Ass 4 (Task)

August 14, 2023

[1]:
```
[1.]
class Vehicle:
    def __init__(self, name_of_vehicle, max_speed, average_of_vehicle):
        self.name_of_vehicle = name_of_vehicle
        self.max_speed = max_speed
        self.average_of_vehicle = average_of_vehicle
```

[2]:
```
my_vehicle = Vehicle("Toyota Camry", 180, 30)
print(my_vehicle.name_of_vehicle)
print(my_vehicle.max_speed)
print(my_vehicle.average_of_vehicle)
```

```
Toyota Camry
180
30
```

[3]:
```
[2.]
class vehicle(Vehicle):
    def seating_capacity(self, capacity):
        return f"{self.name} has a seating capacity of {capacity} passengers."
```

[7]:
```
my_vehicle = vehicle("Toyota", "Camry", 2022)
print(my_vehicle.seating_capacity(5))
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[7], line 1
----> 1 my_vehicle = vehicle("Toyota", "Camry", 2022)
      2 print(my_vehicle.seating_capacity(5))

NameError: name 'vehicle' is not defined
```

[8]:
```
[3.]
class Shape:
    def __init__(self, color):
        self.color = color
```

```
class Rectangle(Shape):
    def __init__(self, width, height, color):
        super().__init__(color)
        self.width = width
        self.height = height

class Circle(Shape):
    def __init__(self, radius, color):
        super().__init__(color)
        self.radius = radius

class Square(Rectangle, Shape):
    def __init__(self, side, color):
        super().__init__(side, side, color)
        self.side = side

my_square = Square(5, "red")
print(my_square.color)
print(my_square.width)
print(my_square.height)
print(my_square.side)
```

```
red
5
5
5
```

[ ]:
```
[4.]  In Python, getters and setters are methods used to access and modify the␣
↪private attributes of a class.
     The primary use of getters and setters is to ensure data encapsulation in␣
↪object-oriented programs
     Private variables in Python are not hidden fields like in other␣
↪object-oriented languages, so getters and
     setters are used to add validation logic around getting and setting a␣
↪value

     A getter is a method that retrieves an object's current attribute value,␣
↪while a setter is a method that changes
       an object's attribute value.
     In Python, we can define getter and setter methods using the @property␣
↪and @<attribute_name>.setter decorators.
```

[9]:
```
class Person:
    def __init__(self, name, age):
        self._name = name
        self._age = age
    @property
    def age(self):
        return self._age

    @age.setter
    def age(self, value):
        if value < 0:
            raise ValueError("Age cannot be negative")
        self._age = value
```

[10]:
```
person = Person("Alice", 25)
print(person.age)

person.age = 30
print(person.age)

person.age = -5
```

```
25
30
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[10], line 7
      4 person.age = 30
      5 print(person.age)
----> 7 person.age = -5

Cell In[9], line 13, in Person.age(self, value)
     10 @age.setter
     11 def age(self, value):
     12     if value < 0:
---> 13         raise ValueError("Age cannot be negative")
     14     self._age = value

ValueError: Age cannot be negative
```

[ ]:
```
[5.] Method overriding in Python is a feature of object-oriented programming␣
 ↪that allows a subclass or child class
    to provide a specific implementation of a method that is already provided␣
 ↪by its superclass or parent class

. When a method in a subclass has the same name, same parameters or signature,␣
 ↪and same return type (or subtype)
```

```
as a method in its superclass, then the method in the subclass is said to␣
↪override the method in the superclass
```

[11]:
```
class Animal:
    def sound(self):
        print("Animal makes a sound")

class Dog(Animal):
    def sound(self):
        print("Dog barks")

class Cat(Animal):
    def sound(self):
        print("Cat meows")

animal = Animal()
animal.sound()

dog = Dog()
dog.sound()

cat = Cat()
cat.sound()
```

```
Animal makes a sound
Dog barks
Cat meows
```

[ ]:
```

```