Diff:

Differences between given skeleton and solution
In order to make the sample solution easier to understand, the differences between it and the given skeleton source code were highlighted with the help of the program `diff`.

Legend:

- Gray: unchanged text (only excerpts).
- Green: new lines
- Yellow: changed lines
- Red: deleted lines

Note: Files not listed have not been changed.

This document was created with the help of diff2html erstellt.

```
diff -u ../course05-object-orientation/exercise/code/01_excercise.py ../course05-object-orientation/exercise/solution/01_excercise.py
      ../course05-object-orientation/exercise/code/01_excercise.py                                   ../course05-object-orientation/exercise/solution/01_excercise.py
⁝                                                                        ⁝
7  # Task 05.1.1                                                         7  # Task 05.1.1
8  class GeometricObject:                                                8  class GeometricObject:
9                                                                        9
10     def __init__(self, middlepoint, XXX):                             10     def __init__(self, middlepoint, color, density, temperature):
11         self.middlepoint = XX                                         11         self.middlepoint = middlepoint
                                                                         12         self.color = color
                                                                         13         self.density = density
                                                                         14         self.temperature = temperature  # in K
12                                                                       15
13         self.check_attributes()                                       16         self.check_attributes()
14                                                                       17
15     # this method is for convenience you can ignore it
16     def __repr__(self):                                               18     def __repr__(self):
17         return str(list(self.__dict__.items()))                       19         return str(list(self.__dict__.items()))
18                                                                       20
19     def check_attributes(self):                                       21     def check_attributes(self):
20         assert isinstance(self.middlepoint, np.ndarray)               22         assert isinstance(self.middlepoint, np.ndarray)
21         assert self.middlepoint.shape == XXX                          23         assert self.middlepoint.shape == (3,)
                                                                         24         assert isinstance(self.color, str)
                                                                         25         assert isinstance(self.density, (float, int))
                                                                         26         assert isinstance(self.temperature, (float, int))
22                                                                       27
23     def calc_volume(self):                                            28     def calc_volume(self):
24         msg = "unavailable for this abstract base class"              29         msg = "unavailable for this abstract base class"
25         raise NotImplementedError(msg)                                30         raise NotImplementedError(msg)
26                                                                       31
27     def XXX():                                                        32     def calc_mass(self):
28         pass                                                          33         return self.calc_volume()*self.density
29                                                                       34
30         # ...                                                         35     def move(self, target_direction):
                                                                         36         assert isinstance(target_direction, np.ndarray)
                                                                         37         assert target_direction.shape == self.middlepoint.shape
                                                                         38         self.middlepoint += target_direction
31                                                                       39
                                                                         40     # Task 05.1.5 (only this method)
                                                                         41     def calc_distance(self, other):
                                                                         42         assert isinstance(other, GeometricObject)
                                                                         43         return np.sqrt(np.sum((self.middlepoint - other.middlepoint)**2))
32                                                                       44
33 exit() # move this line further down or delete it
34                                                                       45
35 class Ellipsoid(XXX):                                                 46 class Ellipsoid(GeometricObject):
36     pass
37                                                                       47
38 # ...                                                                 48     def __init__(self, r1, r2, r3, middlepoint, color="white", density=1, temperature=300):
                                                                         49         self.r1 = r1
                                                                         50         self.r2 = r2
                                                                         51         self.r3 = r3
                                                                         52
                                                                         53         # call the "constructor" of the base class
                                                                         54         GeometricObject.__init__(self, middlepoint, color, density, temperature)
                                                                         55
                                                                         56     def calc_volume(self):
                                                                         57         return 4/3*np.pi*self.r1*self.r2*self.r3
                                                                         58
```

```


# Task 05.1.2
x1 = XXX(np.array([0., 0., 0.]), "black", 2.5, 273)
# ...

# Task 05.1.3 for Cuboid instance x3

assert x3.calc_volume() == XXX


XXX.move(np.array([1, -5, -0.75]))

# check for new position
assert np.all(XXX.middlepoint == XXX)

# Task 05.1.3 for Cuboid instance x3

assert x4.calc_volume() == 4/3*np.pi*x4.r1**3

⋮
x4.move(np.array([0.3, 0.22, 0.111]))

# check for new position (more robust method)
```

```

class Cuboid(GeometricObject):

   def __init__(self, a, b, c, middlepoint, color="white", density=1, temperature=300):
       self.a = a
       self.b = b
       self.c = c

       # call the "constructor" of the base class
       GeometricObject.__init__(self, middlepoint, color, density, temperature)


   def calc_volume(self):
       return self.a*self.b*self.c


class Sphere(Ellipsoid):
   def __init__(self, radius, middlepoint, color="white", density=1, temperature=300):

       # call the "constructor" of the base class (Ellipsoid)
       Ellipsoid.__init__(self, radius, radius, radius, middlepoint, color, density, temperature)


class Cube(Cuboid):
   def __init__(self, a, middlepoint, color="white", density=1, temperature=300):

       # call the "constructor" of the base class (Ellipsoid)
       Cuboid.__init__(self, a, a, a, middlepoint, color, density, temperature)


# Task 05.1.2
x1 = GeometricObject(np.array([0., 0., 0.]), "black", 2.5, 273)
x2 = Ellipsoid(3, 2, 1, np.array([0., 0., 0.]))
x3 = Cuboid(2, 3, 4, np.array([0., 0., 0.]))
x4 = Sphere(2, np.array([0., 0., 0.]))
x5 = Cube(2.5, np.array([0., 0., 0.]))


# Task 05.1.3 for Cuboid instance x3

assert x3.calc_volume() == 24.0


print(x3.calc_volume(), x3.calc_mass())
x3.move(np.array([1, -5, -0.75]))
x3.move(np.array([-3, 7, 0.5]))

# check for new position
assert np.all(x3.middlepoint == np.array([-2, 2, -0.25]))

# Task 05.1.3 for Sphere instance x3

assert x4.calc_volume() == 4/3*np.pi*x4.r1**3

⋮
x4.move(np.array([0.3, 0.22, 0.111]))

# check for new position (more robust method)
```

```
assert np.allclose(XXX.middlepoint, XXX)


# Task 05.1.4

# create empty list
XXX = []
for i in XXX(10):
    XXX.append(XXX)


# Task 05.1.5

my_cube = XXX(...)
my_sphere = XXX(...)

print(my_cube.calc_distance(XXX))
```

```
assert np.allclose(x4.middlepoint, np.array([300.3, 20.22, 1.111]))


# Task 05.1.4

# create empty list
cubes = []
for i in range(10):
    cubes.append(Cube(a=10, middlepoint=np.random.random(3)))


# Task 05.1.5

my_cube = Cube(1, np.array([3, 0, 0]))
my_sphere = Sphere(1, np.array([0, 4, 0]))

print(my_cube.calc_distance(my_sphere))
```