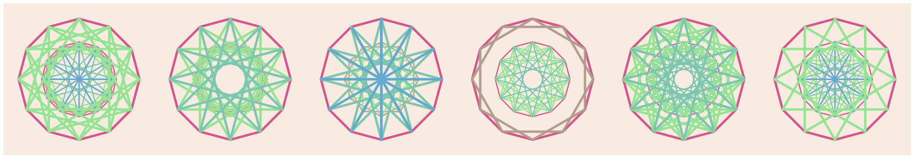
# Exercise 7 for Students of Computer Science

Leon Tabak
l.tabak@ieee.org

11 August 2021



```
# try-logging.py
# Leon Tabak
# l.tabak@ieee.org
# 11 August 2021

# This program is an example and an exercise with which to introduce
# students to Python's logging facility.

import logging

def fun():
    logging.debug( 'I am fun. I will call my friend more fun.' )
    logging.info( 'We are having fun.' )
    more_fun()
# end of fun()

def more_fun():
    logging.debug( 'I am more fun. I will call my friend most fun.' )
    logging.info( 'We are having more_fun.' )
    most_fun()
# end of more_fun()

def most_fun():
    logging.debug( 'I am most fun. You will not find a better friend!' )
    logging.info( 'We are having the most_fun.' )
# end of most_fun()

def try_logging():
    # Call logging.basicConfig() just once.
    # Call it before calling any other logging functions.

    # Specify what kind of messages to print
    # by assigning a value to the level parameter.
    # The choices are:
    #     * logging.DEBUG
    #     * logging.INFO
    #     * logging.WARNING
    #     * logging.ERROR
    #     * logging.CRITICAL
    # Specify the format of the message by assigning
    # a format string to the format parameter.
    # This format string can contain a reference to:
    #     * funcName (Name of function that logged the message.)
    #     * level (This means DEBUG, INFO, WARNING, etc.)
    #     * lineno (Number of the line in the program.)
    #     * message (The content of the log entry.)
    # These variables (funcName, level, and so on) are enclosed
    # in parentheses. A percent sign precedes the left parenthesis.
    # A formatting code ('s' for string, 'd' for decimal integer)
    # follows the right parenthesis.
    # The formatting code can optionally include a number.
    # For example, '16s' means allow 16 characters for a string
    # and '8d' means allow 8 digits for an integer.
    formatString = ('%(levelname)s message in %(funcName)s() ' +
        'on line# %(lineno)d: \n\t%(message)s\n' )
    logging.basicConfig( level = logging.DEBUG,
        format = formatString )

    # TO-DO: Try setting the level in the call to
    # logging.basicConfig() to something other than
    # logging.DEBUG

    # TO-DO: Try constructing a different formatString
    # to assign to the format parameter in the call to
    # logging.basicConfig(). For example, you might choose
    # to leave out the level or put the line number first
    # or fix the width of each part of the log entry.
    # Fixing the width might mean specifying 8 characters
    # for the level or 5 digits for the line number.

    # TO-DO: Learn how to use Python's assert statement
    # by searching on the Web for explanations and examples.
    # Explain how you might use assert statements with
    # logging. Could you use assert statements in place
    # of logging? What do the experts on the Web say?

    # TO-DO: Learn more by reading the documentation
    # that you will find here:
    #    https://docs.python.org/3/library/logging.html

    # TO-DO: Add logging to a program that you wrote
    # as part of another exercise.

    # TO-DO: Learn more by reading the documentation
    # that you will find here:
    #    https://docs.python.org/3/library/logging.html


    logging.debug( 'Here is help for debugging.' )
    logging.info( 'Here is some helpful information.' )
    logging.warning( 'I will give you just a warning this time.' )
    logging.error( 'I cannot let this one go. You have made an error.' )
    logging.critical( 'Now you have done it! Your error is critical.' )

    fun()
# end of try_logging()

if __name__ == '__main__':
    try_logging()
```