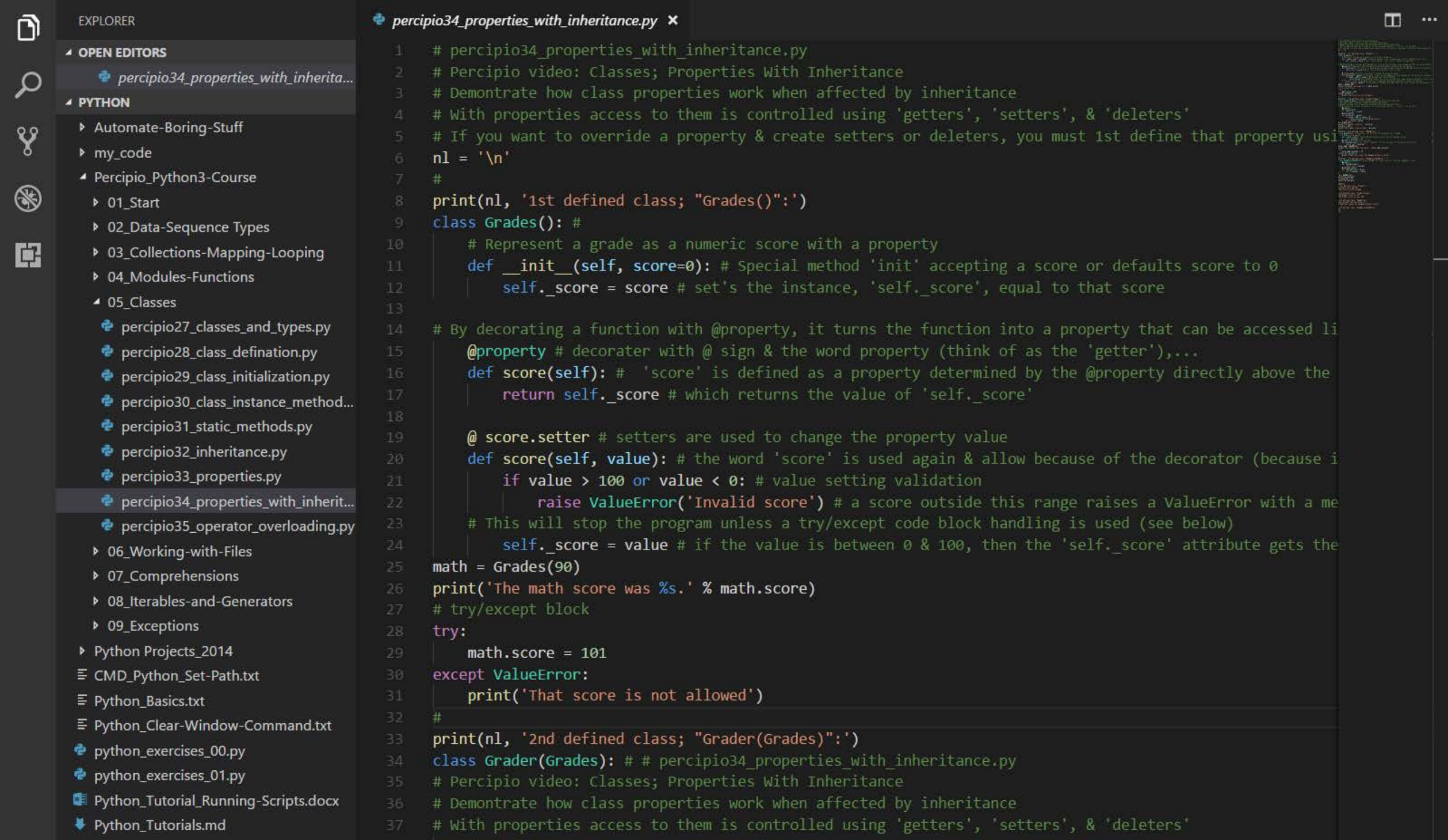
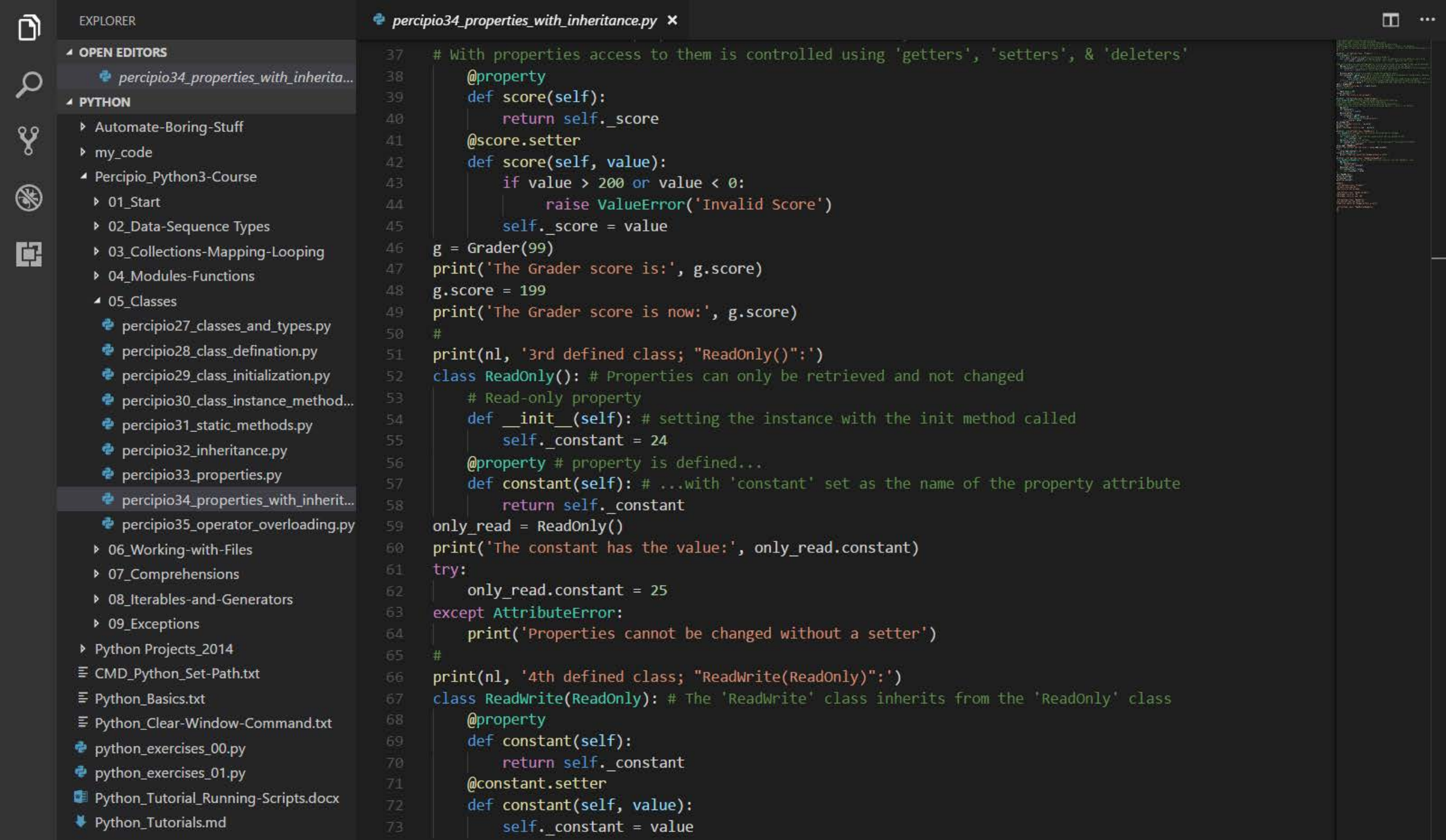
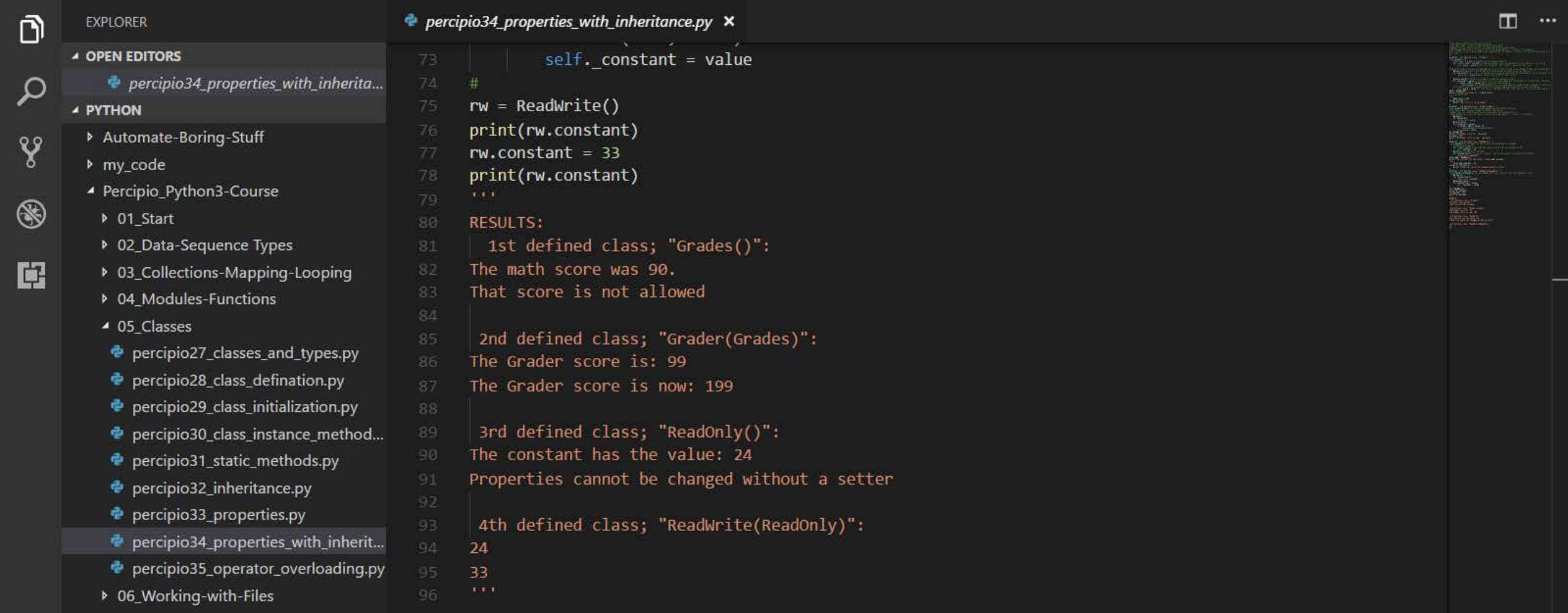

EXPLORER
OPEN EDITORS
percipio34_properties_with_inherita...
PYTHON
Automate-Boring-Stuff
my_code
Percipio_Python3-Course
01_Start
02_Data-Sequence Types
03_Collections-Mapping-Looping
04_Modules-Functions
05_Classes
percipio27_classes_and_types.py
percipio28_class_defination.py
percipio29_class_initialization.py
percipio30_class_instance_method...
percipio31_static_methods.py
percipio32_inheritance.py
percipio33_properties.py
percipio34_properties_with_inherit...
percipio35_operator_overloading.py
06_Working-with-Files
07_Comprehensions
08_Iterables-and-Generators
09_Exceptions
Python Projects_2014
CMD_Python_Set-Path.txt
Python_Basics.txt
Python_Clear-Window-Command.txt
python_exercises_00.py
python_exercises_01.py
Python_Tutorial_Running-Scripts.docx
Python_Tutorials.md
percipio34_properties_with_inheritance.py
# percipio34_properties_with_inheritance.py
# Percipio video: Classes; Properties With Inheritance
# Demontrate how class properties work when affected by inheritance
# With properties access to them is controlled using 'getters', 'setters', & 'deleters'
# If you want to override a property & create setters or deleters, you must 1st define that property usi
nl = '\n'
#
print(nl, '1st defined class; "Grades()":')
class Grades(): #
Represent a grade as a numeric score with a property
def __init__(self, score=0): # Special method 'init' accepting a score or defaults score to 0
self._score = score # set's the instance, 'self._score', equal to that score
# By decorating a function with @property, it turns the function into a property that can be accessed li
@property # decorater with @ sign & the word property (think of as the 'getter'),...
def score(self): # 'score' is defined as a property determined by the @property directly above the
return self._score # which returns the value of 'self._score'
@ score.setter # setters are used to change the property value
def score(self, value): # the word 'score' is used again & allow because of the decorator (because i
if value > 100 or value < 0: # value setting validation
raise ValueError('Invalid score') # a score outside this range raises a ValueError with a me
# This will stop the program unless a try/except code block handling is used (see below)
self._score = value # if the value is between 0 & 100, then the 'self._score' attribute gets the
math = Grades(90)
print('The math score was %s.' % math.score)
# try/except block
try:
math.score = 101
except ValueError:
print('That score is not allowed')
#
print(nl, '2nd defined class; "Grader(Grades)":')
class Grader(Grades): # # percipio34_properties_with_inheritance.py
Percipio video: Classes; Properties With Inheritance
Demontrate how class properties work when affected by inheritance
With properties access to them is controlled using 'getters', 'setters', & 'deleters'


EXPLORER
percipio34_properties_with_inheritance.py
OPEN EDITORS
percipio34_properties_with_inherita...
PYTHON
Automate-Boring-Stuff
my_code
Percipio_Python3-Course
01_Start
02_Data-Sequence Types
03_Collections-Mapping-Looping
04_Modules-Functions
05_Classes
percipio27_classes_and_types.py
percipio28_class_defination.py
percipio29_class_initialization.py
percipio30_class_instance_method...
percipio31_static_methods.py
percipio32_inheritance.py
percipio33_properties.py
percipio34_properties_with_inherit...
percipio35_operator_overloading.py
06_Working-with-Files
07_Comprehensions
08_Iterables-and-Generators
09_Exceptions
Python Projects_2014
CMD_Python_Set-Path.txt
Python_Basics.txt
Python_Clear-Window-Command.txt
python_exercises_00.py
python_exercises_01.py
Python_Tutorial_Running-Scripts.docx
Python_Tutorials.md
37 # With properties access to them is controlled using 'getters', 'setters', & 'deleters'
38 @property
39 def score(self):
40 return self._score
41 @score.setter
42 def score(self, value):
43 if value > 200 or value < 0:
44 raise ValueError('Invalid Score')
45 self._score = value
46 g = Grader(99)
47 print('The Grader score is:', g.score)
48 g.score = 199
49 print('The Grader score is now:', g.score)
50 #
51 print(nl, '3rd defined class; "ReadOnly()":')
52 class ReadOnly(): # Properties can only be retrieved and not changed
53 # Read-only property
54 def __init__(self): # setting the instance with the init method called
55 self._constant = 24
56 @property # property is defined...
57 def constant(self): # ...with 'constant' set as the name of the property attribute
58 return self._constant
59 only_read = ReadOnly()
60 print('The constant has the value:', only_read.constant)
61 try:
62 only_read.constant = 25
63 except AttributeError:
64 print('Properties cannot be changed without a setter')
65 #
66 print(nl, '4th defined class; "ReadWrite(ReadOnly)":')
67 class ReadWrite(ReadOnly): # The 'ReadWrite' class inherits from the 'ReadOnly' class
68 @property
69 def constant(self):
70 return self._constant
71 @constant.setter
72 def constant(self, value):
73 self._constant = value


EXPLORER
percipio34_properties_with_inheritance.py
OPEN EDITORS
percipio34_properties_with_inherita...
PYTHON
Automate-Boring-Stuff
my_code
Percipio_Python3-Course
01_Start
02_Data-Sequence Types
03_Collections-Mapping-Looping
04_Modules-Functions
05_Classes
percipio27_classes_and_types.py
percipio28_class_defination.py
percipio29_class_initialization.py
percipio30_class_instance_method...
percipio31_static_methods.py
percipio32_inheritance.py
percipio33_properties.py
percipio34_properties_with_inherit...
percipio35_operator_overloading.py
06_Working-with-Files
self._constant = value
#
rw = ReadWrite()
print(rw.constant)
rw.constant = 33
print(rw.constant)
'''
RESULTS:
1st defined class; "Grades()":
The math score was 90.
That score is not allowed
2nd defined class; "Grader(Grades)":
The Grader score is: 99
The Grader score is now: 199
3rd defined class; "ReadOnly()":
The constant has the value: 24
Properties cannot be changed without a setter
4th defined class; "ReadWrite(ReadOnly)":
24
33
'''