By Mohamed Taher Othmen

-Logging Library Python-

What's a Logging Library ??

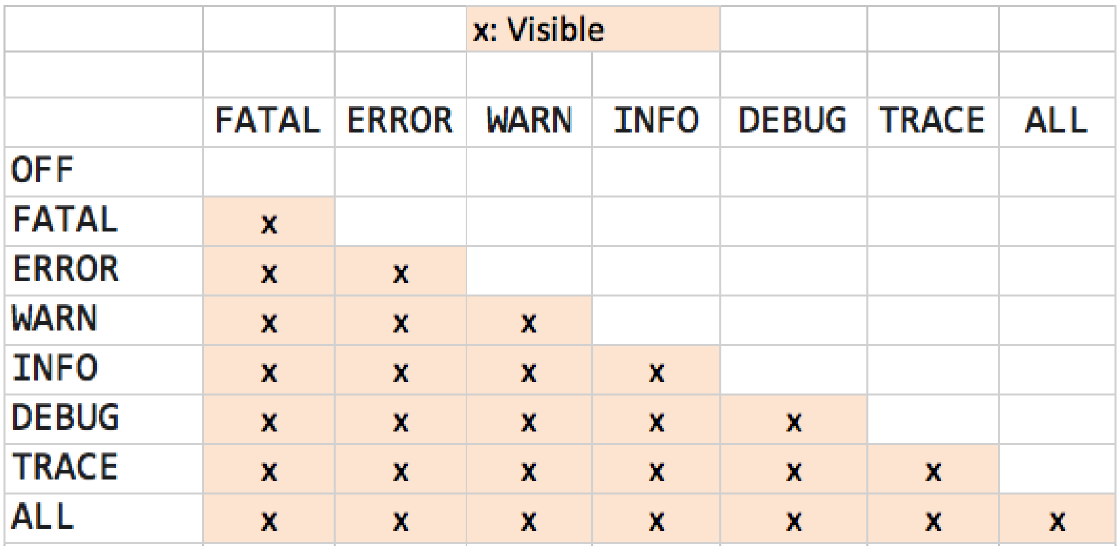
- Imagine it as a journal where your program writes down important events like errors, warning, or just info you want to see while your code is doing his job

What are Logging levels??

- logging messages use different levels based upon severity :
  1. debug: for very detailed output for diagnostic purposes
  2. info: General information about the program's operation.
  3. warning: Indicates a potential issue or something unexpected that doesn't necessarily stop the program.
     logging.warn(), logging.warning()

     `warnings.warn()`
     - **When to Use**: If the code can change something to avoid a problem.
     - **Purpose**: Alerts and suggests changes.
     - **Example**: "Hey, if you do this differently, you might avoid a problem!"

     `logging.warning()`
     - **When to Use:** When the client can't do anything about the situation.
     - **Purpose:** Records an event for reference or investigation.
     - **Example:** "It's out of your control."
  4. error: indicates a more serious issue that might need attention.
     logging.error(), logging.exception()
  5. critical: A critical error that might cause the program to stop working.
     logging.critical()



			x: Visible				
	FATAL	ERROR	WARN	INFO	DEBUG	TRACE	ALL
OFF							
FATAL	x						
ERROR	x	x					
WARN	x	x	x				
INFO	x	x	x	x			
DEBUG	x	x	x	x	x		
TRACE	x	x	x	x	x	x	
ALL	x	x	x	x	x	x	x

Code :

```
import logging

# By Default logging output prompted in Terminal
# Sometimes you want your logging output in a spific log file :
logging.basicConfig(
      filename='/home/taher/Python101/logging/logs.log',
      level=logging.DEBUG
)
logging.debug('This is a debug message')
logging.info('This is an info message')
logging.warning('This a warning message')
logging.error('This is an error message')
logging.critical('This is a critical message')
```

- You can Log variable content to understand what's happening :
  Code:

```
logging.info(f'The value of the Sum variable is {sum}')
```

How to customize Log Messages:

- When you call `basicConfig` multiple times, the subsequent calls don't modify the configuration, as it's already been initialized by the first call.
  Add to the previous code :

```
logging.basicConfig(
   filename='/home/taher/Python101/logging/logs.log',
   format='%(asctime)s - %(name)s - %(levelname)s - %(message)s',
   level=logging.DEBUG
) Content of log.log file :
2023-12-05 00:53:49,069 - root - DEBUG - This is a debug message
2023-12-05 00:53:49,069 - root - INFO - This is an info message
2023-12-05 00:53:49,069 - root - WARNING - This a warning message
2023-12-05 00:53:49,069 - root - ERROR - This is an error message
2023-12-05 00:53:49,069 - root - CRITICAL - This is a critical
message
```

- In the logging module, when we don't specify a logger name, the root logger is used by default. The root logger is at the top of the logger hierarchy and it's the parent of all loggers.
- You can create a custom logger using 'logging.getLogger('logger_name')' and don't forget to assign it to a variable that you'll use in logging later.
  Code :

```
import logging

# define a Logger called Thoura
logger = logging.getLogger('Tahoura')
logging.basicConfig(
   filename='/home/taher/Python101/logging/logs.log',
   format='%(asctime)s - %(name)s - %(levelname)s - %(message)s',
   level=logging.DEBUG
)
# use it to log messages
logger.debug('This is a debug message')
logger.info('This is an info message')
logger.warning('This a warning message')
logger.error('This is an error message')
logger.critical('This is a critical message
```

Libraries:

- var = getattr(object, attribute, default)
- The getattr() function returns the value of the value of the specified attribute from the specified object."default" parameter to write a message when the attribute does not exist
- x = isinstance(object, type) The `isinstance()` function returns `True` if the specified object is of the specified type, otherwise `False`.

What are Loggers??

- Loggers are objects that create log messages. They're like labs for different parts of your code.
- How to create it :

```
logger = logging.getLogger(__name__)
```

   - `__name__` is a built-in variable in Python that contains the name of the current module.
   - when Python runs a script directly (not as an imported module), `__name__` is set to `'__main__'`. otherwise is set to the module's name to help us in identifying the source of the log

```
logger = logging.getLogger(__file__)
```

   - `__file__` is a built-in variable in Python that contains file's absolute adresse.
- `newLogger = logging.getlogger()` this line of code creates a new Logger with the specified name if it is provided, or **root** it not.
- `Logger.setLevel()` specifies lowest-severity log message a logger will handle
- `Logger.addHandler()` adds a handler object to the logger.
- `Logger.removehandler()` removes the handler object from the logger.

- `Logger.addFilter()` adds a filter object to the logger.
- `Logger.removeFilter()` removes the filter object from the logger.
- `Logger.debug(), Logger.info(), Logger.warning(), Logger.error(), Logger.critical().` all create log records with a message and level of severity. The message is actually a format string, which may contain the standard string substitution syntax of `%s`, `%d`, `%f`, and so on.
- `Logger.exception()` we call this method only for an exception. The difference between it and `Logger.error()` is that it dumps a stack trace along with it.
- `Logger.log()` we call this method to log at custom log level with our custom message

  Code : `logger.log(1, 'Taher')`

What are Handlers??

- handlers are responsible for sending log messages. They determine where log messages go based on something (example : severity level)
- handlers are attached to loggers
- How to create it :

```
# send it to the console :
newHandler1 = logging.StreamHandler()
# send it to a specific file :
newHandler2 = logging.FileHandler('fileadresse')
# Let's play something :
handler =
logging.FileHandler('/home/taher/Python101/logging/ad.log',
mode='w', encoding='utf-8')
```

  - Our new handler is called new handler but it's missing something. we must add the level of severity that our handler will control and to attach it to the logger we've just created
- `handlerName.setLevel(logging.<lvl>)` specifies lowest-severity log message
- `handlerName.setFormatter()` specifies lowest-severity log message
- `handlerName.addFilter()` specifies lowest-severity log message
- `handlerName.removeFilter()` specifies lowest-severity log message

What are Formatters??

- Formatter objects configure the final order, structure, and contents of the log message. Simply, it defines how log messages look in the final output.
- How to create it :

```
formatter = logging.Formatter('example')
formatter = logging.Formatter('%(asctime)s - %(name)s - %(levelname)s - %(message)s')
```

- we must to attach it to the handler we've just created

  Code : `handlerName.setFormatter(formatterName)`