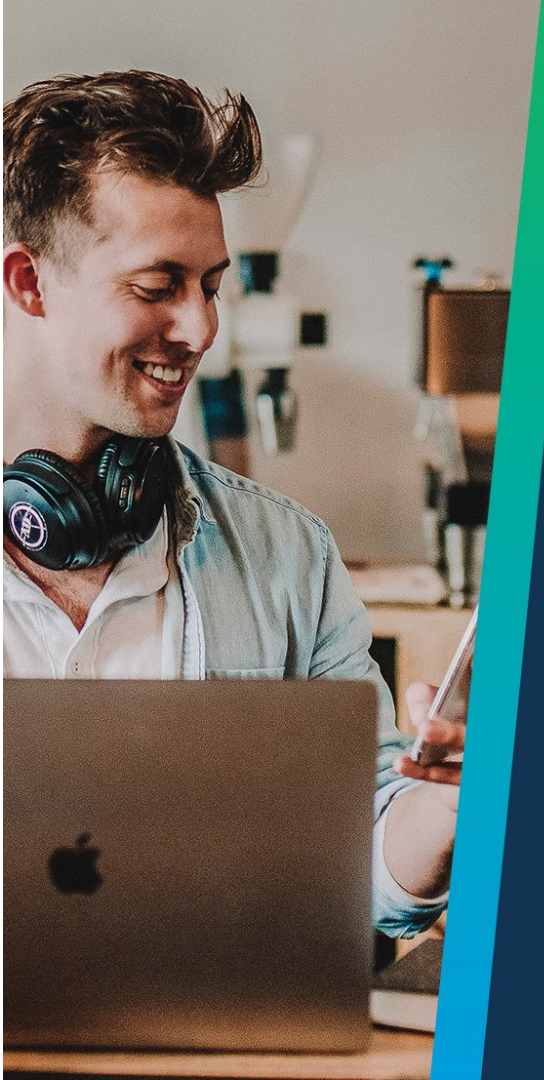


SE Bootcamp

Hyperiondev

Exception Handling

Welcome

Your Lecturer for this session

Nizaam Dalwai

Objectives

1. Understand how to handle Exceptions
2. Learn about 2D lists
3. Understand how to create, index and iterate over 2D lists

Defensive Programming

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

Handling Errors - If Statements

- If statements
  - Easy way of anticipating errors - if input is not correct, then do something to correct it.
  - For example, if user is trying to register a username that already exists, simply prompt the user for another username.

Handling Errors - try-except Blocks

- Recall the stack trace - all methods that were called to generate the error.
- If one of these methods can catch the exception, it is possible to still run the code.

Try-except Example

```
file = None
try:
    file = open('input.txt', 'r')
    # do stuff with file here

except FileNotFoundError:
    print("The file that you are trying to open does not
exist")

finally:
    if file is not None:
        file.close()
```

A Note on try-except Blocks

- 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 input.

Raising Exceptions

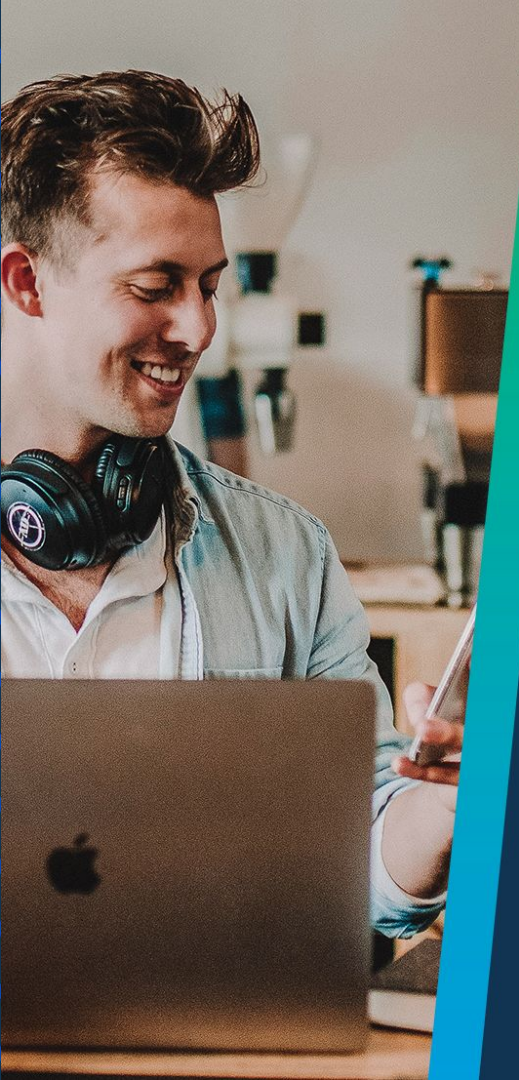
- Uses the raise keyword.
- Requires an object of type Exception

```
num = int(input("Please enter a value greater than 10"))
if num < 10:
    raise Exception('num was less than 10. The value of
num was: {}'.format(num))
```

Hyperiondev

Q & A Section

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



Hyperiondev

Thank you for joining us