## 1. Introduction to Logging

### Why Logging?

- Debugging: Helps identify errors in code without halting program execution.
- Monitoring: Tracks the flow of a program or system.
- Maintenance: Provides detailed insights into what went wrong and where.
- Auditing: Records critical events for security and compliance.

### Key Points to Explain:

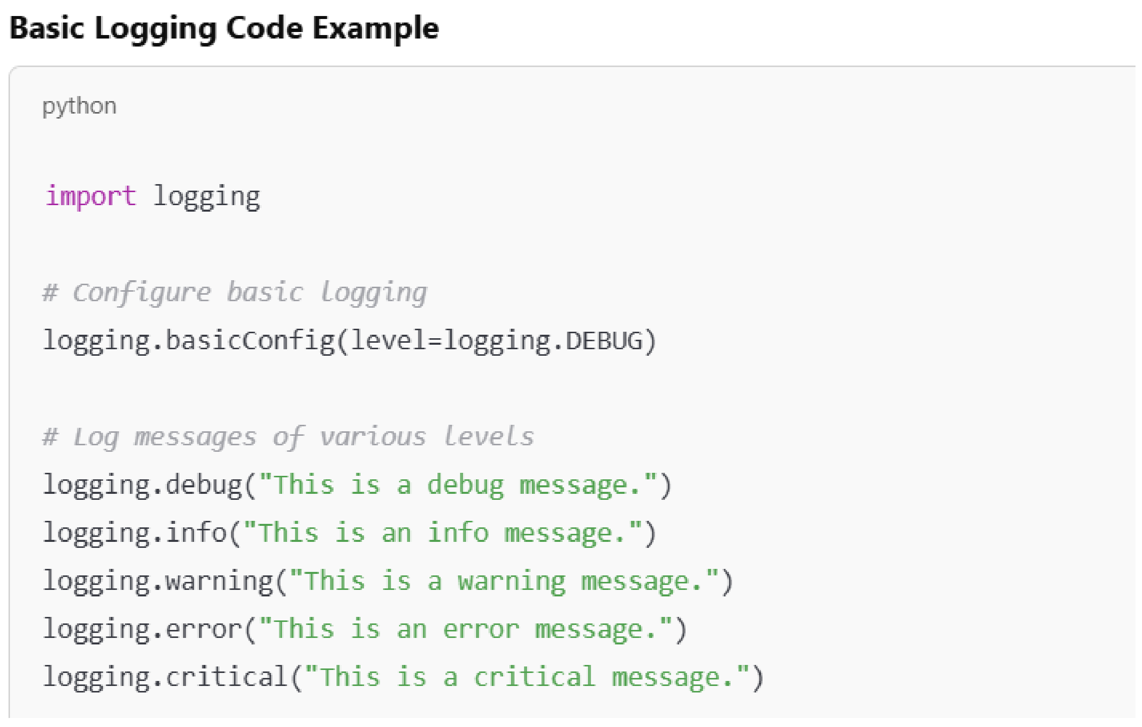
- Logging is more versatile and informative than print() statements.
- It allows the recording of messages with different levels of severity.

## 2. Logging Basics

### The Logging Levels

Explain the five standard levels in logging, emphasizing their use cases:

1. **DEBUG**: Detailed information for diagnosing problems during development.
2. **INFO**: General messages confirming the program is running as expected.
3. **WARNING**: Indicates a potential issue that does not stop the program.
4. **ERROR**: Logs an issue that caused a function or part of the program to fail.
5. **CRITICAL**: Logs a severe problem that may prevent the program from continuing.



### Basic Logging Code Example

python

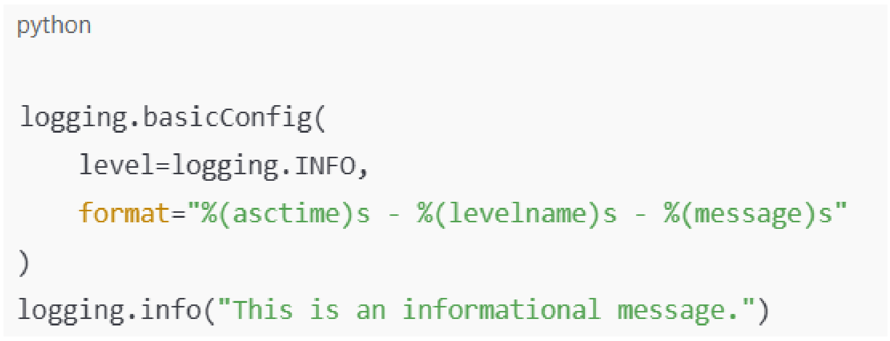
```python
import logging

# Configure basic logging
logging.basicConfig(level=logging.DEBUG)

# Log messages of various levels
logging.debug("This is a debug message.")
logging.info("This is an info message.")
logging.warning("This is a warning message.")
logging.error("This is an error message.")
logging.critical("This is a critical message.")
```

**3. Configuring Logging**

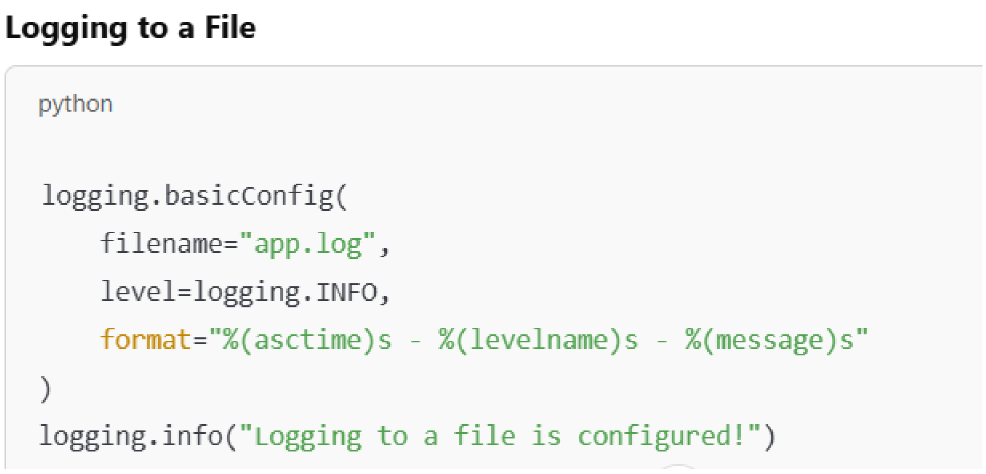
**Customizing Log Format**



python

```python
logging.basicConfig(
    level=logging.INFO,
    format="%(asctime)s - %(levelname)s - %(message)s"
)
logging.info("This is an informational message.")
```

- %(asctime)s: Timestamp of the log.
- %(levelname)s: The severity level.
- %(message)s: The actual log message.



### Logging to a File

python

```python
logging.basicConfig(
    filename="app.log",
    level=logging.INFO,
    format="%(asctime)s - %(levelname)s - %(message)s"
)
logging.info("Logging to a file is configured!")
```