




Topic	DICTIONARIES & CLASSES	
Class Description	The student will learn to use the Dictionary data type. The student will revisit the Object Oriented Programming style and will build Classes and Objects in Python.	
Class	PRO C100	
Class time	45 mins	
Goal	<ul style="list-style-type: none"> <li>• Introduction to Dictionary data type.</li> <li>• Learn about Classes and Objects.</li> <li>• Learn to create a Phonebook application.</li> </ul>	
Resources Required	<ul style="list-style-type: none"> <li>• Teacher Resources:               <ul style="list-style-type: none"> <li>○ Laptop with internet connectivity</li> <li>○ Earphones with mic</li> <li>○ Notebook and pen</li> <li>○ Smartphone</li> </ul> </li> <li>• Student Resources:               <ul style="list-style-type: none"> <li>○ Laptop with internet connectivity</li> <li>○ Earphones with mic</li> <li>○ Notebook and pen</li> </ul> </li> </ul>	
Class structure	<b>Warm-Up</b> <b>Teacher-led Activity</b> <b>Student-led Activity</b> <b>Wrap-Up</b>	<b>05 mins</b> <b>15 mins</b> <b>20 mins</b> <b>05 mins</b>
WARM-UP SESSION - 5 mins		
<div>  </div> <p><b>Teacher Starts Slideshow</b></p> <p><b>Slide 1 to 3</b></p> <p>Refer to speaker notes and follow the instructions on each slide.</p>		
Hey <student's name>. How are you? It's great to see you! Are you excited to learn something new today?		<b>ESR:</b> Hi, thanks! Yes I am excited about it!

<b>Following are the WARM-UP session deliverables:</b> <ul style="list-style-type: none"> <li>• Greet the student.</li> <li>• Revision of previous class activities.</li> <li>• Quizzes.</li> </ul>	Click on the slide show tab and present the slides
<b>WARM-UP QUIZ</b> Click on In-Class Quiz	
<div> <div>Continue WARM-UP Session</div> <div>Slide 4 to 18</div> <div></div> </div>	
<b>Following are the session deliverables:</b> <ul style="list-style-type: none"> <li>• Appreciate the student.</li> <li>• Narrate the story by using hand gestures and voice modulation methods to bring in more interest in students.</li> </ul>	
<div> <div>Teacher Ends Slideshow</div> <div></div> </div>	
<b>TEACHER-LED ACTIVITY 1- 15 mins</b>	
<b>Teacher Initiates Screen Share</b>	
<div> <div><u>ACTIVITY</u></div> <ul style="list-style-type: none"> <li>• Introduce Dictionary</li> <li>• Introduce syntax of a Class</li> </ul> </div>	
Teacher Action	Student Action
You did well!  Where else have we used a for loop in Python so far? Yes!  Do you remember the purpose of using a list?	<b>ESR:</b> While using the <b>list</b> .  <b>ESR:</b> To store multiple data.

<p>Correct!</p> <p>Similar to the list, there is another datatype known as Dictionaries which helps us to store multiple values.</p> <p>In today's class, we will learn about the dictionaries data type and will also understand the syntax of creating a class in Python.</p> <p>Are you excited?</p> <p>Let's get started.</p>	<p><b>ESR: Yes</b></p>
<p><i>Start a Google Colab Notebook using <a href="#">Teacher Activity 1</a>.</i></p> <p><i>The teacher writes the following piece of code into the code cell of the Google Colab.</i></p>	
<div data-bbox="162 1081 503 1249" data-label="Code-Block"> <pre>[ ] person = {     }</pre> </div>	
<p>Which data type do you think it will return?</p> <p>Yes, the syntax is the same as JSON objects, but in Python, it is known as <b>Dictionary</b>.</p> <p>Do you remember which function can be used to check the type of a variable in Python?</p> <p>Let us check the <b>type()</b>.</p>	<p><b>ESR: JSON</b></p> <p><b>ESR: We use <code>type()</code></b></p>

```
[ ] type(person)
```

```
dict
```

The type **dict** stands for **Dictionary** data type.

Similar to a list, a **dictionary** is a collection of many values.

A **dictionary** contains **key-value** pairs.

Each “key-value” pair maps the **key** to its **value**.

### Define a Dictionary:

A dictionary is defined using **curly** brackets, { }, to enclose its elements “**key-value**” pairs.

### Syntax:

```
x = {  
    <key_1>: <value_1>,  
    <key_2>: <value_2>,  
    .  
    .  
    .  
    <key_N>: <value_N>  
}
```

**Note:** The text between < > represents the user defined variable names.

The “**key-value**” pair is formed by a **colon (:)** as a separator between the **key** and respective **value**. Multiple “**key-value**” pairs are added separated by **comma(,)**.

Let’s understand that by an example:

1. Create a **dict** variable **person** to include the contact information.
2. Take “name”, “mobile”, “email”, “address” as keys and assign values of the respective keys.

```
[ ] person = {
    "name": "Darsh"
    "mobile" : 9029190909
    "email": "dummy1email@gmail.com",
    "address": "Sweet Home, Lane, City"
}
```

Can you tell me in the code, which ones are the key and which ones are the value?

Perfect!

For example, the “**name**” is a **key** and “**Darsh**” is the **value**.

We can print the whole dictionary using <dict\_name>. Let’s print the “**persons**” dictionary using **print()** method.

**Note:** The text between < > represents the user defined variable names.

**ESR:** The text on the left side of colon: is key and right side are value.

```
[4] person = {  
    "name": "Darsh",  
    "mobile" : 9029190909,  
    "email": "dummy1email@gmail.com",  
    "address": "Sweet Home, Lane, City"  
}  
  
print(person)  
  
{'name': 'Darsh', 'mobile': 9029190909, 'email': 'dummy1email@gmail.com', 'address': 'Sweet Home, Lane, City'}
```

That was easy, wasn't it?

**ESR:** Yes.

### Accessing Dictionary Element Values:

We can access any one element value of the dictionary, we can use the name of the **key** in a pair of square brackets [ ].

#### Syntax:

```
<dict_name>["<key_name>"]
```

**Note:** The text between < > represents the user defined variable names.

Let's print the value of **"name"** in the **"persons"** dictionary using **print()** method.

We will get the output as **"Darsh"**.

```
person = {
    "name": "Darsh",
    "mobile" : 9029190909,
    "email": "dummy1email@gmail.com",
    "address": "Sweet Home, Lane, City"
}
```

```
print(person["name"])
```

Darsh

### Revisiting Object-Oriented Programming:

Before we can proceed further, can you tell me what Object-oriented programming is and in which games/apps we used the concept of Object-oriented programming?

*A teacher can help the student remember how we used Object-oriented programming in JavaScript Game Applications and React Native Mobile Apps.*

Object-oriented programming is a way of programming where the similar properties and behaviors are grouped into individual objects.

Python is an object-oriented programming language. Object-oriented programming focused on objects.

#### Object:

An object is simply a collection of properties (variables) and methods (functions) to work on those properties.

#### Class:

A class is a blueprint for that object, where all the properties and methods are created.

**ESR:** Varied

We can think of a class as a sketch (blueprint) of a car. It contains all the details about the car material, tyres, size, engine etc.

Based on these descriptions, we build the car. Car is an object. As many cars can be made from a single blueprint of a car, we can create many objects from a class. An object is also called an instance of a class.

### Classes and Objects in Python:



Today, we will be learning how to create Classes and Objects in Python. Let us understand the syntax of a class.

#### Define a Class:

We use a keyword **class** and the name of the class ending with a colon(:) to define a class in Python.

#### Syntax:

```
class className:
    statement(s)
```

DEFINING CLASS	
	
<pre>class className:     #Code Here</pre>	<pre>class className{     //CODE HERE }</pre>



## Class Attributes and Methods:

**Attributes:** Attributes are the properties defined in the class. Each object will have common properties as defined in the class, but their value might change for each object.

E.g. Colors, dimensions, material of a Smartphone could be possible the attributes

**Methods:** User defined functions in the classes are called methods, which identify the behaviors and actions that an object created from the class can perform with its data.

E.g. Calling, Video Chat, Clicking pictures are behaviors/ functionalities of Smartphones.

### DEFINING ATTRIBUTES & METHODS OF A CLASS



```
class className:

    def __init__(self, attribute1, attribute2):

        self.a1 = attribute1
        self.a2 = attribute2

    def show_attributes(self):
        print(self.a1)
        print(self.a2)
```

```
class className {

    constructor(property1, property2) {

        this.p1 = property1;
        this.p2 = property2;

    }

    showProperties() {
        console.log(this.p1)
        console.log(this.p2)
    }

}
```

## Defining Class Attributes:

- The **\_\_init\_\_()** method is used to define the attributes in a class.

- The `__init()` method is written with **two underscores**(`_`) at the beginning and at the end without any spaces in between.

**Syntax:**

```
def __init__(self, attribute1, attribute2):  
    statement(s)
```

The first parameter of the `__init()` method is always reserved as the object of the class.

This is conventionally called **self**, but we can use any name of the first parameter.

Along with **self**, we can also pass other parameters required for the class like attribute1 and attribute2.

- The `__init()` method allows the class to initialize the attributes of the class(also called as class variables) using parameters.

**To initialize attributes of the class:**

We pass can assign the parameters to the attributes using:

```
self.<attribute_name> = <parameter_name>
```

**Note:** The text between `< >` represents the user defined variable names.

The actual values of the parameter values are called arguments, and need to be passed when we create objects of the class to initialize the attributes with parameter values.

We do not need to pass the value of the “self” parameter. This parameter gets automatically initialised with the object of the class.

- The `__init__()` method gets automatically called when an object is created.

```
class className:  
  
    def __init__(self, attribute1, attribute2):  
  
        self.a1 = attribute1  
        self.a2 = attribute2
```

### Defining Class Methods:

- The functions defined inside the class are called methods of the class.
- We can define methods in the class like we define functions in Python using the `def` keyword.
- When you create a method inside a class, you have to always pass at least one argument, a special argument that is “self” which represents the instance/object of the class.

```
def show_attributes(self):  
    print(self.a1)  
    print(self.a2)
```

## Creating Class Objects:

### Syntax:

**When `__init__()` methods has NO parameters:**

```
object_name = ClassName()
```



OR

**When `__init__()` methods has parameters:**

```
object_name = className(argument1, argument2)
```

- When an object is created for a Python class, it will call the `__init__()` method of **class**.
- When the `__init__()` method is defined with the parameters, the actual values of the parameter values are called arguments, and need to be passed when we create objects of the class to initialize the attributes with parameter values.
- We do not need to pass the value of the “self” parameter. This parameter gets automatically initialised with the object of the class.

**Note:** *If we don't specify the **self** argument inside the `__init__()` function. It will create an error, thus the object will not be created for this class due to which we won't be able to access the attributes and functions of the class code.*

CREATING AN OBJECT OF A CLASS	
	
<code>object_name = className()</code>	<code>object_name = new className()</code>

<p><b>Phonebook:</b></p> <p>Do you know how contact details of a person are added to a Phone?</p>     <p>Great!</p> <p>Can you tell what is common for every person's contact details?</p>   <p>Perfect!</p> <p>Suppose we would want to add these contact details to a phone, instead of writing a script of every person's contact details we should be creating a blueprint of a person's contact details properties and functionalities, right?</p> <p>Let's create a class for a person's contact details which will include contact information about all your friends.</p> <p><i>The teacher creates a class named <b>Contact_Details</b>.</i></p>	<p><b>ESR:</b></p> <ul style="list-style-type: none"> <li>• We open up the contact app on the mobile phone.</li> <li>• We can add a new contact with the person's name and phone number.</li> </ul> <p><b>ESR:</b> Every person has a name, phone number, address.</p>     <p><b>ESR:</b> Yes.</p>
---	---

```
# Define Class
class Contact_Details:

    # Define Attributes of class

    # Define Methods of class
```

Can you tell me what the attributes of the **Contact\_Details** class could be?

ESR: Name, Mobile number.

Let's define the attributes of the class:

1. Take **name**, **mobile\_number**, **email**, and **address** along with **self** as the parameters of **\_\_init\_\_()** method of the class.
2. Take **contact\_name**, **contact\_number**, **email**, **contact\_address** as the attributes of the class(also called variables of the class).
3. Assign the parameters to the attributes.

We can assign the parameters to the attributes using:

`self.<attribute_name> = <parameter_name>`

**Note:** The text between **< >** represents the user defined variable names.

```
# Define Class
class Contact_Details:



    # Define Attributes of class
    def __init__(self, name, mobile_number, email, address):
        self.contact_name = name
        self.contact_number = mobile_number
        self.email = email
        self.contact_address = address
```

4. Take a dictionary variable, **person** as the attribute of the class, with "name", "mobile", "email", "address" as the keys.
5. Add other attributes of the class as the value to the keys.

```
# Define Class
class Contact_Details:

    # Define Attributes of class
    def __init__(self, name, mobile_number, email, address):
        self.contact_name = name
        self.contact_number = mobile_number
        self.email = email
        self.contact_address = address

    # Make a dictionary of the contact details
    self.person = {
        "name": self.contact_name,
        "mobile" : self.contact_number,
        "email": self.email,
        "address": self.contact_address
    }
```

<p>We created a class to accept the person's details. We also created a dictionary to assign all the values for one person to one variable.</p> <p>How do we access this class now?</p> <p>Yes, you can create an object for this class, and you will also write a method for this class to add contact details and display the details.</p> <p>Are you excited?</p>	<p><b>ESR:</b> By creating an object for the class.</p> <p><b>ESR:</b> Yes.</p>
<p><b>Teacher Stops Screen Share</b></p>	
<p>So now it's your turn. Please share your screen with me.</p>	
<p><b>Teacher Starts Slideshow</b> </p> <p><b>Slide 19 to 20</b></p> <p>Refer to speaker notes and follow the instructions on each slide.</p>	
<p>We have one more class challenge for you. Can you solve it?</p> <p>Let's try. I will guide you through it.</p>	
<p><b>Teacher Ends Slideshow</b> </p>	
<p><b>STUDENT-LED ACTIVITY - 20 mins</b></p>	
<ul style="list-style-type: none"> <li>• <b>Ask the student to press the ESC key to come back to the panel.</b></li> <li>• <b>Guide the student to start Screen Share.</b></li> <li>• <b>The teacher gets into Fullscreen.</b></li> </ul>	



<b>ACTIVITY</b> <ul style="list-style-type: none"> <li>• Create an object for a class to create a PhoneBook.</li> <li>• Write methods to add and view contact details.</li> </ul>	
Teacher Action	Student Action
<p><i>Guide the student to open and make a copy Google Colab Notebook using <a href="#">Student Activity 1</a>.</i></p> <p><i><b>Note:</b> Let the student go through the code and ask if there is any doubt.</i></p> <p>The <b>Contact_Details</b> class is defined for you. What should we do now?</p> <p><u>The class is just a blueprint/design of contact details of a person that is possible, we haven't actually added the person details!</u></p> <p>Now we will be using blueprint/design to add the details of a person.</p> <p>Remember how we can use class blueprint/design once we define the class?</p> <p>Yes. Perfect!</p> <p>Object represents the actual entity made using the class design.</p> <p>Let's create an object of the <b>Contact_Details</b> class.</p>	<p><i>The student opens the <a href="#">Student Activity 1</a> and makes a copy of the Notebook.</i></p> <p><b>ESR:</b> Varied.</p> <p><b>ESR:</b> We can create an object for the class.</p>
<p>Do you remember how to create the object?</p> <p>Great!</p> <p>Let's take a variable <b>new_contact</b> and create an object of the class <b>Contact_Details</b>.</p>	<p><b>ESR:</b> We can create objects using the class name.</p>

```
new_contact = Contact_Details()
```

Now, let's run the code cell and check the output.

What did we get as the output?

**ESR:** We got an error:  
Missing 4 required positional arguments: 'name', 'mobile\_number', 'email', and 'address'.

Yes!

Remember these are 4 parameters we defined in the `__init__()` method?

**ESR:** Yes.

We get this error because while we defined in the `__init__()` method, we added parameters, but did not pass them while creating an object.

We need to pass the actual values of these parameters to initialize the class attributes when we create the class object.

```
-----
TypeError                                 Traceback (most recent call last)
<ipython-input-4-3177535e7496> in <module>()
----> 1 new_contact = Contact_Details()

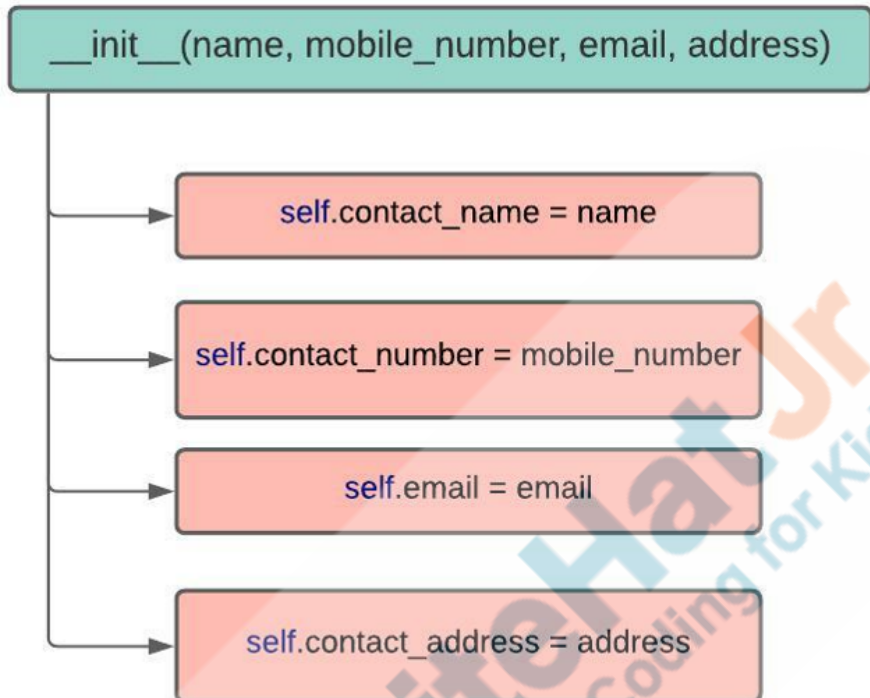
TypeError: __init__() missing 4 required positional arguments: 'name', 'mobile_number', 'email', and 'address'
```

Let's pass the actual values of the 'name', 'mobile\_number', 'email', and 'address' parameters while creating the object and check the output.

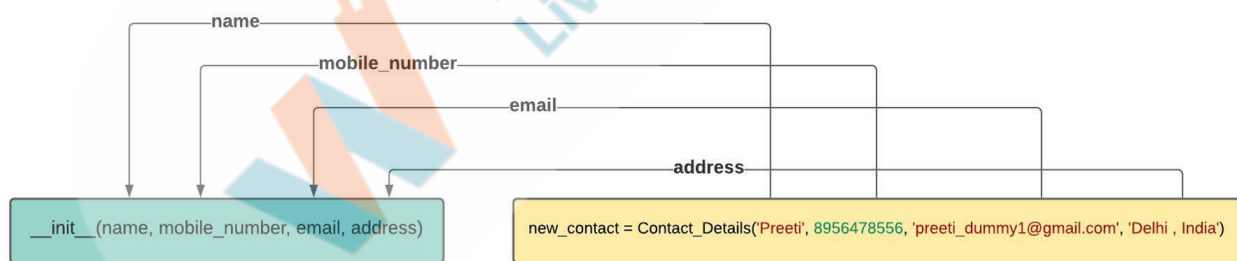
This time we will not get an error!

*Guide the student to add a new code cell and write the code. Make sure to use “ ” for str data type name, email,*

<b>address.</b>	
<pre>new_contact = Contact_Details('Preeti',8956478556,'preeti_dummy1@gmail.com', 'Delhi , India')</pre>	
<p>Now that we have created an object how can we see the details of the person added?</p> <p>Let's try to print the object!</p>	<p><b>ESR:</b> We can print the object.</p>
<pre>new_contact = Contact_Details('Preeti',8956478556,'preeti_dummy1@gmail.com', 'Delhi , India') print(new_contact)  &lt;__main__.Contact_Details object at 0x7f76be8f1c90&gt;</pre>	
<p>But we still can't see the details of the person added. The output only says that it is a <b>Contact_Details</b> object.</p> <p>Before we can view the details let's understand how the attributes are initialised with the arguments passed while we created objects.</p> <p><i>Open <a href="#">Teacher Activity 2</a> (or ask the student to open <a href="#">Student Activity 2</a> ) to show the image explaining the initialization of class attributes.</i></p> <p>We have 4 parameters in the <b>__init__()</b> method.</p>	



While creating an object every parameter has to be passed in the same order.



Each value passed gets assigned to the attributes inside the `__init__()` method.

```
new_contact = Contact_Details('Preeti', 8956478556, 'preeti_dummy1@gmail.com', 'Delhi , India')
```

```
self.contact_name = 'Preeti'
```

```
self.contact_number = 8956478556
```

```
self.email = 'preeti_dummy1@gmail.com'
```

```
self.contact_address = 'Delhi , India'
```

Now that we know how the values are mapped to the attributes, how can we see the details of these?

Amazing!

But, we assigned these attributes to a dictionary attribute called `person` in the `__init__()` method. So, instead of printing each attribute we should directly print the dictionary attribute, right?

```
# Make a dictionary of the contact details
self.person = {
    "name": self.contact_name,
    "mobile" : self.contact_number,
    "email": self.email,
    "address": self.contact_address
}
```

**ESR:** We should print all these attributes.

**ESR:** Yes.

Instead of directly printing the dictionary attribute we can define a method to add the display functionality.

Let's define a method to show the contact details of a person:

1. Take the method variable name,  
**view\_contact\_details.**

2. Pass the **"self"** parameter.

3. Print the person attribute.

Attributes can be printed using:

`self.<attribute_name>`

**Note 1:** The text between < > represents the user defined variable names.

**Note 2:** Make sure the code is intended properly.

```
# Define Class
class Contact_Details:

    # Define Attributes of class
    def __init__(self, name, mobile_number, email, address):
        self.contact_name = name
        self.contact_number = mobile_number
        self.email = email
        self.contact_address = address

    # Make a dictionary of the contact details
    self.person = {
        "name": self.contact_name,
        "mobile" : self.contact_number,
        "email": self.email,
        "address": self.contact_address
    }

    # Define Methods of class
    def view_contact_details(self):
        print(self.person)
```

### Accessing methods of the class:

We can access the method of the class using the object name and dot operator.

### Syntax:

```
<object_name>.<method_name>
```

**Note :** The text between < > represents the user defined variable names.

Let's call the method and see the output.

**Note 2:** Make sure to re-run all the code cells including where class is defined after updating the class definition.

```
# Call Methods of the class

new_contact.view_contact_details()

{'name': 'Preeti', 'mobile': 8956478556, 'email': 'preeti_dummy1@gmail.com', 'address': 'Delhi , India'}
```

Now this is just one person's contact details.

We can create multiple objects with new details every time for a different person.

Let's try to add a new person's contact details and call the method to check the details.

```
new_contact = Contact_Details('Shean',8956476333,'shean_dummy1@gmail.com', 'California , USA')

new_contact.view_contact_details()

{'name': 'Shean', 'mobile': 8956476333, 'email': 'shean_dummy1@gmail.com', 'address': 'California , USA'}
```

Well the output shows only the recent details, we cannot see the details of the person we added before this!

But, to create a phonebook we need to keep each person's contact details together.

Any ideas, how can we keep multiple records together?

We have used lists to store multiple records like numbers and strings in a sequence earlier. A list can also have a dictionary type. We can create a list which can keep the details of every person together.

**ESR: Varied.**



Let's write a method to add the contact details into a phonebook list:

1. Define a new method called **add\_contact\_details(<list\_name>)** for adding more records. We will use the list as a parameter of the function.
2. Pass the **contact\_list** as a parameter, along with the **"self"** parameter.
3. Use **append()** method to add the elements in the list inside the method.

The **append()** can be used to add elements in the list at the end.(Reference: [Teacher Activity 3](#) )

**Syntax:**

```
<list_name>.append(<element>)
```

**Note 1:** The text between < > represents the user defined variable names.

**Note 2:** Make sure the code is intended properly.

*The student can refer to [Student Activity 3](#)*

```
# Define Class
class Contact_Details:

    # Define Attributes of class
    def __init__(self, name, mobile_number, email, address):
        self.contact_name = name
        self.contact_number = mobile_number
        self.email = email
        self.contact_address = address

    # Make a dictionary of the contact details
    self.person = {
        "name": self.contact_name,
        "mobile" : self.contact_number,
        "email": self.email,
        "address": self.contact_address
    }

    # Define Methods of class
    def view_contact_details(self):
        print(self.person)

    # Add the contact details to the list
    def add_contact_details(self, contact_list):
        contact_list.append(self.person)
```

Once we add the method for adding persons details to the list, modify the **view\_contact\_details()** method as **view\_contact\_details(<list\_name>)** to print the **contact\_list** instead of the **person** attribute.

```
# Define Class
class Contact_Details:

    # Define Attributes of class
    def __init__(self, name, mobile_number, email, address):
        self.contact_name = name
        self.contact_number = mobile_number
        self.email = email
        self.contact_address = address

    # Make a dictionary of the contact details
    self.person = {
        "name": self.contact_name,
        "mobile" : self.contact_number,
        "email": self.email,
        "address": self.contact_address
    }

    # Define Methods of class
    def view_contact_details(self, contact_list):
        print(contact_list)

    # Add the contact details to the list
    def add_contact_details(self, contact_list):
        contact_list.append(self.person)
```

Now that we have defined the methods of the class, let's add each person's contact details to a list using the **add\_contact\_details(<list\_name>)** method of the class:

1. Take a variable, **phonebook\_list** outside the class.

**Note:** We need to create this list outside the class.

This is because, every time an object is created all

attributes of the class get updated with new values of the parameters. If we create a list as the class attribute inside the class, then the list will be updated every time only with the latest values of the parameters.

2. Create class objects with all the arguments.
3. Call the method to add contact details by passing **phonebook\_list** as the parameter.
4. Call the method to view contact details by passing **phonebook\_list** as the parameter.

**Note:** Make sure to re-run all the code cells including where class is defined after updating the class definition.

```
phonebook_list=[]

new_contact = Contact_Details('Preeti',8956478556,'preeti_dummy1@gmail.com', 'Delhi , India')

new_contact.view_contact_details(phonebook_list)

[{'name': 'Preeti', 'mobile': 8956478556, 'email': 'preeti_dummy1@gmail.com', 'address': 'Delhi , India'}]
```

We can see that the person's contact details are added to the list as a dictionary object.

Now let's try to add one more person's contact details to the list.

5. Repeat the Steps 2 to 4 above to add the one more person's contact details and check if the details of the second person is appended at the end of the **phonebook\_list**.

```
# Create the object with new values
new_contact = Contact_Details('Shean',8956476333,'shean_dummy1@gmail.com', 'California , USA')

# Add details
new_contact.add_contact_details(phonebook_list)

# View details
new_contact.view_contact_details(phonebook_list)
```

```
[ {'name': 'Preeti', 'mobile': 8956478556, 'email': 'preeti_dummy1@gmail.com', 'address': 'Delhi , India'}, {'name': 'Shean', 'mobile': 8956476333, 'email': 'shean_dummy1@gmail.com', 'address': 'California , USA'} ]
```

We can see that the second person's contact is also added in the list as a dictionary object.  
You did amazing work today!

It was really interesting how by Object oriented programming we could design a contact details blueprint and use it to create a phonebook!

Do you have any doubts?



As a challenge, why don't you try and add information about 5 more friends, and show me your **phonebook\_list** in the next class?




**ESR:** Varied

**ESR:** Sure

**Teacher Guides Student to Stop Screen Share**

**WRAP UP SESSION - 5 mins**

<div>  <p><b>Teacher Starts Slideshow</b> Slide 21 to 26</p> </div>	
<p><b>Activity details</b> Following are the <b>WRAP-UP</b> session deliverables:</p> <ul style="list-style-type: none"> <li>• Appreciate the student.</li> <li>• Revise the current class activities.</li> <li>• Discuss the quizzes.</li> </ul>	
<p><b>WRAP-UP QUIZ</b> Click on In-Class Quiz</p>	
<div>  <p><b>Continue WRAP-UP Session</b> Slide 27 to 32</p> </div>	
<p><b>Activity Details</b> Following are the session deliverables:</p> <ul style="list-style-type: none"> <li>• Explain the facts and trivia</li> <li>• Next class challenge</li> <li>• Project for the day</li> <li>• Additional Activity (Optional)</li> </ul>	
<p><b><u>FEEDBACK</u></b></p> <ul style="list-style-type: none"> <li>• <b>Appreciate and compliment</b> the student for trying to learn a difficult concept.</li> <li>• <b>Get to know how they</b> are feeling after the session.</li> <li>• <b>Review and check their understanding.</b></li> </ul>	
Teacher Action	Student Action
You get Hats off for your excellent work!	<i>Make sure you have given at least 2 Hats Off during the class for:</i>

<p>In the next class, we will install Python to run on a local machine. We will also get introduced to a few modules of Python.</p>	<div>Creatively Solved Activities  +10</div> <div>Great Question  +10</div> <div>Strong Concentration  +10</div>
<p align="center"><b>PROJECT OVERVIEW DISCUSSION</b> Refer the document below in Activity Links Sections</p>	
<p align="center"> <b>Teacher Clicks</b> <span>✕ End Class</span> </p>	

ACTIVITY LINKS		
Activity Name	Description	Link
Teacher Activity 1	Google Colab Notebook	<a href="#">Google Colaboratory</a>
Teacher Activity 2	Class Attributes Initialization	<a href="https://s3-whjr-curriculum-uploads.whjr.online/bc739d05-d46e-4f10-be1d-d1013d186af0.png">https://s3-whjr-curriculum-uploads.whjr.online/bc739d05-d46e-4f10-be1d-d1013d186af0.png</a>
Teacher Activity 3	Reference Code	<a href="https://colab.research.google.com/drive/1wC_s0hG1NPRp7G_HXBJxo31YXKa_p8pCL?usp=sharing">https://colab.research.google.com/drive/1wC_s0hG1NPRp7G_HXBJxo31YXKa_p8pCL?usp=sharing</a>
Student Activity 1	Student Activity Boilerplate	<a href="https://colab.research.google.com/drive/13WkTKY63GC2Dx33nwnDypW0KZYwLOmIY?usp=sharing">https://colab.research.google.com/drive/13WkTKY63GC2Dx33nwnDypW0KZYwLOmIY?usp=sharing</a>
Student Activity 2	Class Attributes Initialization	<a href="https://s3-whjr-curriculum-uploads.whjr.online/bc739d05-d46e-4f10-be1d-d1013d186af0.png">https://s3-whjr-curriculum-uploads.whjr.online/bc739d05-d46e-4f10-be1d-d1013d186af0.png</a>

Teacher Reference 1	Reference To Append Method.	<a href="#">5. Data Structures — Python 3.9.6 documentation</a>
Teacher Reference 2	Project Document	<a href="https://s3-whjr-curriculum-uploads.whjr.online/bd360c83-1772-410f-b523-232cf6ee9eb0.pdf">https://s3-whjr-curriculum-uploads.whjr.online/bd360c83-1772-410f-b523-232cf6ee9eb0.pdf</a>
Teacher Reference 3	Project Solution	<a href="https://colab.research.google.com/drive/1M3VSINM-IT0mj4CsTUU6UETF5d7kO093?usp=sharing">https://colab.research.google.com/drive/1M3VSINM-IT0mj4CsTUU6UETF5d7kO093?usp=sharing</a>
Teacher Reference 4	Visual-Aid	<a href="https://s3-whjr-curriculum-uploads.whjr.online/82f434f9-9504-44c6-bdf7-d20229cc8459.html">https://s3-whjr-curriculum-uploads.whjr.online/82f434f9-9504-44c6-bdf7-d20229cc8459.html</a>
Teacher Reference 5	In-Class Quiz	<a href="https://s3-whjr-curriculum-uploads.whjr.online/e1a12c7e-057f-468a-b770-badf3a8b82b7.pdf">https://s3-whjr-curriculum-uploads.whjr.online/e1a12c7e-057f-468a-b770-badf3a8b82b7.pdf</a>
Student Reference 1	Reference To Append Method	<a href="#">5. Data Structures — Python 3.9.6 documentation</a>