**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;

}

CSS:

IST WAY:-

Color:red;

2nd WAY:-

color: rgb(255,255,255)🡪white

color: rgb(0,0,0)🡪black

3rd WAY:-

Hexadecimal:-

color:#10ff33

rgba a is alpha for transparency

color: rgba(255,22,122,0.1)

Range is ( 0.1-1.0 )

img{

border : blue 20px solid;

}

body{

background: url();

background-repeat: no-repeat;

background-size: cover;

}

**CSS basic selectors:-**

1-Elementary Selectors

2-Id Selectors

Use #

<h1 id="spl" align="center">Welcome to new learnING INdIA</h1>

#spl{

    color:green

}

3-Class Selectors

.class\_name{

}

<h1 class="new" align="center">Welcome to new learnING Usa</h1>

.new {

    color:blue;

}

Advanced CSS Selectors:

1-\*

2-Descendent

3-Adjacent

4-Attribute

5-Nth of type selector

<html>

<head>

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

</head>

<body>

<h1>Advanced Css Selectors</h1>

<a href="www.google.com">Click here to go to Google</a>

<ul>

<li>DOG</li>

<li>RAT</li>

<li>MOUSE</li>

</ul>

<h4>List of top movies</h4>

<ul>

<li>Bahuballi</li>

<li>KGF</li>

<li>Parmanu</li>

</ul>

<h4>List of top websites</h4>

<ul>

<li><a href="www.google.com">Flipkart</a></li>

<li><a href="www.snapdeal.com">Snapdeal</a></li>

<li><a href="www.amazon.com">Amazon</li>

<li><a href="www.twitter.com">Twitter</a></li>

</ul>

</body>

</html>

1-\* Selector

\*{

color:blue

}

Output:

**Advanced Css Selectors**

[Click here to go to Google](file:///D:\web-techy\www.google.com)

* DOG
* RAT
* MOUSE

**List of top movies**

* Bahuballi
* KGF
* Parmanu

**List of top websites**

* [Flipkart](file:///D:\web-techy\www.google.com)
* [Snapdeal](file:///D:\web-techy\www.snapdeal.com)
* [Amazon](file:///D:\web-techy\www.amazon.com)
* [Twitter](file:///D:\web-techy\www.twitter.com)

2-Descendent Selectors:

li  a{

background: blue;

color: red;

}

Output:

**Advanced Css Selectors**

[Click here to go to Google](file:///D:\web-techy\www.google.com)

* DOG
* RAT
* MOUSE

**List of top movies**

* Bahuballi
* KGF
* Parmanu

**List of top websites**

* [Flipkart](file:///D:\web-techy\www.google.com)
* [Snapdeal](file:///D:\web-techy\www.snapdeal.com)
* [Amazon](file:///D:\web-techy\www.amazon.com)
* [Twitter](file:///D:\web-techy\www.twitter.com)

**Attribute Selector:-**

a[href]{

    color:red;

}

**Output:**

**Advanced Css Selectors**

[Click here to go to Google](file:///D:\web-techy\www.google.com)

* DOG
* RAT
* MOUSE

**List of top movies**

* Bahuballi
* KGF
* Parmanu

**List of top websites**

* [Flipkart](file:///D:\web-techy\www.google.com)
* [Snapdeal](file:///D:\web-techy\www.snapdeal.com)
* [Amazon](file:///D:\web-techy\www.amazon.com)
* [Twitter](file:///D:\web-techy\www.twitter.com)

**Font and text in CSS:-**

1-font-family 🡪 cssfontstack.com

2- font-size: x px

3- font-weight 🡪 font-weight:100-900;

4-font-height:1.5 🡪the space between two line

5-text-align 🡪 text-align: center

6-text-decoration:line-through;

**Box Model:**

4 Edges

1-content

2-padding

3-border

4-margin

The space between content and border padding.

Margin

Padding

Hello

Content

**BOOTSTRAP**

Current version :5

getbootstrap.com 🡪website

**How to connect bootstrap with html:**

1-By using CDN(online)

2-Locally

**By using CDN:**

Content Delivery Network (Internet is required)

<!-- Latest compiled and minified CSS --> <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css" integrity="sha384-BVYiiSIFeK1dGmJRAkycuHAHRg32OmUcww7on3RYdg4Va+PmSTsz/K68vbdEjh4u" crossorigin="anonymous">

**Locally**

1-Download the bootstrap

2-Link in head:--<link rel=”stylesheet” href=”…….”>

**Buttons:**

**Jumbotron:**

**Form:**

**Form-group:** To maintain proper space between elements

**Navbar:** Navigation bar

<nav class="navbar navbar-default">

</nav>

How to add logo/brand:

<a href="https://durgasoftonline.com" class="navbar-brand"> Informatics</a>

**How to add remaining items we should use unordered list:**

**<ul>**

**</ul>**

**List-style:none; to remove the dots of list**

**Points to revise**

Functions types 🡪**Done 03/01/2020**

Lambda, map, filters 🡪**Done 22/12/2020**

Decorators, Generator 🡪**Done 31/12/2020**

Iterator

Pickling

Logging

Global , local

**Oops**

Encapsulation 🡪**Done/22/12/2020**

Variables & Methods 🡪 **Done/22/12/2020**

Super

Inheritance

Polymorphism

Abstraction(Abstract class & Interface)

Exception Handling

File Handling

**High Order Functions:**

#### Passing Function as an argument to other function

**Properties of higher-order functions:**

* A function is an instance of the Object type.
* You can store the function in a variable.
* You can pass the function as a parameter to another function.
* You can return the function from a function.
* You can store them in data structures such as hash tables, lists, …

def square(x):

return x\*x

def cal(fact):

print(fact)

cal(square(4))

**Decorator:**

def dec(func):

def inner(x,y):

print('The addition of a is:')

func(x,y)

return inner

@dec

def add(x,y):

print(x+y)

add(12,14)

**Generator:**

Here is how a generator function differs from a normal function.

• Generator function contains one or more yield statements.

• When called, it returns an object (iterator) but does not start execution immediately.

• Methods like \_\_iter\_\_() and \_\_next\_\_() are implemented automatically. So we can

iterate through the items using next().

• Once the function yields, the function is paused and the control is transferred to the caller.

• Local variables and their states are remembered between successive calls.

• Finally, when the function terminates, StopIteration is raised automatically on further calls.

**Mongo Db:-**

Humonguos 🡪 Extremely large

To handle extremely large amount of data

MongoDB is the most popular and trending database.

Vendor—MongoDB

MDB internally uses Mozilla’s spider monkey java script engine

Databases:

1-Relational database🡪Tables (Row & Column) Relation between data of the tables. Fixed schema/structure

2-No SQL/document database🡪

🡪Data will be stored in documents.

🡪Each document is independent from each other

🡪Documents are like row/record if compared in context of relational database.

**MongoDB structure:-**

MongoDB database contains several logical databases.

Each database contains several collections.

Collections contain several documents. Collection are like table in context of RDBMS

**How data represented in MongoDB?**

In Java script object notation [JSON].

**Features of MongoDB:-**

**1-Performance is good because retrieval is fast.**

**2-Schema less**

**3-**

**MongoDB shell/client vs MongoDB Server:-**

Once we install MongoDB we will get MongoDB shell and MongoDB server.

These are Javascript based applications

**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>')

**Urls.py**

from django.contrib import admin

from django.urls import path

from testapp import views

urlpatterns = [

    path('admin/', admin.site.urls),

    path('home/',views.show),

]

**Define your app in settings.py**

INSTALLED\_APPS = [

    'django.contrib.admin',

    'django.contrib.auth',

    'django.contrib.contenttypes',

    'django.contrib.sessions',

    'django.contrib.messages',

    'django.contrib.staticfiles',

    'testapp',

]

**How to define urls pattern to application level instead of project level:**

1-🡪Create a urls.py at application level.

2-🡪include the application level urls in project level

Create urls.py at application level:-

from django.contrib import admin

from django.urls import path

from testapp import views

urlpatterns = [

    path('admin/', admin.site.urls),

    path('home/',views.show),

    path('home1/',views.show1),

    path('home2/',views.show2),

    path('home3/',views.show3),

    path('home4/',views.show4),

]

Include this urls.py to project level:

from django.contrib import admin

from django.urls import path,include

from testapp import views

urlpatterns = [

    path('admin/', admin.site.urls),

    path('testapp/',include('testapp.urls')),

]

from django.urls import path, **include**

**And in path inside urlpatterns**

**path(‘application\_name’,include(‘application\_name.urls’)**

**Template:**

1-Create templates folder in the main project inside templates folder create a separate folder named application in that application folder we need to create our templates that is html files.

2-Add templates folder path to settings.py

print(\_\_file\_\_) 🡪 D:/PYTHON/uggjh.py

print(os.path.abspath(\_\_file\_\_)) 🡪 D:\PYTHON\uggjh.py

print(os.path.dirname(os.path.abspath(\_\_file\_\_))) 🡪 D:\PYTHON

**Programming:-**

import os

print(os.path.dirname(os.path.abspath(\_\_file\_\_))) To know current directory location

settings.py:

TEMPLATE\_DIR=os.path.join(BASE\_DIR,'templates')

TEMPLATES = [

    {

        'BACKEND': 'django.template.backends.django.DjangoTemplates',

        'DIRS': [TEMPLATE\_DIR],

**MVT**

There is no separate controller and complete application is based on Model View and Template. That’s why it is called MVT application.

See the following graph that shows the MVT based control flow.



Here, a user **requests** for a resource to the Django, Django works as a controller and check to the available resource in URL.

If URL maps, **a view is called** that interact with model and template, it renders a template.

Django responds back to the user and sends a template as a **response**.

**Working with static files:**

Static files are those files which are like fixed data which will not change in most of the cases.

Types of static files:

1-CSS

2-Javascript

3-Images

4-Fonts

There are 2 types of static directories:

1-Generic

2- Specific

1-Generic directory will be consisting of the static files that are related to the project.

2- Specific directory will be consisting of the static files that are related to the application level.

**Generic Static Directory**

Create a folder inside the main project named **static.**

**Go to settings.py:**

As we have done for the templates we need to do for static file also

STATIC\_DIR= os.path.join(BASE\_DIR,’static’)

After that create a list named **STATICFILES\_DIRS** below **STATIC\_URL=’/static/’**

**Note: STATICFILES\_DIRS this name must be this only any kind of change in the name will give error.**

**STATICFILES\_DIRS=[]**

Add the above STATIC\_DIR in the list as:

**STATICFILES\_DIRS=[** STATIC\_DIR,**]**

After that save a image in the static folder and start the server and type the url:

127.0.0.1:8000/static/image.extension

If image is displayed that means connection is working fine.

For image we use:

1-Load the html file using the Jinja tag:

**{% load static %}**

**2-Include in the html body:-**

**<img src=”{%static ‘path of the image’%}”>**

**<img src=”{%static ‘directory/image\_file\_name.extenstion%}”>**

**{% load static %}**

<html>

<body>

<h3>How to display image using static folder and Jinja Tags</h3>

<img src=" {% static 'image/a1.jpg'%} " alt="Not found" >

<img src=" {%static 'image/a1.jpg'%} " alt="Not found" >

<body>

</html>

If u use {% load staticfiles %} instead {% load static %} it will give error:

'staticfiles' is not a registered tag library

**Model**

Model is a class that represent one database table name fields and behaviour.

from django.db import models

class Employee(models.Model):

eno=models.IntegerField();

ename=models.models.CharField(max\_length=20)

esal=models.FloatField()

eaddr=models.CharField(max\_length=50)

makemigrations🡪convert python model class /code to sql code

migrate 🡪 how to execute the sql code

Run the command:

**makemigrations🡪**

python manage.py makemigrations.

Id is a default field/column that is autofield that acts as primary key

**migrate:**

python manage.py migrate🡪

inbuilt application

'django.contrib.admin',

    'django.contrib.auth',

    'django.contrib.contenttypes',

    'django.contrib.sessions',

    'django.contrib.messages',

    'django.contrib.staticfiles',

    'newsapp'

for every application table will be created.

To get the sql query that generated using makemigrations use the commands:

python manage.py sqlmigrate newsapp 0001

python manage.py sqlmigrate app\_name file\_name

**How to create superuser:**

python manage.py createsuperuser

We need to create superuser if you want to open the admin panel provided by Django:

**To display several parameters as table**

Write in admin.py

class model\_nameAdmin (admin.ModelAdmin):

list\_display=[‘eno’,’ename’,’esal’,’eaddr’]

admin.site.register(Employee,EmployeeAdmin)

Displaying data stored in database to End user

Select \* from table\_name --- Employee.object.all() 🡪 read all data from db

Fetch.html

html>

<body>

<h1>Employees Information</h1><hr>

{% if empl %}

<table border="2">

<thead>

<th>Employee Number </th>

<th>Employee Name</th>

<th>Employee Salary</th>

<th>Employee  Address</th>

</thead>

   {% for emp in empl %}

  <tr>

  <td>{{emp.eno}}</td>

  <td>{{emp.ename}}</td>

  <td>{{emp.esal}}</td>

  <td>{{emp.eaddr}}</td>

  </tr>

{% endfor %}

</table>

{% else%}

<p>No info</p>

{% endif %}

</body>

</html>

Url.py

from django.contrib import admin

from django.urls import path

from newsapp import views

urlpatterns = [

    path('admin/', admin.site.urls),

    path('home/',views.display),

    path('h2/',views.emp\_info),

]

Views.py

from django.shortcuts import render

from django.http import HttpResponse

from newsapp.models import Employee

# Create your views here.

def display(request):

    return render(request,'newsapp/result.html')

def emp\_info(request):

    employees=Employee.objects.all()

    return render(request,'newsapp/fetch.html',{'empl':employees})

models.py

from django.db import models

# Create your models here.

class Employee(models.Model):

    eno=models.IntegerField();

    ename=models.CharField(max\_length=20)

    esal=models.FloatField()

    eaddr=models.CharField(max\_length=50)

admin.py

from django.contrib import admin

from newsapp.models import Employee

# Register your models here.

class EmployeeAdmin(admin.ModelAdmin):

    list\_display=['eno','ename','esal','eaddr']

admin.site.register(Employee,EmployeeAdmin)