```
"""
1. A class is collection/set of functions that carry out various operations o
"Instances"

2. Instances are the actual objects/data that your function manipulate on.
"""
# Different ways of storing data using built-in data structures.
a = [1, 2]
t = (1, 2)
d = {'a': 1, 'b': 2}    # advantage of storing data in dict is that you can a

# Performing different operations on the data stored in the list.
# ------------------------------------------------------------
# 1. sort
# 2. reverse
# 3. __len__()
# 4. __getitem__()
# 5. __contains__()
# ------------------------------------------------------------
# Want to perform other operations apart from built-in
# ------------------------------------------------------------
# 1. Adding a[0] = a[0] + 0.5, a[1] = a[1] + 0.5
# 2. Get the total of two co-ordinates total = a[0] + a[1]
# 3. Swap two co-ordinates temp = a[0], a[0] = a[1], a[1] = temp
# 4. Sorting two co-ordinates a.sort()
# 5. Resetting the co-ordinates a[0] = 0, a[1] = 0
# ------------------------------------------------------------
```

```python
# User defined class or datatype
class Point:
    # Data is being saved inside a dictionary
    def __init__(self, a, b):
        self.a = a
        self.b = b

p1 = Point(1, 2)
p2 = Point(10, 20)

# The values are internally stored in a dictionary. It is also called instanc
print(p1.__dict__)  # {"a": 1, "b": 2}
print(p2.__dict__)  # {"a": 10, "b": 20}
# ================================================================================
# "Point" class with a some methods
class Point:
    def __init__(self, a, b):
        self.a = a
        self.b = b

    # Takes data from instance dictionary
    def move(self, dx, dy):
        self.a += dx
        self.b += dy

    # resets the value of self.a and self.b to zero
    def reset(self):
        self.a = 0
        self.b = 0

    # Method that sorts points
    def sort(self):
        if self.a < self.b:
            return (self.a, self.b)
        return (self.b, self.a)

    # Method that swaps values of 'a' and 'b'
    def swap_points(self):
        temp = self.a
        self.a = self.b
        self.b = temp
        return (self.a, self.b)

    def total(self):
        return self.a + self.b
```

```
73
74  p1 = Point(1, 2)
75  p2 = Point(1.4, 1.2)
76
77  # The information about data is present in instance dictionary  (obj.__dict__
78  # The information about methods is present in class dictionary (class_name.__
79  # ============================================================================
80  class Calculator:
81      def __init__(self, x, y):
82          self.a = x
83          self.b = y
84
85      def add(self):
86          return self.a + self.b
87
88      def sub(self):
89          return self.a - self.b
90
91      def mul(self):
92          return self.a * self.b
93
94  c1 = Calculator(1, 2)
95  c2 = Calculator(4, 5)
96  c3 = Calculator(10, 20)
97  # ============================================================================
98  # Employee Class
99  class Employee:
100     def __init__(self, fname, lname, pay):
101         self.fname = fname
102         self.lname = lname
103         self.pay = pay
104
105     def email(self):
106         return f'{self.fname}.{self.lname}@company.com'
107
108 e1 = Employee("Steve", "Jobs", 1000)
109 e2 = Employee("Bill", "Gates", 2000)
110 # ============================================================================
111 class Player:
112     def __init__(self, x, y):
113         self.x = x
114         self.y = y
115         self.health = 100
116
117     def move(self, dx, dy):
```

```
118         self.x += dx
119         self.y += dy
120 
121     def attack(self, pts):
122         self.health -= pts
123 
124 p1 = Player(1, 2)
125 p2 = Player(3, 4)
126 p3 = Player(5, 6)
127 
128 print(p1.__dict__)
129 print(p1.__class__.__dict__)
130 print(Player.__dict__)
131 # Please note that __dict__ attribute is available only for custom classes an
132 # ============================================================================
133 class Point:
134     # a and b with default values
135     def __init__(self, a=0, b=0):
136         self.a = a
137         self.b = b
138 
139 p1 = Point()
140 p2 = Point()
141 # ============================================================================
142 class Employee:
143     def __init__(self, fname, lname, pay, *args):
144         self.fname = fname
145         self.lname = lname
146         self.pay = pay
147         self.args = args
148 
149 e1 = Employee('steve', 'jobs', 1000, 'python', 26, '2200 valley view lane')
150 # ============================================================================
151 # Overloading constructor using optional arguments
152 class Point:
153     def __init__(self, a=0, b=0, c=0):
154         self.a = a
155         self.b = b
156         self.c = c
157 
158 p1 = Point()
159 p2 = Point(1)
160 p3 = Point(1, 2)
161 p4 = Point(1, 2, 3)
162 # ============================================================================
```

```
# We can have multiple __init__ method's, but the latest implementation of __
class Point:
    def __init__(self, a, b):
        self.a = a
        self.b = b

    # Re-defining __init__ method (new implementation)
    def __init__(self, a, b, c):
        self.a = a
        self.b = b
        self.c = c

# Python maintains the information about methods in class dictionary.
# we can access the dictionary using Point.__dict__
# Method name will be the key of the dictionary and the reference of the meth
# e.g.
# >>> Point.__dict__
# >>> mappingproxy({'__module__': '__main__', '__init__': <function Point.__i
# '__weakref__': <attribute '__weakref__' of 'Point' objects>, '__doc__': Non

# when you have multiple method's with the same name, the key of the class di
# So the new reference of the function object (with new implementation) would
# Point.__dict__
# >>> mappingproxy({'__module__': '__main__', '__init__': <function Point.__i
# '__weakref__': <attribute '__weakref__' of 'Point' objects>, '__doc__': Non
# ========================================================================
```