LOGGING IN PYTHON:

```
def main():
    # Set logging level based on DEBUG environment variable
    DEBUG = os.getenv('DEBUG', 'false').lower() == 'true'

    logging.basicConfig(
        level=logging.DEBUG if DEBUG else logging.INFO,
        format='%(asctime)s - %(levelname)s - %(message)s'
    )

    # The rest of the code remains the same
    ...
```

EXPLANATION:

The line `DEBUG = os.getenv('DEBUG', 'false').lower() == 'true'` is a way to set the `DEBUG` variable in your Python script based on an environment variable.

Explanation:

1. **`os.getenv('DEBUG', 'false')`:** This function retrieves the value of the environment variable `DEBUG`. If `DEBUG` is not set, it defaults to `'false'`.
2. **`.lower()`:** Converts the value to lowercase to ensure consistency in comparison.
3. **`== 'true'`:** Compares the lowercase value to the string `'true'`. If they match, `DEBUG` is set to `True`; otherwise, it is set to `False`.

Usage:

- In your Jenkins job, you can set the `DEBUG` environment variable to `true` or `false`.
- This line of code will ensure that the `DEBUG` variable in your script is `True` if `DEBUG` is `true` and `False` otherwise, allowing you to control the debug mode from Jenkins.

```
logging.basicConfig(
    level=logging.DEBUG if DEBUG else logging.INFO,
    format='%(asctime)s - %(levelname)s - %(message)s'
)
```

Explanation:

1. **`logging.basicConfig(...)`**: This function configures the logging module. It sets up the basic configuration for logging, specifying the level of severity and the format of the log messages.

2. **`level=logging.DEBUG if DEBUG else logging.INFO`**:
    - **`level`**: This parameter sets the threshold for the logger. Messages which are less severe than this level will be ignored.
    - **`logging.DEBUG if DEBUG else logging.INFO`**: This conditional expression sets the logging level based on the `DEBUG` variable:
        - If `DEBUG` is `True`, it sets the logging level to `DEBUG`. This means all messages of level `DEBUG` and higher (DEBUG, INFO, WARNING, ERROR, CRITICAL) will be logged.
        - If `DEBUG` is `False`, it sets the logging level to `INFO`. This means only messages of level `INFO` and higher (INFO, WARNING, ERROR, CRITICAL) will be logged. `DEBUG` messages will be ignored.
3. **`format='%(asctime)s - %(levelname)s - %(message)s'`**:
    - **`format`**: This parameter specifies the format of the log messages.
    - **`'%(asctime)s - %(levelname)s - %(message)s'`**: This format string includes three pieces of information in each log message:
        - **`%(asctime)s`**: The date and time when the log message was created.
        - **`%(levelname)s`**: The log level of the message (e.g., DEBUG, INFO, WARNING, ERROR, CRITICAL).
        - **`%(message)s`**: The actual log message.

Usage:

- When **`DEBUG`** is set to `True`:
    - The logging level is set to `DEBUG`.
    - All log messages, including debug messages, will be shown.
- When **`DEBUG`** is set to `False`:
    - The logging level is set to `INFO`.
    - Only messages of level `INFO` and above will be shown, and debug messages will be ignored.

This setup allows you to control the verbosity of your logging output by setting the `DEBUG` variable, making it easier to switch between detailed debugging information and regular informational logging.

`asctime` stands for "ASCII time" and refers to a human-readable string representation of the current time. In the context of the `logging` module in Python, `asctime` is used as part of the log message format to include the timestamp when the log entry was created.

Breakdown of the Format String

- **`%(asctime)s`**:
  - This placeholder is replaced with the current time in a readable format, such as `2024-08-04 12:00:00`.
- **`%(levelname)s`**:
  - This placeholder is replaced with the log level (e.g., `DEBUG`, `INFO`, `ERROR`).
- **`%(message)s`**:
  - This placeholder is replaced with the actual log message.

Example Log Entry

Given the format string above, a log entry might look like this:

yaml

Copy code

```
2024-08-04 12:00:00 - DEBUG - App_Name: ExampleApp
```

ERROR:

The statement logging.error(e) is used to log an error message in Python. Here's a breakdown of what it does:

1. **logging.error**:
   - This function is part of Python's logging module and is used to log messages with a severity level of ERROR. The ERROR level is used for logging error messages, indicating a serious problem that might prevent the program from continuing to run.
2. **e**:
   - e is typically an exception object that has been caught in an except block. When you catch an exception, you usually assign it to a variable (commonly e) that contains information about the exception, such as its type, message, and stack trace.