DJANGO :—

↳ Server side web framework
↳ MVT (Model view Template)
- Model → data structure + database
- View → Python function
- Template → HTML file + Django syntax.

✓ ORM
✓ Admin panel
✓ Authentication.
✓ DRY (Maintainable + reusable)
↳ Don't repeat yourself.

Virtual environment

✓ python -m venv myenv

Activate virtual environment

✓ myenv\Scripts\activate.bat

Install django

✓ pip install django

Verifying django

✓ python -m django version

enter into virtual environment

✓ cd myenv

create project

✓ django-admin startproject myproject
↳ project name.

folder structure

manage.py ⇒ running development server, creating apps.

myproject
↳ myproject
- ↳ settings.py ⇒ database setting, apps, middlewares.
- ↳ urls.py
- ↳ wsgi.py ⇒ deploying (web server gateway interface)
- ↳ asgi.py ⇒ asynchronous gateway interface
- ↳ init.py ⇒ tell that its python package.

enter into project

✓ cd myproject

running development server

✓ python manage.py runserver.

creating app

✓ python manage.py startapp myapp,
→ app name

myapp (views.py) ⇒ functions.

```
from django.http import HttpResponse
def hello(request):
    return HttpResponse("Hello, World!")
                        ("<h1> Welcome </h1>")
```

✓ Create a file (urls.py) inside myapp

```
from django.urls import path
from . import views
urlpatterns = [
    path('hello/', views.hello),
]
```

✓ (urls.py of myproject)

```
from django.urls import path, include.
urlpatterns = [
    path('', include(myapps.urls)),
]
```

views.py

```
def menuitems(request):
    items = {
        'pizza' : 'cost $10',
        'burger' : 'cost $5',
        'pasta' : 'cost $8'
    }
    content = '<h1> Menu Items </h1>'
    for item, cost in items.items():
        content += f'<li> {item} : {cost} </li>'
    return HttpResponse(content)
```

comments ⇒ # in python.

→ used for string concatenation.

URLs

- Static
- Dynamic
  - Route Parameter
  - Query Strings

Route parameters :-

views.py

```
def greet(request, name):
    return HttpResponse(f"Good Morning! {name}")
```

urls.py

```
path('greet/<str:name>/', views.greet);
```

<int: ___>

Query parameters :-

```
def addition(request)
    value1 = request.GET.get('num1')
    value2 = request.GET.get('num2')
    try:
        value1 = int(value1)
        value2 = int(value2)
    except ValueError:
        return HttpResponse("Enter valid numbers")
    result = value1 + value2
    return HttpResponse(result)
```

```
path('calculator/', views.addition)
```

Check at :-
/calculator/? num1=5 & num2=1

Regular Expressions → sequence of characters for search pattern

^ (caret) :- pattern match from beginning
$ (dollar) :- " " until end
/ (forward slash) :- act as separator
() (parentheses) :- encloses group of patterns
?P<name>
\d :- digit
\s :- spaces
[0-9]
[a-z A-Z]
[\w] :- alpha numeric character
[-] :- -, _
+ :- one or more matches.
* :- zero or more matches

Regular Expressions in Urls

urls.py

```
from django.urls import re_path
```

⇒ Check in urls.py file in my project

Error handling → 2 ways to handle errors.

- debug Mode
- Custom Error Views

debug Mode ⇓ settings.py ⇓

```
DEBUG = FALSE
ALLOWED_HOSTS = ["*"]
```

⇓ Shows Message "NOT found"

Custom Error Views

① Project level

Create `views.py` in myproject

```
from django.http import HttpResponse
def handler404(request, exception):
    return HttpResponse("<h1 style='color:red;'>
                        Dear user, page not exist </h1>
                        , status=404)
```

`urls.py` of project

```
handler404 = 'myproject.view.handler404'
```

myproject → Project name

② App level

↳ we mention in views.

- Check a condition, where an error might occur
- Return an error response manually.
- Specify the HTTP status code

e.g

```
If not subcategory:
    return HttpResponse("No subcategory", status=404)
```

↳ in a view ~~only~~ function.

Template

- HTML + Django template language
- use for generating dynamic web pages.
- we can use variables passed by views by {{ variable_name }}
- we can also use loops, conditional statements.
- loops, logics are written inside {% %}

Create a folder **templates** inside **myapp**

- in this folder, we will create html files e.g. home.html

settings.py ⇒

```
TEMPLATES = [
    {
        ...
        'DIRS': ['templates'],
        ...
    }
]
```

⇒

```
INSTALLED_APPS = [
    'myapp',
]
```

Basic e.g.

home.html | urls.py

e.g

```
def menus(request):
    items = [
        {'name': 'pizza', 'price': '$10'},
        .
        .
        .
    ]
    return render(request, 'menus.html',
                  {'items': 'items'})
```

↳ ∵ items is list, so we wrap it in dictionary.

menus.html

```
<ul>
  {% for item in items %}
    {% if item.price == 'free' %}
      <li> {{ item.name }} - {{ item.price }} </li>
    {% endif %}
  {% endfor %}
</ul>
```

for and if syntaxes

template inheritance

Base.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <title>
        {% block title %}
        {% endblock %}
    </title>

    <style>
        {% block css %}
        {% end block %}
    </style>
</head>
<body>
    {% include "restHeader.html" %}
    {% block content %}
    {% endblock %}
</body>
</html>
```

used to include other file.

Using Base.html in other files

```
{% extends Base.html %}

{% block title %}
    HOME PAGE
{% endblock %}

{% block css %}
    h1{ color : red; }
{% end block %}

{% block content %}
    <h1> This is Home </h1>
{% end block %}
```

{% url %} template tag

Syntax :- {% url 'RouteName' parameter = value %}

e.g.

① Home

urls.py → path ('Home/', views.home, name = 'Home'),

②

urls.py → path ('menu/<str:item_name>/', views.menu, name = 'Menu'),

Ways to work with forms

1. Returning HTML markup as HTTP response
2. HTML forms with Django templates
3. Django form
4. Model forms

CSRF
Cross-site Request Forgery
⇒ Security measure.

~~from django~~

(1) Returning HTML markup as HTTP response

```
from django.middleware.csrf import get_token

def simpleform(request):
    csrf_token = get_token(request)
    if(request.method == 'POST')
        textbox1 = request.POST.get('textbox1')
        textbox2 = request.POST.get('textbox2')
        return HttpResponse(f"You submitted {textbox1}, and {textbox2}");

    return HttpResponse(f'''
        <form method="POST">
        <input type="hidden" name="csrfmiddlewaretoken"
            value="{csrf_token}">
        <___>
        <___>
        <submit>
        </form>
```

name attributes of input fields in form

(2) HTML forms with Django templates

```
def templateform(request):
    if (request.method == 'POST'):
        name = request.POST.get('name')
        email = ''.''.''('email')
        Password = ''.''.''('Password')
        if name and email and password:
            return HttpResponse("Successful");
    return render(request, 'form.html')
```

name attributes of input fields in form.

'form.html' :-

```
<form method="POST">
    {% csrf_token %}
    ==
</form>
```

views.py
↳ Same as normal form.

(3) Django forms → build in feature.

* Create a separate file (forms.py)
  ↳ Here, we create Class

forms.py

```
from django import forms
class InputForm ((forms.Form)):
    name = forms.CharField(max_length=3)
    email = forms.EmailField()
    password = forms.CharField(widget=forms.PasswordInput())
```

→ class from django.forms module.

↳ used to hide passwords like [....]

* Create a (form1.html) in templates

form1.html

```
<form method="post" novalidate>
    {% csrf_token %}
    {{ form as_p }}
    <input type="submit" value="Submit">
</form>
```

novalidate → This attribute is used to validate Django forms

as_p → Paragraph tag

same variable used in views.py file to create instance of ~~Inputclass~~ ~~create~~ Inputform

Views.py

⇒ Here, we create function

forms.py → class we created in forms.py

```
from forms import InputForm

def form1(request):
    if (request.method == 'POST'):
        form = InputForm(request.POST)   # Creating instance of class of forms.py file.
        if (form.is_valid()):            # Check all fields are required and check their datatype
            return HttpResponse
            (f"Data submited is
            {form.cleaned_data['name']}")   # dictionary that contain validated form data
    else:
        return render(request,
                      form1.html,
                      {'form': form})   # The instance we created

    form = InputForm()   # first time rendering empty form by GET method.
    return render(request,
                  form1.html,
                  {'form': form})
```

urls.py

```
path('djangoform/', views.form1)
```

Form validation ⇒ HTML form validation

views.py

```
def Validation(request):
    # Creating variable for submitted details
    Submitted_details = None
    if (request.method == 'POST'):
        name = request.POST.get('name')
        email = ".".get('email')
        password = ".".get('password')
    # Creating variables to hold error messages.
    name_error = email_error = password_error = None
    # Validating
    if not name:
        name_error = "Name is required"
    if not email or "@" not in email:
        email_error = "Valid email required"
    if len(password) < 6:
        password_error = "Password must be at
                          least 6 char long"
    # if there are any error, ~~recode~~ re-render
      the form with error messages.
    if name_error or email_error or password_error:
        return render(request, 'validate.html', {
            'name_error': name_error,
            'email_error': email_error,
            'password_error': password_error
            'name': name,
            'email': email,
            'password': password,
        })
    else:
        Submitted details = {'name': name,
                             'email': email,
                             'password': password
```

return render(request, 'validation.html',
{'submitted_details': submitted_details})

validation.html

Here, we use Django **if-else** for showing error.

e.g

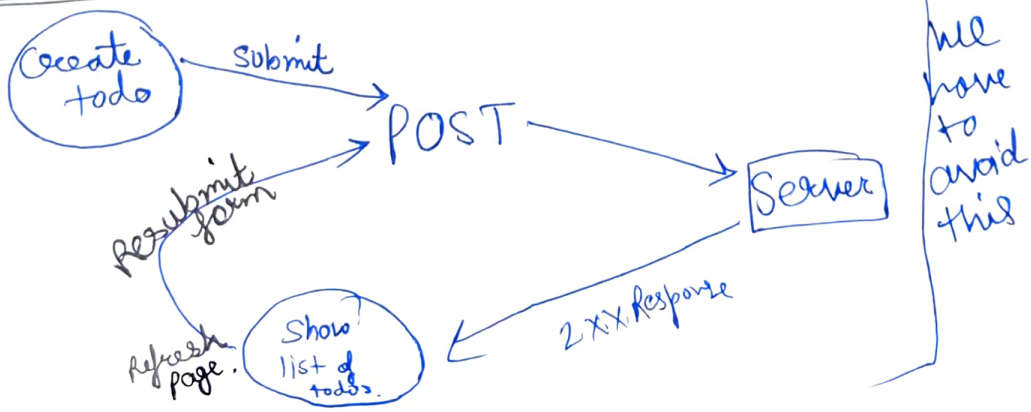
```
{% if name_error %}
    <span style="color: red;">{{ name_error }}</span>
{% endif %}
```

Django form validation

→ use attribute "**novalidate** in html template

e.g `<form method="post" novalidate>`

Post Redirect Refresh pattern



from django.shortcuts import redirect

|

return redirect('success')

Similar to 'name' attribute in url path.

* Basically, Here we redirect url to another page after successful submission of form.

Models

↳ Python class representing database table.

models.py

```
from django.db import models.
class Emp(models.Model):
    firstname = models.CharField(max_length=255)
    lastname = models.CharField( " " " " )
    salary = models.IntegerField()
```

Commands

1. python manage.py makemigrations
   ↳ This will create a file inside migration folder. We have to execute code of this file in order to create table
2. python manage.py migrate.

ORM → object-Relational Mapping

↳ Allows developers to interact with database using python code instead of writing SQL queries.

CRUD operations using python shell :—

Command :- python manage.py shell

Shell opened

Inserting from

```
myapp.models import Emp → class we created.
Emp.objects.create(firstname='Aneesh',
                   lastname='Kumar',
                   salary=50000)
```

`Emp.objects.all()` → Used to check data is inserted or not.

exit() → to exit from shell

Updating

from djangoapp → app/feature we created in project.

```
from myapp.models import Emp
e = Emp.objects.get(pk=2)   → storing row whose primary key is 2.
e.salary = 520000   # updating
e.save()            # saving
```

Deleting

```
from myapp.models import Emp
e = Emp.objects.get(pk=1)
e.delete()
```

---

django-admin

→ Command line utility to perform Administrative tasks

- manage projects
- application
- Migrations
- databases.

url :- 127.0.0.1:8000/admin

↳ it will ask for login Id password

We have to create it

Command to create

→ python manage.py migrate

→ python manage.py createsuperuser

↳ ask for username, email, pass, confirmpass.

(raghav
1234567890)

✓ Now try to login.

django Admin

→ Add users
→ Add permissions
→ Add groups.

Two ways to perform these → Interface
→ ~~djan~~ python shell.

Python shell ⇒ python manage.py shell.

① Add users.

```
from django.contrib.auth.models import User
user = User.objects.create_user(username='nav',
                                email='nav@gmail.com', password=123)
```

Raghav@123

```
user.save()
```

② Add permissions.

```
from django.contrib.auth.models import Permission, User
from django.contrib.contenttypes.models import ContentType
from myapp.models import Emp
```

Emp ⇒ we create model before.

```
user = User.objects.get(username='nav')
ContentType = ContentType.objects.get_for_model(Emp)
view_permission = Permission.objects.get(codename='view_employee',
                                         ContentType=ContentType)
```

~~user.user_permissions.add(view_permission)~~

user.user_permissions.add(view_permission).

Now, to see Table in admin, we have to register model in admin.py

```
from django.contrib import admin
from .models import Emp
admin.site.register(Emp).
```

in models.py

```
def __str__(self):
    return f'{self.firstname} {self.lastname}'
```

(3) group Add :—

```
from django.contrib.auth.models import Group
group = Group(name='newGroup')
group.save()
Group.objects.all()
```

Giving permissions to Groups

```
from django.contrib.auth.models import Permission, User
from django.contrib.contenttypes.models import ContentType
from myapp.models import Emp
```

```
group = Group.objects.get(name='newGroup')
content_type = ContentType.objects.get_for_model(Emp)
view_permission = Permission.objects.get(codename='view_emp', content_type=content_type)
group.permission.add(view_permission).
```

Setting up database connection

1) Singup form (Html)
2) django form
3) Model form

1) Singupform

models.py

Step 1) Create model of user

```
class Useer(models.Model):
    username = models.CharField(max_length=100)
    email = models.EmailField(unique=True)
    password = models.CharField(max_length=255)
```

commands :- i) python manage.py makemigrations
ii) " " migrate

views.py

Step 2 Create fcn

```
from .models import User
def signup(request):
    account_created = False,
    if (request.method == 'POST'):
        username = request.POST.get('username')
        email = "  .  "  .  "
        password = "  .  "  .  "
```

name attribute in .html file

('username') ('email') ('password')

```
# Assigning into database Table.
        user = User.objects.create(
            username = username,
            email = email
            password = password
        )
        user.save()
        account_created = True
    return render(request, 'signup.html', {'account_created': account_created})
```

column name used in models.

{'account_created': 'account_created'}

signup.html

Step 3 Create signup.html in template folder.

```
<form method = "POST">
    {% csrf_token %}
```

user ____

email ____

passw. ____

____

{% if account_created %}

 account created successfully

{% endif %}

urls.py

(2) django form :- → we will use same model used before.

forms.py

```
class SignupForm(forms.Form):
    username = forms.CharField(max_length=3)
    email = forms.EmailField()
    Password = forms.CharField(widget=forms.PasswordInput)
```

views.py

```
def signup1(request):
    form_submitted = False
    if (request.method == 'POST'):
        form = SignupForm(request.POST)
        if form.is_valid():
            username = form.cleaned_data['username']
            email = "." "['email']
            Password = "." "['Password']

            USER = User.objects.create(model we create)
                (username = username,
                 email = email,
                 Password = password)

            user.save()
            form_submitted = True.
    else:
        form = SignupForm()

    return render(request, 'signup1.html',
        {'form': form, 'form_submitted': form_submitted})
```

signup1.html

```
<form method="POST">
    {% csrf_token %}
    {{ form.as_p }}
    <input type="submit" value="Signup">

    {% if form_submitted %}
        <p style="color: green;">Account created successfully.</p>
    {% endif %}
</form>
```

③ Model form (CRUD operations)

① Model.py ⇒ use already created Model. (User.)

② forms.py → used to create form based on Model.

```
from .models import User
class SignupForm1(forms.ModelForm):
    class Meta:
        model = User
        fields = "__all__"
```

③ views.py

```
from .forms import SignupForm1
from .models import User.

def signup2(request):
    User = User.objects.all()   # fetch all records from table
    form_submitted = False
```

```
if (request.method == 'POST'):
    form = SignupForm1(request.POST)
    if (form.is_valid()):
        form.save()
        form_submitted = True.
        return render (request,
                       'signup2.html',
                       {'form': form, 'form_submitted':
                        form_submitted})
else:
    form = SignupForm1()
return render (request, 'signup2.html',
               {'form': form, 'users': user})
```

signup2.html

```
<form method="post">
    {% csrf_token %}
    {{ form.as_p }}
    <input type="submit" value="Signup">
    {% if form_submitted %}
        <P> Account Successful </P>
    {% end if %}
</form>
<div>
    <h1> All users </h1>
    {% if users %} {% for user in users %}
    <P> {{ user.username }} </P>
    <a href={% url 'delete' user.id %}"> delete </a>
    <a href={% url 'edit' user.id %}"> Edit </a>
    {% end for %}
    {% end if %}
</div>
```

Views.py

```
# delete
def delete(request, id):
    user = User.objects.get(pk=id)
    user.delete()
    return redirect('signup2')
```

```
# update
def edit(request, id):
    user = user.objects.get(pk=id)
    if(request.method == 'POST'):
        form = SignupForm1(request.POST, instance=user)
        if form.is_valid():
            form.save()
            return redirect('signup2')
    else:
        form = SignupForm1(instance=user)
    return render(request, 'update.html',
                  {'user': user})
```

helps to get editable data in form.

Update.html

```
<form method="post">
    {% csrf_token %}
    {{ form.as_p }}
    <button type="submit">Update</button>
</form>
<a href="{% url 'signup2' %}">Back to Signup.</a>
```

urls.py

```
path ('signup2/', views.signup2, name='signup2'),
  "  ('delete/<int:id>/', views.delete, name='delete'
  "  ('edit/<int:id>/', views.edit, name='edit'),
```

Blog POST

Model.py

```
class Blogpost (models.Model):
    title = models.CharField (max_length=200)
    post  =   "    .   "    (   "    "   , default="" )
    thumbnail = " .ImageField (upload_to='images/',
                               null=True,
                               blank=True)

    def __str__(self):
        return self.title.
```

command :- python manage.py make migration
"   "   "   migrate

forms.py

```
from .models import BlogPost

class BlogPostForm (forms.ModelForm)
    Class Meta :
        model = BlogPost
        field = "__all__"
```

views.py

```
from .forms import BlogPostForm

def insertblogpost(request):
    blog_created = False
    if(request.method == 'POST'):
        form = BlogPostForm(request.POST,
                            request.FILES)
                                  → Handle file uploads
        if form.is_valid():
            form.save()
            blog_created = True
    else:
        form = BlogPostForm()
    return render(request, 'insertblogposts.html',
                  {'form': form, 'blog_created': blog_created})
```

Retrieval of Data on Template from Models

```
from .models import BlogPost

def displayblogposts(request):
    posts = BlogPost.objects.all()
    return render(request, 'displayblogposts.
                  html', {'posts': posts})

def singleblogpost(request, id):
    post = Blogpost.objects.get(pk=id)
    return render(request, 'singleblogpost.html',
                  {'post': post})
```

insertblogpost.html

<form method='POST' enctype="multipart/form-data">

↳ for file upload to server.

{% blog.created %}

{{ }}

displayblogposts.html

↳ we display title of each post in

using for loop.

and a button for each title to display.details

Singleblogpost.html

detail of a post along with a back button.

urls.py

three urls path will be created.

To display image properly

In urls.py at project level

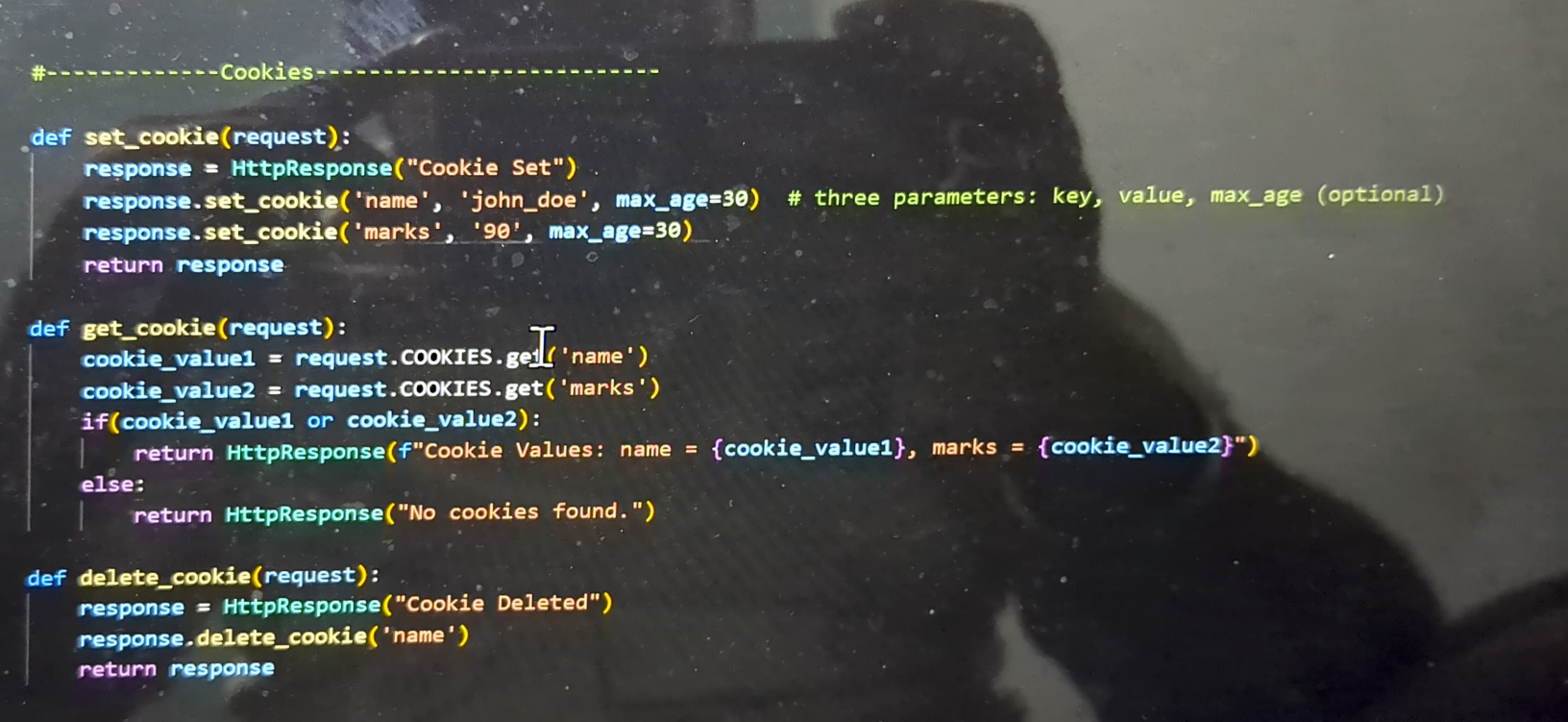
```
from django.conf import settings
from django.conf.urls.static import static

urlpatterns = [

] + static(settings.MEDIA_URL,
           document_root = settings.MEDIA_ROOT)
```

DEBUG = True in settings.py.

DJANGO COOKIES AND SESSIONS

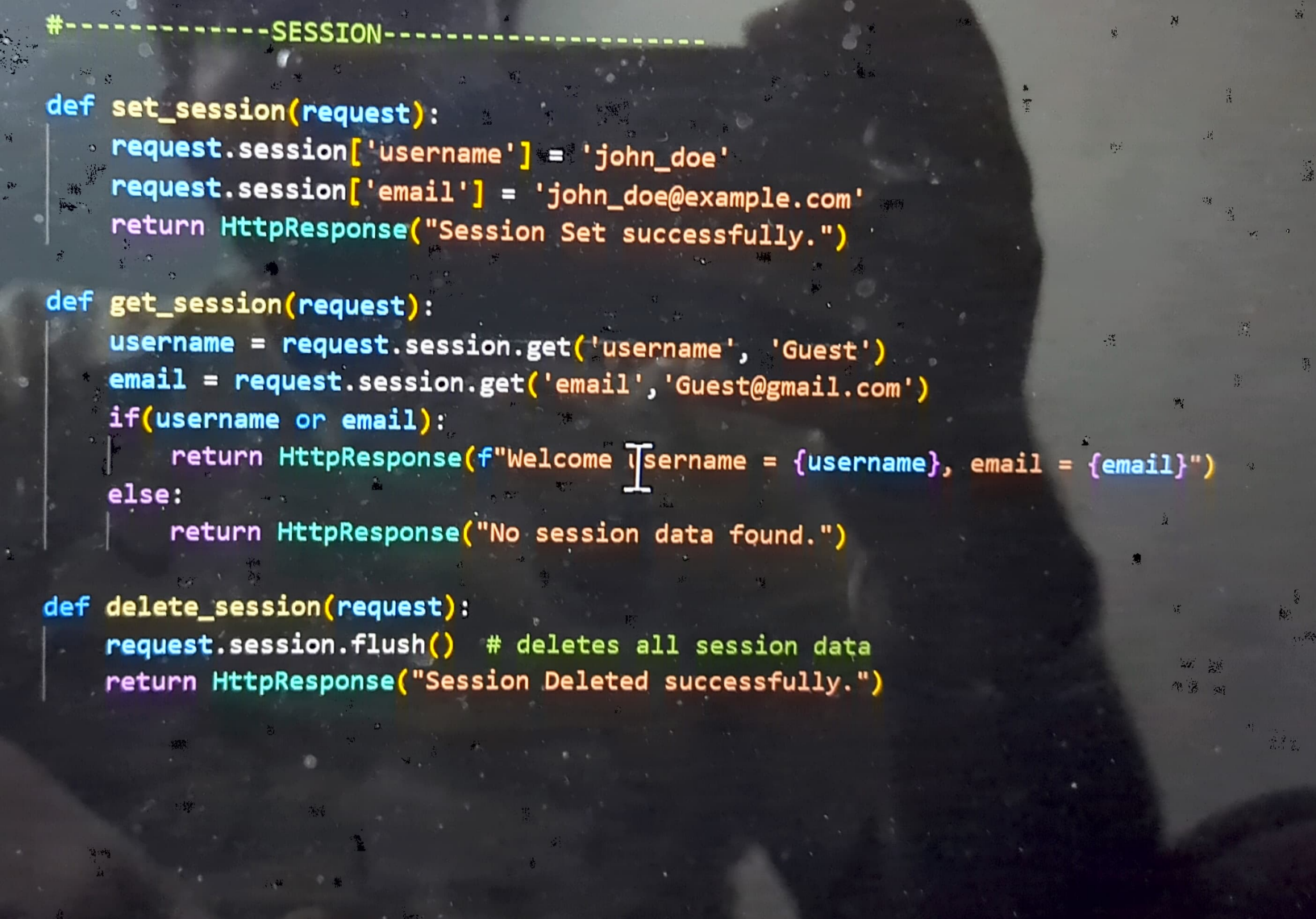


```python
#------------Cookies-------------------------

def set_cookie(request):
    response = HttpResponse("Cookie Set")
    response.set_cookie('name', 'john_doe', max_age=30)  # three parameters: key, value, max_age (optional)
    response.set_cookie('marks', '90', max_age=30)
    return response

def get_cookie(request):
    cookie_value1 = request.COOKIES.get('name')
    cookie_value2 = request.COOKIES.get('marks')
    if(cookie_value1 or cookie_value2):
        return HttpResponse(f"Cookie Values: name = {cookie_value1}, marks = {cookie_value2}")
    else:
        return HttpResponse("No cookies found.")

def delete_cookie(request):
    response = HttpResponse("Cookie Deleted")
    response.delete_cookie('name')
    return response
```



```python
#-------------SESSION--------------------

def set_session(request):
    request.session['username'] = 'john_doe'
    request.session['email'] = 'john_doe@example.com'
    return HttpResponse("Session Set successfully.")

def get_session(request):
    username = request.session.get('username', 'Guest')
    email = request.session.get('email','Guest@gmail.com')
    if(username or email):
        return HttpResponse(f"Welcome username = {username}, email = {email}")
    else:
        return HttpResponse("No session data found.")

def delete_session(request):
    request.session.flush()  # deletes all session data
    return HttpResponse("Session Deleted successfully.")
```