```
========
Logging
========

=> The process of storing application execution details to a file.

=> To understand runtime behaviour of the application we will use logging in the application.

=> With the help of logging we can identify root cause of the exception.

=> To implement logging, we have several logging frameworks

                                1) Log4J
                                2) Log4J2
                                3) Logback

=======================
Logging Architecture
=======================

1) Logger

2) Layout

3) Appender


=> Logger is a class which is providing methods to write log msgs

        ex : trace ( ), debug (), info (), warn (), error ( )

Note: For every java class we will create one logger object.


=> Layout represents log msg pattern

        Ex:   Date time  log-level   thread  class - msg

=> Appender is used to write log msg to destination

        Ex : ConsoleAppender, FileAppender

======================
Spring Boot logging
=======================

1) Create boot app with web starter

2) Create service class like below

@Service
public class MsgService {

        private Logger logger = LoggerFactory.getLogger(MsgService.class);

        public String getWelcomeMsg() {

                logger.info("getWelcomeMsg() - started...");

                String msg = "Welcome to Ashok IT";

                logger.info("getWelcomeMsg() - ended....");
```

```
		return msg;
	}
}
```

3) Create Rest Controller class like below

```
@RestController
public class MsgRestController {

	@Autowired
	private MsgService msgService;

	@GetMapping("/welcome")
	public String getMsg() {
		return msgService.getWelcomeMsg();
	}

}
```

4) Run the application and send the request

```
		URL : http://localhost:port/welcome
```

Note: To print logs in the log file add below property in application.properties file

```
		logging.file.name=app.log
```

===============
Logging Levels
===============

=> Log msgs will be stored to log file based on log level

=> Logging having several levels

```
	#### TRACE > DEBUG > INFO > WARN > ERROR ####
```

=> TRACE is used to store every line execution details

=> DEBUG is used to store execution flow at low level

=> INFO is used to store execution flow at high level

=> WARN is used to store warnings in code execution flow

=> ERROR is used to store execptions occured in code flow

Note: When we set log level, log msgs will be printed from that level to all higher levels also.

```
		Log level = INFO  => INFO + WARN + ERROR

		Log Level = WARN  => WARN + ERROR

		Log Level = DEBUG => DEBUG + INFO + WARN + ERROR

		Log Level = ERROR => ERROR
```

Note: In springboot, default log level is INFO.

=> We can change log level in the application using below property

```
		logging.level.root = WARN
```

```
======================
Logging with Rolling
======================

=> If we use single log file to store log msgs then after few days log file size will become very
big.

=> If file size is keep on increasing then it will become difficult to open/read log file data.

Note: To avoid this problem we will use rolling mechanism.

=> Rolling we can implement in 2 ways

                        1) Size Based Rolling
                        2) Time Based Rolling

=> Size Based Rolling is used to create new log file once old log is reached to given limit.

                Ex: 1 GB

=> Time based rolling is used to create every day new log file.

=> To configure rolling, we will use RollingFileAppender.


=> We can configure rolling in 2 ways

                        1) application.properties

                        2) logback.xml

================================================================

=> We will keep logback.xml under src/main/resources folder

=> logback.xml is used to customize logging in our application

=> In logback.xml we will configure below components

1) RollingFileAppender with policy
2) Log msg Pattern
3) Level

=======================logback.xml=========================
<configuration>

        <appender name="RollingFile"
                class="ch.qos.logback.core.rolling.RollingFileAppender">
                <file>sbi.log</file>
                <encoder>
                        <pattern>%d [%thread] %-5level %-50logger{40} - %msg%n</pattern>
                </encoder>
                <rollingPolicy
                        class="ch.qos.logback.core.rolling.SizeAndTimeBasedRollingPolicy">
                        <fileNamePattern>sbi-%d{yyyy-MM-dd}.%i.log</fileNamePattern>
                        <maxFileSize>1MB</maxFileSize>
                        <maxHistory>30</maxHistory>
                        <totalSizeCap>10MB</totalSizeCap>
                </rollingPolicy>
        </appender>

        <root level="INFO">
                <appender-ref ref="RollingFile" />
        </root>
```

```
</configuration>
=================================================================

==============================================
Q) What is the diff between SLF4J and Log4J ?
==============================================

=> SL4J stands for Simple Logging Facade for Java (SLF4J)

=> It serves as a simple facade or abstraction for various logging frameworks (e.g.
java.util.logging, logback, log4j)

=> If we use Sl4J for logging then our application will be loosely coupled with logging frameworks.

=> By using SL4J we can plugin the desired logging framework at deployment time.

================
Log Monitoring
================

=> It is the process of reading or observing log msgs available in log file.

Note: In Real-Time log msgs will be available in linux machines.

=> To get log msgs from log file we have several tools

                        1) Putty / MobaXterm
                        2) WinScp

                        3) ELK
                        4) Splunk

==========
Summary
==========

1) What is Logging
2) Why Logging
3) Logging Architecture
                - Logger
                - Layout
                - Appender
4) Log Levels
5) Logback
6) Logging in Spring Boot
7) Rolling Policy
8) SLF4J vs Log4J
9) Log Monitoring
                - Putty/WinScp
                - ELK
                - Splunk
```