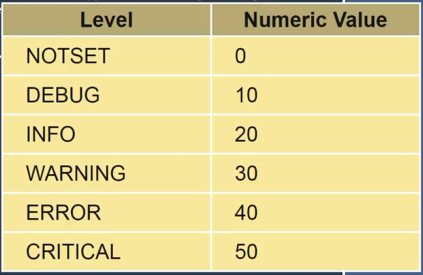
Logging

* Loggers will only write messages with the level greater than or equal to set level.
* Default logging level is warning (30)
* You can use the following formats to include in your log messages:
  + %(levelname)s – Name of logger (logging channel)
  + %(asctime)s – Textual time when the log record was created
  + %(message)s – the result of record.getMessage(), computed just as the record is emitted.
* By default log files are created in **append** mode.



* DURGA: You use **logging** in your daily life. Prior to going in any library there is a log book wherein you are logging your information in it, namely you write/log your name, phone number, the time your arrived and the time your left libarary. This log file is important you can extract useful information like keeping track of number of uers of library, what times are rush hours, what is the average time library users are spending their time in and many more …

Now, that log book is very important to keep track of any thing that goes wrong in that libarary. For example assuming a keyboard of computer on cabine number #5 is missing from library on 8 August. You can search your log file at date 8 August who used computer in that cabine and then you can do your investigation.

Another example, you’re working in a company as server maintainer/ engineer and all of a sudden server goes down for five minutes from 10:30 to 10:35.

The CEO of the company asks you to prepare a report that why servers went down. If you have a log file that keeps track of the status and progress of server then it would be much easier for you to spot the problem and prepare a RCA report (root cause analysis report) of this problem and how to make temporary and permanent solution for this problem in future.

* Advantages of logging:
  + to perform debugging and to prepare RCA
  + To provide statistics like the number of requests per day
* Handler**:** Handlers send the LogRecord to the output destination (console or a file) . Handler is a base for subclasses like StreamHandler, FileHandler, SMTPHandler, HTTPHandler, and more. These subclasses send the logging outputs to corresponding destinations (console, file)
* to disable logging run logging.disable(logging.CRITICAL), within this parenthesis you can pass a logging level and it will supress all log messages at that level or lower.

# Socratica

import logging

LOG\_FORMAT = "%(levelname)s %(asctime)s - %(message)s"

logging.basicConfig(filename='log\_file.log',

level=logging.DEBUG,

format=LOG\_FORMAT,

filemode='w')

# getLogger() method to create a logger object

logger = logging.getLogger()

# print(logger.level)

logger.debug("Info message logged.")

logger.info("Info message logged.")

logger.warning("Info message logged.")

logger.error("Info message logged.")

logger.critical("Info message logged.")

import logging

import math

logging.basicConfig(filename='quadratic\_formula.log',

format="%(levelname)s %(asctime)s - %(message)s ",

level=logging.DEBUG,

filemode='w')

logger = logging.getLogger()

def quadratic\_formula(a, b, c):

""" Quadratic formula """

logger.info("quadratic\_formula({0}, {1}, {2})".format(a, b, c))

logger.debug('Calculate the discriminant')

disc = b\*\*2 - 4\*a\*c

logger.debug('Calculate the roots')

root1 = (-b + math.sqrt(disc)) / 2\*a

root2 = (-b - math.sqrt(disc)) / 2\*a

logger.debug("Output")

return (root1, root2)

roots = quadratic\_formula(1, 0, -4)

print(roots)

# CoreyMS, First Part

import logging

*#* ***DEBUG****: Detailed information, typically of interest only when diagnosing problems.*

*#* ***INFO****: Confirmation that things are working as expected.*

*#* ***WARNING****: An indication that something unexpected happened, or indicates some problem in the near future (e.g.‘disk space low’) & software is still working as expected.*

*#* ***ERROR****: Due to a serious problem, the software isn’t able to perform some function.*

*#* ***CRITICAL****: A serious error, indicates that the program may be unable to run.*

*# this logging.basicConfig() is configuring the root level.*

logging.basicConfig(filename='test.log', level=logging.DEBUG,

format='%(asctime)s:%(levelname)s:%(message)s')

def add(x, y):

"""Add Function"""

return x + y

def subtract(x, y):

"""Subtract Function"""

return x - y

def multiply(x, y):

"""Multiply Function"""

return x \* y

def divide(x, y):

"""Divide Function"""

return x / y

num\_1 = 20

num\_2 = 10

add\_result = add(num\_1, num\_2)

logging.debug('Add: {} + {} = {}'.format(num\_1, num\_2, add\_result))

sub\_result = subtract(num\_1, num\_2)

logging.debug('Sub: {} - {} = {}'.format(num\_1, num\_2, sub\_result))

mul\_result = multiply(num\_1, num\_2)

logging.debug('Mul: {} \* {} = {}'.format(num\_1, num\_2, mul\_result))

div\_result = divide(num\_1, num\_2)

logging.debug('Div: {} / {} = {}'.format(num\_1, num\_2, div\_result))

import logging

logging.basicConfig(filename='employee.log',

format="%(asctime)s:%(levelname)s:%(message)s",

filemode='w',

level=logging.INFO)

class employee:

def \_\_init\_\_(self, firstname, lastname):

self.firstname = firstname

self.lastname = lastname

logging.info("Created Employee: {0} - {1}".format(self.fullname, self.email))

@property

def fullname(self):

return "{0} {1}".format(self.firstname, self.lastname)

@property

def email(self):

return "{0}{1}@gmail.com".format(self.firstname, self.lastname)

emp\_1 = employee('John', 'Doe')

emp\_2 = employee('Mike', 'Smith')

emp\_3 = employee('Marry', 'Doe')

Since we haven’t specified a specific logger we are working with root logger, but it is a good habit to work with specific loggers that can be logged separately.

Shell:

DEBUG:root:2019-08-23 13:43:14,293 - Debug message logged.

INFO:root:2019-08-23 13:43:14,294 - Info message logged.

WARNING:root:2019-08-23 13:43:14,294 - Warning message logged.

**ERROR**:root:2019-08-23 13:43:14,294 - **Error** message logged.

**CRITICAL**:root:2019-08-23 13:43:14,294 - Critical message logged.

**2nd Part - Advanced Logging**

# Creating a separate logger:

import logging

logger = logging.getLogger(\_\_name\_\_) # creating logging object

logger.setLevel(logging.INFO)

*# FileHandler() outputs log records to* ***employee.log***

file\_handler = logging.FileHandler('employee.log')

format = logging.Formatter("%(asctime)s:%(name)s:%(message)s")

file\_handler.setFormatter(format) *# we are adding the format to file handler and not logger*

logger.addHandler(file\_handler)

# logging.basicConfig(filename='calculation.log', level=logging.DEBUG,

# format="%(asctime)s:%(name)s:%(message)s")

class employee:

def \_\_init\_\_(self, firstname, lastname):

self.firstname = firstname

self.lastname = lastname

logger.info("Created Employee: {0} - {1}".format(

self.fullname, self.email))

@property

def fullname(self):

return "{0} {1}".format(self.firstname, self.lastname)

@property

def email(self):

return "{0}{1}@gmail.com".format(self.firstname, self.lastname)

emp\_1 = employee('John', 'Doe')

emp\_2 = employee('Mike', 'Smith')

emp\_3 = employee('Marry', 'Doe')

## Log only errors in file

import logging

logger = logging.getLogger(\_\_name\_\_)

logger.setLevel(logging.WARNING)

file\_handler = logging.FileHandler('new\_file.log')

format = logging.Formatter('%(levelname)s:%(name)s:%(asctime)s:%(message)s')

# we want only errors and worst to be logged in our file!

file\_handler.setLevel(logging.ERROR)

file\_handler.setFormatter(format)

logger.addHandler(file\_handler)

def division(a, b):

try:

result = a / b

except ZeroDivisionError:

logger.error('Cannot divide by zero.')

# logger.exception('Cannot divide by zero.') # to log the traceback of error

else:

return result

logger.debug(division(10,0))

## Stream Handler: to print the output in the screen rather than file

import logging

import employee

logger = logging.getLogger(\_\_name\_\_) # creating an logging object

logger.setLevel(logging.DEBUG)

file\_handler = logging.FileHandler('calculation.log')

file\_handler.setLevel(logging.ERROR) # log only errors in 'calculation.log' file

format = logging.Formatter("%(asctime)s:%(name)s:%(message)s")

file\_handler.setFormatter(format)

logger.addHandler(file\_handler)

stream\_handler = logging.StreamHandler()

stream\_handler.setFormatter(format)

logger.addHandler(stream\_handler)

def add(a, b):

return a+b

def subtract(a, b):

return a-b

def multiply(a, b):

return a\*b

def divide(a, b):

try:

result = a/b

except ZeroDivisionError:

logger.error('Tried to divide by zero')

else:

return result

a = 10

b = 0

logger.debug('Add: {0} + {1} = {2} '.format(a, b, add(a, b)))

logger.debug('Subtract: {0} - {1} = {2} '.format(a, b, subtract(a, b)))

logger.debug('Multiply: {0} \* {1} = {2} '.format(a, b, multiply(a, b)))

logger.debug('Divide: {0} / {1} = {2} '.format(a, b, divide(a, b)))

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

import logging

logging.basicConfig()

logging.debug('This is a debug message.')

logging.info('This is an info message.')

logging.warning('This is an warning message.')

logging.error('This is an error message.')

logging.critical('This is a critical message.')

# WARNING:root:This is an warning message.

# ERROR:root:This is an error message.

# CRITICAL:root:This is a critical message.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

