**Encapsulation in Python:-**

Capsule--->

🡪Grouping/combining data member and member function in single unit is known as Encapsulation.

🡪Restrict access to method and variable

🡪Prevent data from direct modification

Advantage:

1-Security

2-Enhancement will be easy{without affecting outside we can able to perform any type of change internally }

3-Maintainability

4-Modularity

Disadvantage:

1-Time consuming process slows the speed

2-Username/password/OTP

Real example for encapsulation

Consider a real-life example of encapsulation, in a company, there are different sections like the accounts section, finance section, sales section etc. The finance section handles all the financial transactions and keeps records of all the data related to finance. Similarly, the sales section handles all the sales-related activities and keeps records of all the sales. Now there may arise a situation when for some reason an official from the finance section needs all the data about sales in a particular month. In this case, he is not allowed to directly access the data of the sales section. He will first have to contact some other officer in the sales section and then request him to give the particular data. This is what encapsulation is. Here the data of the sales section and the employees that can manipulate them are wrapped under a single name “sales section”. Using encapsulation also hides the data. In this example, the data of the sections like sales, finance, or accounts are hidden from any other section.

**\_\_ private variable**

Type Description

private variables 🡪 Accessible only in their own class starts with two underscore

private methods 🡪 Accessible only in their own class or by a method defined with two underscore

class Computer:

def \_\_init\_\_(self):

self.\_\_maxprice = 900

def sell(self):

print("Selling Price: {}".format(self.\_\_maxprice))

def setMaxPrice(self, price):

self.\_\_maxprice = price

c = Computer()

c.sell()

# change the price

c.\_\_maxprice = 1000

c.sell()

# using setter function

c.setMaxPrice(1000)

c.sell()

**Output:**

Selling Price: 900

Selling Price: 900

Selling Price: 1000

9511113102

**HTML**

**Forms:**

Every data goes in backend as string.

For input elements we should define label that means the box will be associated with the label.

<label for="">Enter name</label>

<input id="uname" type="text" name="username" value="enter name">

Mandatory fields:

< input type="...." ..... required>

Placeholder vs value:

<placeholder ="Enter name">

Program for form :-

<html >

<head>

  <link rel="stylesheet" href="styles.css">

</head>

<body>

    <h1 align ="center" >Welcome to informatics</h1>

    <table  align="center" cellspacing="0" cellpading="0">

        <form action="submit\_to.html">

           <tr>

              <div class="block"> <td> <label for="n">Name</label></td><td></td></div>

               <td><input id="n" type="text" name="first\_name"  placeholder="Enter your name></td>

           </tr>

           <tr>

              <div class="block"><td> <label for="em">Email</label></td><td></td></div>

              <td><input id="em" type="email" name="em\_ail"  placeholder="Enter your mail account" required></td>

           </tr>

            <tr>

                <div class="block"><td><label for="f\_name">Father's Name</label></td><td></td></div>

                <td><input id="f\_name" type="text" name="father's\_name"  placeholder="Enter your father's name"></td>

            </tr>

            <tr>

                <td><label for="qual">Qualification</label></td><td></td>

                <td><input id="qual" type="text" name="qualification"  placeholder="Enter your qualification's"></td>

            </tr>

            <tr>

                <td>Gender</td>

                <td></td>

                <td><label for="male">Male</label>

                <input id="male" type="radio" name="gender" >

                <label for="fmale">Female</label>

                <input id="fmale" type="radio" name="gender" >

                </td>

            <tr>

                <td></td>

                <td></td><td><input type="submit"  placeholder="submit"></td>

            </tr>

        </form>

    </table>

</body>

</html>

.h1

{

font-color:red;

}

block.label

{

font:bold;

font-size: 65;

}

Output:

**Welcome to informatics**

|  |  |  |
| --- | --- | --- |
| Name |  |  |
| Email |  |  |
| Father's Name |  |  |
| Qualification |  |  |
| Gender |  | Male  Female |
|  |  |  |

Dropdown :

<select name=’state’>

<option value=”1”>Value</option>

<option value=”1”>Value</option>

<option value=”1”>Value</option>

<option value=”1”>Value</option>

<option value=”1”>Value</option>

</select>

**CSS**

CSS is of 3 types:

**1-Inline CSS**

<h1 style=”color:red”>Welcome</h1>

**2-Internal CSS:**

<head>

<style type=”text/css”>

h1{

color: red ;

}

</style>

</head>

**3-External CSS**

**<head>**

<link rel=”stylesheet” href=”style1.css”>

**</head>**

style1.css

h1{

color: blue;

}

**Points to revise**

Functions types

Lambda,map,filters

Decorators,Generator,Iterator

Logging

**Oops**

Encapsulation -->Done/22/12/2020

Variables & Methods--> Done/22/12/2020

Super

Inheritance

Polymorphism

Abstraction(Abstract class & Interface)

Exception Handling

File Handling

**Django:**

To create a project command:

**django-admin startproject project\_name**

**django-admin startproject firstproject**

Inside the project a subfolder will be created and a file named manage.py will be created once the above code is run as shown below:

**firstproject**

**manage.py**

**firstproject**

**\_\_init\_\_.py –** It is used as a python package file.Initially it will be empty

**asgi.py –** Asynchronous Server Gateway Interface

**urls.py –** All urls is managed by this file

**wsgi.py--** webserver gateway interface /used when application is moved to production.

**settings.py** --All project level setting is managed by this file.(databases and all)

**How to create an app :**

Go inside the project directory and then we need to run the command:

**python manage.py startapp app\_name**

**python manage.py startapp testapp**

Once the app is created default files will be automatically generated inside the app as shown below

**manage.py**

**firstproject**

**\_\_init\_\_.py –** It is used as a python package file.Initially it will be empty

**asgi.py –** Asynchronous Server Gateway Interface

**urls.py –** All urls is managed by this file

**wsgi.py--** webserver gateway interface /used when application is moved to production.

**settings.py** --All project level setting is managed by this file.(databases and all)

**testapp**

**migrations**

**\_\_init\_\_.py --**

**admin.py --**

**apps.py --**

**models.py --**

**tests.py --**

**views.py --**

Go to the directory where the file manage.py is there in command prompt then run the below command:

**python manage.py runserver**

**View**

**http request**

**http response**

**Class based**

**Function based**

**Views.py**

We need to import the HttpResponse from Django.http

from django.http import HttpResponse

Instead of request we can use any argument like r, req based on your mood

def hello\_world\_view(request):

    return HttpResponse('<h1>This is response from django application</h1>')