

Python Logging

Logging is a crucial aspect of any application, providing a way to track events, errors, and operational information. Python's built-in logging module offers a flexible framework for emitting log messages from Python programs. In this lesson, we will cover the basics of logging, including how to configure logging, log levels, and best practices for using logging in Python applications.

Configuring Logging

```
import logging

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

# Log messages with different severity 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')
```

This code configures the logging module to display messages of level DEBUG and higher.
Key Definition: `logging.basicConfig(**kwargs)` - Configures the logging module with the specified parameters.

Log Levels

Python's logging module has several log levels indicating the severity of events. The default levels are:

Level Description

DEBUG Detailed information, typically of interest only when diagnosing problems.

INFO Confirmation that things are working as expected.

WARNING An indication that something unexpected happened.

ERROR Due to a more serious problem, the software has not been able to perform some function.

CRITICAL A very serious error, indicating that the program itself may be unable to continue running.

Logging to a File

```
import logging

# Configure logging to a file
logging.basicConfig(
    filename='app.log',
    filemode='w',
    level=logging.DEBUG,
    format='%(asctime)s - %(name)s - %(levelname)s - %(message)s',
    datefmt='%Y-%m-%d %H:%M:%S')
```

```
)

# Log messages
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')
```

This configuration logs messages to a file named 'app.log' with a specific format.

Key Definition: `logging.basicConfig(filename='app.log', ...)` - Configures logging to output to a specified file.

Logging with Multiple Loggers

```
import logging

# create a logger for module1
logger1 = logging.getLogger('module1')
logger1.setLevel(logging.DEBUG)

# create a logger for module2
logger2 = logging.getLogger('module2')
logger2.setLevel(logging.WARNING)

# Configure logging settings
logging.basicConfig(
    level=logging.DEBUG,
    format='%(asctime)s - %(name)s - %(levelname)s - %(message)s',
    datefmt='%Y-%m-%d %H:%M:%S'
)

# log messages with different loggers
logger1.debug('This is a debug message for module1')
logger1.critical('This is a critical message for logger1')
logger2.warning('This is a warning message for module2')
logger2.error('This is an error message')
```

This example demonstrates how to create and use multiple loggers for different modules in your application.

Key Definition: `logging.getLogger(name)` - Retrieves a logger with the specified name.

Conclusion

Logging is an essential part of application development, allowing developers to monitor and debug their applications effectively.