|  |  |
| --- | --- |
| **class** kettle(object):  **def** \_\_init\_\_(self, make, price):  self.make = make  self.price = price  self.on = **False** potter=kettle(**"potter"**, 80) print(potter.price) print(potter.make)  potter.price=100.36 print(potter.price)  print(**"price : {} maker : {} cost: {}"** .format(potter.price, potter.make, potter.price\*5)) | 80  potter  100.36  price : 100.36 maker : potter cost: 501.8 |
| **class** kettle(object):  **def** \_\_init\_\_(self, make, price):  self.make = make  self.price = price  self.on = **False** *#implementing a method* **def** switch\_on(self):  self.on=**True**;  potter=kettle(**"potter"**, 80) *#checking the method* print(potter.on) #it is initialized with False potter.switch\_on() #calling the method print(potter.on) #assigned into TRUE | False  True |
| THE DYNAMIC FEATURE IN PYTHON :  One can define any field, for an instance, anywhere in the code, that gonna work for particular instance quite well;  The dynamic field cant be seen from other instances, although they bith uses the same method as well; | |
| **class** kettle(object):  **def** \_\_init\_\_(self, make, price):  self.make = make  self.price = price  self.on = **False** *#implementing a method* **def** switch\_on(self):  self.on=**True**;  potter=kettle(**"potter"**, 80) potter.power=1.5 *#this is defined here* print(potter.power) *#getting o/p* pot=kettle(**"pot"**,78)  print(pot.power) *#power is never defined for pot, so throws an ERROR* | 1.5  ERROR |
| CLASSES CAN BE REPRESENTED AS DICTIONARY | |
| **class** kettle(object):  **def** \_\_init\_\_(self, make, price):  self.make = make  self.price = price  self.on = **False** *#implementing a method* **def** switch\_on(self):  self.on=**True**;  ket=kettle(**"shit"**, 50) kon=kettle(**"poty"**, 80) print(kettle.\_\_dict\_\_) print(ket.\_\_dict\_\_ ) print(kon.\_\_dict\_\_) | |
| {'\_\_module\_\_': '\_\_main\_\_', '\_\_init\_\_': <function kettle.\_\_init\_\_ at 0x01851468>, 'switch\_on': <function kettle.switch\_on at 0x01851420>, '\_\_dict\_\_': <attribute '\_\_dict\_\_' of 'kettle' objects>, '\_\_weakref\_\_': <attribute '\_\_weakref\_\_' of 'kettle' objects>, '\_\_doc\_\_': None}  {'make': 'shit', 'price': 50, 'on': False}  {'make': 'poty', 'price': 80, 'on': False} | |

Being Dynamic we can assign a assigned value inside the class;

But re assigning will print the value newly assighed

|  |  |
| --- | --- |
| **class** kettle(object):  power\_supply=**"eletronic"  def** \_\_init\_\_(self, make, price):  self.make = make  self.price = price  self.on = **False** *#implementing a method* **def** switch\_on(self):  self.on=**True**;  ket=kettle(**"shit"**, 50) kon=kettle(**"poty"**, 80) print(ket.power\_supply) print(kon.power\_supply)  print(**"\*"**\*50) ket.power\_supply=**"BOMB"** *#gonna update only the instance* print(ket.power\_supply) print(kon.power\_supply) *#will never update aother instanc*  only python allows to access a class member using class, rather than instanceprint(**"\*"**\*50) kettle.power\_supply=**"IRON"** *#REASSIGNING the value will update the value that has not been updated previously* print(ket.power\_supply) print(kon.power\_supply) *#so only kon will be updated* | eletronic  eletronic  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  BOMB  eletronic  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  BOMB  IRON |
| **class** kettle(object):  power\_supply=**"eletronic"  def** \_\_init\_\_(self, make, price):  self.make = make  self.price = price  self.on = **False** *#implementing a method* **def** switch\_on(self):  self.on=**True**;  ket=kettle(**"shit"**, 50) kon=kettle(**"poty"**, 80)  print(**"\*"**\*50) kettle.power\_supply=**"IRON"** *#no previous updation, gonna update the value of power\_supply for both case* print(ket.power\_supply) print(kon.power\_supply) | IRON  IRON |