**Logging and debugging**

Logging and debugging are essential aspects of software development in Python. Here's an overview of how to use Python's built-in logging module and debugging techniques:

**Logging**

Logging allows you to **record information** about the execution of your program. Python's logging module provides a flexible framework for logging messages with different severity levels.

**basicConfig()** configures the logging system.

**level=logging.DEBUG** sets the threshold level for logging.

**filename='example.log'** specifies the log file.

**filemode='w'** specifies that the log file will be overwritten each time.

**format='%(asctime)s - %(levelname)s - %(message)s'** defines the format of log messages.

**Different levels of Logging:**

In Python's logging module, several levels of logging allow you to categorize log messages based on their severity. These levels help you control which types of messages get logged and provide a way to filter and manage log output effectively. Here are the standard logging levels in Python:

Each level has a numerical value associated with it, which is used to determine the severity of the message. The numerical values are as follows:

**DEBUG**: 10  
**INFO**: 20  
**WARNING**: 30  
**ERROR**: 40  
**CRITICAL**: 50

**DEBUG**: Detailed information, typically useful for debugging purposes.

**INFO**: General information about the program's execution. This is typically used to confirm that things are working as expected.

**WARNING**: Indicates a potential issue or unexpected behaviour that does not necessarily cause the program to fail but may require attention.  
**ERROR**: Indicates a serious issue or error that has occurred. These messages typically indicate problems that might cause the program to fail or behave unexpectedly.  
**CRITICAL**: Indicates a critical error that requires immediate attention. These are typically used to log severe errors that may lead to the termination of the program or data loss.

By default, the logging module is configured to log messages at the WARNING level and above. You can configure logging to capture messages at specific levels by setting the appropriate logging level threshold.

**Debugging:**

Debugging involves identifying and fixing errors or bugs in your code. Python offers several debugging techniques:

* Print Statement
* Debugger
* IDE
* Logging

**Print Statements**:

Insert **print** statements at strategic points in your code to inspect variable values and control flow.

**Debugger**:

Python's built-in debugger module **pdb** allows you to set breakpoints, step through code, and inspect variables interactively.

**Integrated Development Environments (IDEs)**:

IDEs like PyCharm, Visual Studio Code, or Spyder provide powerful debugging features such as breakpoints, variable inspection, step-by-step execution, etc.

**Logging for Debugging**:

You can also use logging statements instead of print for debugging purposes. By logging variable values, function calls, or program states.

Choose the debugging technique that best fits your workflow and the complexity of the debugging task. For simple debugging tasks can often be handled with **print** statements. while complex debugging scenarios, IDEs, the **pdb** debugger, **logging** might be more effective or use all of the above.