Python Logging: Comprehensive Guide

Assignment 1: Basic Logging

```
import logging
def basic_logger():
    logging.basicConfig(filename='app.log', level=logging.DEBUG)
    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')
# Test the function
basic_logger()
```

This function sets up a basic logger that logs messages of various levels to a file named 'app.log'.

Assignment 2: Logging with Different Handlers

```
def logger_with_handlers():
    logger = logging.getLogger('my_logger')
    logger.setLevel(logging.DEBUG)
    file_handler = logging.FileHandler('app.log')
    console_handler = logging.StreamHandler()
    file_handler.setLevel(logging.DEBUG)
    console_handler.setLevel(logging.DEBUG)
    formatter = logging.Formatter('%(asctime)s - %(name)s - %(levelname)s - %(message)s')
    file_handler.setFormatter(formatter)
    console_handler.setFormatter(formatter)
    logger.addHandler(file_handler)
    logger.addHandler(console_handler)
    logger.debug('This is a debug message')
    logger.info('This is an info message')
    logger.warning('This is a warning message')
    logger.error('This is an error message')
    logger.critical('This is a critical message')
# Test the function
logger_with_handlers()
```

This function creates a logger that logs messages to both a file and the console.

Assignment 3: Formatting Log Messages

```
def logger_with_custom_format():
    logger = logging.getLogger('custom_logger')
    logger.setLevel(logging.DEBUG)
    file_handler = logging.FileHandler('custom_app.log')
    console_handler = logging.StreamHandler()
    formatter = logging.Formatter('%(asctime)s - %(levelname)s - %(message)s')
    file_handler.setFormatter(formatter)
    console_handler.setFormatter(formatter)
    logger.addHandler(file_handler)
    logger.addHandler(console_handler)
    logger.debug('This is a debug message')
    logger.info('This is an info message')
    logger.warning('This is a warning message')
    logger.error('This is an error message')
    logger.critical('This is a critical message')
# Test the function
logger_with_custom_format()
```

This function sets up a logger with a custom format for log messages.

Task 2: Different Formats

```
def logger_with_different_formats():
    logger = logging.getLogger('multi_format_logger')
    logger.setLevel(logging.DEBUG)
    file_handler = logging.FileHandler('multi_format_app.log')
    console_handler = logging.StreamHandler()
    file_formatter = logging.Formatter('%(asctime)s - %(name)s - %(levelname)s - %(messag
    console_formatter = logging.Formatter('%(asctime)s - %(levelname)s - %(message)s')
    file_handler.setFormatter(file_formatter)
    console_handler.setFormatter(console_formatter)
    logger.addHandler(file_handler)
    logger.addHandler(console_handler)
    logger.debug('This is a debug message')
    logger.info('This is an info message')
    logger.warning('This is a warning message')
    logger.error('This is an error message')
    logger.critical('This is a critical message')
# Test the function
logger_with_different_formats()
```

This function uses different formats for the file and console handlers.

Assignment 4: Rotating Log Files

```
from logging.handlers import RotatingFileHandler
def logger_with_rotating_file_handler():
    logger = logging.getLogger('rotating_logger')
    logger.setLevel(logging.DEBUG)
    rotating_handler = RotatingFileHandler('rotating_app.log', maxBytes=2000, backupCount
    formatter = logging.Formatter('%(asctime)s - %(name)s - %(levelname)s - %(message)s')
    rotating_handler.setFormatter(formatter)
    logger.addHandler(rotating_handler)
    for i in range(100):
        logger.debug('This is debug message number {}'.format(i))
logger_with_rotating_file_handler()
```

This function creates a logger that uses a rotating file handler to manage log file sizes.

Assignment 5: Logging Exceptions

```
def log_exception():
    logger = logging.getLogger('exception_logger')
    logger.setLevel(logging.ERROR)
    file_handler = logging.FileHandler('exception_app.log')
    formatter = logging.Formatter('%(asctime)s - %(name)s - %(levelname)s - %(message)s')
    file_handler.setFormatter(formatter)
    logger.addHandler(file_handler)
    try:
        1 / 0
    except Exception as e:
        logger.exception('An exception occurred')
log_exception()
```

This function logs an exception stack trace to a log file when an exception occurs.

Assignment 6: Contextual Logging

```
def logger_with_context():
    logger = logging.getLogger('context_logger')
    logger.setLevel(logging.DEBUG)
    file_handler = logging.FileHandler('context_app.log')
    formatter = logging.Formatter('%(asctime)s - %(levelname)s - %(message)s - %(funcName
    file_handler.setFormatter(formatter)
    logger.addHandler(file_handler)
    def test_func():
        logger.debug('This is a debug message')
        logger.info('This is an info message')
        logger.warning('This is a warning message')
        logger.error('This is an error message')
        logger.critical('This is a critical message')
    test_func()
# Test the function
logger_with_context()
```

This function creates a logger that includes contextual information in the log messages.

Task 2: Additional Context

```
def logger_with_additional_context(user_id, session_id):
    logger = logging.getLogger('additional_context_logger')
    logger.setLevel(logging.DEBUG)
    file_handler = logging.FileHandler('additional_context_app.log')
    formatter = logging.Formatter('%(asctime)s - %(name)s - %(levelname)s - %(message)s -
    file_handler.setFormatter(formatter)
    logger.addHandler(file_handler)
    extra = {'user_id': user_id, 'session_id': session_id}
    def test_func():
        logger.debug('This is a debug message', extra=extra)
        logger.info('This is an info message', extra=extra)
        logger.warning('This is a warning message', extra=extra)
        logger.error('This is an error message', extra=extra)
        logger.critical('This is a critical message', extra=extra)
    test_func()
# Test the function
logger_with_additional_context('user123', 'session456')
```

This function logs messages with additional contextual information such as user ID and session ID.

Assignment 7: Configuring Logging with a Dictionary

```
import logging.config
def configure_logging_with_dict():
    log_config = {
        'version': 1,
        'formatters': {
            'default': {
                'format': '%(asctime)s - %(name)s - %(levelname)s - %(message)s'
            },
            'detailed': {
                'format': '%(asctime)s - %(name)s - %(levelname)s - %(message)s - %(funcN
            }
        },
        'handlers': {
            'file': {
                'class': 'logging.FileHandler',
                'filename': 'dict_config_app.log',
                'formatter': 'detailed',
                'level': 'DEBUG'
            },
            'console': {
                'class': 'logging.StreamHandler',
                'formatter': 'default',
                'level': 'DEBUG'
            }
        },
        'root': {
            'handlers': ['file', 'console'],
            'level': 'DEBUG'
        }
    }
    logging.config.dictConfig(log_config)
    logger = logging.getLogger('')
    logger.debug('This is a debug message')
    logger.info('This is an info message')
    logger.warning('This is a warning message')
    logger.error('This is an error message')
    logger.critical('This is a critical message')
# Test the function
```

```
configure_logging_with_dict()
```

This function configures logging using a dictionary to set up handlers and formatters.

Assignment 8: Logging in a Multi-Module Application

```
import logging
from module_a import module_a_function
from module_b import module_b_function
def setup_logging():
    log_config = {
        'version': 1,
        'formatters': {
            'default': {
                'format': '%(asctime)s - %(name)s - %(levelname)s - %(message)s'
            }
        },
        'handlers': {
            'file': {
                'class': 'logging.FileHandler',
                'filename': 'multi_module_app.log',
                'formatter': 'default',
                'level': 'DEBUG'
            },
            'console': {
                'class': 'logging.StreamHandler',
                'formatter': 'default',
                'level': 'DEBUG'
            }
        },
        'root': {
            'handlers': ['file', 'console'],
            'level': 'DEBUG'
        }
    }
    logging.config.dictConfig(log_config)
# Main script
if __name__ == '__main__':
    setup_logging()
    logger = logging.getLogger(__name__)
    logger.info('Main module started')
    module_a_function()
    module_b_function()
    logger.info('Main module finished')
```

This script sets up logging for a multi-module application, allowing each module to log messages.

Module A Function

```
import logging
def module_a_function():
    logger = logging.getLogger(__name__)
    logger.info('Module A function started')
    logger.debug('This is a debug message from Module A')
    logger.info('Module A function finished')
```

This function logs messages related to Module A's execution.

Module B Function

```
import logging
def module_b_function():
    logger = logging.getLogger(__name__)
    logger.info('Module B function started')
    logger.debug('This is a debug message from Module B')
    logger.info('Module B function finished')
```

This function logs messages related to Module B's execution.

Assignment 9: Logging Performance

```
import logging
import time
from logging.handlers import RotatingFileHandler
def benchmark_logging_performance():
    logger = logging.getLogger('performance_logger')
    logger.setLevel(logging.DEBUG)
    # File handler
    file_handler = logging.FileHandler('performance_file.log')
    file_handler.setLevel(logging.DEBUG)
    logger.addHandler(file_handler)
    start_time = time.time()
    for i in range(10000):
        logger.debug('This is a debug message')
    end_time = time.time()
    print('File handler logging time: {} seconds'.format(end_time - start_time))
    logger.removeHandler(file_handler)
    # Console handler
    console_handler = logging.StreamHandler()
    console_handler.setLevel(logging.DEBUG)
    logger.addHandler(console_handler)
    start_time = time.time()
    for i in range(10000):
        logger.debug('This is a debug message')
    end_time = time.time()
    print('Console handler logging time: {} seconds'.format(end_time - start_time))
    logger.removeHandler(console_handler)
    # Rotating file handler
    rotating_handler = RotatingFileHandler('performance_rotating.log', maxBytes=2000, bac
    rotating_handler.setLevel(logging.DEBUG)
    logger.addHandler(rotating_handler)
    start_time = time.time()
    for i in range(10000):
        logger.debug('This is a debug message')
    end_time = time.time()
    print('Rotating file handler logging time: {} seconds'.format(end_time - start_time))
    logger.removeHandler(rotating_handler)
# Test the function
benchmark_logging_performance()
```

This script benchmarks the performance of logging with different handlers.

Task 2: Benchmark Logging Formatting Performance

```
def benchmark_logging_formatting_performance():
    logger = logging.getLogger('formatting_performance_logger')
    logger.setLevel(logging.DEBUG)
    # File handler without formatting
    file_handler = logging.FileHandler('performance_no_format.log')
    file_handler.setLevel(logging.DEBUG)
    logger.addHandler(file_handler)
    start_time = time.time()
    for i in range(10000):
        logger.debug('This is a debug message')
    end_time = time.time()
    print('File handler logging time without formatting: {} seconds'.format(end_time - st
    logger.removeHandler(file_handler)
    # File handler with formatting
    file_handler = logging.FileHandler('performance_with_format.log')
    file_handler.setLevel(logging.DEBUG)
    formatter = logging.Formatter('%(asctime)s - %(name)s - %(levelname)s - %(message)s')
    file_handler.setFormatter(formatter)
    logger.addHandler(file_handler)
    start_time = time.time()
    for i in range(10000):
        logger.debug('This is a debug message')
    end_time = time.time()
    print('File handler logging time with formatting: {} seconds'.format(end_time - start
    logger.removeHandler(file_handler)
# Test the function
benchmark_logging_formatting_performance()
```

This script compares the performance of logging with and without message formatting.

Assignment 10: Advanced Logging Configuration

```
import logging.config
def setup_logging_from_file():
    logging.config.fileConfig('logging.conf')
    logger = logging.getLogger(__name__)
    logger.debug('This is a debug message')
    logger.info('This is an info message')
    logger.warning('This is a warning message')
    logger.error('This is an error message')
    logger.critical('This is a critical message')
# Test the function
setup_logging_from_file()
```

This function configures logging using an external configuration file.

Key Definitions of Functions and Methods Used in Logging

logging.basicConfig(): Configures the logging system with basic settings, such as the log file name and log level.

logging.getLogger(name): Retrieves a logger instance with the specified name. If no logger with that name exists, it creates one.

logger.setLevel(level): Sets the logging level for the logger, determining the severity of messages that will be logged.

logging.FileHandler(filename): Creates a handler that writes log messages to a specified file.

logging.StreamHandler(): Creates a handler that writes log messages to the console (standard output).

logging.Formatter(format): Creates a formatter that specifies the layout of log messages.

handler.setFormatter(formatter): Assigns a formatter to a handler, defining how log messages will be formatted.

logger.addHandler(handler): Adds a handler to the logger, allowing it to send log messages to the specified destination (file, console, etc.).

logger.debug(message): Logs a message with the DEBUG level, used for detailed diagnostic information.

logger.info(message): Logs a message with the INFO level, used for general information about program execution.

logger.warning(message): Logs a message with the WARNING level, indicating a potential problem.

logger.error(message): Logs a message with the ERROR level, indicating a serious problem that prevented a function from performing its task.

logger.critical(message): Logs a message with the CRITICAL level, indicating a very serious error that may prevent the program from continuing.

logger.exception(message): Logs a message with the ERROR level, including the stack trace of the exception that was caught.

RotatingFileHandler(filename, maxBytes, backupCount): Creates a handler that writes log messages to a file, rotating the log file when it reaches a specified size, and keeping a specified number of backup files.

logging.config.dictConfig(config): Configures logging using a dictionary that specifies the configuration for loggers, handlers, and formatters.

logging.config.fileConfig(filename): Configures logging using an external configuration file.