**ARRANQUE DEL PROYECTO**

1.-Crear la carpeta del proyecto, en cualquier lugar

2.-ejecutar en la consola c:\django-admin startproject <nombre del proyecto>

Django comienza hacer su trabajo

Se crea una carpeta con el nombre del proyecto, al abrir la carpeta vemos que se han creado una serie de archivos

Manage.py y una subcarpeta que contiene los archivos \_\_init\_\_.py settings.py urls.py y wsgi.py

El archivo manage.py nos mpermite interactuar de diversas formas en nuestro proyecto con django

3.-ejecutamos manage.py help nos muestra un listado de instructiones y todo lo que podemos hacer con ese archivo py

4.- archivo \_\_init\_\_.py es un archivo necesario para que Python trate el directorio del sitio como un paquete

Settings.py contiene las configuraciones de nuestro proyecto en django

Urls.py donde se almacenan las url de nuestro proyecto,

Wsgi.py es relativo a los servidores que se usaran el proyecto django

5.-INSTALLED\_APPS = [

‘Django.contrib.admin’ ---- la interfaz administrativa

‘Django.contrib.auth’, ----- sistema de autenticación

‘Django.contrib.contenttypes’ ----- framewors para manejar contenido

‘Django.contrib.sessions’, ----- frameworks para manejar mensajes

‘Django.contrib.staticfiles’ ------- frameworks para manejar archivos estáticos

]

6.-ejecutamos el comando Python manage.py migrate

A partir de este momento el proyecto esta ejecutando

7.- arrancamos el servidor con el comando Python manage.py runserver

Si todo esta bien vemos la línea con nuestra dirección local

Starting development server at <http://127.0.0.1:8000/>

**CREACION DE LA PRIMERA PAGINA**

8.-Crear archivo view.py en la subcarpeta del proyecto nuestro proyecto

9.-importamos el modelo Httprequest from django.http import HttpResponse

10.-creamos la función de lo que queremos ver con “request” como primer argumento en un archivo view

11.-agregamos las url en el archivo urls.py en ls tupla urlpatterns sin olvidar importar la función

12.-arancer el servidor

13.localhost/<dirección>

**PARAMETROS EN URL**

Contenido diamico

Uso de URL freindly,

Creamos una function y agregamos atributos además de request

Def calculateage(request, year):

En url patter, las url se considera texto, hay que pasarlo a enteros usando int:<atributo>

Si quieres un segundo parámetro, agregar a la función los parámetros y separar

En url patter los parámetros a introducir separados por /

**USO DE PLANTILLAS**

Se usa principalmente para separa la parte lógica de la parte visual

Se crea en 3 pasos

1.-Creación de objetos de tipo template plt=Template(doc\_externo.read())

2.-Creación de contexto ctx=Context()

Datos adicionales para el template(variable, funciones, etc)

Rendetrizar el objeto template

3.-Documento=plt.render(ctx)

1.- crear el archivo html

Regresar al archivo view de Python y en el método creamos una variable con la dirección del documento html

Creamos una variable de apertura de plantilla

Cerramos el documento con .close()

Creamos otra variable con el contexto Context()

**VARIABLES EN PLANTILLAS**

1.-Crear variable

2.-Agregar a la variable ctx agregar en los argumentos del valor de la variable un diccionario

3.-en el documento html agregar el valor de la variable entre {{ }}

**BUCLES Y CONDICIONALES**

**CARGADORES**

En el archivo views.py **from django.template import loader**, ya podemos usar este cargador con todos sus métodos

Especificar la carpeta donde están nuestras plantillas en el archivo settings.py “DIRS”[]

Pasar al método render() los argumentos en diccionarios

Para simplificarlo mas podemos importar

from django.template.loader import get\_template

**PLANTILLAS INCRUSTADAS**

Paquetes Shorcut

Importamos por from django.shorcuts import render

Y que nos retorne return render(request, <nombre de la plantilla>, diccionario)

Incrutamos plantillas

Se incluye en el archivo html principal con la sintaxis {% include <catpetas y nombre del archivo> %}

**HERENCIAS EN PLANTILLAS**

se hace con la siguiente sintaxis {% extends “padre.html” %}

registrar la vista en el archivo views.py

resgistrar la url en el archivo urls.py

**BASE DE DATOS 1**

Diferencias entre proyecto y aplicación

Creación de aplicación

Con el comando Python manage.py startapp <nombvre del proyecto>

En el archivo models.py se crea una clase por cada base de datos

Para crear la clase, class <Nombre de la clase> (models.Model):

Nombre=models.<tipo de dato>(<parámetros del tipo de dato>)

Comunicarle a django las tablas en el archivo settings.py

Para chequear con la siguiente sintaxis

Python manage.py check <nombre del proyecto>

Para crear las base de datos con la siguiente sintaxi

Python manage.py makemigrations

Para django para crear las tablas para la base de datos

Python manage.py sqlmigrate <nombre de la aplicacion> <numero de migracion>

Ahora hay que agregar las tablas creadas en la base de datos

Python manage.py migrate

Django crea in campo ID como clave primaria

**INSERTAR REGISTROS, ACTIALIZAR REGISTROS, BORRAR REGISTROS**

Abrimos el Shell

Python manage.py Shell

Importamos la aplicación

>>>from <nombre de la aplicación>.models import <nombre de la tabla a importar>

Para llenar la tabla

>>><nombre de la variable>=<nombre de la tabla>(rellenar los argumentos)

Para guardar los cambios ingresados

>>><nombre de la variable>.save()

Para ingresar datos a la tabla en la base de datos en un solo paso

>>><nombre de la variable>=<nombre de la tabla>.objects.create(argumentos)

Actualizar datos de la base de dato

>>><nombre de la variable original>.<nombre de la columna a actualizar>= <nuevo valor>

>>><nombre de la variable>.save()

Para Borrar campo

Para Borrar registro de una tabla

>>><nombre de la variable>=<nombre de la table>.objects.get(criterio del elemento a borrar)

>>><nombre de la variable>.delete()

Para consultar SQL del tipo Select

>>><nombre de la variable> = <nombre de la table>.objects.all()

>>><nombre de la variable>.query.\_\_str\_\_

**CONFIGURAR POSTGRESQL**

Instalar postgress

Click derecho sobre la base de datos en QueryTool, intruduces sintaxis SQL para crear base de datos y click en el boton del rayo

Se actualiza el panel de las bases de datos y al hacer click en la nueva base de datos se activa

Vincular nuestras tablas con la base de datos

Se necesita instalar la librería psycopg2 con el comando pip install psycopg2

Configurar el archivo setting.py con la nueva base de datos y agregar

'ENGINE':'django.db.backends.postgresql\_psycopg2',

'NAME':'CustumerArticle', <nombre de la base de datos>

'USER':'postgres',

'PASSWORD':'1234',

'HOST':'127.0.0.1',

'DATABASE\_PORT':'5432',

Después migrar nuestras tablas del proyecto a la base de datos con el comando

Python manage.py makemigrations

Después migrar todas las tablas a la base de datos

Python manage.py migrate

Al refrescar, se actualiza todas nuestras tablas a la base de datos postgress

Ingresamos datos en consola

Abrimos el Shell

From <nombre de la aplicación>.models import <nombre de la tabla>

Agregamos datos a la base de datos

<nombre de la variable>=<nombre de la tabla>(dato1=”valor dato”, dato2=’valor dato 2’)

<nombre de la variable>.save()

Se refresca en el gestor de la base de datos

**CONSULTA DE BASE DE DATOS CON CRITERIOS SELECT**

Abrir la Shell de django

Con la sintaxis

<nombre de la tabla>.objects.filter(<nombre de la columna>=’criterio de selecion’)

Pero para obtener un resultado legible debemos transforme los resultados a una cadena de caracteres con la función \_\_str\_\_()

Def \_\_str\_\_(self):

Return “%s, %s, %s”%(self.columna1, self.columna2, self.columna3)

Actualizar las migraciones

Python manage.py makemigrations

Python manage.py migrate

No importa a si indica que no hay cambios

Y para leer lo indicado en el método \_\_str\_\_

<nombre de la tabla>.object.filter(<nombre de la columna>=’<criterio de seleccion>’)

Si vamos a usar 2 criterios

<nombre de la tabla>.object.filter(<nombre de la columna1>=’<criterio de seleccion1, <nombre de la columna2>=’<criterio de selección2>’)

Si queremos usar selección por rangos numéricos

<nombre de la tabla>.object.filter(<nombre de la columna>=’<criterio de selección>’, <nombre de la columna>\_\_gte=’rango’) \_\_gte igual o mayor que \_\_lte igual o menor que

Para un rango de valores

\_\_range(valor menor, valor mayor)

Si se dese aun criterio de ordenación

<nombre de la tabla>.object.filter(<nombre de la columna>\_\_gte=<valor>).order\_by(<ordenar por valor de la columna>)

Si se desea ordenar en forma decendente <-ordenar por valor de la columna>

**PANEL DE ADMINISTRACION**

Se corre el servidor Python manage.py runserver

Abre la pagina de administrador 127.0.0.1/admin

Crear el super usuario con el comando Python manage.py createsuperuser

Se introduce nombre de usuario, correo, contraseña y confirma contraseña

Corre nuevamente el servidor

**MANIPULAR LAS TABLAS DEL PANEL DE ADMINISTRACION**

Abrir el archivo admin.py

Importamos la tabla from <nombre de la aplicacion>.models import <nombre de la tabla>

Registramos el modelo por

Admin.site.register(<nombre de la tabla>) nota: siempre se agrega al final una s

Todos los campos deben ser obligatorios, si se desea que un campo sea opcional en el archivo

Models.py

<nombre de la columna>.models.<tipo de dato>(blank=True, null=True)

Se hace makemigrate y migrate

**PANEL DE ADMINISTRACION III**

En el archivo models.py creamos el método def \_\_str\_\_(self):

Y que retorne

En el archivo admin.py creamos la clase

class <nombre de la clase>(admin.ModelAdmin):

List\_display=(<”campo1”>, <”campo2”>, <”campo3”>)

Campos que quieres que aparezcan en el panel de administrador

Administrar formas de buqueda de registros

En el archivo admin agregamos el artributo

Search\_fields = (<”campo1”>, <”campo2”>)