

Class Design Strategy In Python

Crompton Fan Class

```
1 -----
2
3
4 """
5     @Author:Kunal Narkhede
6     @Date:22/12/2023
7     @Goal:To Implement Class Fan
8     Capture Real Life Product on Amazon
9     http://surl.li/onctm
10 """
11 -----
12
13 import sys
14
15 class ProductDimension:
16     """
17     This Class Implement The Dimension Of Crompton Fan
18     @__init__(self, length: float, width: float, height: float, weight: float):
19     Constructor
20     @get_length(self)
21     getter of attribute length
22     @get_width(self)
23     getter of attribute width
24     @get_height(self)
25     getter of attribute height
26     @get_weight(self)
27     getter of attribute weight
28     -----
29     @set_length(self):
30     setter of attribute length
31     @set_width(self):
32     setter of attribute width
33     @set_height(self):
34     setter of attribute height
```

```
35     @set_weight(self):
36         setter of attribute weight
37     """
38
39     def __init__(
40         self,
41         length:float,
42         width:float,
43         height:float,
44         weight:float
45     ):
46
47         """
48         Constructor of ProductDimension class:
49         @__init__(self, length: float, width: float, height: float, weight: float):
50
51         @self:newly created class object of ProductDimension
52         @length:Client specified value for attribute length
53         @width:Client specified value for attribute width
54         @height:Client specified value for attribute height
55         @weight:Client specified value for attribute weight
56
57         """
58         if type(length)!=float:
59             raise TypeError("Bad type:length")
60         if type(width)!=float:
61             raise TypeError("Bad type:width")
62         if type(height)!=float:
63             raise TypeError("Bad type:height")
64         if type(weight)!=float:
65             raise TypeError("Bad type:weight")
66         if length<=0.0:
67             raise ValueError("Length must be positive")
68         if width<=0.0:
69             raise ValueError("Width must be positive")
```

```
70     if height<=0.0:
71         raise ValueError("Height must be positive")
72     if weight<=0.0:
73         raise ValueError("Weight must be positive")
74
75     self.length=length
76     self.width=width
77     self.height=height
78     self.weight=weight
79
80     #getter method
81
82     def get_length(self) -> float:
83         """
84         Returns the length attribute of the calling object
85         """
86         return self.length
87
88     def get_width(self) -> float:
89         """
90         Returns the width attribute of the calling object
91         """
92         return self.width
93
94     def get_height(self) -> float:
95         """
96         Returns the height attribute of the calling object
97         """
98         return self.height
99
100     def get_weight(self) -> float:
101         """
102         Returns the weight attribute of the calling object
103         """
104         return self.weight
```

```
105
106 #setter method
107
108 def set_length(self,new_length:float):
109     """
110     Sets the length attribute of the calling object to @new_length
111     Before setting, TypeCheck and ValueCheck is performed.
112     """
113     if type(new_length)!=float:
114         raise TypeError("new_length must be an float")
115     if new_length <= 0.0:
116         raise TypeError("new_length must be positive")
117     self.length=new_length
118
119 def set_width(self,new_width:float):
120     """
121     Sets the width attribute of the calling object to @new_width
122     Before setting, TypeCheck and ValueCheck is performed.
123     """
124     if type(new_width)!=float:
125         raise TypeError("new_width must be an float")
126     if new_width <= 0.0:
127         raise ValueError("new_width must be positive")
128     self.width=new_width
129
130 def set_height(self,new_height:float):
131     """
132     Sets the height attribute of the calling object to @new_height
133     Before setting, TypeCheck and ValueCheck is performed.
134     """
135     if type(new_height)!=float:
136         raise TypeError("new_height must be an float")
137     if new_height <= 0.0 :
138         raise ValueError("new_height must be positive")
139     self.height=new_height
```

```
140
141 def set_weight(self,new_weight:float):
142     """
143     Sets the weight attribute of the calling object to @new_weight
144     Before setting, TypeCheck and ValueCheck is performed.
145     """
146     if type(new_weight)!=float:
147         raise TypeError("new_weight must be an float")
148     if new_weight <= 0.0:
149         raise ValueError("new_weight must be positive")
150     self.weight=new_weight
151
152
153 class Fan:
154     def __init__(
155         self,
156         fan_brand:str,
157         fan_colour:str,
158         electric_fan_design:str,
159         fan_power_source:str,
160         fan_style:str,
161         fan_prod_dimensions:ProductDimension,
162         room_type:[str],
163         fan_feature:str,
164         fan_mounting_type:str,
165         fan_controller_type:str,
166         fan_material:str,
167         fan_nr_of_speed:int,
168         fan_wattage:int,
169         fan_finish_type:str,
170         fan_nr_of_blades:int,
171         fan_air_flow_capacity:str,
172         fan_speed:int,
173         fan_switch_type:str,
174         fan_included_components:[str],
```

```
175         fan_model_name:str,
176         fan_specification_met:str,
177         fan_blade_material:str,
178         fan_manufacturer:str,
179         fan_country_of_origin:str,
180         fan_model_num:str,
181         fan_ASIN:str
182     ):
183
184
185     if type(fan_brand)!=str:
186         raise TypeError("brand must be in str")
187     if type(fan_colour)!=str:
188         raise TypeError("colour must be in str")
189     if type(electric_fan_design)!=str:
190         raise TypeError("electric_fan_design must be in str")
191     if type(fan_power_source)!=str:
192         raise TypeError("power source must be in str")
193     if type(fan_style)!=str:
194         raise TypeError("style must be in str")
195     if type(fan_prod_dimensions)!=ProductDimension:
196         raise TypeError("Product dimension must in Product Dimension")
197     if '__iter__' not in dir(type(room_type)):
198         raise TypeError("room type must be iterable")
199     for room in room_type:
200         if type(room)!=str:
201             raise TypeError("room must be str")
202     if type(fan_feature)!=str:
203         raise TypeError("feature must be in str")
204     if type(fan_mounting_type)!=str:
205         raise TypeError("mounting type must be in str")
206     if type(fan_controller_type)!=str:
207         raise TypeError("controller type must be in str")
208     if type(fan_material)!=str:
209         raise TypeError("material must be in str")
```

```
210     if type(fan_nr_of_speed)!=int:
211         raise TypeError("number of speed must be in int")
212     if fan_nr_of_speed<=0:
213         raise ValueError("number of speed must be positive")
214     if type(fan_wattage)!=int:
215         raise TypeError("wattage must be in int")
216     if fan_wattage<=0:
217         raise ValueError("wattage must be in positive")
218     if type(fan_finish_type)!=str:
219         raise TypeError("finish_type must be in str")
220     if type(fan_nr_of_blades)!=int:
221         raise TypeError("number of blades must be int")
222     if fan_nr_of_blades<=0:
223         raise ValueError("number of blades must be positive")
224     if type(fan_air_flow_capacity)!=str:
225         raise TypeError("air flow capacity must be in str")
226     if type(fan_speed)!=int:
227         raise TypeError("speed must be in int")
228     if fan_speed<=0:
229         raise ValueError("speed must be positive")
230     if type(fan_switch_type)!=str:
231         raise TypeError("switch type must be in str")
232     if '__iter__' not in dir(type(fan_included_components)):
233         raise TypeError("included components must be iterable")
234     for component in fan_included_components:
235         if type(component)!=str:
236             raise TypeError("component must be str")
237     if type(fan_model_name)!=str:
238         raise TypeError("model name must be str")
239     if type(fan_specification_met)!=str:
240         raise TypeError("specification must be str")
241     if type(fan_blade_material)!=str:
242         raise TypeError("blade materail must be str")
243     if type(fan_manufacturer)!=str:
244         raise TypeError("manufacturer must be str")
```

```
245         if type(fan_country_of_origin)!=str:
246             raise TypeError("country of origin must be str")
247         if type(fan_model_num)!=str:
248             raise TypeError("model number must be str")
249         if type(fan_ASIN)!=str:
250             raise TypeError("ASIN must be str")
251
252         self.fan_brand=fan_brand
253         self.fan_colour=fan_colour
254         self.electric_fan_design=electric_fan_design
255         self.fan_power_source=fan_power_source
256         self.fan_style=fan_style
257         self.fan_prod_dimensions=fan_prod_dimensions
258         self.room_type=room_type
259         self.fan_feature=fan_feature
260         self.fan_mounting_type=fan_mounting_type
261         self.fan_controller_type=fan_controller_type
262         self.fan_material=fan_material
263         self.fan_nr_of_speed=fan_nr_of_speed
264         self.fan_wattage=fan_wattage
265         self.fan_finish_type=fan_finish_type
266         self.fan_nr_of_blades=fan_nr_of_blades
267         self.fan_air_flow_capacity=fan_air_flow_capacity
268         self.fan_speed=fan_speed
269         self.fan_switch_type=fan_switch_type
270         self.fan_included_components=fan_included_components
271         self.fan_model_name=fan_model_name
272         self.fan_specification_met=fan_specification_met
273         self.fan_blade_material=fan_blade_material
274         self.fan_manufacturer=fan_manufacturer
275         self.fan_country_of_origin=fan_country_of_origin
276         self.fan_model_num=fan_model_num
277         self.fan_ASIN=fan_ASIN
278     #Getter method
279     def get_fan_brand(self)->str:
```



```
280     """
281     Returns the fan_brand attribute of the calling object
282     """
283     return self.fan_brand
284 def get_fan_colour(self)->str:
285     """
286     Returns the fan_colour attribute of the calling object
287     """
288     return self.fan_colour
289
290 def get_electric_fan_design(self)->str:
291     """
292     Returns the electric_fan_design attribute of the calling object
293     """
294     return self.electric_fan_design
295 def get_fan_power_source(self)->str:
296     """
297     Returns the fan_power_source attribute of the calling object
298     """
299     return self.fan_power_source
300 def get_fan_style(self)->str:
301     """
302     Returns the fan_style attribute of the calling object
303     """
304     return self.fan_style
305 def get_fan_prod_dimensions(self)->ProductDimension:
306     """
307     Returns the fan_prod_dimensions attribute of the calling object
308     """
309     return self.fan_prod_dimensions
310 def get_room_type(self)->[str]:
311     """
312     Returns the room_type attribute of the calling object
313     """
314     return self.room_type
```

```
315 def get_fan_feature(self)->str:
316     """
317     Returns the fan_feature attribute of the calling object
318     """
319     return self.fan_feature
320 def get_fan_mounting_type(self)->str:
321     """
322     Returns the fan_mounting_type attribute of the calling object
323     """
324     return self.fan_mounting_type
325 def get_fan_controller_type(self)->str:
326     """
327     Returns the fan_controller_type attribute of the calling object
328     """
329     return self.fan_controller_type
330 def get_fan_material(self)->str:
331     """
332     Returns the fan_material attribute of the calling object
333     """
334     return self.fan_material
335 def get_fan_nr_of_speed(self)->int:
336     """
337     Returns the fan_nr_of_speed attribute of the calling object
338     """
339     return self.fan_nr_of_speed
340 def get_fan_wattage(self)->int:
341     """
342     Returns the fan_wattage attribute of the calling object
343     """
344     return self.fan_wattage
345 def get_fan_finish_type(self)->str:
346     """
347     Returns the fan_finish_type attribute of the calling object
348     """
349     return self.fan_finish_type
```

```
350 def get_fan_nr_of_blades(self)->int:
351     """
352     Returns the fan_nr_of_blades attribute of the calling object
353     """
354     return self.fan_nr_of_blades
355 def get_fan_air_flow_capacity(self)->str:
356     """
357     Returns the fan_air_flow_capacity attribute of the calling object
358     """
359     return self.fan_air_flow_capacity
360 def get_fan_speed(self)->int:
361     """
362     Returns the fan_speed attribute of the calling object
363     """
364     return self.fan_speed
365 def get_fan_switch_type(self)->str:
366     """
367     Returns the fan_switch_type attribute of the calling object
368     """
369     return self.fan_switch_type
370 def get_fan_included_components(self)->[str]:
371     """
372     Returns the fan_included_components attribute of the calling object
373     """
374     return self.fan_included_components
375 def get_fan_model_name(self)->str:
376     """
377     Returns the fan_model_name attribute of the calling object
378     """
379     return self.fan_model_name
380 def get_fan_specification_met(self)->str:
381     """
382     Returns the fan_specification_met attribute of the calling object
383     """
384     return self.fan_specification_met
```

```

385 def get_fan_blade_material(self)->str:
386     """
387     Returns the fan_blade_material attribute of the calling object
388     """
389     return self.fan_blade_material
390 def get_fan_manufacturer(self)->str:
391     """
392     Returns the fan_manufacturer attribute of the calling object
393     """
394     return self.fan_manufacturer
395 def get_fan_country_of_origin(self)->str:
396     """
397     Returns the fan_country_of_origin attribute of the calling object
398     """
399     return self.fan_country_of_origin
400 def get_fan_model_num(self)->str:
401     """
402     Returns the fan_model_num attribute of the calling object
403     """
404     return self.fan_model_num
405 def get_fan_ASIN(self)->str:
406     """
407     Returns the fan_ASIN attribute of the calling object
408     """
409     return self.fan_ASIN
410 #Setter method
411 def set_fan_brand(self,new_fan_brand)->None:
412     """
413     Sets the fan_brand attribute of the calling object to @new_fan_brand
414     Before setting, TypeCheck is performed.
415     """
416     if type(new_fan_brand)!=str:
417         raise TypeError("new_fan_brand must be str")
418     self.fan_brand=new_fan_brand
419

```

```
420 def set_fan_colour(self,new_fan_colour)->None:
421     """
422     Sets the fan_colour attribute of the calling object to @new_fan_colour
423     Before setting, TypeCheck is performed.
424     """
425     if type(new_fan_colour)!=str:
426         raise TypeError("new_fan_colour must be str")
427     self.fan_colour=new_fan_colour
428
429 def set_electric_fan_design(self,new_electric_fan_design)->None:
430     """
431     Sets the electric_fan_design attribute of the calling object to @new_electric_fan_design
432     Before setting, TypeCheck is performed.
433     """
434     if type(new_electric_fan_design)!=str:
435         raise TypeError("new_electric_fan_design must be str")
436     self.electric_fan_design=new_electric_fan_design
437
438 def set_fan_power_source(self,new_fan_power_source)->None:
439     """
440     Sets the fan_power_source attribute of the calling object to @new_fan_power_source
441     Before setting, TypeCheck is performed.
442     """
443     if type(new_fan_power_source)!=str:
444         raise TypeError("new_fan_power_source must be str")
445     self.fan_power_source=new_fan_power_source
446
447 def set_fan_style(self,new_fan_style)->None:
448     """
449     Sets the fan_style attribute of the calling object to @new_fan_style
450     Before setting, TypeCheck is performed.
451     """
452     if type(new_fan_style)!=str:
453         raise TypeError("new_fan_style must be str")
454     self.fan_style=new_fan_style
```

```
455
456 def set_fan_prod_dimensions(self,new_fan_prod_dimensions)->None:
457     """
458         Sets the fan_prod_dimensions attribute of the calling object to @new_fan_prod_dimensions
459         Before setting, TypeCheck is performed.
460     """
461     if type(new_fan_prod_dimensions)!=str:
462         raise TypeError("new_fan_prod_Dimensions must be ProductDimension")
463     self.fan_prod_dimensions=new_fan_prod_dimensions
464
465 def set_room_type(self,new_room_type)->None:
466     """
467         Sets the room_type attribute of the calling object to @new_room_type
468         Before setting, TypeCheck is performed.
469     """
470     if '__iter__' not in dir(type(new_room_type)):
471         raise TypeError("new_room_type must be iterable")
472     for room in new_room_type:
473         if type(room)!=str:
474             raise TypeError("room must be str")
475     self.room_type=new_room_type
476
477 def set_fan_feature(self,new_fan_feature)->None:
478     """
479         Sets the fan_feature attribute of the calling object to @new_fan_feature
480         Before setting, TypeCheck is performed.
481     """
482     if type(new_fan_feature)!=str:
483         raise TypeError("new_fan_feature must be str")
484     self.fan_feature=new_fan_feature
485
486 def set_fan_mounting_type(self,new_mounting_type)->None:
487     """
488         Sets the mounting_type attribute of the calling object to @new_mounting_type
489         Before setting, TypeCheck is performed.
```

```

490     """
491     if type(new_mounting_type)!=str:
492         raise TypeError("new_mounting_type must be str")
493     self.fan_mounting_type=new_mounting_type
494
495     def set_fan_controller_type(self,new_fan_controller_type)->None:
496         """
497         Sets the fan_controller_type attribute of the calling object to @new_fan_controller_type
498         Before setting, TypeCheck is performed.
499         """
500         if type(new_fan_controller_type)!=str:
501             raise TypeError("new_fan_controller_type must be str")
502         self.fan_controller_type=new_fan_controller_type
503
504     def set_fan_material(self,new_fan_material)->None:
505         """
506         Sets the fan_material attribute of the calling object to @new_fan_material
507         Before setting, TypeCheck is performed.
508         """
509         if type(new_fan_material)!=str:
510             raise TypeError("new_fan_material must be str")
511         self.fan_material=new_fan_material
512
513     def set_fan_nr_of_speed(self,new_fan_nr_of_speed)->None:
514         """
515         Sets the fan_nr_of_speed attribute of the calling object to @new_fan_nr_of_speed
516         Before setting, TypeCheck and ValueCheck is performed.
517         """
518         if type(new_fan_nr_of_speed)!=int:
519             raise TypeError("new_fan_nr_of_speed must be int")
520         if new_fan_nr_of_speed<=0:
521             raise ValueError("new_fan_nr_of_speed must be positive")
522         self.fan_nr_of_speed=new_fan_nr_of_speed
523
524     def set_fan_wattage(self,new_fan_wattage)->None:

```

```

525     """
526     Sets the fan_wattage attribute of the calling object to @new_fan_wattage
527     Before setting, TypeCheck and ValueCheck is performed.
528     """
529     if type(new_fan_wattage)!=int:
530         raise TypeError("new_fan_wattage must be int")
531     if new_fan_wattage<=0:
532         raise ValueError("new_fan_wattage must be positive")
533     self.fan_wattage=new_fan_wattage
534
535 def set_fan_finish_type(self,new_fan_finish_type)->None:
536     """
537     Sets the fan_finish_type attribute of the calling object to @new_fan_finish_type
538     Before setting, TypeCheck is performed.
539     """
540     if type(new_fan_finish_type)!=str:
541         raise TypeError("new_fan_finish_type must be str")
542     self.fan_finish_type=new_fan_finish_type
543
544 def set_fan_nr_of_blades(self,new_fan_nr_of_blades):
545     """
546     Sets the fan_nr_of_blades attribute of the calling object to @new_fan_nr_of_blades
547     Before setting, TypeCheck and ValueCheck is performed.
548     """
549     if type(new_fan_nr_of_blades)!=int:
550         raise TypeError("new_fan_nr_of_blades must be int")
551     if new_fan_nr_of_blades<=0:
552         raise ValueError("new_fan_nr_of_blades must be positive")
553     self.fan_nr_of_blades=new_fan_nr_of_blades
554
555 def set_fan_air_flow_capacity(self,new_air_flow_capacity)->None:
556     """
557     Sets the air_flow_capacity attribute of the calling object to @new_air_flow_capacity
558     Before setting, TypeCheck is performed.
559     """

```



```

560     if type(new_air_flow_capacity)!=str:
561         raise TypeError("new_air_flow_capacity must be str")
562     self.fan_air_flow_capacity=new_air_flow_capacity
563
564     def set_fan_speed(self,new_fan_speed)->None:
565         """
566         Sets the fan_speed attribute of the calling object to @new_fan_speed
567         Before setting, TypeCheck and ValueCheck is performed.
568         """
569         if type(new_fan_speed)!=int:
570             raise TypeError("new_fan_speed must be int")
571         if new_fan_speed<=0:
572             raise ValueError("new_fan_speed must be positive")
573         self.fan_speed=new_fan_speed
574
575     def set_fan_switch_type(self,new_fan_switch_type)->None:
576         """
577         Sets the fan_switch_type attribute of the calling object to @new_fan_switch_type
578         Before setting, TypeCheck is performed.
579         """
580         if type(new_fan_switch_type)!=str:
581             raise TypeError("new_fan_switch_type must be str")
582         self.fan_switch_type=new_fan_switch_type
583
584     def set_fan_included_components(self,new_fan_included_components)->None:
585         """
586         Sets the fan_included_components attribute of the calling object to
587         @new_fan_included_components
588         Before setting, TypeCheck is performed.
589         """
590         if '__iter__' not in dir(type(new_fan_included_components)):
591             raise TypeError("new_fan_included_components must iterable")
592         for component in new_fan_included_components:
593             if type(component)!=str:
594                 raise TypeError("feature must be str")
595         self.fan_included_components=new_fan_included_components

```

```
596
597
598 def set_fan_model_name(self,new_fan_model_name)->None:
599     """
600         Sets the fan_model_name attribute of the calling object to @new_fan_model_name
601         Before setting, TypeCheck is performed.
602     """
603     if type(new_fan_model_name)!=str:
604         raise TypeError("new_fan_model_name must be str")
605     self.fan_model_name=new_fan_model_name
606
607 def set_fan_specification_met(self,new_fan_specification_met)->None:
608     """
609         Sets the fan_specification_met attribute of the calling object to @new_fan_specification_met
610         Before setting, TypeCheck is performed.
611     """
612     if type(new_fan_specification_met)!=str:
613         raise TypeError("new_fan_specification_met must be str")
614     self.fan_specification_met=new_fan_specification_met
615
616 def set_fan_blade_material(self,new_fan_blade_material)->None:
617     """
618         Sets the fan_blade_material attribute of the calling object to @new_fan_blade_material
619         Before setting, TypeCheck is performed.
620     """
621     if type(new_fan_blade_material)!=str:
622         raise TypeError("new_fan_blade_material must be str")
623     self.fan_blade_material=new_fan_blade_material
624
625 def set_fan_manufacturer(self,new_fan_manufacturer):
626     """
627         Sets the fan_manufacturer attribute of the calling object to @new_fan_manufacturer
628         Before setting, TypeCheck is performed.
629     """
630     if type(new_fan_manufacturer)!=str:
```

```

631         raise TypeError("new_fan_manufacturer must be str")
632     self.fan_manufacturer=new_fan_manufacturer
633
634     def set_fan_country_of_origin(self,new_fan_country_of_origin)->None:
635         """
636         Sets the fan_country_of_origin attribute of the calling object to @new_fan_country_of_origin
637         Before setting, TypeCheck is performed.
638         """
639         if type(new_fan_country_of_origin)!=str:
640             raise TypeError("new_fan_country_of_origin must be str")
641         self.fan_country_of_origin=new_fan_country_of_origin
642
643     def set_fan_model_num(self,new_model_num)->None:
644         """
645         Sets the model_num attribute of the calling object to @new_model_num
646         Before setting, TypeCheck is performed.
647         """
648         if type(new_model_num)!=str:
649             raise TypeError("new_model_num must be str")
650         self.fan_model_num=new_model_num
651
652     def set_fan_ASIN(self,new_fan_ASIN)->None:
653         """
654         Sets the fan_ASIN attribute of the calling object to @new_fan_ASIN
655         Before setting, TypeCheck is performed.
656         """
657         if type(new_fan_ASIN)!=str:
658             raise TypeError("new_fan_ASIN must be str")
659         self.fan_ASIN=new_fan_ASIN
660     def show(self)->None:
661
662         """
663         This function display all the characterstics of Fan class
664         """
665         print("Brand:{}".format(self.fan_brand))

```

```
666     print("Colour:{}".format(self.fan_colour))
667     print("Electric fan design:{}".format(self.electric_fan_design))
668     print("Power Source:{}".format(self.fan_power_source))
669     print("Style:{}".format(self.fan_style))
670     print("Product Dimension:{}".format(self.fan_prod_dimensions.__dict__))
671     print("Room type:{}".format(self.room_type))
672     print("Special Feature:{}".format(self.fan_feature))
673     print("Mounting type:{}".format(self.fan_mounting_type))
674     print("Controller type:{}".format(self.fan_controller_type))
675     print("Material:{}".format(self.fan_material))
676
677     print("Number of Speeds:{}".format(self.fan_nr_of_speed))
678     print("Wattage:{}".format(self.fan_wattage))
679     print("Finish Type:{}".format(self.fan_finish_type))
680     print("Number of blades:{}".format(self.fan_nr_of_blades))
681     print("Air flow capacity:{}".format(self.fan_air_flow_capacity))
682     print("Speed :{}".format(self.fan_speed))
683     print("Switch type :{}".format(self.fan_switch_type))
684     print("Included Components:{}".format(self.fan_included_components))
685     print("Model Name:{}".format(self.fan_model_name))
686
687     print("Specification Met:{}".format(self.fan_specification_met))
688     print("Blade Material:{}".format(self.fan_blade_material))
689     print("Country of Origin:{}".format(self.fan_country_of_origin))
690     print("Model number:{}".format(self.fan_model_num))
691     print("ASIN number:{}".format(self.fan_ASIN))
692
693
694
695 def main():
696     fan_obj=Fan(
697         "Crompton",
698         "Lustre Brown",
699         "Ceiling Fan",
700         "Electricity",
```

```

701         "Sea Sapphira(1 Star Rated)",
702         ProductDimension(54.5,25.5,19.4,3.4),
703         ["Living Room","Bedroom","Daning Room"],
704         "High Velocity",
705         "downrod mount",
706         "Regulator Control",
707         "Aluminium",
708         3,
709         51,
710         "Power Coated",
711         3,
712         "210 Cubic Matres Minute",
713         380,
714         "Push Button",
715         ["Fan motor","Balanced blade set","downrod","canopies","safety cable and shackle kit
716 assembly"],
717         "CROMPTON SUREBREEZE SEA",
718         "CE",
719         "CRCA",
720         "Crompton Greaves Consumer Electricals Limited",
721         "India",
722         "CFSBSSP48LB1S",
723         "B0BTS9GG2V"
724     )
725     print("FAN PRODUCT DETAILS:")
726     fan_obj.show()
727     #we can also get the attribute using getter method and
728     #set the specific attribute using setter method
729     sys.exit(0)
730 main()
731
732 """
733 Ouput:
734 -----
735 FAN PRODUCT DETAILS:
736 Brand:Crompton

```

737 Colour:Lustre Brown
738 Electric fan design:Ceiling Fan
739 Power Source:Electricity
740 Style:Sea Sapphira(1 Star Rated)
741 Product Dimension:{'length': 54.5, 'width': 25.5, 'height': 19.4, 'weight': 3.4}
742 Room type:['Living Room', 'Bedroom', 'Daning Room']
743 Special Feature:High Velocity
744 Mounting type:downrod mount
745 Controller type:Regulator Control
746 Material:Aluminium
747 Number of Speeds:3
748 Wattage:51
749 Finish Type:Power Coated
750 Number of blades:3
751 Air flow capacity:210 Cubic Matres Minute
752 Speed :380
753 Switch type :Push Button
754 Included Components:['Fan motor', 'Balanced blade set', 'downrod', 'canopies', 'safety cable and
755 shackle kit assembly']
756 Model Name:CROMPTON SUREBREEZE SEA
757 Specification Met:CE
758 Blade Material:CRCA
759 Country of Origin:India
760 Model number:CFSBSSP48LB1S
761 ASIN number:B0BTS9GG2V
762 ""
763