



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

```
1 # percipio34_properties_with_inheritance.py
2 # Percipio video: Classes; Properties With Inheritance
3 # Demonstrate how class properties work when affected by inheritance
4 # With properties access to them is controlled using 'getters', 'setters', & 'deleters'
5 # If you want to override a property & create setters or deleters, you must 1st define that property using
6 nl = '\n'
7 #
8 print(nl, '1st defined class; "Grades()":')
9 class Grades(): #
10     # Represent a grade as a numeric score with a property
11     def __init__(self, score=0): # Special method 'init' accepting a score or defaults score to 0
12         self._score = score # set's the instance, 'self._score', equal to that score
13
14 # By decorating a function with @property, it turns the function into a property that can be accessed like
15 @property # decorator with @ sign & the word property (think of as the 'getter'),...
16 def score(self): # 'score' is defined as a property determined by the @property directly above the
17     return self._score # which returns the value of 'self._score'
18
19 @ score.setter # setters are used to change the property value
20 def score(self, value): # the word 'score' is used again & allow because of the decorator (because it
21     if value > 100 or value < 0: # value setting validation
22         raise ValueError('Invalid score') # a score outside this range raises a ValueError with a message
23     # This will stop the program unless a try/except code block handling is used (see below)
24     self._score = value # if the value is between 0 & 100, then the 'self._score' attribute gets the
25
26 math = Grades(90)
27 print('The math score was %s.' % math.score)
28 # try/except block
29 try:
30     math.score = 101
31 except ValueError:
32     print('That score is not allowed')
33
34 #
35 print(nl, '2nd defined class; "Grader(Grades)":')
36 class Grader(Grades): # # percipio34_properties_with_inheritance.py
37     # Percipio video: Classes; Properties With Inheritance
38     # Demonstrate how class properties work when affected by inheritance
39     # With properties access to them is controlled using 'getters', 'setters', & 'deleters'
```




EXPLORER

percipio34_properties_with_inheritance.py



OPEN EDITORS

percipio34_properties_with_inheritance.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_method...
 - percipio31_static_methods.py
 - percipio32_inheritance.py
 - percipio33_properties.py
 - percipio34_properties_with_inheritance.py
 - 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



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

percipio34_properties_with_inheritance.py x

```
73         self._constant = value
74     #
75     rw = ReadWrite()
76     print(rw.constant)
77     rw.constant = 33
78     print(rw.constant)
79     '''
80 RESULTS:
81     1st defined class; "Grades()":
82     The math score was 90.
83     That score is not allowed
84
85     2nd defined class; "Grader(Grades)":
86     The Grader score is: 99
87     The Grader score is now: 199
88
89     3rd defined class; "ReadOnly()":
90     The constant has the value: 24
91     Properties cannot be changed without a setter
92
93     4th defined class; "ReadWrite(ReadOnly)":
94     24
95     33
96     '''
```



```
class Grades:
    """1st defined class; "Grades()":
    The math score was 90.
    That score is not allowed"""
    def __init__(self, score):
        self._score = score
    def __str__(self):
        return f"Grades score: {self._score}"
    def __repr__(self):
        return f"Grades(score={self._score})"
    def __setattr__(self, name, value):
        if name == '_score':
            self._score = value
        else:
            raise AttributeError("Grades object has no attribute '%s'" % name)
    def __del__(self):
        print("Grades object is destroyed")

class Grader(Grades):
    """2nd defined class; "Grader(Grades)":
    The Grader score is: 99
    The Grader score is now: 199"""
    def __init__(self, score):
        super().__init__(score)
    def __str__(self):
        return f"Grader score: {self._score}"
    def __repr__(self):
        return f"Grader(score={self._score})"
    def __setattr__(self, name, value):
        if name == '_score':
            self._score = value
        else:
            raise AttributeError("Grader object has no attribute '%s'" % name)
    def __del__(self):
        print("Grader object is destroyed")

class ReadOnly:
    """3rd defined class; "ReadOnly()":
    The constant has the value: 24
    Properties cannot be changed without a setter"""
    def __init__(self, constant):
        self._constant = constant
    def __str__(self):
        return f"ReadOnly constant: {self._constant}"
    def __repr__(self):
        return f"ReadOnly(constant={self._constant})"
    def __del__(self):
        print("ReadOnly object is destroyed")

class ReadWrite(ReadOnly):
    """4th defined class; "ReadWrite(ReadOnly)":
    24
    33"""
    def __init__(self, constant):
        super().__init__(constant)
    def __str__(self):
        return f"ReadWrite constant: {self._constant}"
    def __repr__(self):
        return f"ReadWrite(constant={self._constant})"
    def __del__(self):
        print("ReadWrite object is destroyed")
    def set_constant(self, value):
        self._constant = value
    def get_constant(self):
        return self._constant
```