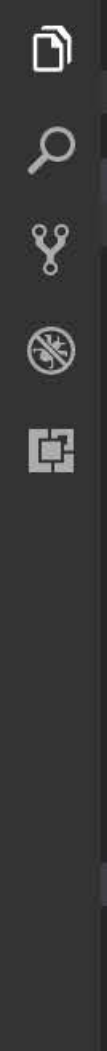


EXPLORER

OPEN EDITORS
- Welcome
- percipio33_properties.py Percipio_Py...

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_methods....
    - percipio31_static_methods.py
    - percipio32_inheritance.py
    - percipio33_properties.py
    - percipio34_properties_with_inherita...
    - 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

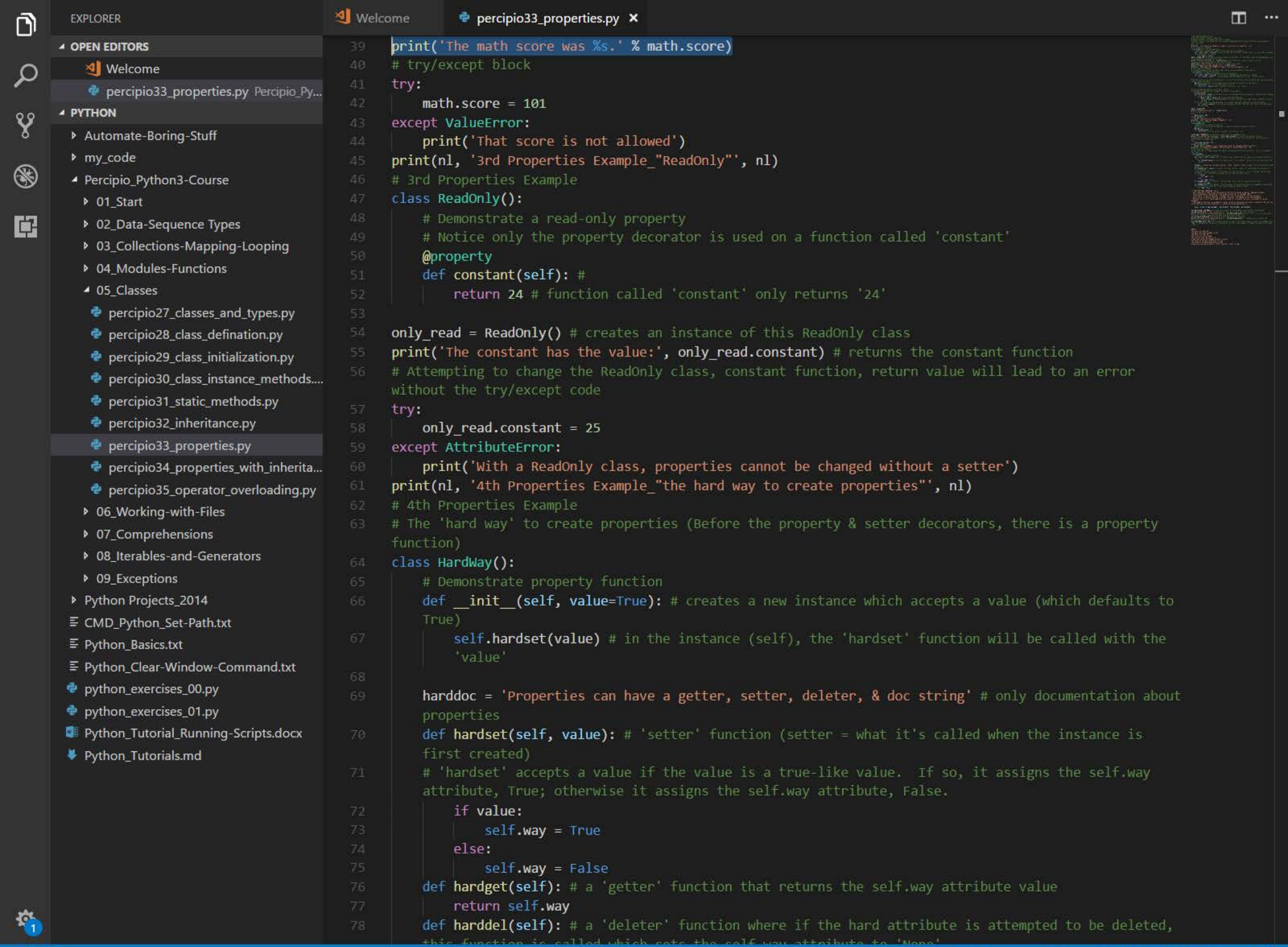
Welcome | percipio33_properties.py

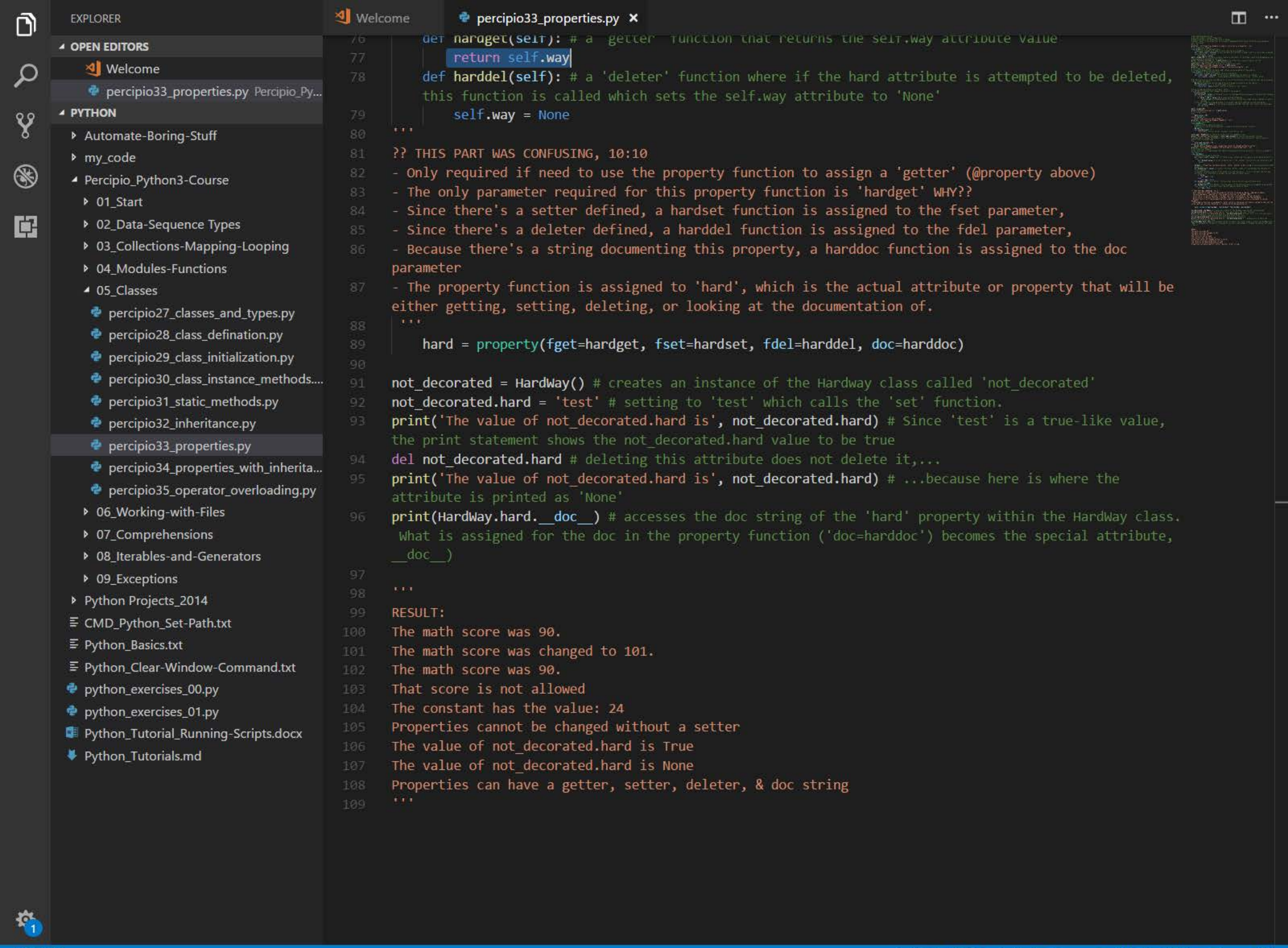
```python
# percipio33_properties.py
# Percipio video: Classes; Properties
# Demontrate how class properties work in Python
# A main reason to use properties is to prevent manipulating attributes directly (& assigning an incorrect value)
nl = '\n'
print(nl, '1st Properties Example_"a numeric score not as a property"', nl)
# 1st Properties Example
class Grader(): # 1st defined class
    # Represent a grade as a numeric score, but not as a property
    def __init__(self, score=0): # this instance will be able to keep track of a score that is passed when the object is created
        self.score = score
math = Grader(90) # This Math instance creates a new object of the Grader class with 90 passed as the score value & is assigned to self.score
print('The math score was %s.' % math.score) # to access this class attribute, use the instance-dot-attribute-name to view the attribute
math.score = 101 # Example of an incorrect property attribute
print('The math score was changed to %s.' % math.score)
print(nl, '2nd Properties Example_"a numeric score as a property"', nl)
# 2nd Properties Example
# A property example that validates what values can be assigned to the property
class Grades(): # 2nd defined class
    # Represent a grade as a numeric score as a property
    def __init__(self, score=0): # The score is passed when the object is created,...
        self._score = score # ...and assigned a private underscore 'score' (_score) value

# By decorating a function with @property, it turns the function into a property that can be accessed like a normal attribute.
    @property # decorater with @ sign & the word property (think of as the 'getter'),...
    def score(self): #  ...a function named 'score',...
        return self._score # which returns the value of 'self._score'

# Also through decoration is setting a 'setter'.
# a SETTER is a method that changes the value of the property
    @ score.setter
    def score(self, value): # the word 'score' is used again & allow because of the decorator (because it's local, not global?)
        if value > 100 or value < 0: # value setting validation
            raise ValueError('Invalid score') # a score outside this range raises a ValueError with a message
    # 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 new value

math = Grades(90)
print('The math score was %s.' % math.score)
# try/except block
```


EXPLORER
OPEN EDITORS
Welcome
percipio33_properties.py Percipio_Py...
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_methods....
percipio31_static_methods.py
percipio32_inheritance.py
percipio33_properties.py
percipio34_properties_with_inherita...
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
Welcome
percipio33_properties.py
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, '3rd Properties Example_"ReadOnly"', nl)
# 3rd Properties Example
class ReadOnly():
# Demonstrate a read-only property
# Notice only the property decorator is used on a function called 'constant'
@property
def constant(self): #
return 24 # function called 'constant' only returns '24'
only_read = ReadOnly() # creates an instance of this ReadOnly class
print('The constant has the value:', only_read.constant) # returns the constant function
# Attempting to change the ReadOnly class, constant function, return value will lead to an error without the try/except code
try:
only_read.constant = 25
except AttributeError:
print('With a ReadOnly class, properties cannot be changed without a setter')
print(nl, '4th Properties Example_"the hard way to create properties"', nl)
# 4th Properties Example
# The 'hard way' to create properties (Before the property & setter decorators, there is a property function)
class HardWay():
# Demonstrate property function
def __init__(self, value=True): # creates a new instance which accepts a value (which defaults to True)
self.hardset(value) # in the instance (self), the 'hardset' function will be called with the 'value'
harddoc = 'Properties can have a getter, setter, deleter, & doc string' # only documentation about properties
def hardset(self, value): # 'setter' function (setter = what it's called when the instance is first created)
# 'hardset' accepts a value if the value is a true-like value. If so, it assigns the self.way attribute, True; otherwise it assigns the self.way attribute, False.
if value:
self.way = True
else:
self.way = False
def hardget(self): # a 'getter' function that returns the self.way attribute value
return self.way
def harddel(self): # a 'deleter' function where if the hard attribute is attempted to be deleted,
0 0
Ln 39, Col 1 (44 selected)
Spaces: 4
UTF-8
CRLF
Python


EXPLORER
OPEN EDITORS
Welcome
percipio33_properties.py Percipio_Py...
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_methods....
percipio31_static_methods.py
percipio32_inheritance.py
percipio33_properties.py
percipio34_properties_with_inherita...
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
Welcome
percipio33_properties.py
def hardget(self): # a 'getter' function that returns the self.way attribute value
return self.way
def harddel(self): # a 'deleter' function where if the hard attribute is attempted to be deleted, this function is called which sets the self.way attribute to 'None'
self.way = None
'''
?? THIS PART WAS CONFUSING, 10:10
- Only required if need to use the property function to assign a 'getter' (@property above)
- The only parameter required for this property function is 'hardget' WHY??
- Since there's a setter defined, a hardset function is assigned to the fset parameter,
- Since there's a deleter defined, a harddel function is assigned to the fdel parameter,
- Because there's a string documenting this property, a harddoc function is assigned to the doc parameter
- The property function is assigned to 'hard', which is the actual attribute or property that will be either getting, setting, deleting, or looking at the documentation of.
'''
hard = property(fget=hardget, fset=hardset, fdel=harddel, doc=harddoc)
not_decorated = HardWay() # creates an instance of the Hardway class called 'not_decorated'
not_decorated.hard = 'test' # setting to 'test' which calls the 'set' function.
print('The value of not_decorated.hard is', not_decorated.hard) # Since 'test' is a true-like value, the print statement shows the not_decorated.hard value to be true
del not_decorated.hard # deleting this attribute does not delete it,...
print('The value of not_decorated.hard is', not_decorated.hard) # ...because here is where the attribute is printed as 'None'
print(HardWay.hard.__doc__) # accesses the doc string of the 'hard' property within the HardWay class. What is assigned for the doc in the property function ('doc=harddoc') becomes the special attribute, __doc__)
'''
RESULT:
The math score was 90.
The math score was changed to 101.
The math score was 90.
That score is not allowed
The constant has the value: 24
Properties cannot be changed without a setter
The value of not_decorated.hard is True
The value of not_decorated.hard is None
Properties can have a getter, setter, deleter, & doc string
'''
0 0
Ln 77, Col 24 (16 selected)
Spaces: 4
UTF-8
CRLF
Python