

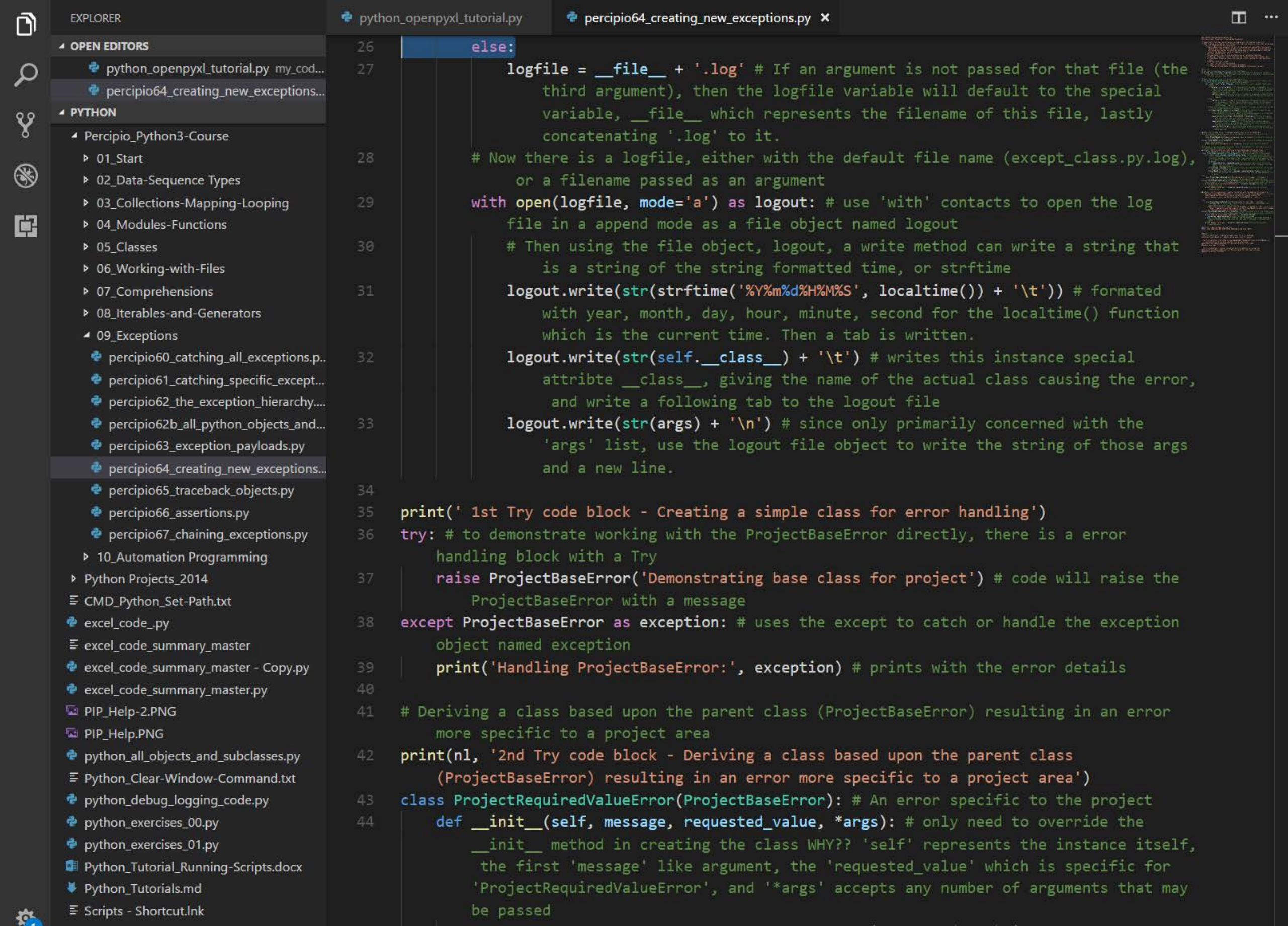


EXPLORER


```
'''
percipio64_creating_new_exceptions.py
Percipio video: Exceptions; Creating New Exceptions

* Demonstrate creating a new Exception so program will not end due to an error
* When creating an exception, it's Pythonic to create an Exception class that can be used
    for all exceptions within the project.
    * Create a Base class exception for all of the exceptions used within the project,
        beng able to subclass off that base class to use for specific errors, and with
        all exceptions (parent and children) being able to take advantage of anything
        done in the base class, such as logging to a file.
    * The class should end in 'Error'
    * This way features can be incorporated across all exception handling
    * The general 'Exception' object should be sub-classed within the created class.
    * In creating an Exception class, override the __init__ method for that particular
        class
    * Exception class will accept:
        * 'self' to refer to the instance
        * *args for any number of positional arguements
        * **kwargs for any number of keyword arguements (pronounced k-w-args)
'''
nl = '\n'
# Use time to timestamp log files with the time module
from time import strftime, localtime # import the Time module to be able to format &
    obtain the current time. The strftime() function formats a time object. The localtime
    () function returns a time object

# Base class for all errors in project code logs errors
class ProjectBaseError(Exception): # Being a generic base class for other classes, it's
    Pythonic to not name any specific positional arguments
    def __init__(self, *args, **kwargs): # overriding the __init__ method, *args, and
        *kwargs
        if len(args) > 2 and args[2]: # If the length of the number of arguments is
            greater than 2, and, there is a third argument that has a true value (args[2])
            , then the variable logfile will be set to the args of 2 (args[2]) (the 3rd
            argument in the tuple of args)
            logfile = args[2] # This will allow a filename to be passed instead of using
                a default file, allowing a user to specify the file that will be used for
                logging.
        else:
```

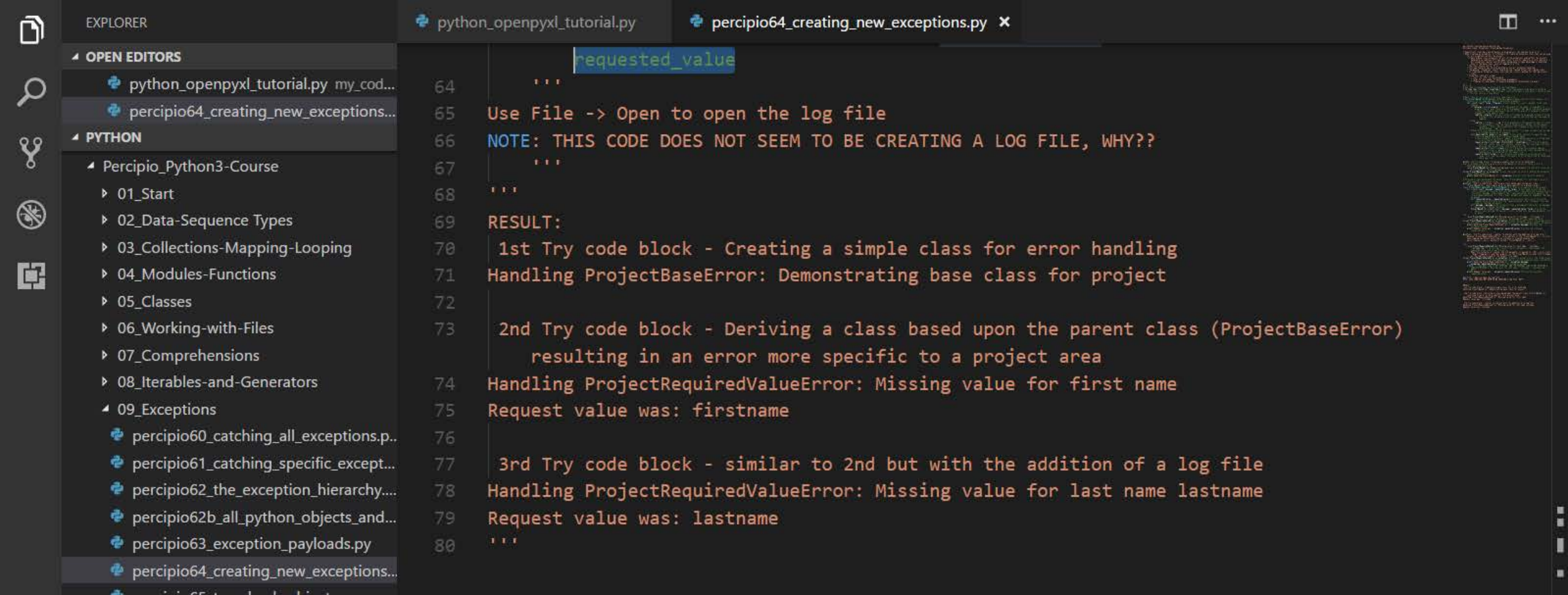

EXPLORER
python_openpyxl_tutorial.py
percipio64_creating_new_exceptions.py
OPEN EDITORS
python_openpyxl_tutorial.py my_cod...
percipio64_creating_new_exceptions...
PYTHON
Percipio_Python3-Course
01_Start
02_Data-Sequence Types
03_Collections-Mapping-Looping
04_Modules-Functions
05_Classes
06_Working-with-Files
07_Comprehensions
08_Iterables-and-Generators
09_Exceptions
percipio60_catching_all_exceptions.p...
percipio61_catching_specific_except...
percipio62_the_exception_hierarchy....
percipio62b_all_python_objects_and...
percipio63_exception_payloads.py
percipio64_creating_new_exceptions...
percipio65_traceback_objects.py
percipio66_assertions.py
percipio67_chaining_exceptions.py
10_Automation Programming
Python Projects_2014
CMD_Python_Set-Path.txt
excel_code_.py
excel_code_summary_master
excel_code_summary_master - Copy.py
excel_code_summary_master.py
PIP_Help-2.PNG
PIP_Help.PNG
python_all_objects_and_subclasses.py
Python_Clear-Window-Command.txt
python_debug_logging_code.py
python_exercises_00.py
python_exercises_01.py
Python_Tutorial_Running-Scripts.docx
Python_Tutorials.md
Scripts - Shortcut.lnk
26 else:
27 logfile = __file__ + '.log' # If an argument is not passed for that file (the third argument), then the logfile variable will default to the special variable, __file__ which represents the filename of this file, lastly concatenating '.log' to it.
28 # Now there is a logfile, either with the default file name (except_class.py.log), or a filename passed as an argument
29 with open(logfile, mode='a') as logout: # use 'with' contacts to open the log file in a append mode as a file object named logout
30 # Then using the file object, logout, a write method can write a string that is a string of the string formatted time, or strftime
31 logout.write(str(strftime('%Y%m%d%H%M%S', localtime()) + '\t')) # formated with year, month, day, hour, minute, second for the localtime() function which is the current time. Then a tab is written.
32 logout.write(str(self.__class__) + '\t') # writes this instance special attribte __class__, giving the name of the actual class causing the error, and write a following tab to the logout file
33 logout.write(str(args) + '\n') # since only primarily concerned with the 'args' list, use the logout file object to write the string of those args and a new line.
34
35 print(' 1st Try code block - Creating a simple class for error handling')
36 try: # to demonstrate working with the ProjectBaseError directly, there is a error handling block with a Try
37 raise ProjectBaseError('Demonstrating base class for project') # code will raise the ProjectBaseError with a message
38 except ProjectBaseError as exception: # uses the except to catch or handle the exception object named exception
39 print('Handling ProjectBaseError:', exception) # prints with the error details
40
41 # Deriving a class based upon the parent class (ProjectBaseError) resulting in an error more specific to a project area
42 print(nl, '2nd Try code block - Deriving a class based upon the parent class (ProjectBaseError) resulting in an error more specific to a project area')
43 class ProjectRequiredValueError(ProjectBaseError): # An error specific to the project
44 def __init__(self, message, requested_value, *args): # only need to override the __init__ method in creating the class WHY?? 'self' represents the instance itself, the first 'message' like argument, the 'requested_value' which is specific for 'ProjectRequiredValueError', and '*args' accepts any number of arguments that may be passed





EXPLORER


```
            be passed
45          self.requested_value = requested_value # the instance (self) itsef stores the
                request_value attribute with the requested_value (after the equal) that was
                used in creating the object
46          self.message = message # the instance stores a message attribute for the message
                passed when creating the object
47          ProjectBaseError.__init__(self, message, requested_value, *args) # using the
                parent class calling its special __init__ method with the same parameter list
                as above
48  try:
49      raise ProjectRequiredValueError('Missing value for first name', 'firstname') #
            raising this error with 2 parameters, the message and the requested_value
50  except ProjectRequiredValueError as exception: # when the except occurs, the except block
        for the specific error (ProjectRequiredValueError) as the exception object (exception)
51      print('Handling ProjectRequiredValueError:', exception.message) # access the
            instances message
52      print('Request value was:', exception.requested_value) # access the instances
            requested_value
53
54  print(nl, '3rd Try code block - similar to 2nd but with the addition of a log file')
55  ''' Because 'ProjectRequiredValueError' is a subclass of the 'ProjectBaseError', it still
        does logging operations each time an instance is created because the
        parent-special-__init__ method is called ('ProjectBaseError.__init__')
56  '''
57  try:
58      raise ProjectRequiredValueError('Missing value for last name', 'lastname',
            'lastname.log') #  # raising this error with 3 parameters, the message, the
            requested_value, and a log-file-entry.
59      ''' When this instance is created, the 3rd parameter is passed in ('def __init__(self,
             *args, **kwargs)'), and the 'args[2]' in the ProjectBaseError uses this filename
            as the logfile instead of the default file name. '''
60  except ProjectRequiredValueError as exception: # when the except occurs, the except block
        for the specific error (ProjectRequiredValueError) as the exception object (exception)
61      print('Handling ProjectRequiredValueError:', exception.message,
            exception.requested_value) # access the instances message
62      # NOTE: ?? THIS NEXT LINE IS NOT PRINTING, AND THE PREVIOUS LINE IS NOT PRINTING A
            COMPLETE MESSAGE (('Missing value for last name', 'lastname', 'lastname.log'))
            WHY??
63      print('Request value was:', exception.requested_value) # access the instances
            requested_value
```


EXPLORER
python_openpyxl_tutorial.py
percipio64_creating_new_exceptions.py
OPEN EDITORS
python_openpyxl_tutorial.py my_cod...
percipio64_creating_new_exceptions...
PYTHON
Percipio_Python3-Course
01_Start
02_Data-Sequence Types
03_Collections-Mapping-Looping
04_Modules-Functions
05_Classes
06_Working-with-Files
07_Comprehensions
08_Iterables-and-Generators
09_Exceptions
percipio60_catching_all_exceptions.p...
percipio61_catching_specific_except...
percipio62_the_exception_hierarchy....
percipio62b_all_python_objects_and...
percipio63_exception_payloads.py
percipio64_creating_new_exceptions...
requested_value
'''
Use File -> Open to open the log file
NOTE: THIS CODE DOES NOT SEEM TO BE CREATING A LOG FILE, WHY??
'''
'''
RESULT:
1st Try code block - Creating a simple class for error handling
Handling ProjectBaseError: Demonstrating base class for project
2nd Try code block - Deriving a class based upon the parent class (ProjectBaseError)
resulting in an error more specific to a project area
Handling ProjectRequiredValueError: Missing value for first name
Request value was: firstname
3rd Try code block - similar to 2nd but with the addition of a log file
Handling ProjectRequiredValueError: Missing value for last name lastname
Request value was: lastname
'''