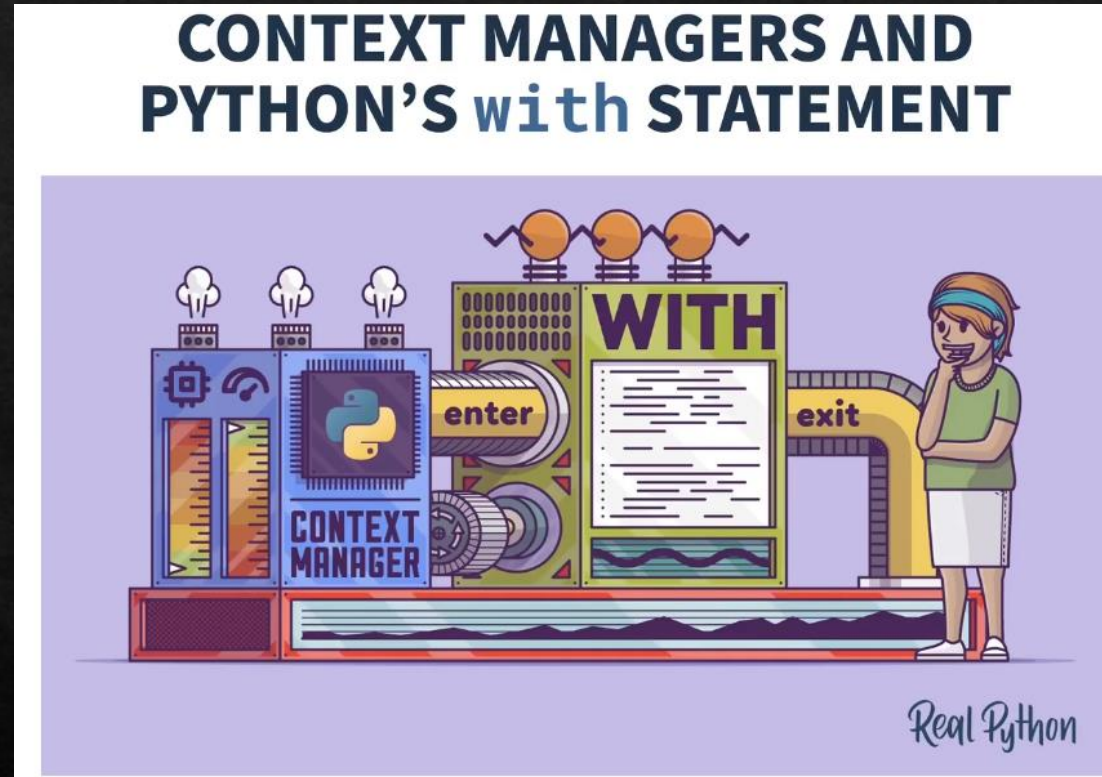


# Context Manager

With Statement

# Context-Managers-Introduction





# Overview

- A key part of programming is managing resources:
  - Files
  - Network connections
  - Database connections
- Have to ensure that you clean up after yourself
- Can use a **finally** section of a **try** block, or,
- Use a *context manager*
- When the block goes out of scope, a clean-up method is called, even if there is an exception

❖ contextlib — with 문 컨텍스트를 위한 유틸리티  
(<https://docs.python.org/ko/dev/library/contextlib.html>)

# Context-Managers-Introduction

- ❖ Click the topic to listen it for a while.
- ❖ The basic Data Stream communication method

```
with open("input.txt") as in_file, open("output.txt", "w") as out_file:  
    # Read content from input.txt  
    # Transform the content  
    # Write the transformed content to output.txt  
    pass
```

# Context Manager and Python's with Statement

## Table of Contents

1. Managing Resources in Python
  1. The try ... finally Approach
  2. The with Statement Approach
2. Using the Python with Statement
  1. Working With Files
  2. Traversing Directories
  3. Performing High-Precision Calculations
  4. Handling Locks in Multithreaded Programs
  5. Testing for Exceptions With pytest
3. Summarizing the with Statement's Advantages
4. Using the async with Statement
5. Creating Custom Context Managers
6. Coding Class-Based Context Managers
  6. Writing a Sample Class-Based Context Manager
  7. Handling Exceptions in a Context Manager
  8. Opening Files for Writing: First Version
  9. Redirecting the Standard Output
  10. Measuring Execution Time
7. Creating Function-Based Context Managers
  6. Opening Files for Writing: Second Version
  7. Mocking the Time
8. Writing Good APIs With Context Managers
9. Creating an Asynchronous Context Manager
10. Conclusion

