PART-1

django Web Framework

Geeky Shows YouTube Channel Learning Notes

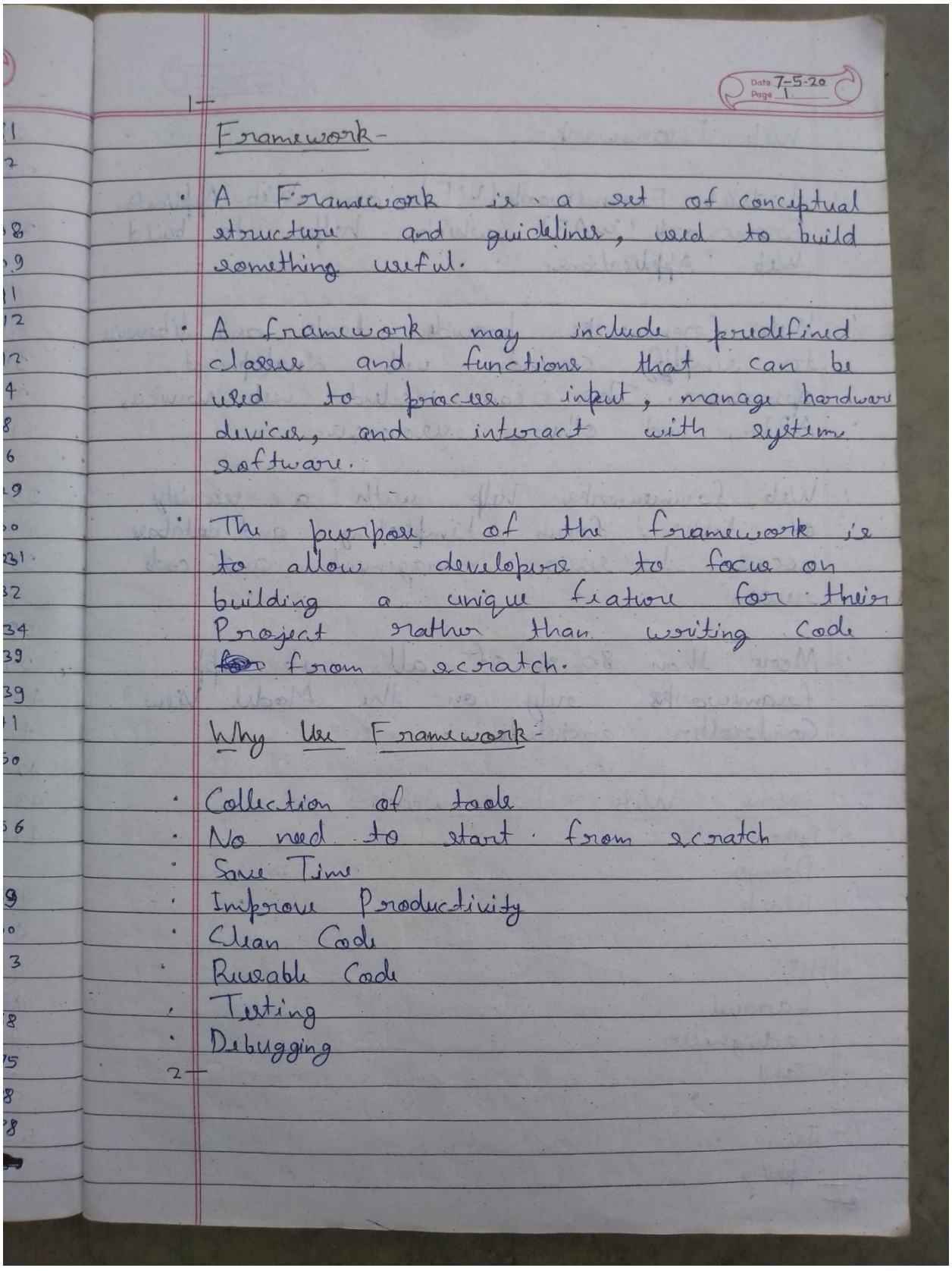
SATYAM SETH

14/09/2020

Source Code – https://github.com/satyam-seth/django_learning

Playlist Link – https://www.youtube.com/playlist?list=PLbGui_ZYuhigchy8DTw4pX4duTTpvqlh6

INDEX



Date 7-5-20
Page 1

1—

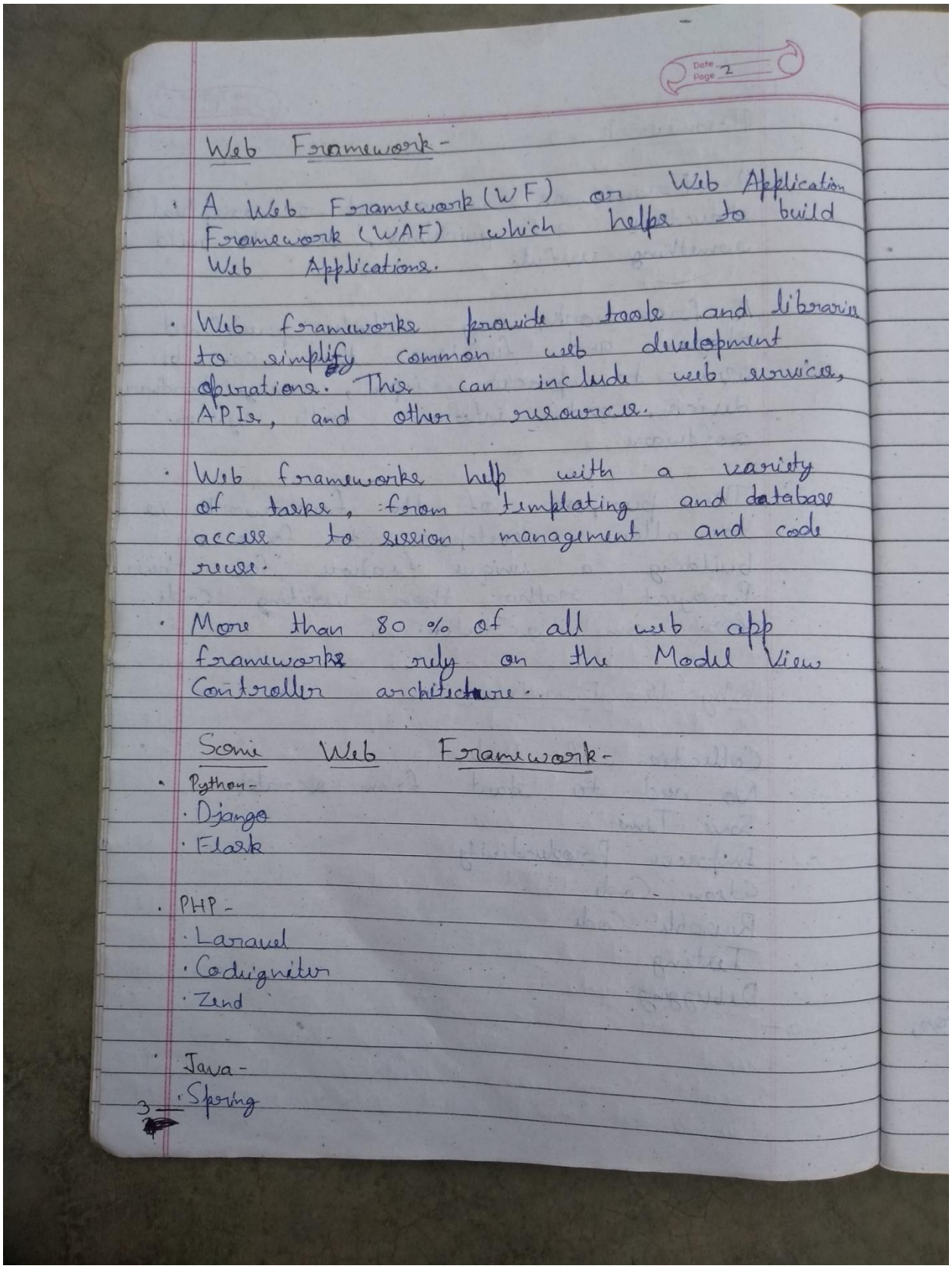
Framework-

- A Framework is a set of conceptual structure and guidelines, used to build something useful.
- A framework may include predefined classes and functions that can be used to process input, manage hardware devices, and interact with system software.
- The purpose of the framework is to allow developers to focus on building a unique feature for their Project rather than writing code from scratch.

Why Use Framework-

- Collection of tools
- No need to start from scratch
- Save Time
- Improve Productivity
- Clean Code
- Reusable Code
- Testing
- Debugging

2—



Date
Page 2

Web Framework-

- A Web Framework (WF) or Web Application Framework (WAF) which helps to build Web Applications.

- Web frameworks provide tools and libraries to simplify common web development operations. This can include web services, APIs, and other resources.

- Web frameworks help with a variety of tasks, from templating and database access to session management and code reuse.

- More than 80 % of all web app frameworks rely on the Model View Controller architecture.

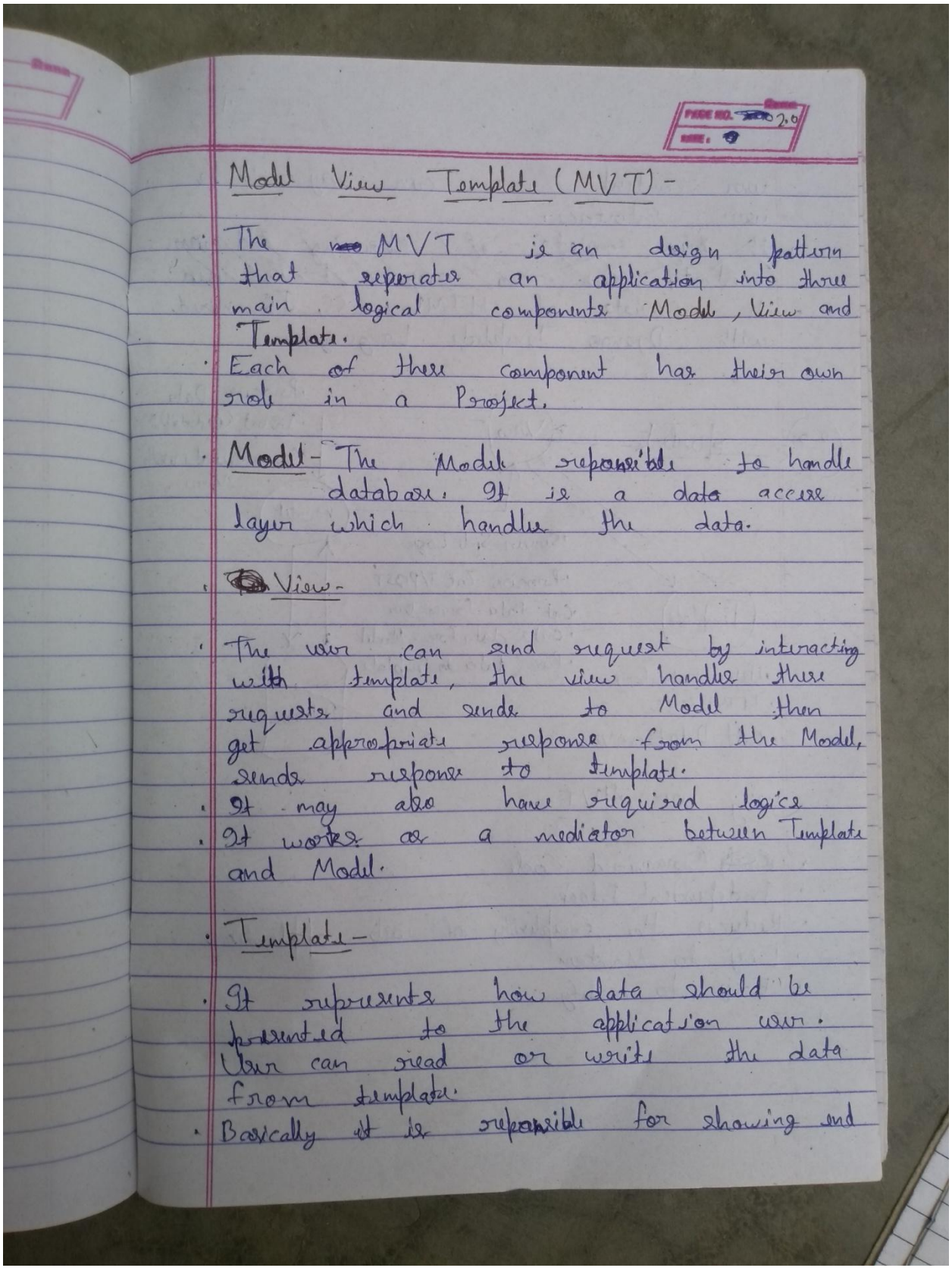
Some Web Framework-

- Python-
  - Django
  - Flask

- PHP-
  - Laravel
  - Codigniter
  - Zend

- Java-
  - Spring

3



Model View Template (MVT) -

- The MVT is an design pattern that seperates an application into three main logical components Model, View and Template.
- Each of these component has their own role in a Project.

- **Model** - The Model responsible to handle database. It is a data access layer which handles the data.

- **View** -

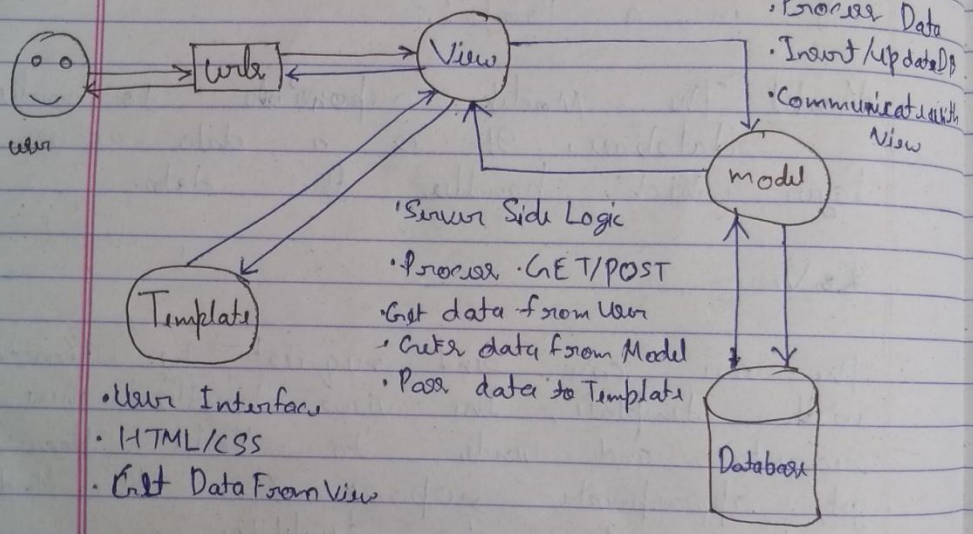
- The user can send request by interacting with template, the view handles these requests and sends to Model then get appropriate response from the Model, sends respone to template.
- It may also have required logics
- It works as a mediator between Template and Model.

- **Template** -

- It represents how data should be presented to the application user. User can read or write the data from template.
- Basically it is responsible for showing end

user content, we can say it is user interface.

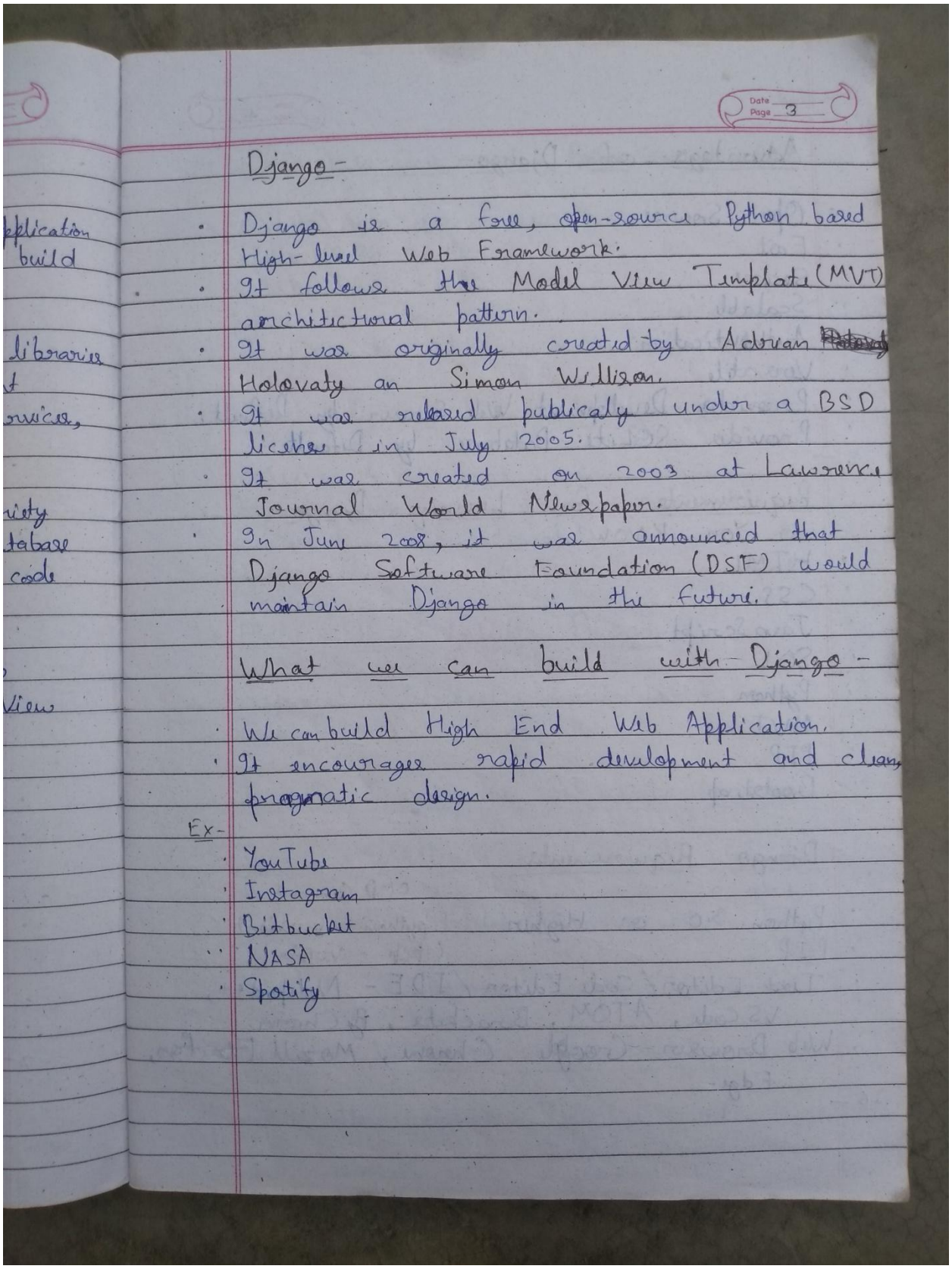
- ~~It may consists of showing and user content, we can say it is user~~
- It consists of HTML, CSS, JS mixed with Django Template Language.



Why we MVT-

- Organized Code
- Independent Block
- Reduces the complexity of web applications
- Easy to Maintain
- Easy to modify.

4 —



Date
Page 3

Django -

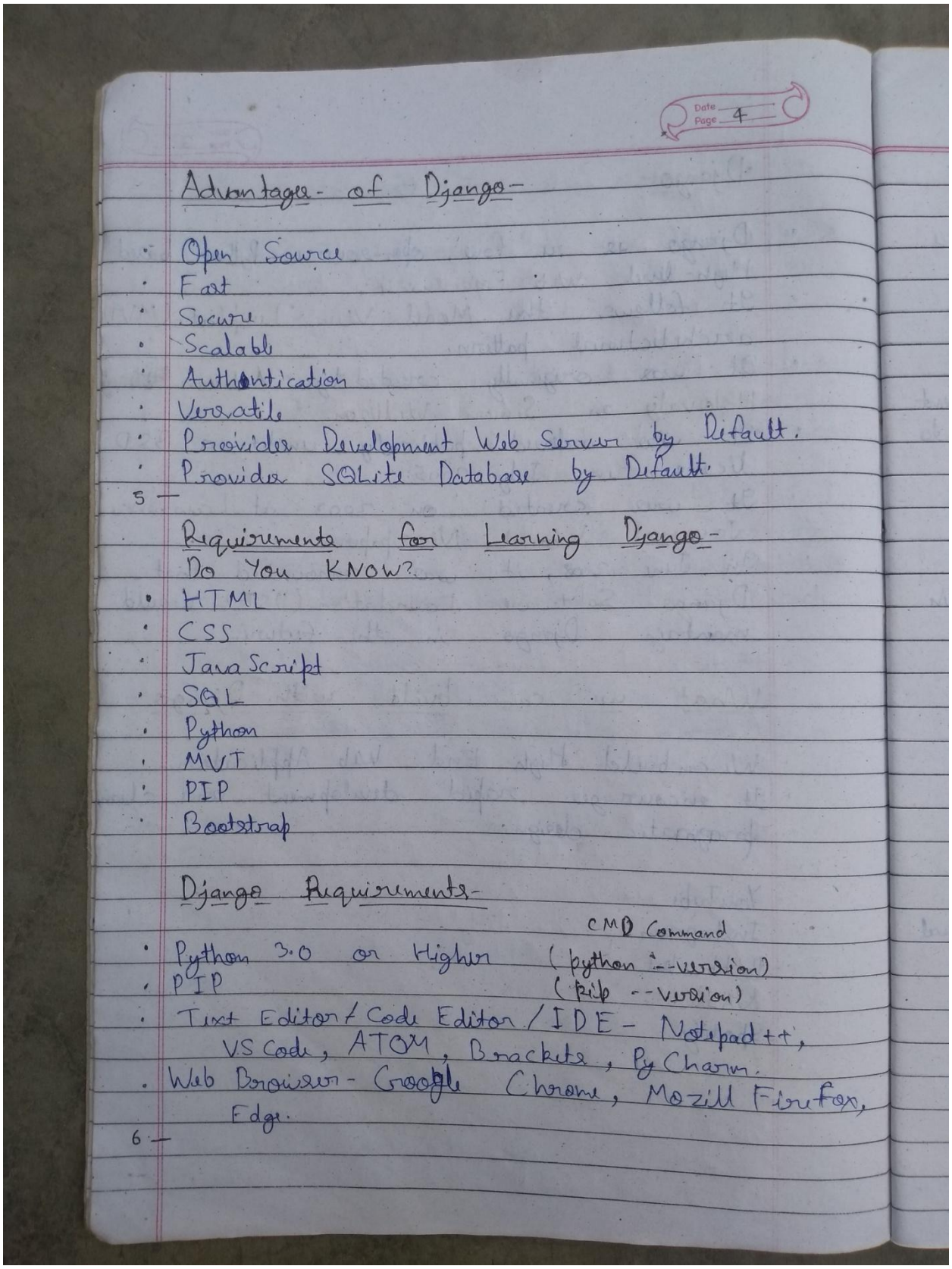
- Django is a free, open-source Python based High-level Web Framework.
- It follows the Model View Template (MVT) architectural pattern.
- It was originally created by Adrian Holovaty an Simon Willison.
- It was released publicaly under a BSD license in July 2005.
- It was created on 2003 at Lawrence Journal World Newspaper.
- In June 2008, it was announced that Django Software Foundation (DSF) would maintain Django in the future.

What we can build with Django -

- We can build High End Web Application.
- It encourages rapid development and clean, pragmatic design.

Ex -

- YouTube
- Instagram
- Bitbucket
- NASA
- Spotify



Date
Page 4

Advantages of Django-

- Open Source
- Fast
- Secure
- Scalable
- Authentication
- Versatile
- Provides Development Web Server by Default.
- Provides SQLite Database by Default.

5 —

Requirements for Learning Django-

Do You KNOW?

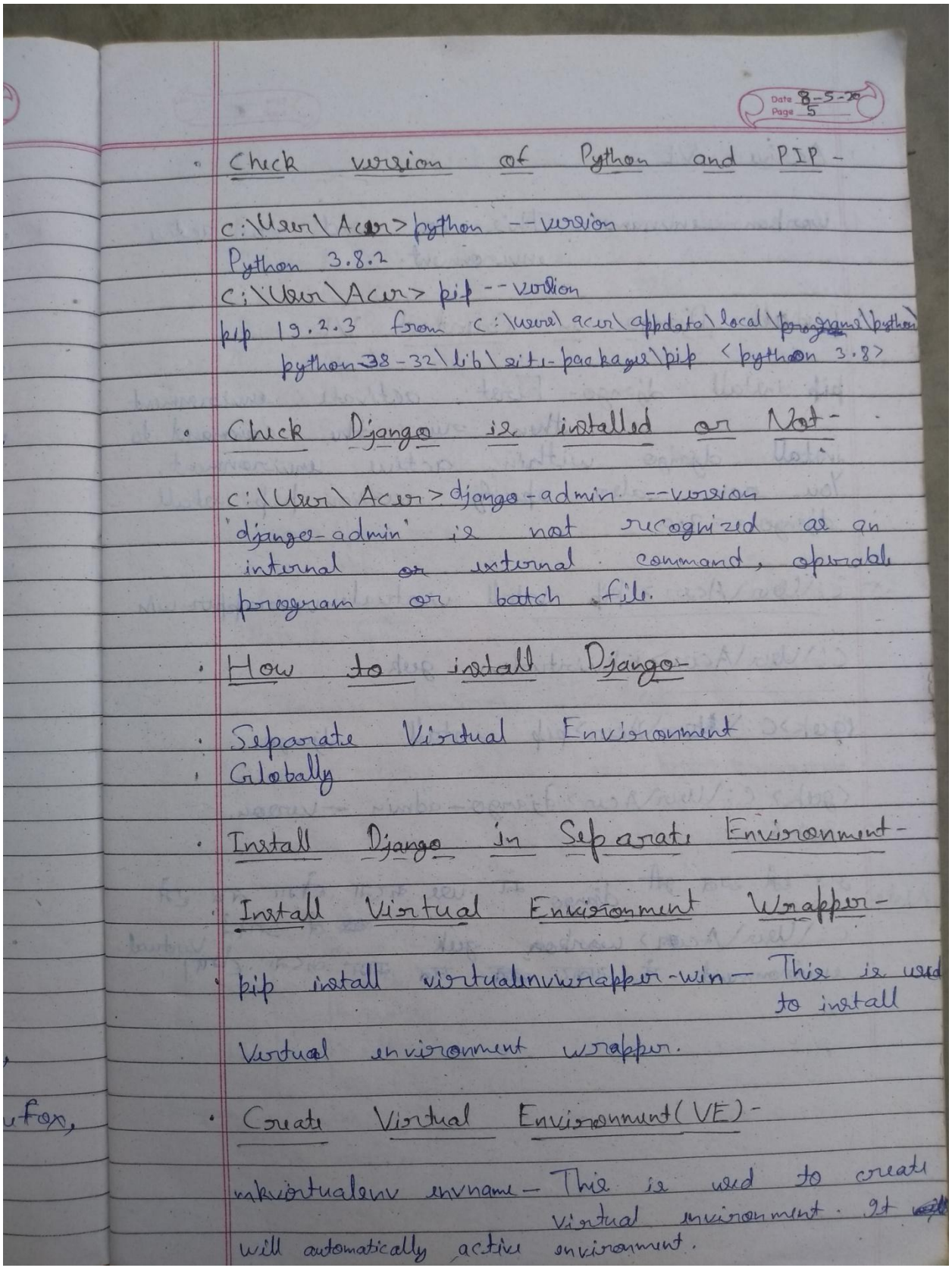
- HTML
- CSS
- JavaScript
- SQL
- Python
- MVT
- PIP
- Bootstrap

Django Requirements-

CMD Command

- Python 3.0 or Higher (python --version)
- PIP (pip --version)
- Text Editor / Code Editor / IDE - Notepad++, VS Code, ATOM, Brackets, PyCharm.
- Web Browser - Google Chrome, Mozill Firefox, Edge.

6 —



Date 8-5-20
Page 5

- Check version of Python and PIP -

c:\Users\Acer>python --version
Python 3.8.2
c:\Users\Acer>pip --version
pip 19.2.3 from c:\users\acer\appdata\local\programs\python\python38-32\lib\site-packages\pip (python 3.8)

- Check Django is installed or Not -

c:\Users\Acer>django-admin --version
'django-admin' is not recognized as an internal or external command, operable program or batch file.

- How to install Django -

- Separate Virtual Environment
- Globally

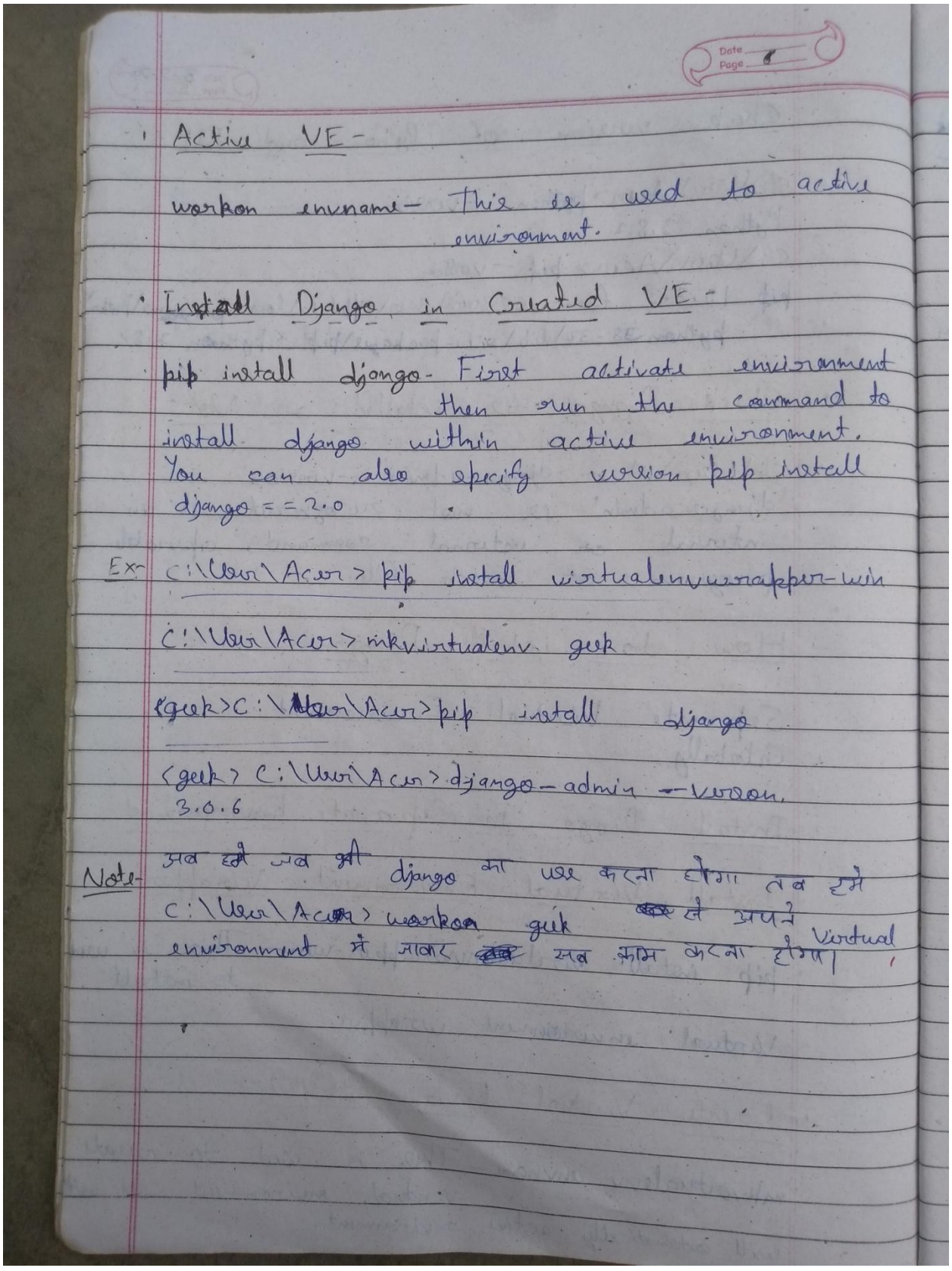
- Install Django in Separate Environment -

- Install Virtual Environment Wrapper -

- pip install virtualenvwrapper-win — This is used to install Virtual environment wrapper.

- Create Virtual Environment (VE) -

mkvirtualenv envname — This is used to create Virtual environment. It will automatically active environment.



• Active VE -

workon envname - This is used to active environment.

• Install Django in Created VE -

pip install django - First activate environment then run the command to install django within active environment. You can also specify version pip install django==2.0

Ex- C:\Users\Acer> pip install virtualenvwrapper-win

C:\Users\Acer> mkvirtualenv geek

(geek) C:\Users\Acer> pip install django

(geek) C:\Users\Acer> django-admin --version
3.0.6

Note- जब हमें जब भी django का use करना होगा तब हमें C:\Users\Acer> workon geek से अपने Virtual environment में जाकर सब काम करना होगा।

Date Page 7

Django Project-

- A Django Project may contain multiple Project Applications, which means a group of Application and file is called as Django Project.
- An Application is a Part of Django Project.

Ex- SchoolProject-

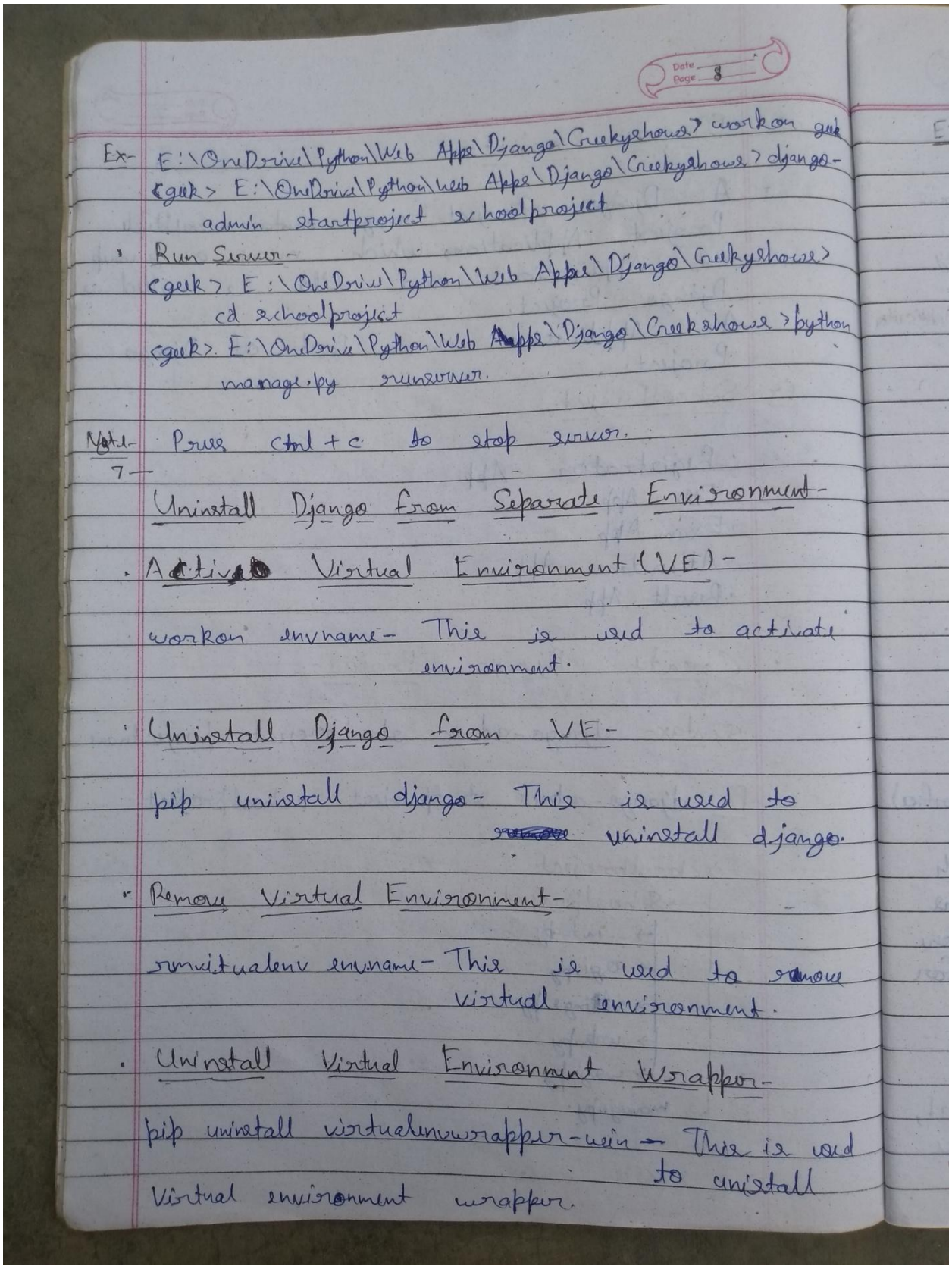
- Registration App
- Fee App
- Exam App
- Attendance App
- Result App

Create Django Project-

Syntax- django-admin startproject projectname

Ex- django-admin startproject schoolproject.

schoolproject
- schoolproject
  - __init__.py
  - asgi.py
  - settings.py
  - urls.py
  - wsgi.py
- manage.py



Ex- E:\OneDrive\Python\Web Appe\Django\Geekyshows> workon geek
(geek) E:\OneDrive\Python\Web Appe\Django\Geekyshows> django-admin startproject schoolproject

. Run Server-
(geek) E:\OneDrive\Python\Web Appe\Django\Geekyshows> cd schoolproject
(geek) E:\OneDrive\Python\Web Appe\Django\Geekyshows\schoolproject> python manage.py runserver.

Note- Press ctrl + c to stop server.

7- Uninstall Django from Separate Environment-

. Activate Virtual Environment (VE)-

workon envname- This is used to activate environment.

. Uninstall Django from VE-

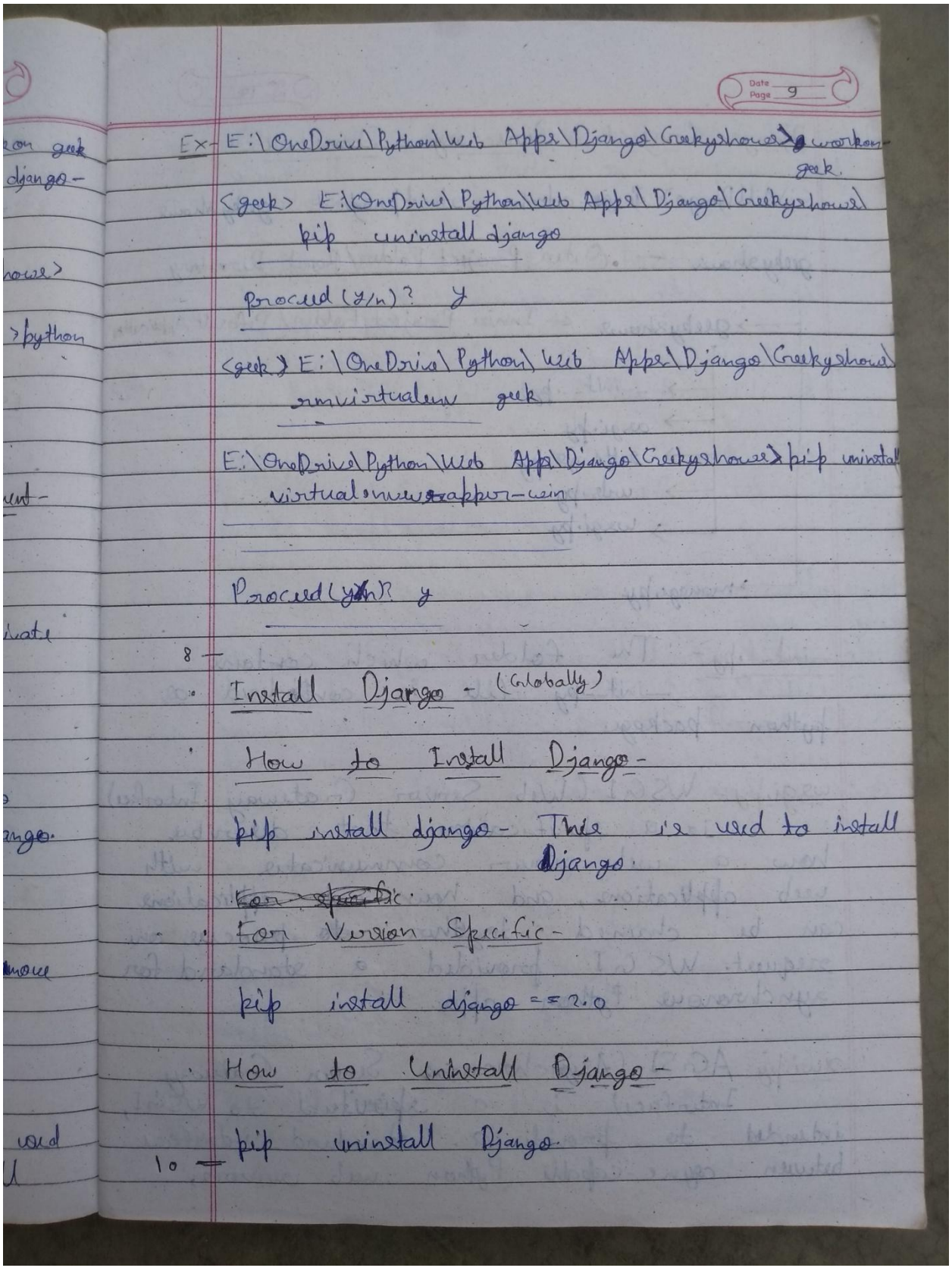
pip uninstall django- This is used to uninstall django.

. Remove Virtual Environment-

rmvirtualenv envname- This is used to remove virtual environment.

. Uninstall Virtual Environment Wrapper-

pip uninstall virtualenvwrapper-win - This is used to uninstall Virtual environment wrapper.



Ex- E:\OneDrive\Python\Web Appz\Django\Geekyshows> workon-geek.

(geek) E:\OneDrive\Python\Web Appz\Django\Geekyshows\ pip uninstall django

Proceed (y/n)? y

(geek) E:\OneDrive\Python\Web Appz\Django\Geekyshows\ rmvirtualenv geek

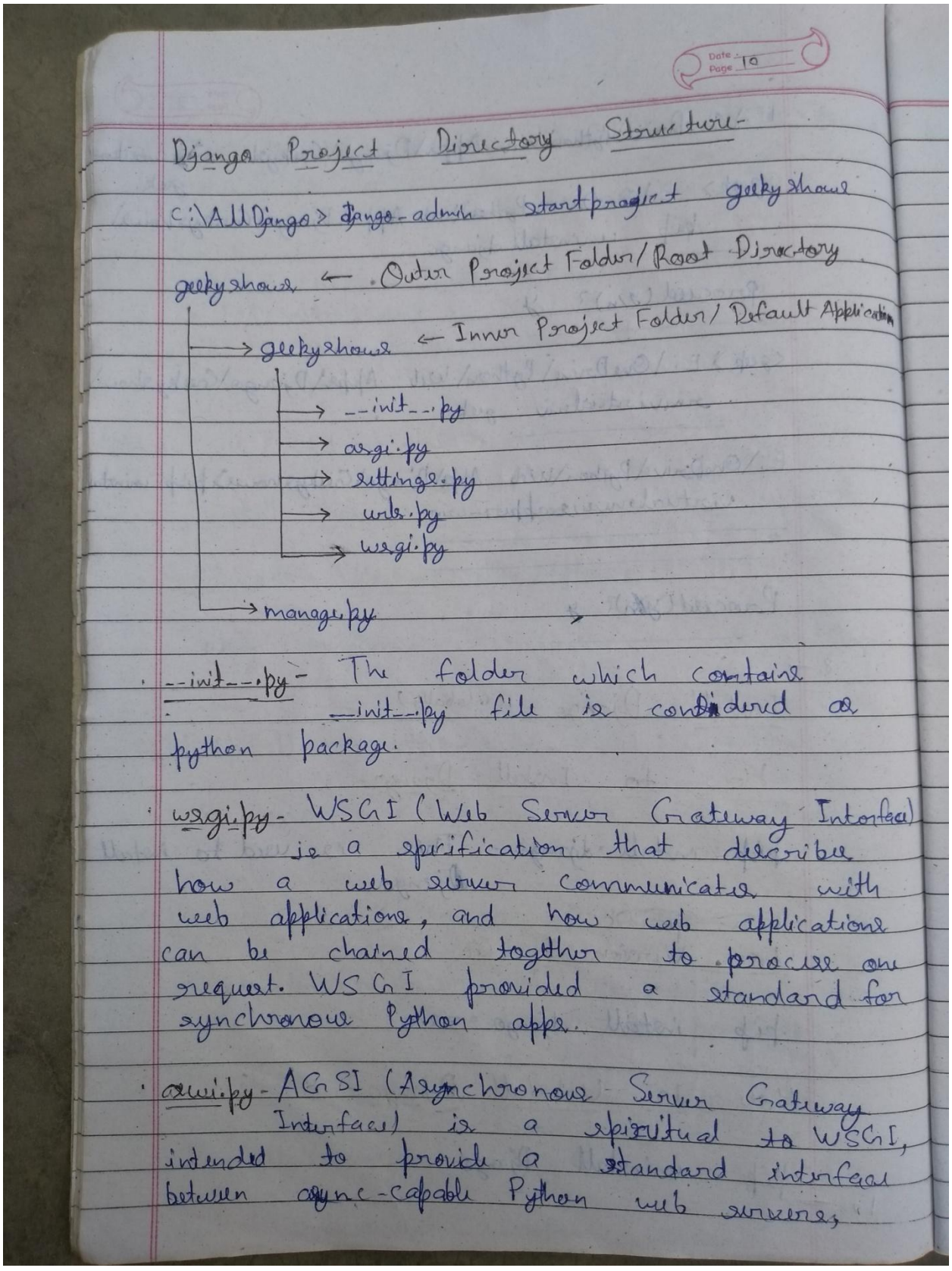
E:\OneDrive\Python\Web Appz\Django\Geekyshows> pip uninstall virtualenvwrapper-win

Proceed (y/n)? y

8 —

- Install Django - (Globally)
- How to Install Django -
  - pip install django - This is used to install Django.
  - For Version Specific -
    - pip install django==2.0
- How to Uninstall Django -
  - pip uninstall Django.

10 —



Date
Page 10

Django Project Directory Structure-

c:\AllDjango> django-admin startproject geekyshows

geekyshows ← Outer Project Folder / Root Directory

→ geekyshows ← Inner Project Folder / Default Application

- → __init__.py
- → asgi.py
- → settings.py
- → urls.py
- → wsgi.py

→ manage.py

- __init__.py - The folder which contains __init__.py file is considered as python package.

- wsgi.py - WSGI (Web Server Gateway Interface) is a specification that describe how a web server communicates with web applications, and how web applications can be chained together to process one request. WSGI provided a standard for synchronous Python apps.

- asgi.py - AGSI (Asynchronous Server Gateway Interface) is a spiritual to WSGI, intended to provide a standard interface between async-capable Python web servers,

frameworks, and applications. ASGI provides standard for both asynchronous and synchronous apps.

- settings.py - This file contains all the information or data about project settings.
  Ex- Database Config information, Templates, Installed Application, Validators etc.

- urls.py - This file contains all the information of urls attached with application.

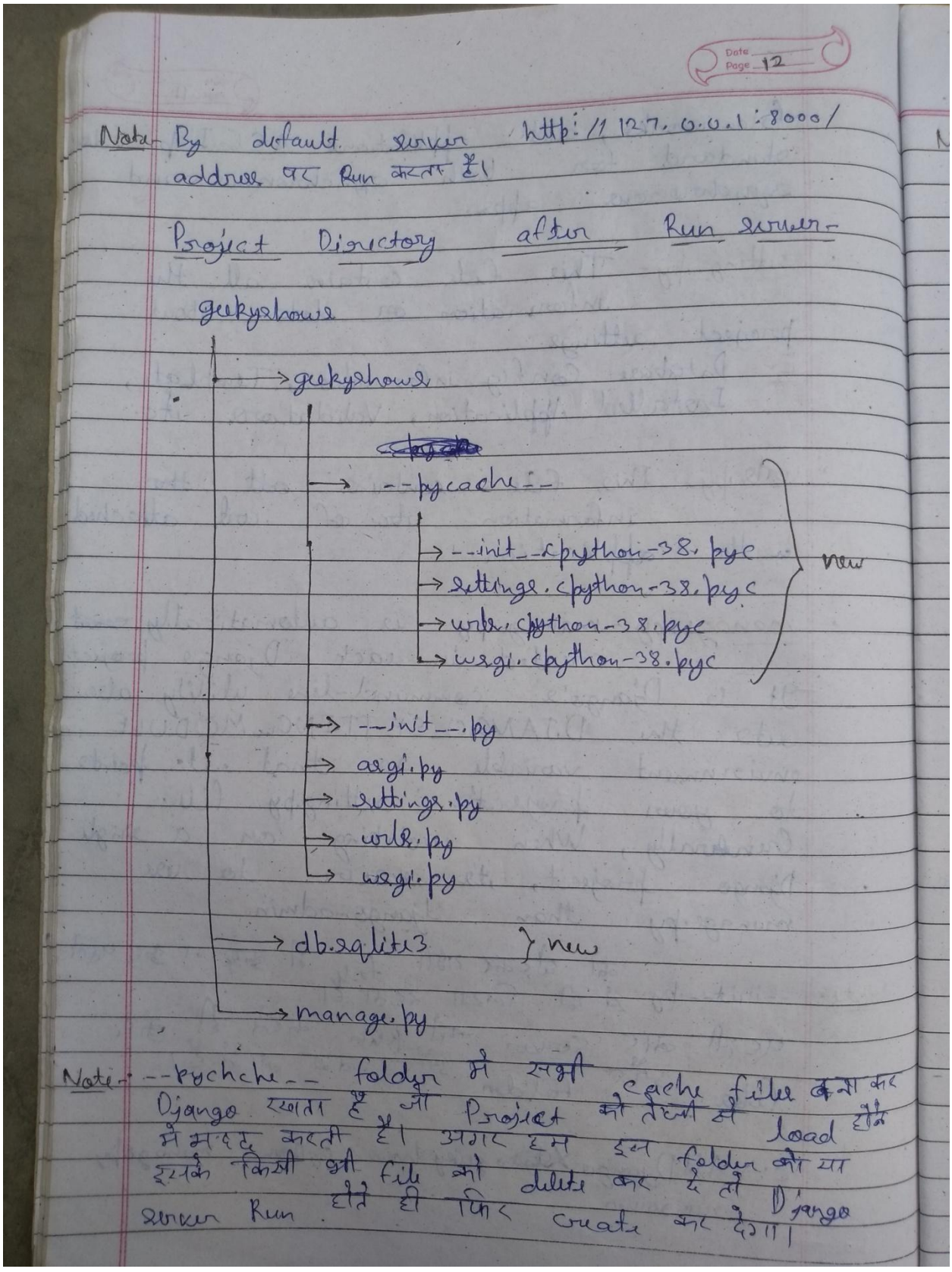
- manage.py - manage.py is automatically created in each Django project. It is Django's command-line utility also sets the DJANGO_SETTING_MODULE environment variable so that it points to your project's setting.py file. Eventually, When working on a single Django project, its easier to use manage.py than django-admin.

Note- --init--.py को छोड़कर सभी files में कुछ न कुछ पहले से ही लिखा रहता है।
पहली बार Server को Run करते ही कुछ files और Folder अपने आप बन जाते हैं।

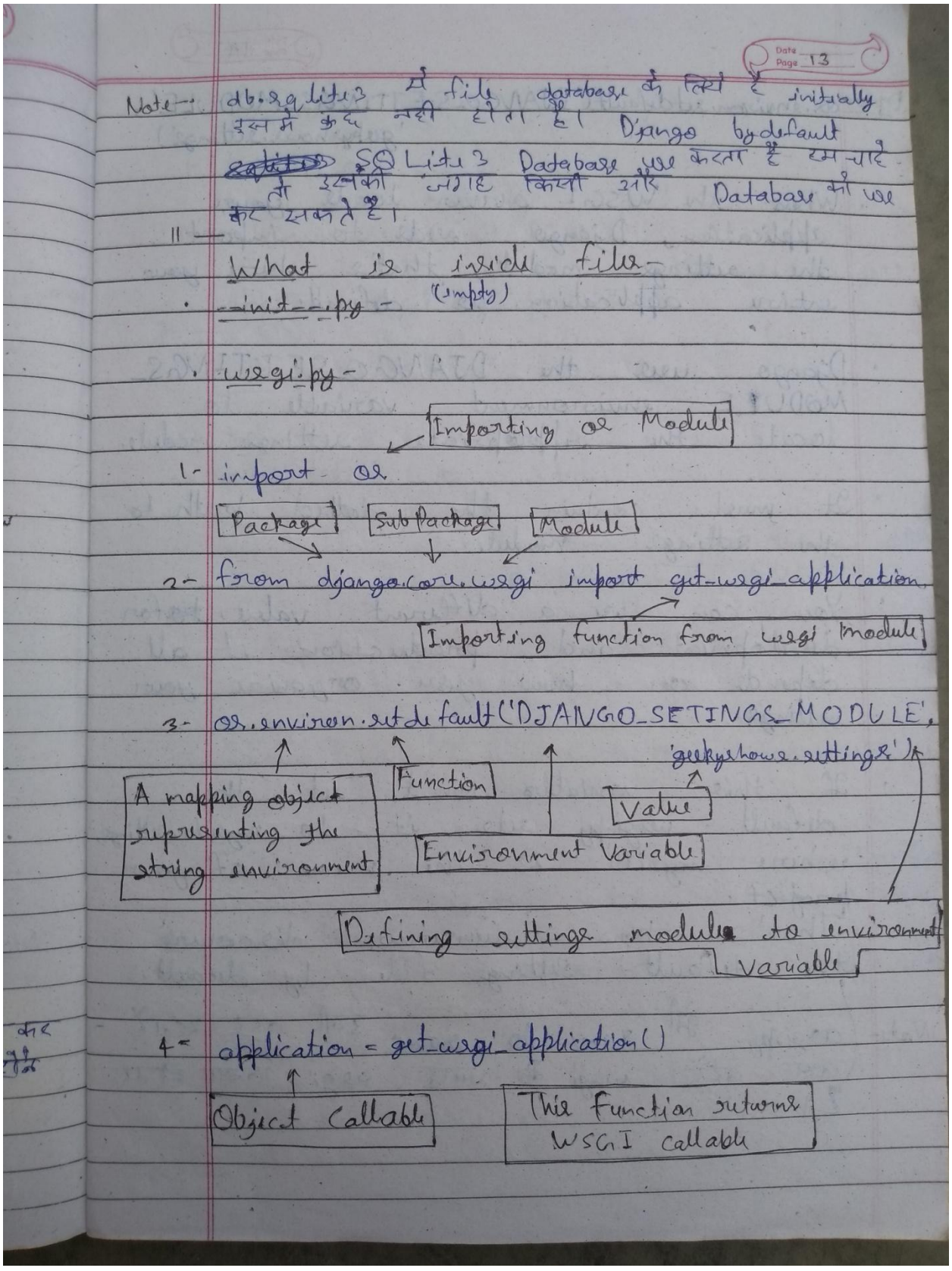
c:\AllDjango\geekyshows> python manage.py runserver

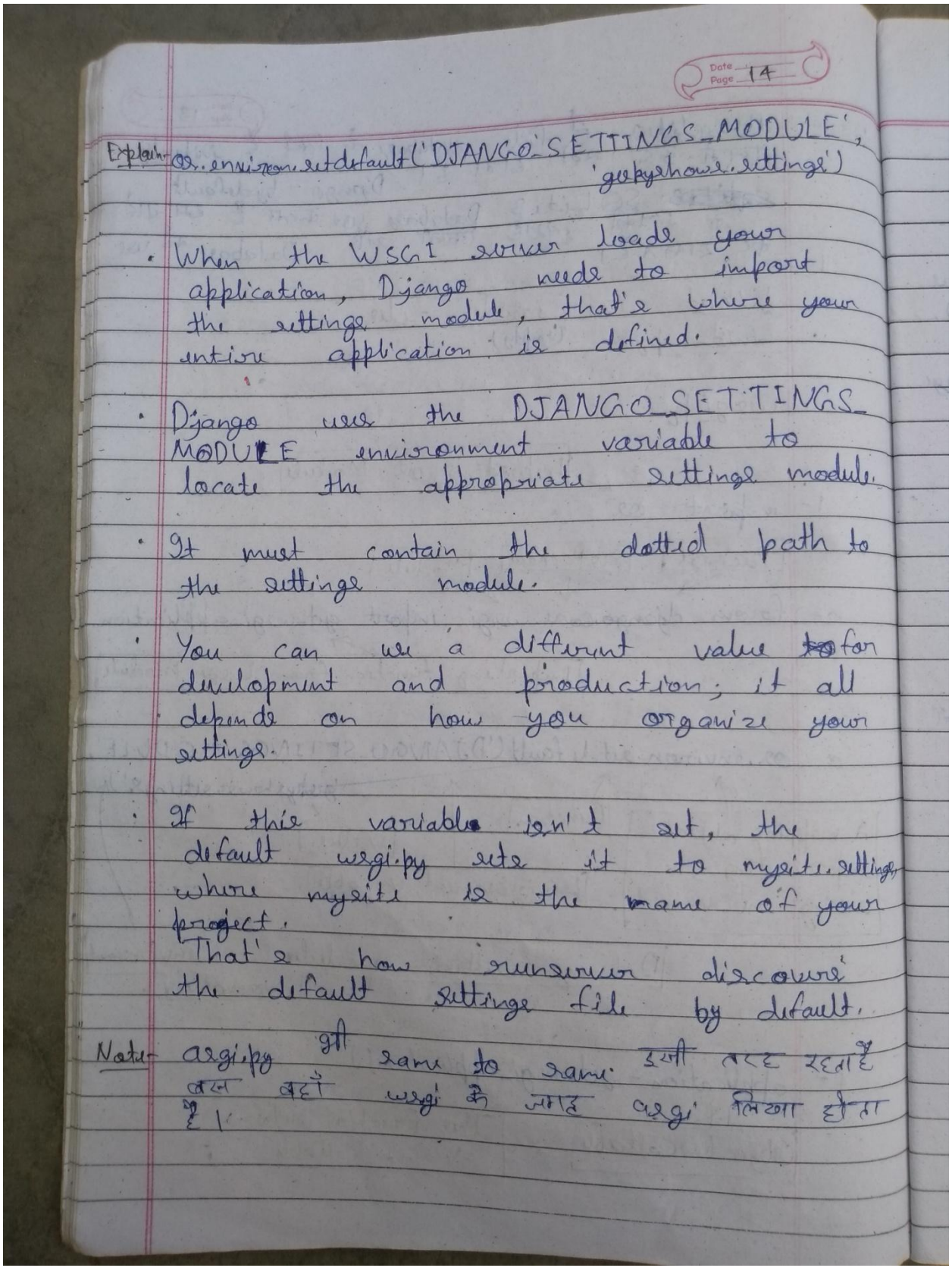
Note- By default server http://127.0.0.1:8000/ address पर Run करता है।

Project Directory after Run server-



Note- __pychche__ folder में सभी cache files को कर Django रखता है जो Project को तेजी से load होने में मदद करती है। अगर हम इस folder को या इसके किसी भी file को delete कर दे तो Django server Run होते ही फिर create कर देगा।


Note - db.sqlite3 ये file database को लिए है initially इसमें कुछ नहीं होता है। Django by default SQLite3 Database use करता है हम चाहे तो इसकी जगह किसी और Database को use कर सकते है।
II —
What is inside file-
. __init__.py - (empty)
. wsgi.py -
Importing os Module
1- import os
Package
Sub Package
Module
2- from django.core.wsgi import get_wsgi_application
Importing function from wsgi module
3- os.environ.setdefault('DJANGO_SETINGS_MODULE', 'geekyshows.settings')
A mapping object representing the string environment
Function
Value
Environment Variable
Defining settings module to environment variable
4- application = get_wsgi_application()
Object Callable
This function returns WSGI callable

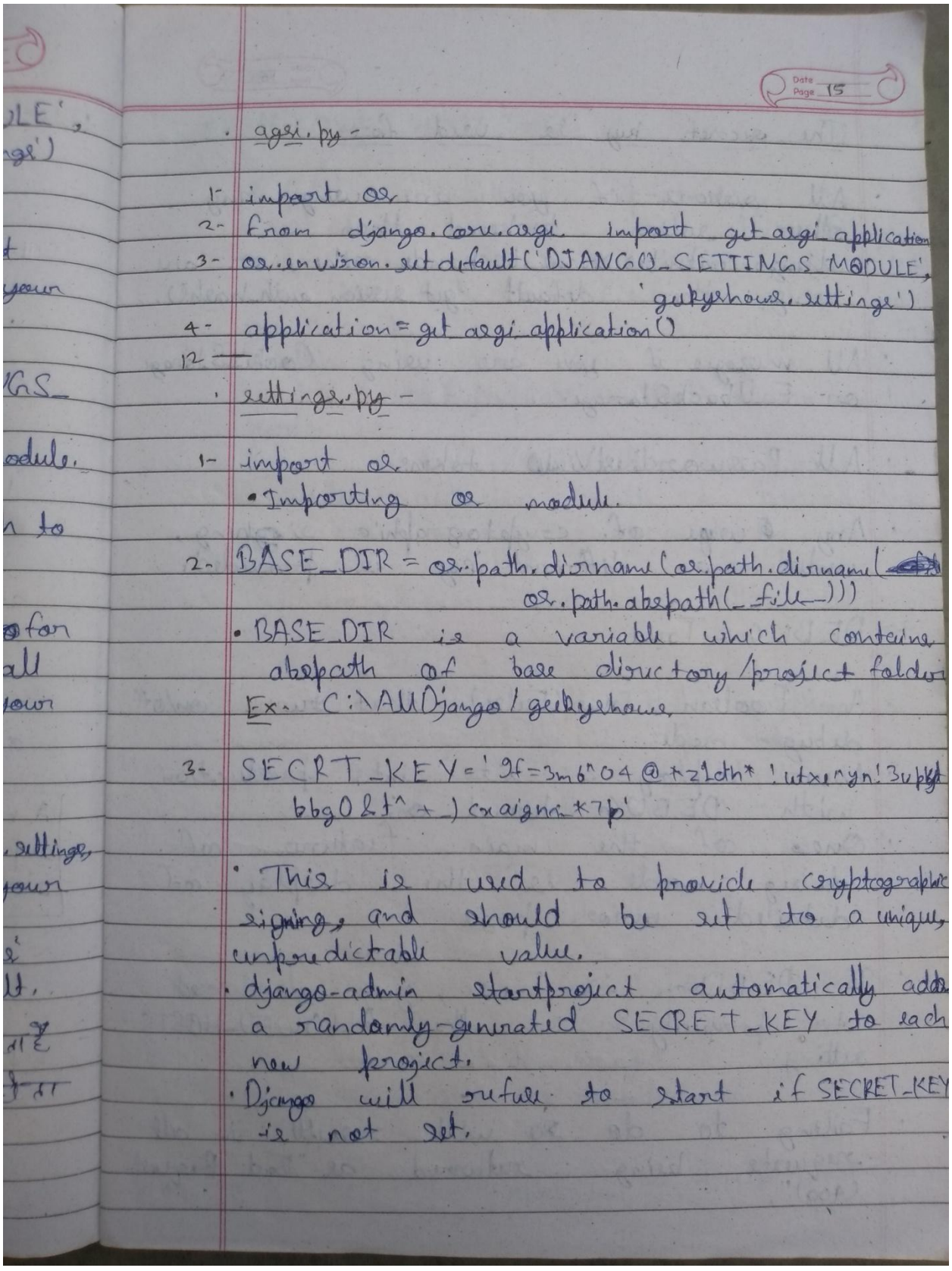


Date Page 14

Explain- os.environ.setdefault('DJANGO_SETTINGS_MODULE', 'geekyshows.settings')

- When the WSGI server loads your application, Django needs to import the settings module, that's where your entire application is defined.
- Django uses the DJANGO_SETTINGS_MODULE environment variable to locate the appropriate settings module.
- It must contain the dotted path to the settings module.
- You can use a different value ~~for~~ for development and production; it all depends on how you organize your settings.
- If this variable isn't set, the default wsgi.py sets it to mysite.settings, where mysite is the name of your project.
  That's how runserver discovers the default settings file by default.

Note- wsgi.py में name to name इसी तरह रहता है बस वहाँ wsgi के जगह asgi लिखा होता है।



Date Page 15

- wsgi.py -

1- import os
2- from django.core.wsgi import get_wsgi_application
3- os.environ.setdefault('DJANGO_SETTINGS_MODULE', 'geekyshows.settings')
4- application = get_wsgi_application()

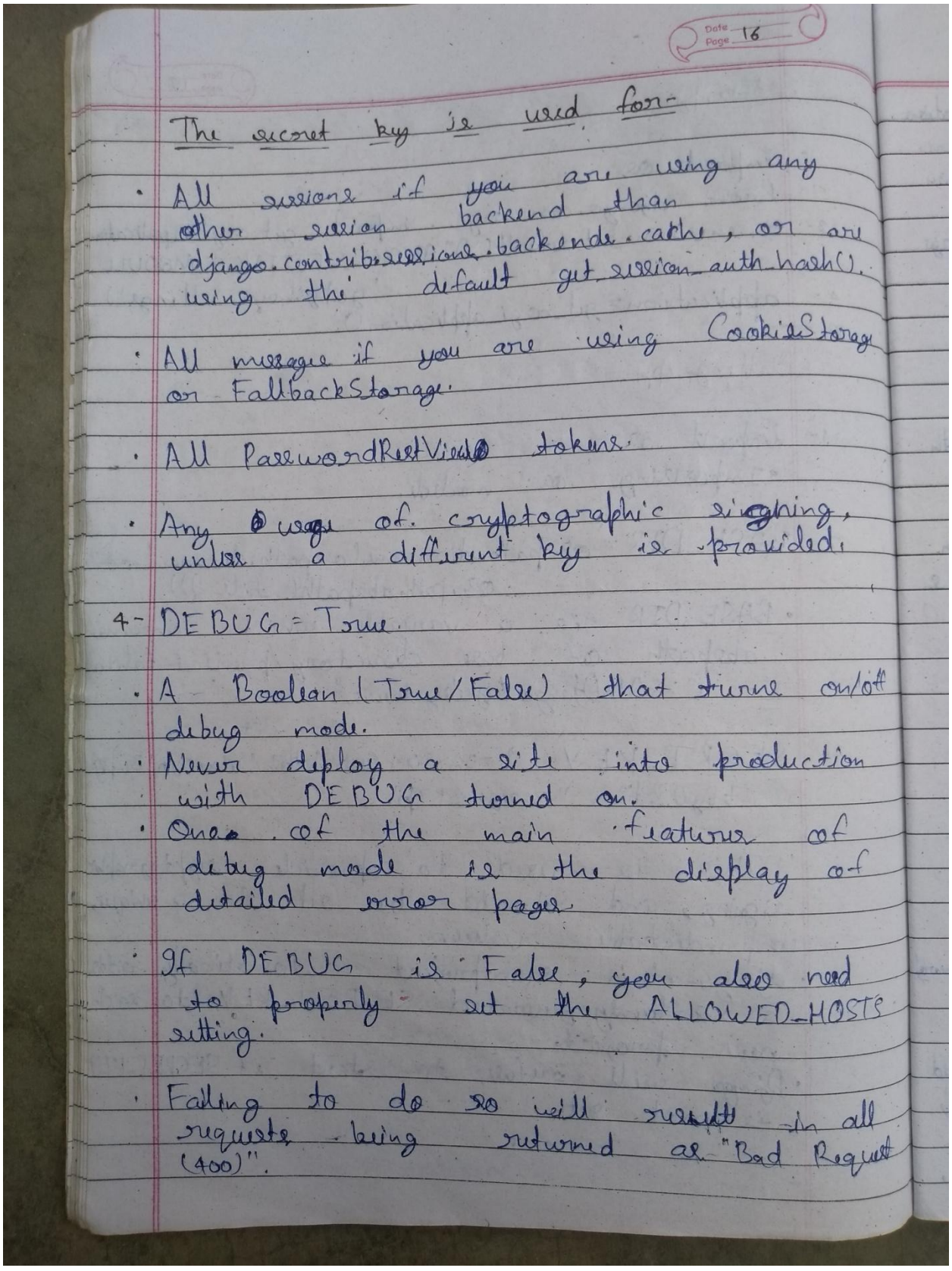
12 —

- settings.py -

1- import os
- Importing os module.

2- BASE_DIR = os.path.dirname(os.path.dirname(os.path.abspath(__file__)))
- BASE_DIR is a variable which contains abspath of base directory/project folder
Ex. C:\All Django\geekyshows

3- SECRET_KEY = '9f=3m6"04@*z1dh*!utxe^yn!3vpkf bbg0&t^+)cxaigna*7p'
- This is used to provide cryptographic signing, and should be set to a unique, unpredictable value.
- django-admin startproject automatically adds a randomly-generated SECRET_KEY to each new project.
- Django will refuse to start if SECRET_KEY is not set.



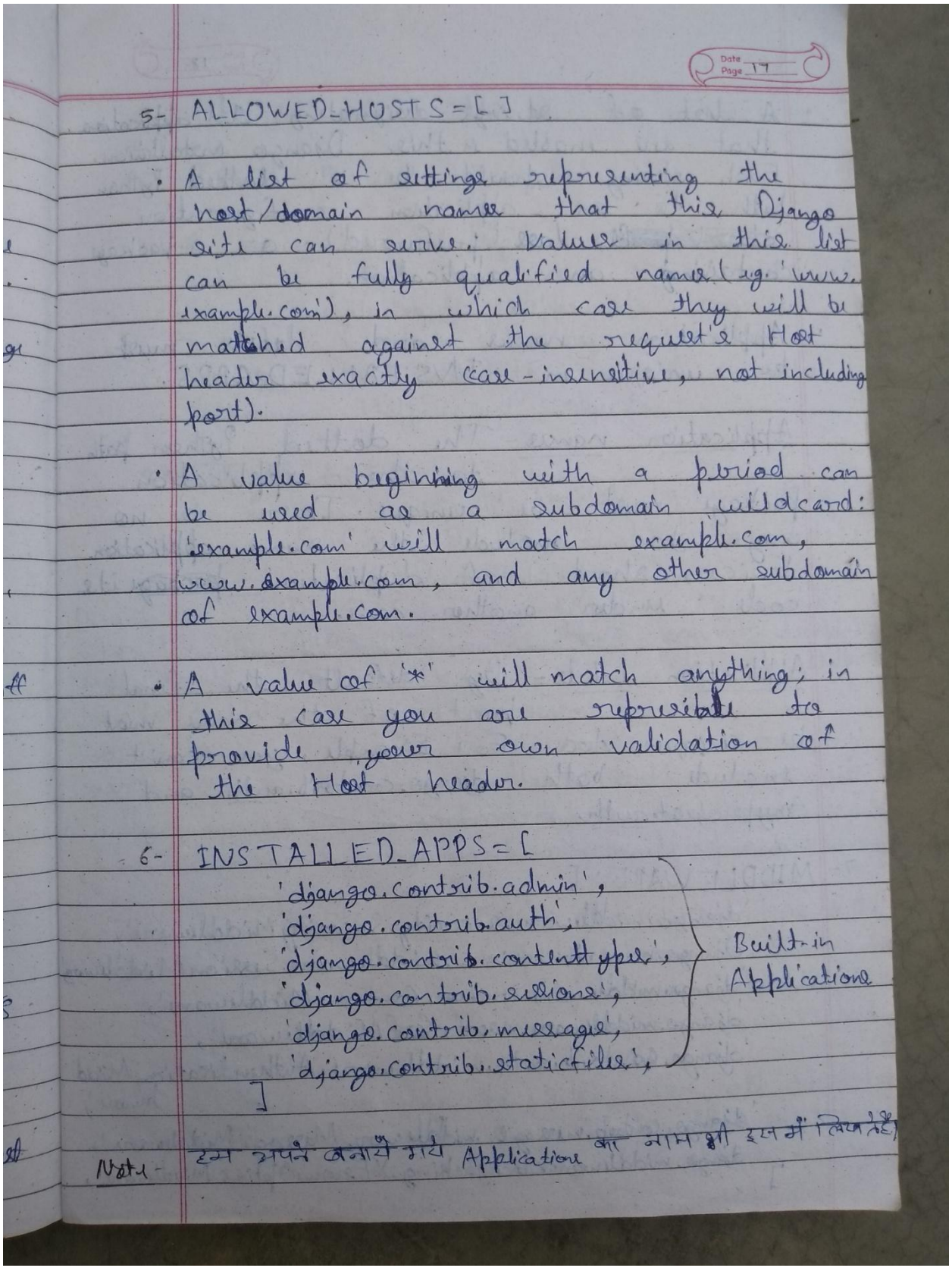
Date Page 16

The secret key is used for-

- All sessions if you are using any other session backend than django.contrib.sessions.backends.cache, or are using the default get_session_auth_hash().
- All messages if you are using CookieStorage or FallbackStorage.
- All PasswordResetView tokens.
- Any usage of cryptographic signing, unless a different key is provided.

4- DEBUG = True.

- A Boolean (True/False) that turns on/off debug mode.
- Never deploy a site into production with DEBUG turned on.
- One of the main features of debug mode is the display of detailed error pages.
- If DEBUG is False, you also need to properly set the ALLOWED_HOSTS setting.
- Failing to do so will result in all requests being returned as "Bad Request (400)".

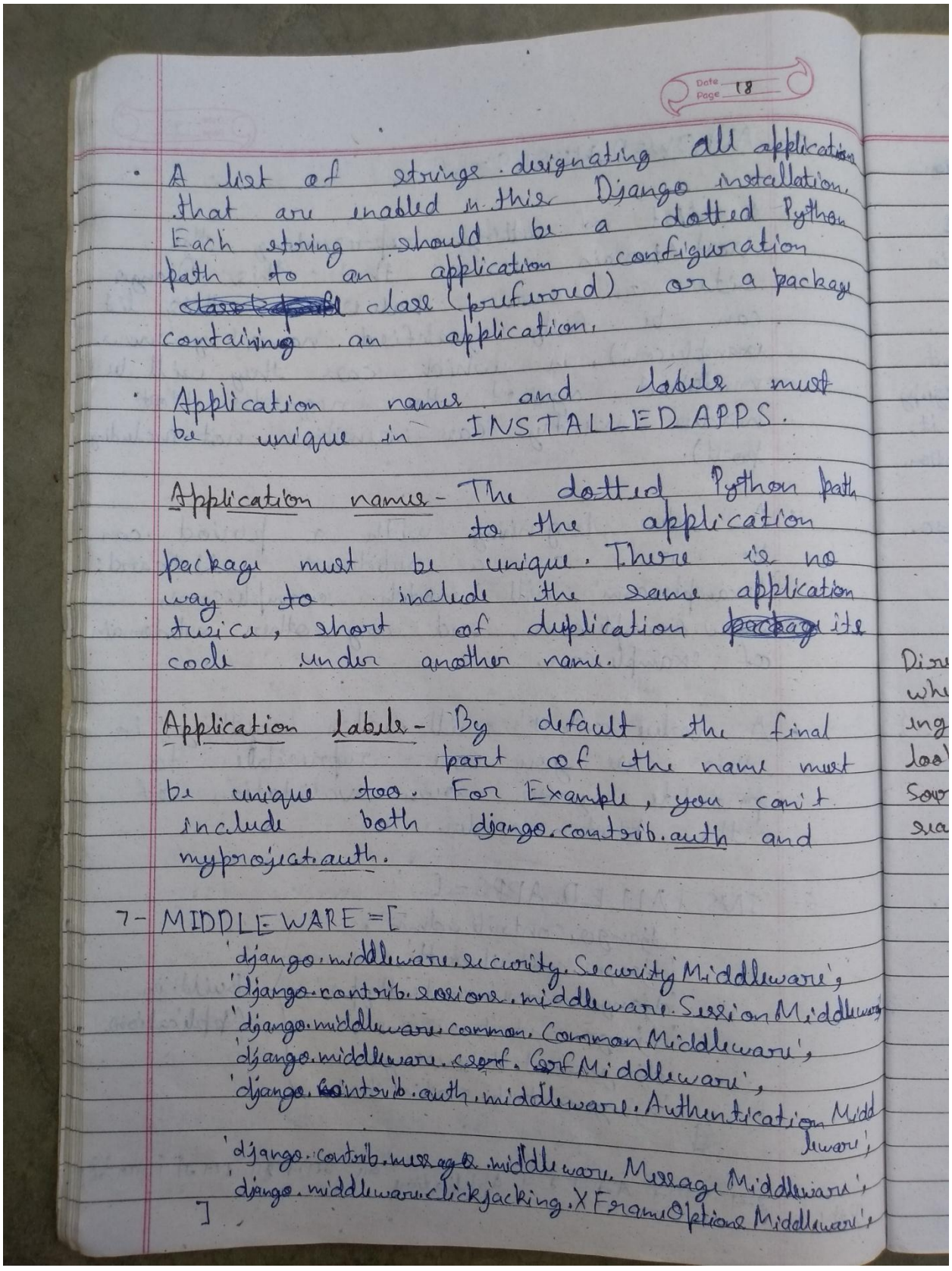


5- ALLOWED_HOSTS = [ ]

- A list of strings representing the host/domain names that this Django site can serve. Values in this list can be fully qualified names (e.g. 'www.example.com'), in which case they will be matched against the request's Host header exactly (case-insensitive, not including port).
- A value beginning with a period can be used as a subdomain wildcard: '.example.com' will match example.com, www.example.com, and any other subdomain of example.com.
- A value of '*' will match anything; in this case you are responsible to provide your own validation of the Host header.

6- INSTALLED_APPS = [
'django.contrib.admin',
'django.contrib.auth',
'django.contrib.contenttypes',
'django.contrib.sessions',
'django.contrib.message',
'django.contrib.staticfiles',
] } Built-in Applications

Note - इस तरह बनाये गये Applications का नाम भी इस में लिखते हैं।



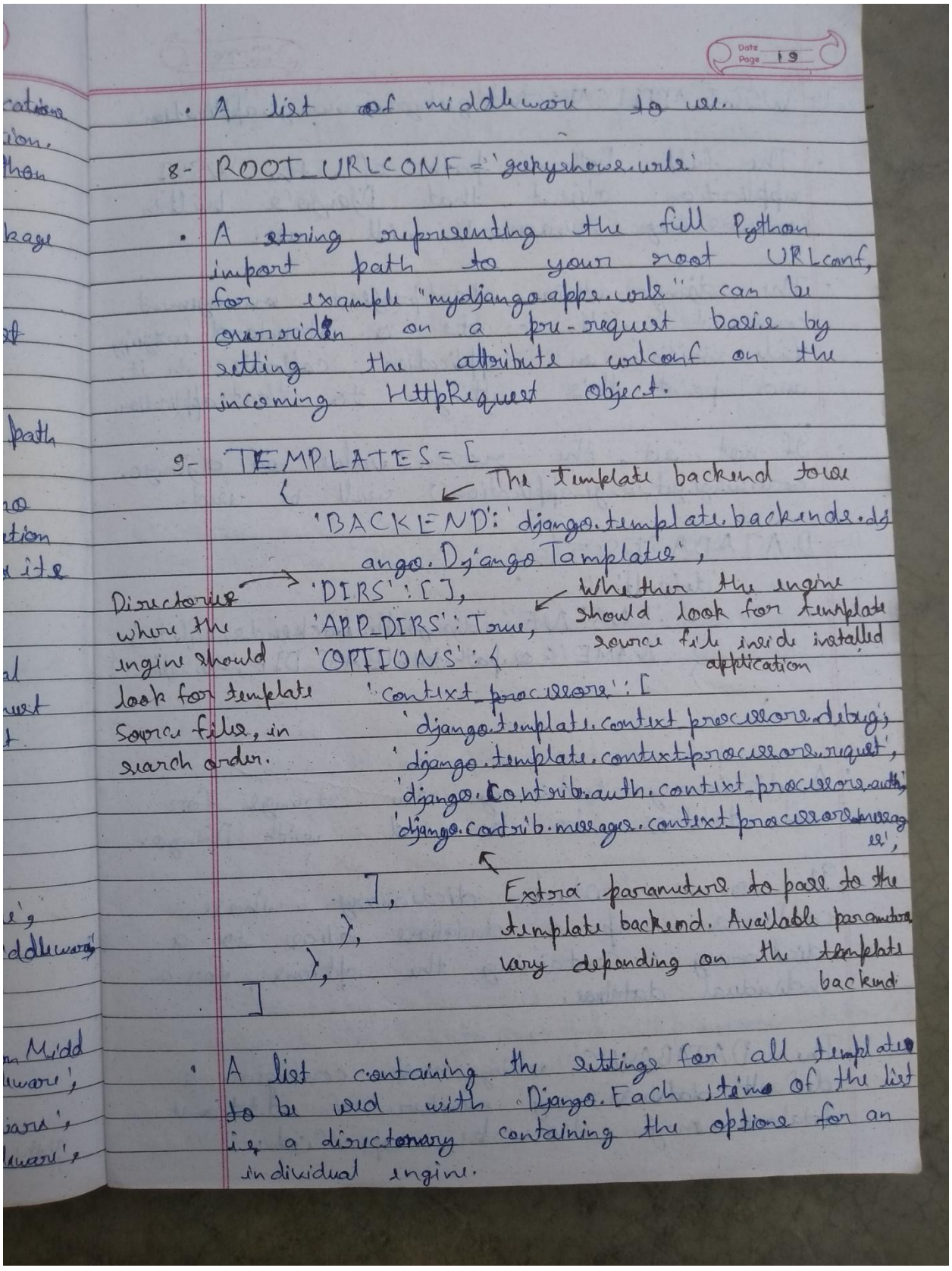
- A list of strings designating all applications that are enabled in this Django installation. Each string should be a dotted Python path to an application configuration class (preferred) or a package containing an application.

- Application names and labels must be unique in INSTALLED_APPS.

Application names - The dotted Python path to the application package must be unique. There is no way to include the same application twice, short of duplicating its code under another name.

Application labels - By default the final part of the name must be unique too. For Example, you can't include both django.contrib.auth and myproject.auth.

7- MIDDLEWARE = [
    'django.middleware.security.SecurityMiddleware',
    'django.contrib.sessions.middleware.SessionMiddleware',
    'django.middleware.common.CommonMiddleware',
    'django.middleware.csrf.CsrfMiddleware',
    'django.contrib.auth.middleware.AuthenticationMiddleware',
    'django.contrib.messages.middleware.MessageMiddleware',
    'django.middleware.clickjacking.XFrameOptionsMiddleware',
]



- A list of middleware to use.

8- ROOT_URLCONF = 'geekyshows.urls'

- A string representing the full Python import path to your root URLconf, for example "mydjango.apps.urls". can be overridden on a per-request basis by setting the attribute urlconf on the incoming HttpRequest object.

9- TEMPLATES = [
{
'BACKEND': 'django.template.backends.django.DjangoTemplates',
'DIRS': [],
'APP_DIRS': True,
'OPTIONS': {
'context_processors': [
'django.template.context_processors.debug',
'django.template.context_processors.request',
'django.contrib.auth.context_processors.auth',
'django.contrib.messages.context_processors.messages',
],
},
},
]

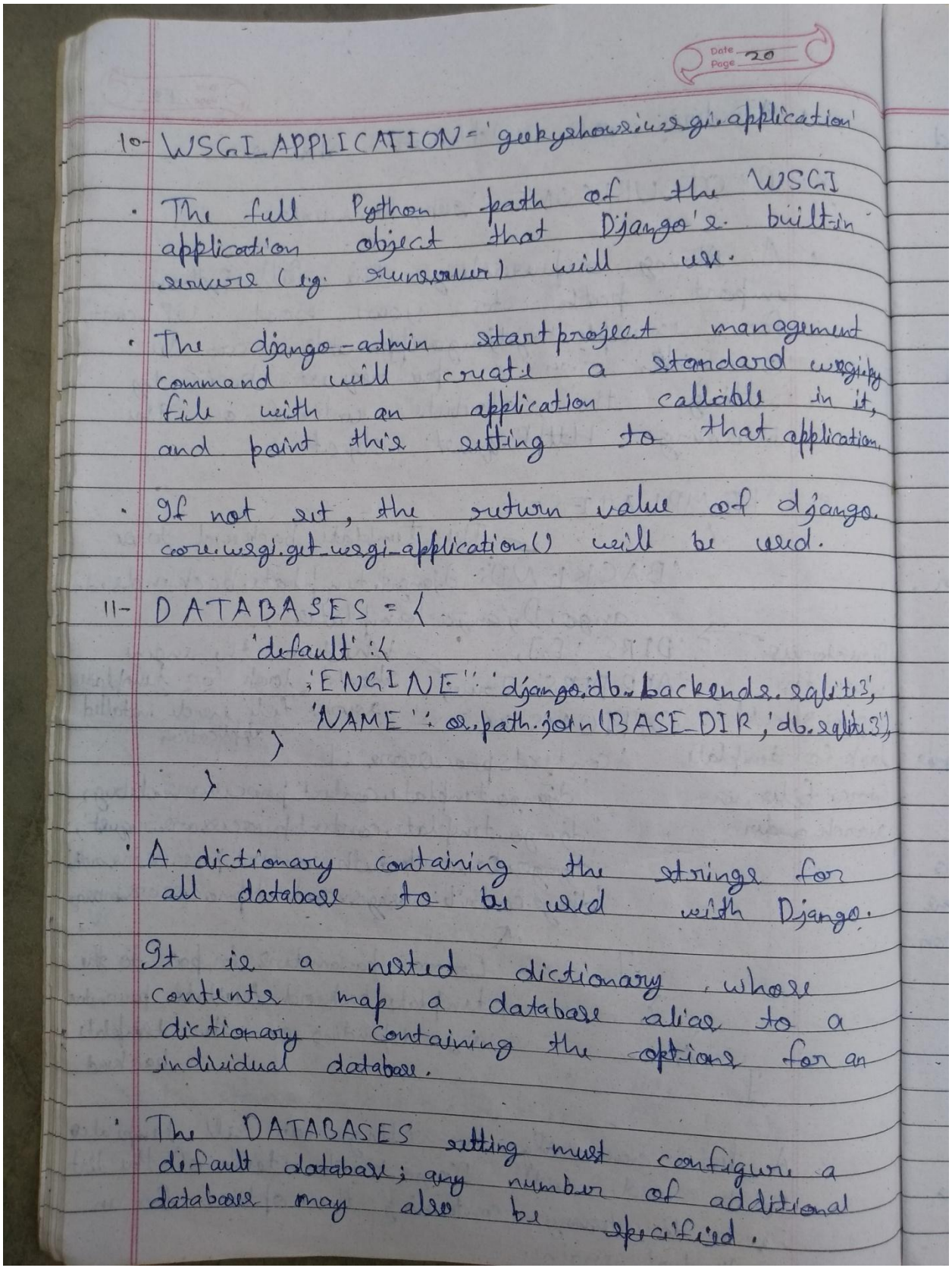
The template backend to use

Directories where the engine should look for template source files, in search order.

Whether the engine should look for template source file inside installed application

Extra parameters to pass to the template backend. Available parameters vary depending on the template backend

- A list containing the settings for all templates to be used with Django. Each item of the list is a directionary containing the options for an individual engine.



10- WSGI_APPLICATION = 'geekyshows.wsgi.application'

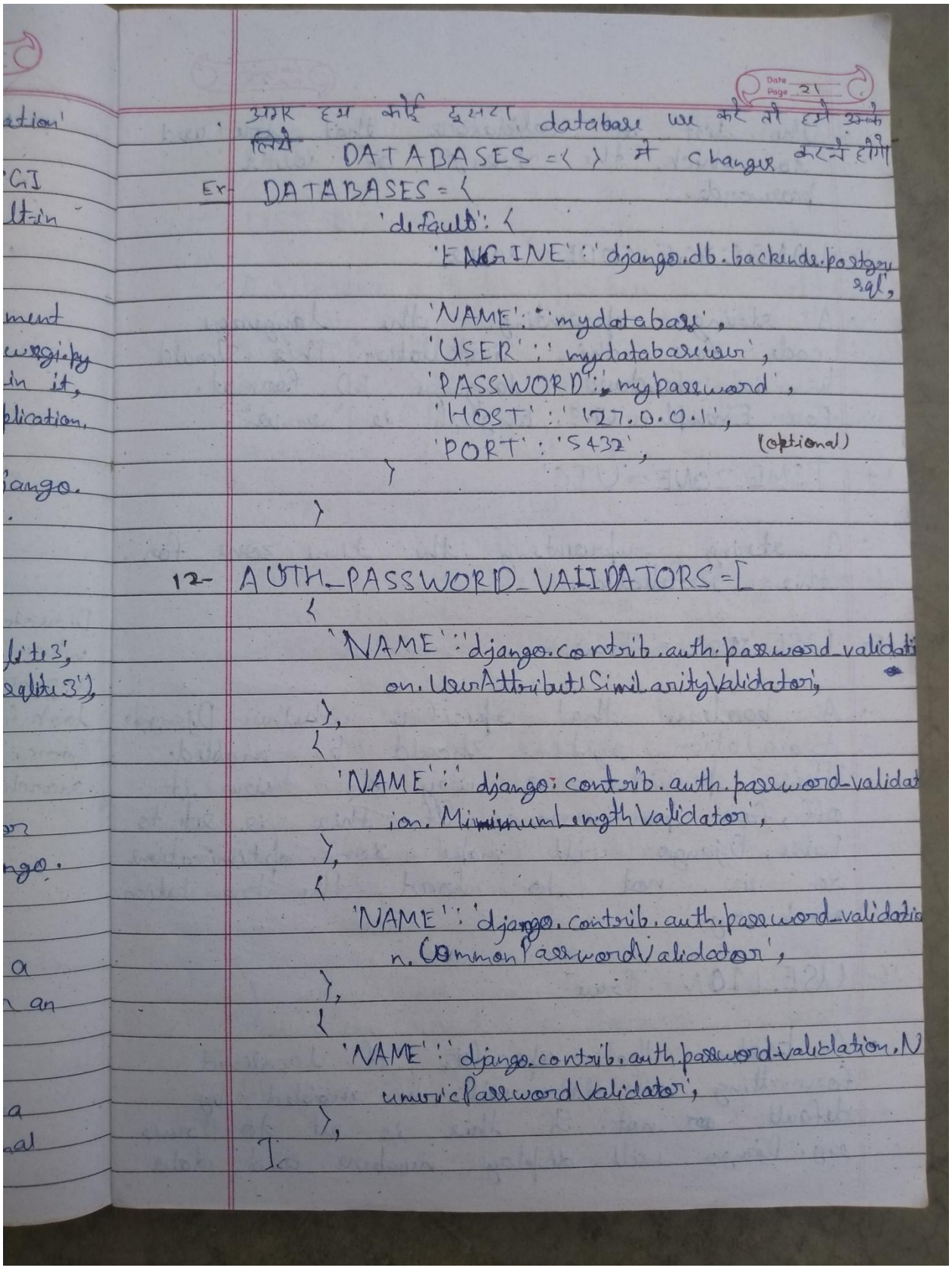
- The full Python path of the WSGI application object that Django's built-in servers (e.g. runserver) will use.
- The django-admin startproject management command will create a standard wsgi.py file with an application callable in it, and point this setting to that application.
- If not set, the return value of django.core.wsgi.get_wsgi_application() will be used.

11- DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': os.path.join(BASE_DIR, 'db.sqlite3'),
    }
}

- A dictionary containing the strings for all databases to be used with Django.

It is a nested dictionary whose contents map a database alias to a dictionary containing the options for an individual database.

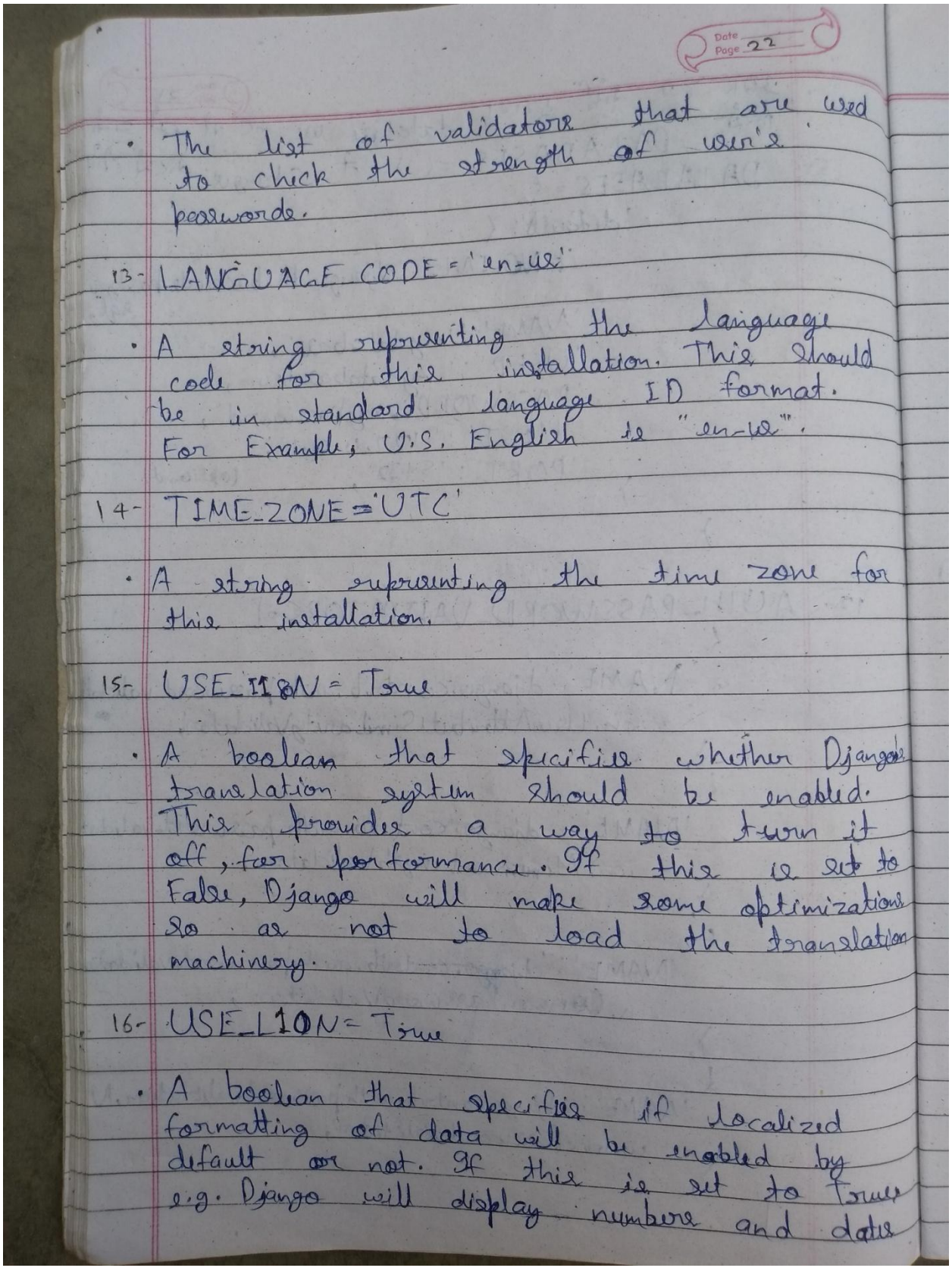
- The DATABASES setting must configure a default database; any number of additional databases may also be specified.



अगर हम कोई दूसरा database use करें तो हमें उसके लिये DATABASES = { } में changes करने होंगे।

Ex- DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.postgresql',
        'NAME': 'mydatabase',
        'USER': 'mydatabaseuser',
        'PASSWORD': 'mypassword',
        'HOST': '127.0.0.1',
        'PORT': '5432', (optional)
    }
}

12- AUTH_PASSWORD_VALIDATORS = [
    {
        'NAME': 'django.contrib.auth.password_validation.UserAttributeSimilarityValidator',
    },
    {
        'NAME': 'django.contrib.auth.password_validation.MinimumLengthValidator',
    },
    {
        'NAME': 'django.contrib.auth.password_validation.CommonPasswordValidator',
    },
    {
        'NAME': 'django.contrib.auth.password_validation.NumericPasswordValidator',
    },
]



Date
Page 22

- The list of validators that are used to check the strength of user's passwords.

13- LANGUAGE_CODE = 'en-us'

- A string representing the language code for this installation. This should be in standard language ID format. For Example, U.S. English is "en-us".

14- TIME_ZONE = 'UTC'

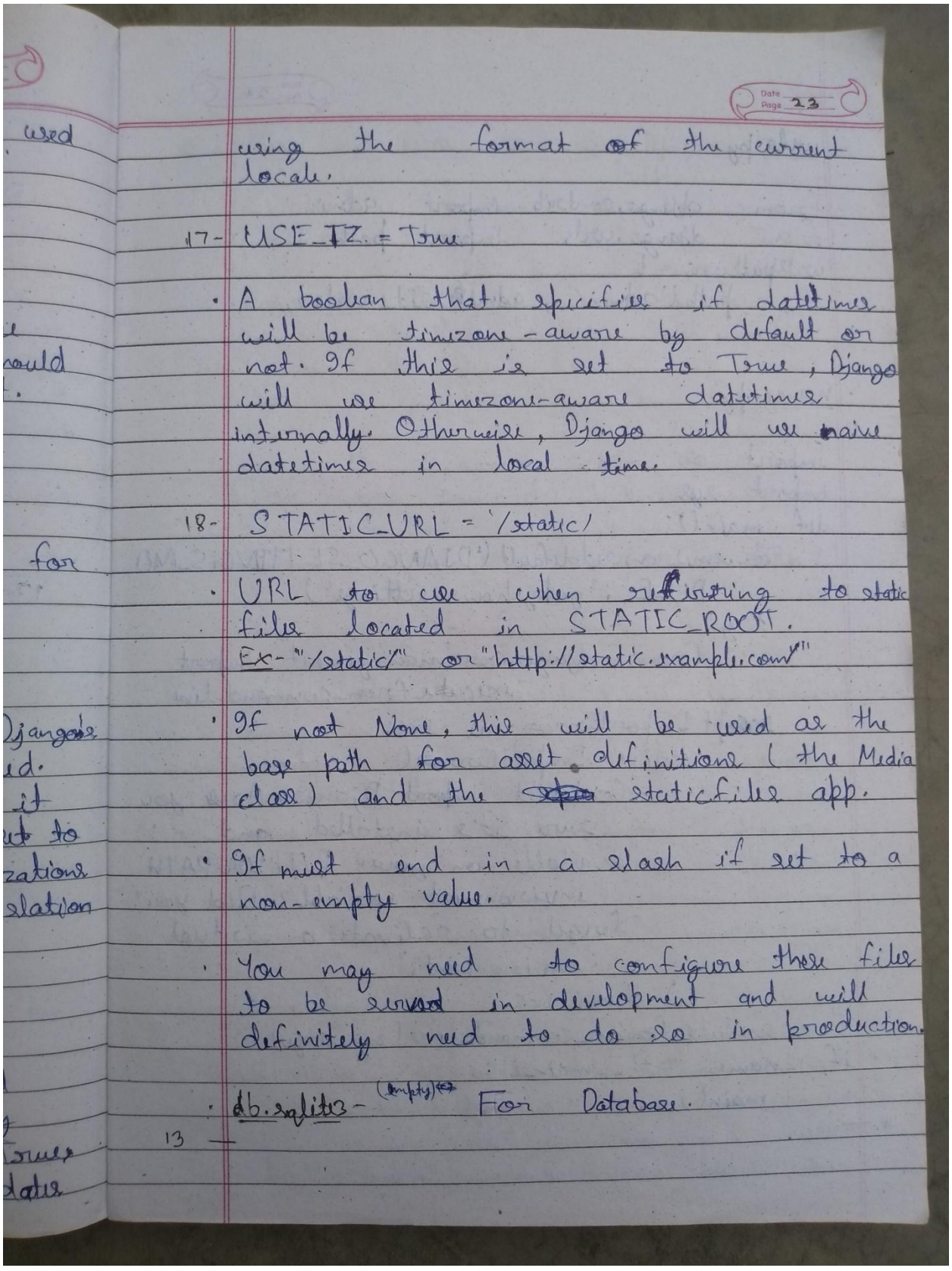
- A string representing the time zone for this installation.

15- USE_I18N = True

- A boolean that specifies whether Django's translation system should be enabled. This provides a way to turn it off, for performance. If this is set to False, Django will make some optimizations so as not to load the translation machinery.

16- USE_L10N = True

- A boolean that specifies if localized formatting of data will be enabled by default or not. If this is set to True e.g. Django will display numbers and dates



using the format of the current locale.

17- USE_TZ = True

- A boolean that specifies if datetimes will be timezone-aware by default or not. If this is set to True, Django will use timezone-aware datetimes internally. Otherwise, Django will use naive datetimes in local time.

18- STATIC_URL = '/static/'

- URL to use when referring to static files located in STATIC_ROOT.
  Ex- "/static/" or "http://static.example.com/"

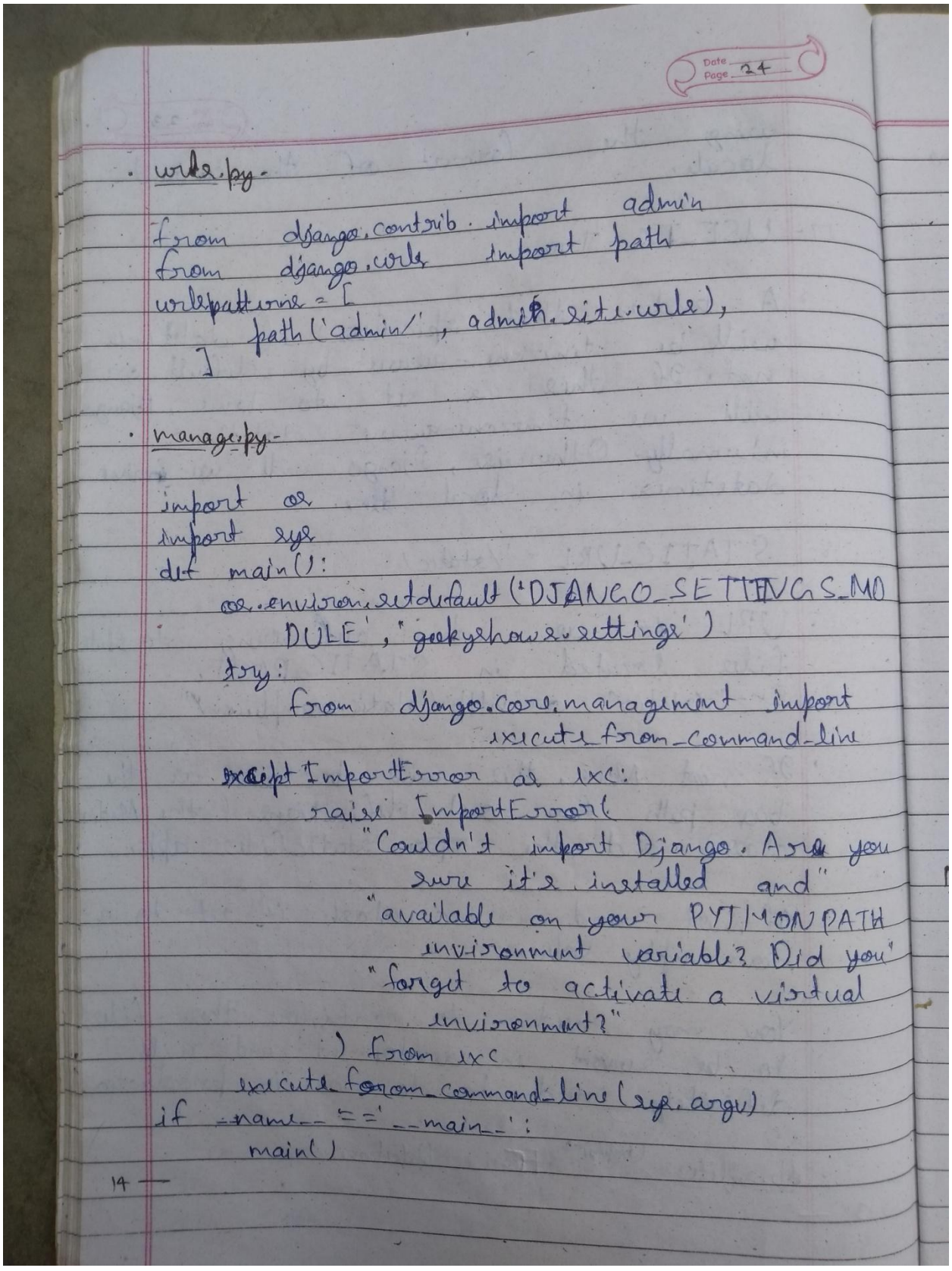
- If not None, this will be used as the base path for asset definitions (the Media class) and the staticfiles app.

- It must end in a slash if set to a non-empty value.

- You may need to configure these files to be served in development and will definitely need to do so in production.

- db.sqlite3 - (empty) For Database.

13 —

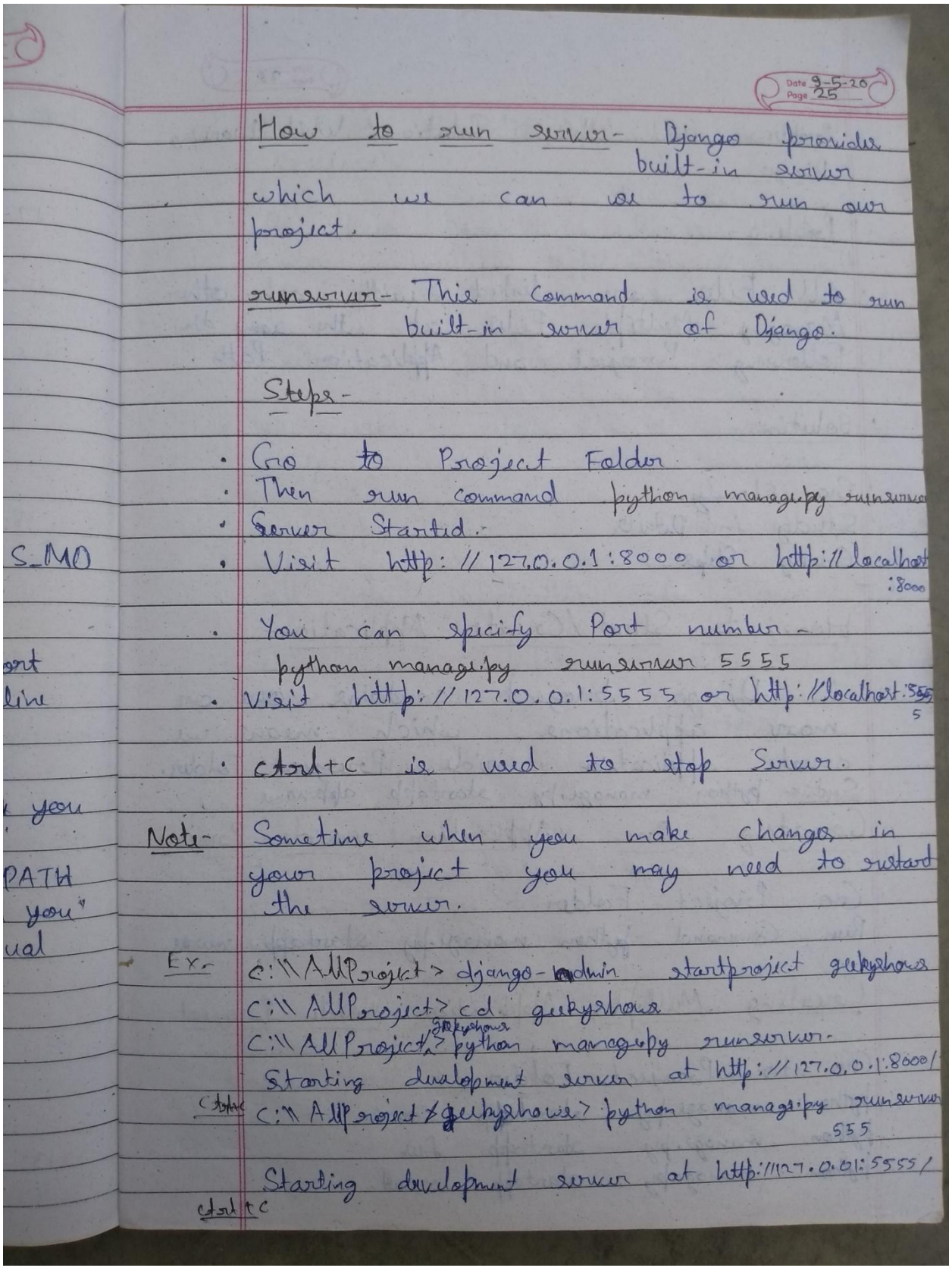


- urls.py -

```
from django.contrib import admin
from django.urls import path
urlpatterns = [
    path('admin/', admin.site.urls),
]
```

- manage.py -

```
import os
import sys
def main():
    os.environ.setdefault('DJANGO_SETTINGS_MODULE', 'geekyshows.settings')
    try:
        from django.core.management import execute_from_command_line
    except ImportError as exc:
        raise ImportError(
            "Couldn't import Django. Are you sure it's installed and "
            "available on your PYTHONPATH environment variable? Did you "
            "forget to activate a virtual environment?"
        ) from exc
    execute_from_command_line(sys.argv)
if __name__ == '__main__':
    main()
```



Date 9-5-20
Page 25

How to run server- Django provides built-in server which we can use to run our project.

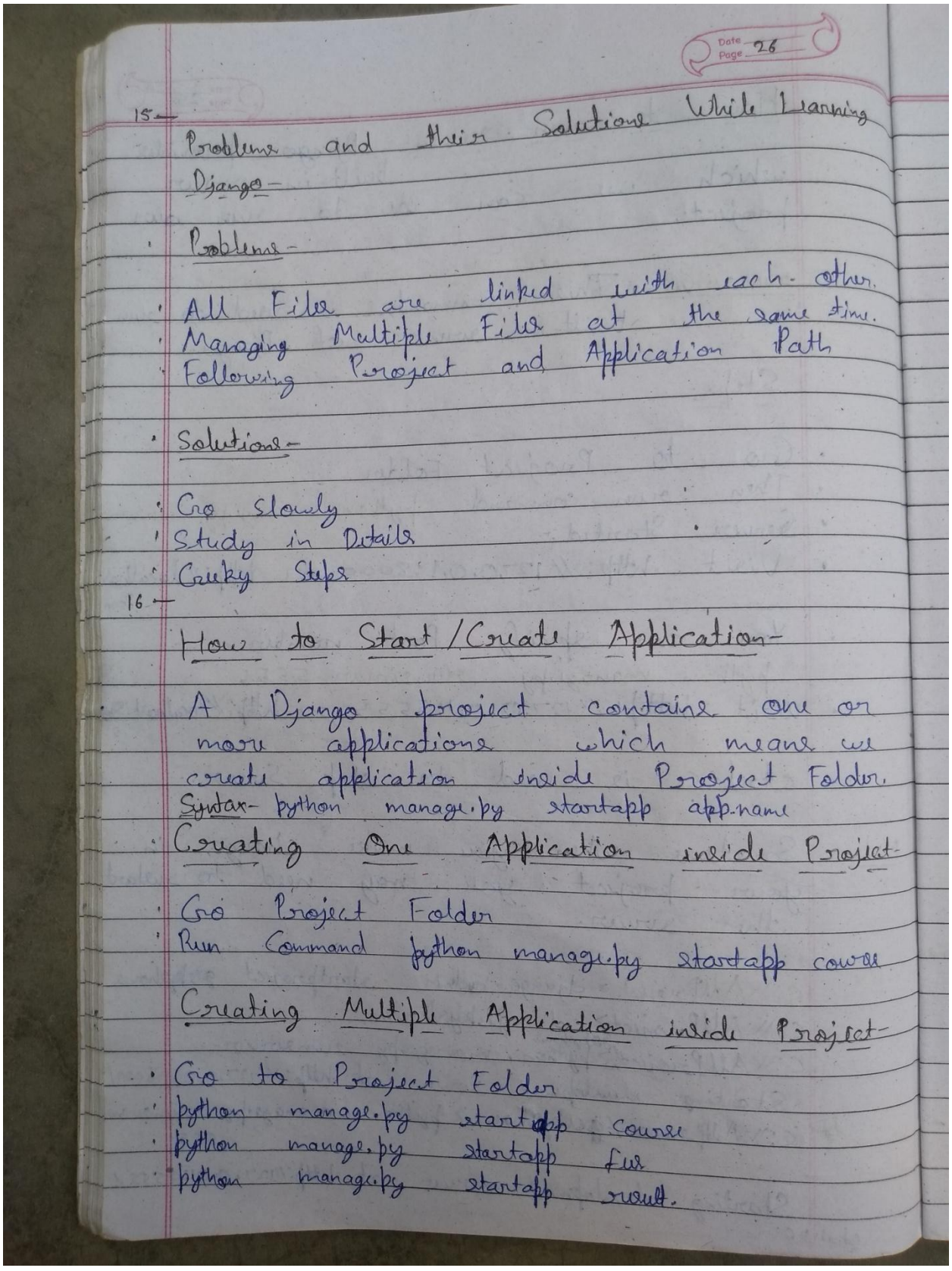
runserver- This command is used to run built-in server of Django.

Steps-

- Go to Project Folder
- Then run command python manage.py runserver
- Server Started.
- Visit http://127.0.0.1:8000 or http://localhost:8000
- You can specify Port number - python manage.py runserver 5555
- Visit http://127.0.0.1:5555 or http://localhost:5555
- ctrl+c is used to stop Server.

Note- Sometime when you make changes in your project you may need to restart the server.

Ex- c:\AllProject> django-admin startproject geekyshows
c:\AllProject> cd geekyshows
c:\AllProject\geekyshows> python manage.py runserver.
Starting development server at http://127.0.0.1:8000/
ctrl+c
c:\AllProject\geekyshows> python manage.py runserver 555
Starting development server at http://127.0.0.1:5555/
ctrl+c



Date Page 26

15 —

Problems and their Solutions while Learning Django -

- Problems -

- All Files are linked with each other.
- Managing Multiple Files at the same time.
- Following Project and Application Path

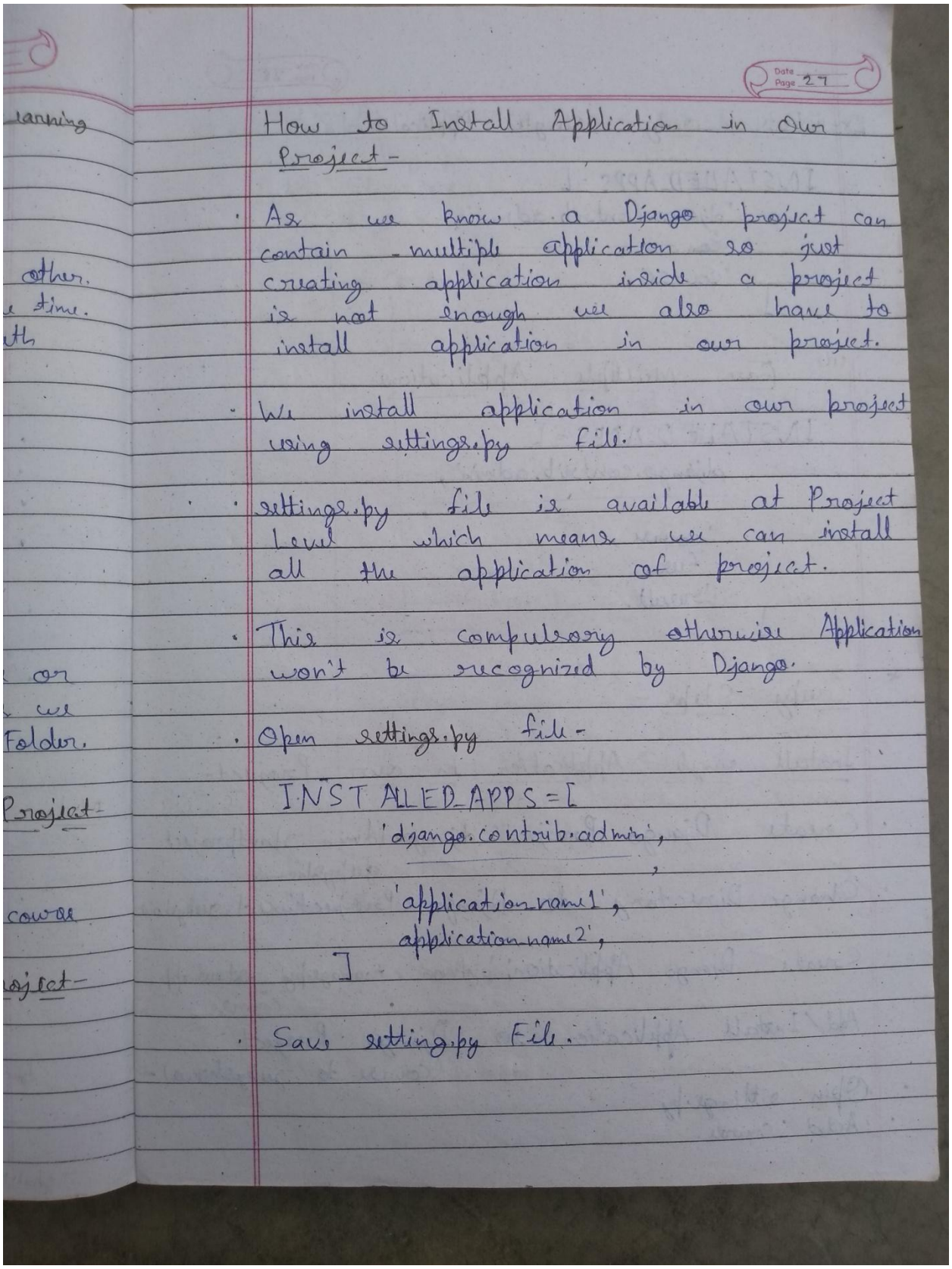
- Solutions -

- Go slowly
- Study in Details
- Geeky Steps

16 —

How to Start / Create Application -

- A Django project contains one or more applications which means we create application inside Project Folder.
  Syntax - python manage.py startapp app_name
- Creating One Application inside Project -
- Go Project Folder
- Run Command python manage.py startapp course
- Creating Multiple Application inside Project -
- Go to Project Folder
- python manage.py startapp course
- python manage.py startapp fun
- python manage.py startapp result.

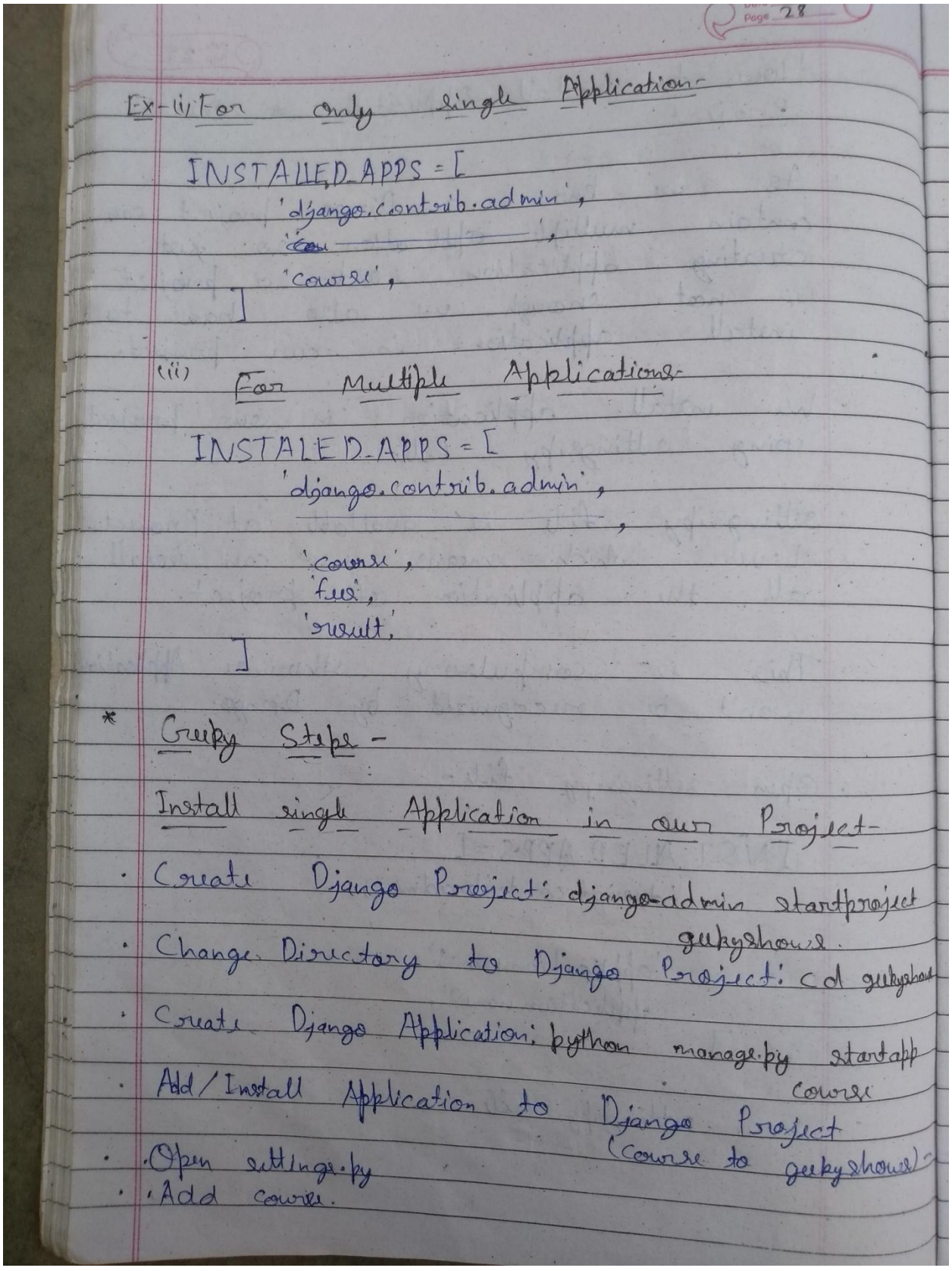


How to Install Application in Our Project -

- As we know a Django project can contain multiple application so just creating application inside a project is not enough we also have to install application in our project.
- We install application in our project using settings.py file.
- settings.py file is available at Project Level which means we can install all the application of project.
- This is compulsory otherwise Application won't be recognized by Django.
- Open settings.py file -

```
INSTALLED_APPS = [
    'django.contrib.admin',
    'application_name1',
    'application_name2',
]
```

- Save setting.py File.



Page 28

Ex- (i) For only single Application-

```
INSTALLED_APPS = [
    'django.contrib.admin',
    'course',
]
```

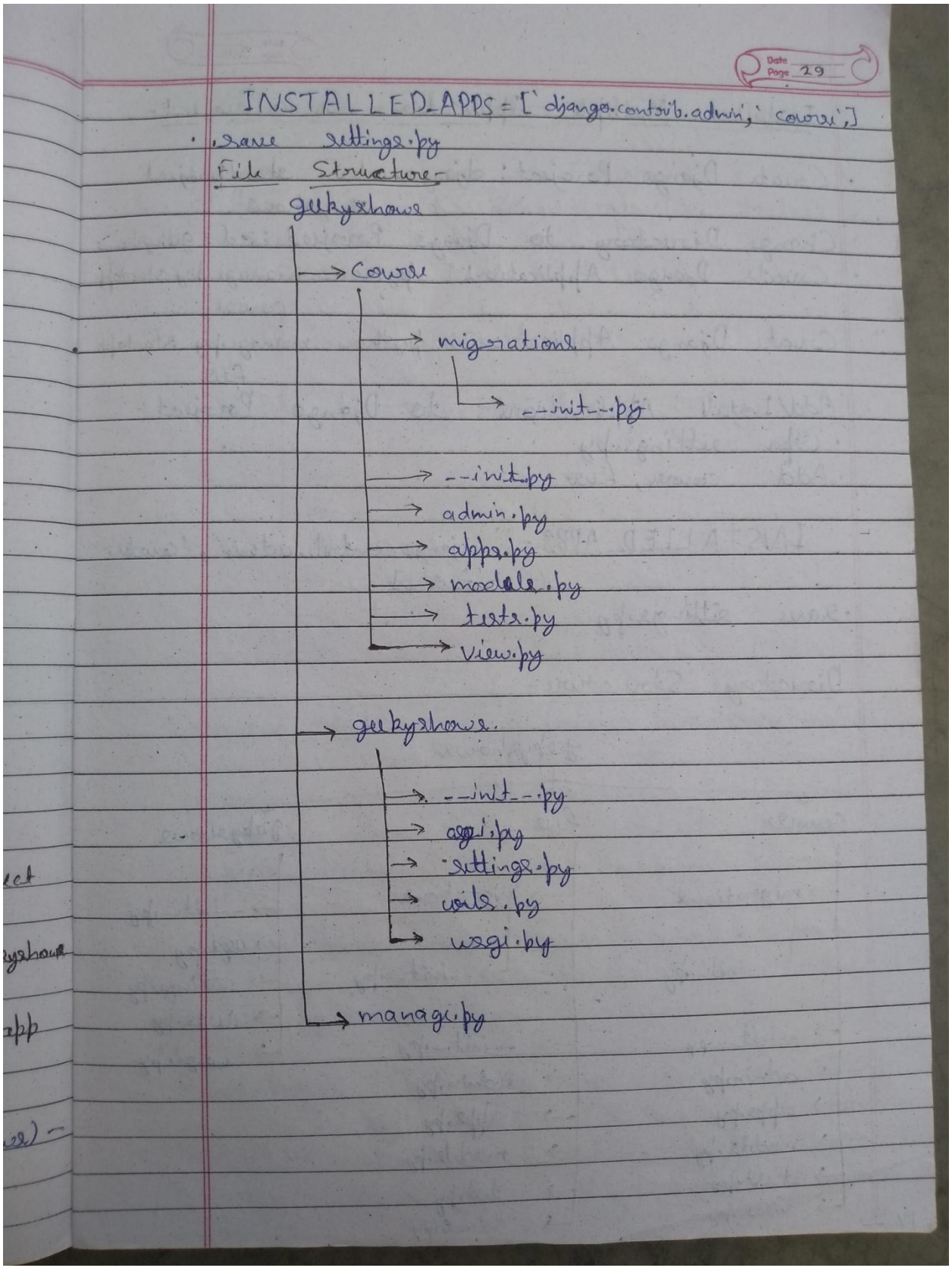
(ii) For Multiple Applications-

```
INSTALED_APPS = [
    'django.contrib.admin',
    'course',
    'fee',
    'result',
]
```

* Geeky Steps -

Install single Application in our Project-

- Create Django Project: django-admin startproject geekyshows.
- Change Directory to Django Project: cd geekyshows
- Create Django Application: python manage.py startapp course
- Add/Install Application to Django Project (course to geekyshows)-
  - Open settings.py
  - Add course.


Date
Page 29
INSTALLED_APPS = [' django.contrib.admin', 'course',]
, save settings.py
File Structure-
geekyshows
course
migrations
__init__.py
__init.py
admin.py
apps.py
models.py
tests.py
view.py
geekyshows
__init__.py
asgi.py
settings.py
urls.py
wsgi.py
manage.py

Date
Page 30

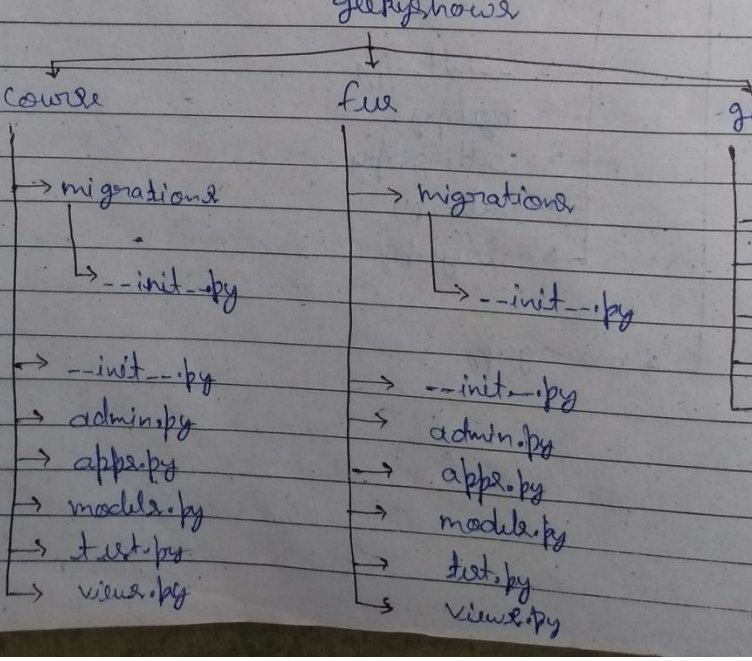
Install Multiple Applications in our Project -

- Create Django Project: django-admin startproject geekyshows
- Change Directory to Django Project: cd geekyshows
- Create Django Application1: python manage.py startapp course
- Create Django Application2: python manage.py startapp fun
- Add/Install Applications to Django Project -
  - Open settings.py
  - Add course, fun

```
INSTALLED_APPS = [ 'django.contrib.admin', 'course', 'fun', ]
```

- Save settings.py.

Directory Structure -

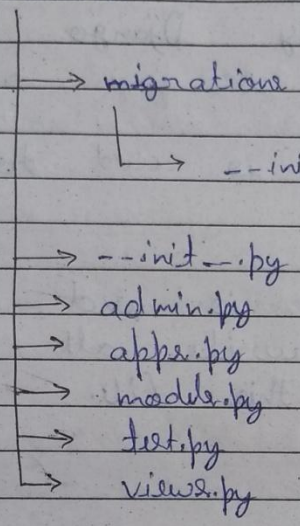


17

Date Page 31

Application Directory Structure -

- Go To Project Folder
- Run Command python manage.py startapp course



- **migrations** - This folder contains __init__.py file which means it's a python package. It also contains all file which are created after running makemigration command.

- **__init__.py** - This Folder which contains __init__.py file is considered as python Package.

- **admin.py** - This file is used to register sql tables so we could perform CRUD operation from Admin Application. Admin Application is provided by Django to perform CRUD operation.

Date Page 32

- apps.py - This file is used to config app.
- models.py - This file is used to create our own model class later these classes will be converted into database table by Django for our application.
- tests.py - This file is used to create tests.
- views.py - This file is used to create view. We write all the business related code in this file.

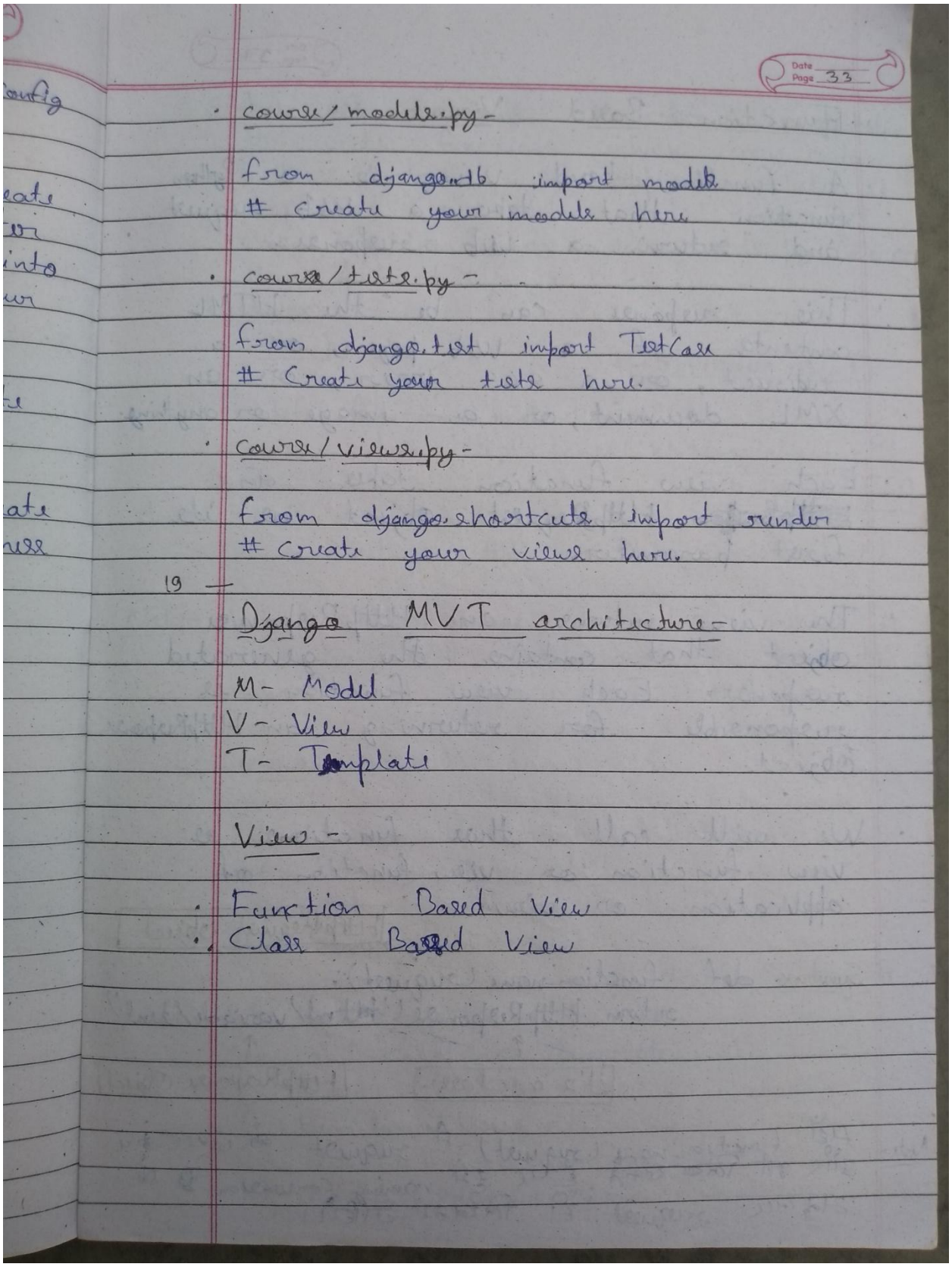
Code inside files -

- course/migrations/__init__.py - (empty)
- course/__init__.py - (empty)
- course/admin.py -

```
from django.contrib import admin
# Register your models here.
```

- course/apps.py -

```
from django.apps import AppConfig
class CourseConfig(AppConfig):
    name = 'course'
```



Date
Page 33

- course/models.py -

```
from django.db import models
# Create your models here.
```

- course/tests.py -

```
from django.test import TestCase
# Create your tests here.
```

- course/views.py -

```
from django.shortcuts import render
# Create your views here.
```

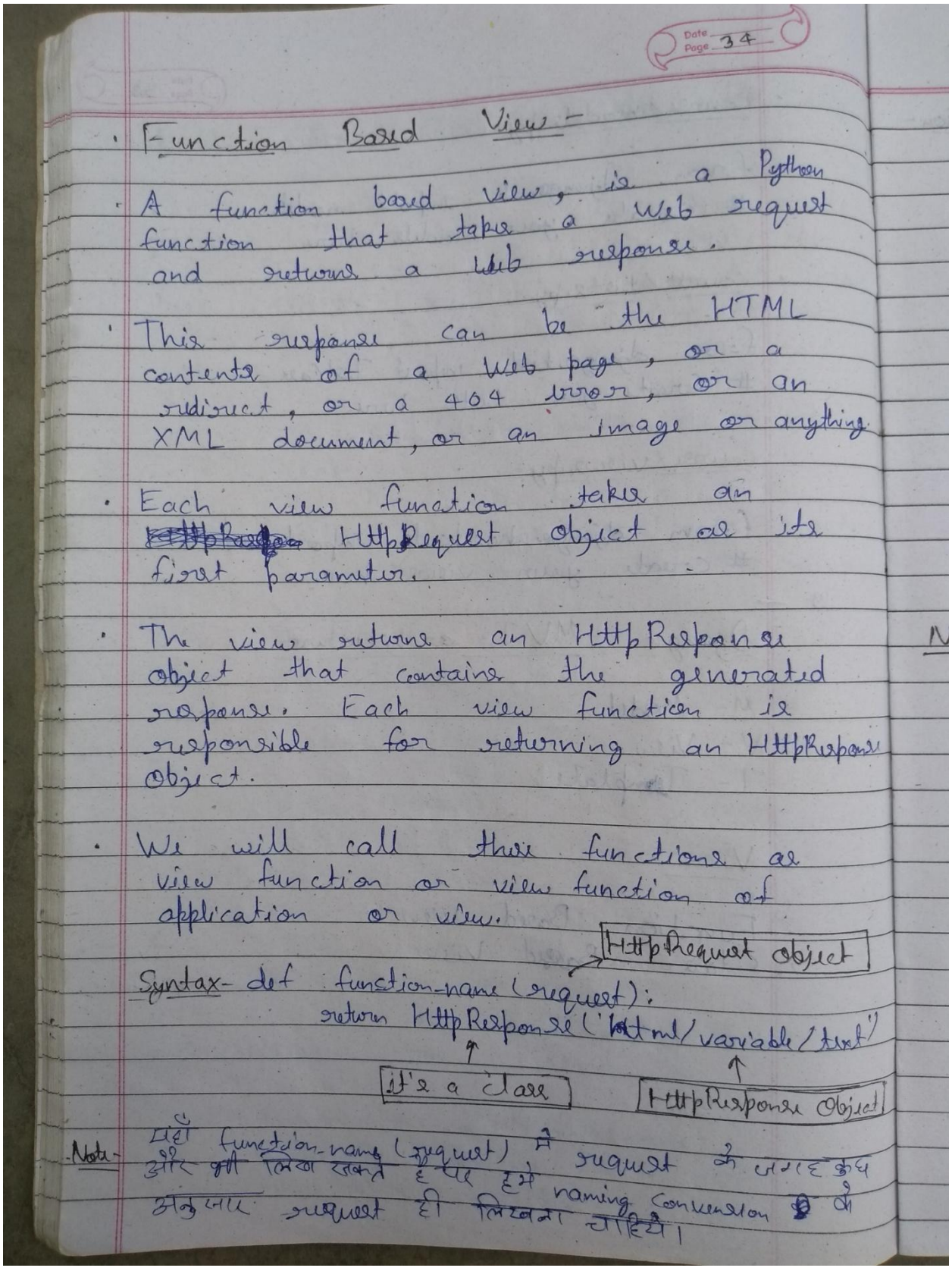
19 —

Django MVT architecture -

M - Model
V - View
T - Template

View -

- Function Based View
- Class Based View



Date
Page 34

- Function Based View -

- A function based view, is a Python function that take a Web request and returns a Web response.

- This response can be the HTML contents of a Web page, or a redirect, or a 404 error, or an XML document, or an image or anything.

- Each view function takes an ~~HttpRequest~~ HttpRequest object as its first parameter.

- The view returns an HttpResponse object that contains the generated response. Each view function is responsible for returning an HttpResponse object.

- We will call this functions as view function or view function of application or view.

HttpRequest object

Syntax- def function-name(request):
    return HttpResponse('html/variable/text')

it's a class

HttpResponse Object

Note- यहाँ function-name(request) में request के जगह कुछ और भी लिख सकते है पर हमें naming convention के अनुसार request ही लिखना चाहिये।

Date Page 35

We use views.py file of the application to write functions which may contain business logic of application, later it required to define url name for this function in the urls.py file of the project.

Syntax -

views.py -

HttpRequest object

```
def function_name1(request):
    return HttpResponse('html/variable/text')
```

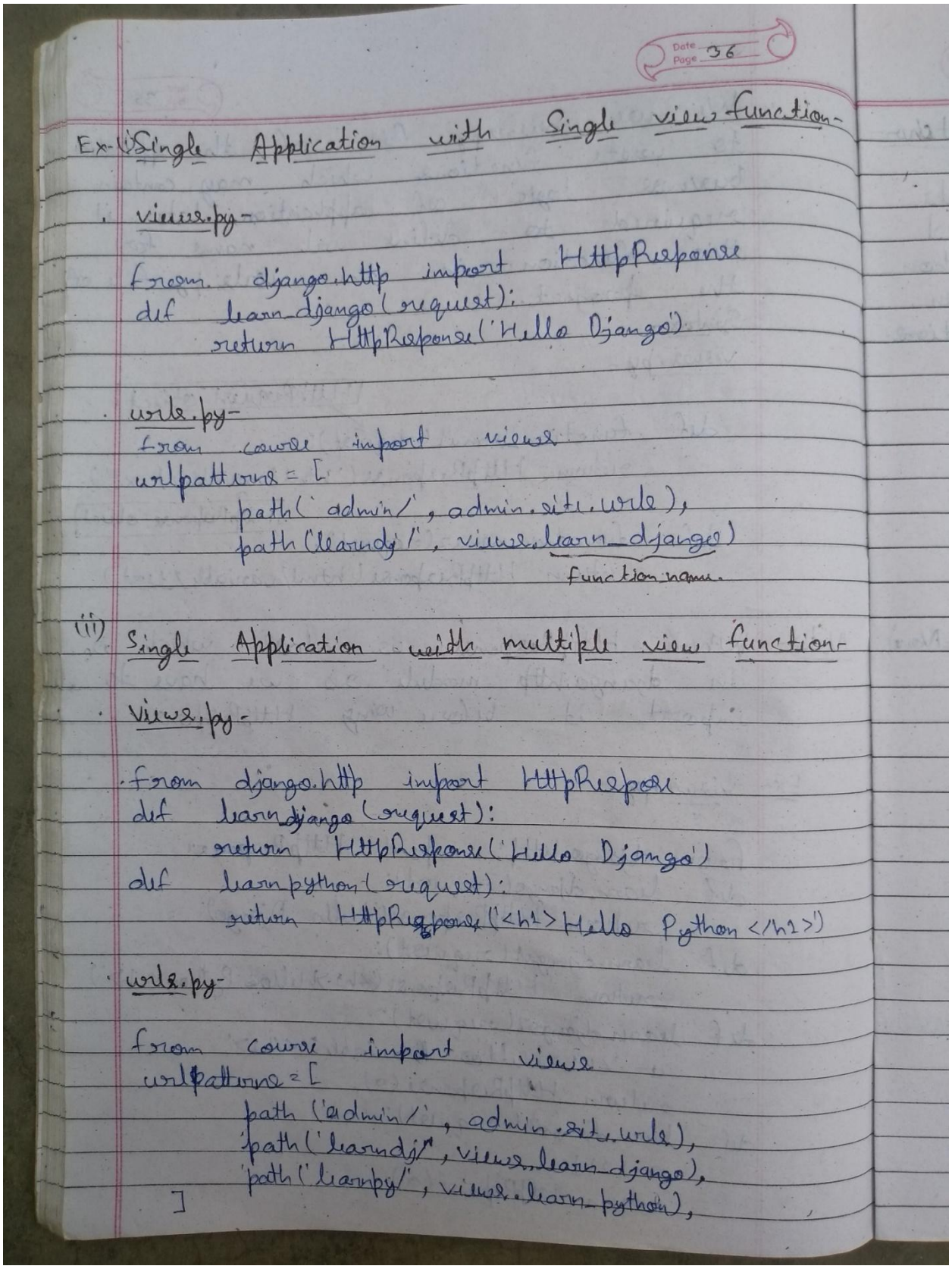
HttpResponse object

```
def function_name2(request):
    return HttpResponse('html/variable/text')
```

Note- Where HttpResponse is class which is in django.http module so we have to import it before using HttpResponse.

Ex- views.py -

```
from django.http import HttpResponse
def learn_django(request):
    return HttpResponse('Hello Django')
def learn_django1(request):
    return HttpResponse('<h1>Hello Python</h1>')
def learn_django2(request):
    a='<h1>Hello Variable</h1>'
    return HttpResponse(a)
def learn_django3(request):
    a=10+10
    return HttpResponse(a)
```



Date Page 36

Ex- (i) Single Application with Single view function-

. views.py-

```
from django.http import HttpResponse
def learn_django(request):
    return HttpResponse('Hello Django')
```

. urls.py-

```
from course import views
urlpatterns = [
    path('admin/', admin.site.urls),
    path('learndj/', views.learn_django)
                     function name.
```

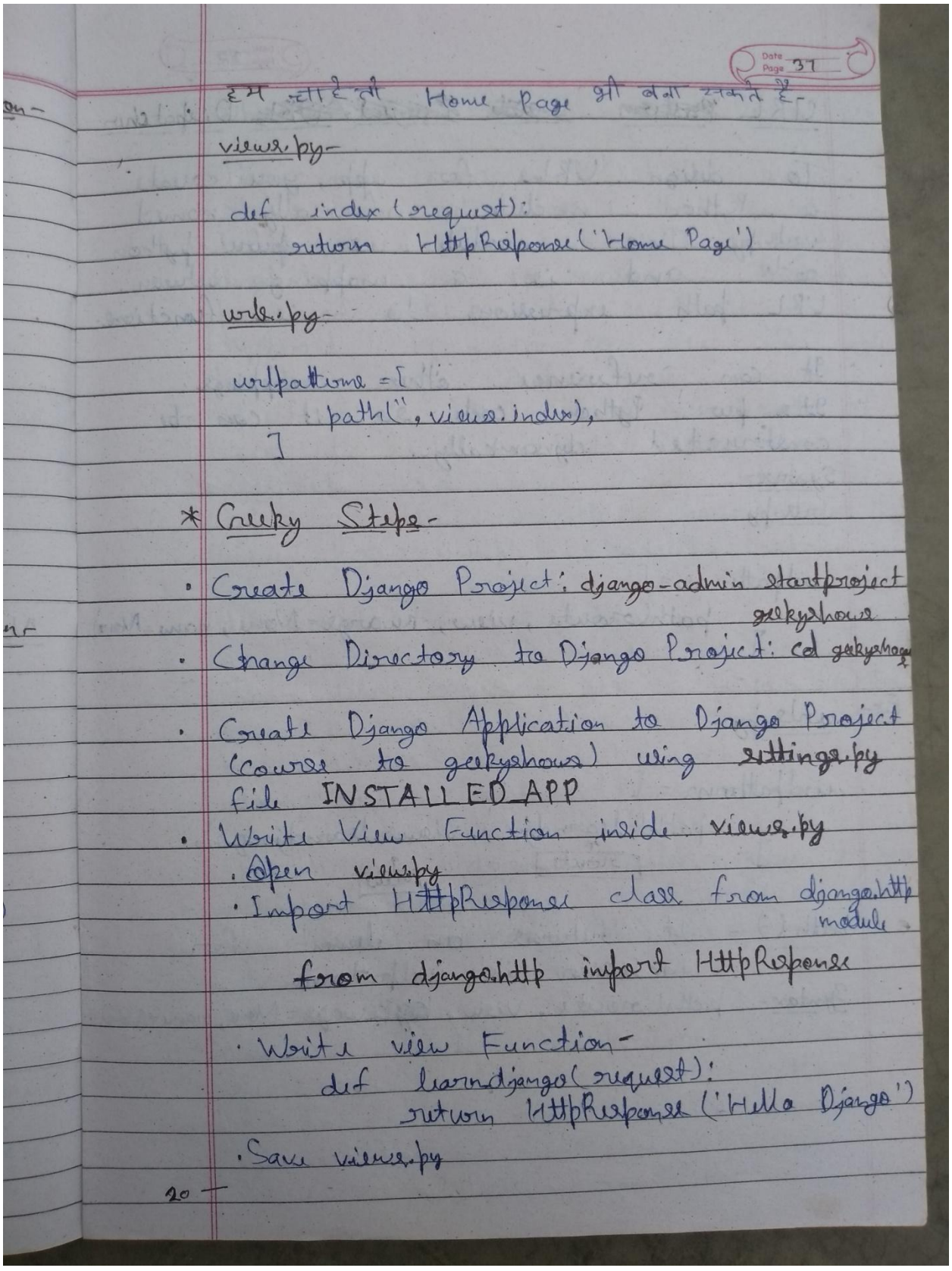
(ii) Single Application with multiple view function-

. Views.py-

```
from django.http import HttpRespons
def learndjango(request):
    return HttpResponse('Hello Django')
def learnpython(request):
    return HttpResponse('<h1>Hello Python</h1>')
```

. urls.py-

```
from course import views
urlpatterns = [
    path('admin/', admin.site.urls),
    path('learndj/', views.learn_django),
    path('learnpy/', views.learn_python),
]
```



Date Page 37

इस तरीके से Home Page भी बना सकते है-

views.py-

```
def index(request):
    return HttpResponse('Home Page')
```

urls.py-

```
urlpatterns = [
    path('', views.index),
]
```

* Quicky Steps-

- Create Django Project: django-admin startproject geekyshows
- Change Directory to Django Project: cd geekyshows
- Create Django Application to Django Project (Course to geekyshows) using settings.py file INSTALLED_APP
- Write View Function inside views.py
  - Open views.py
  - Import HttpResponse class from django.http module

```
from django.http import HttpResponse
```

- Write view Function-

```
def learndjango(request):
    return HttpResponse('Hello Django')
```

- Save views.py

Date Page 38

URL ~~Pattern~~ ~~Input~~ ~~Project/URLs~~ Dispatcher

- To design URLs for app, you create a Python module informally named urls.py. This module is pure python code and is a mapping between URL path expressions to view functions.
- It can reference other mappings.
- It's pure Python code so it can be constructed dynamically.

Syntax-

urls.py-

```
urlpatterns = [
        path(route, view, kwargs=None, name=None),
    ]
```

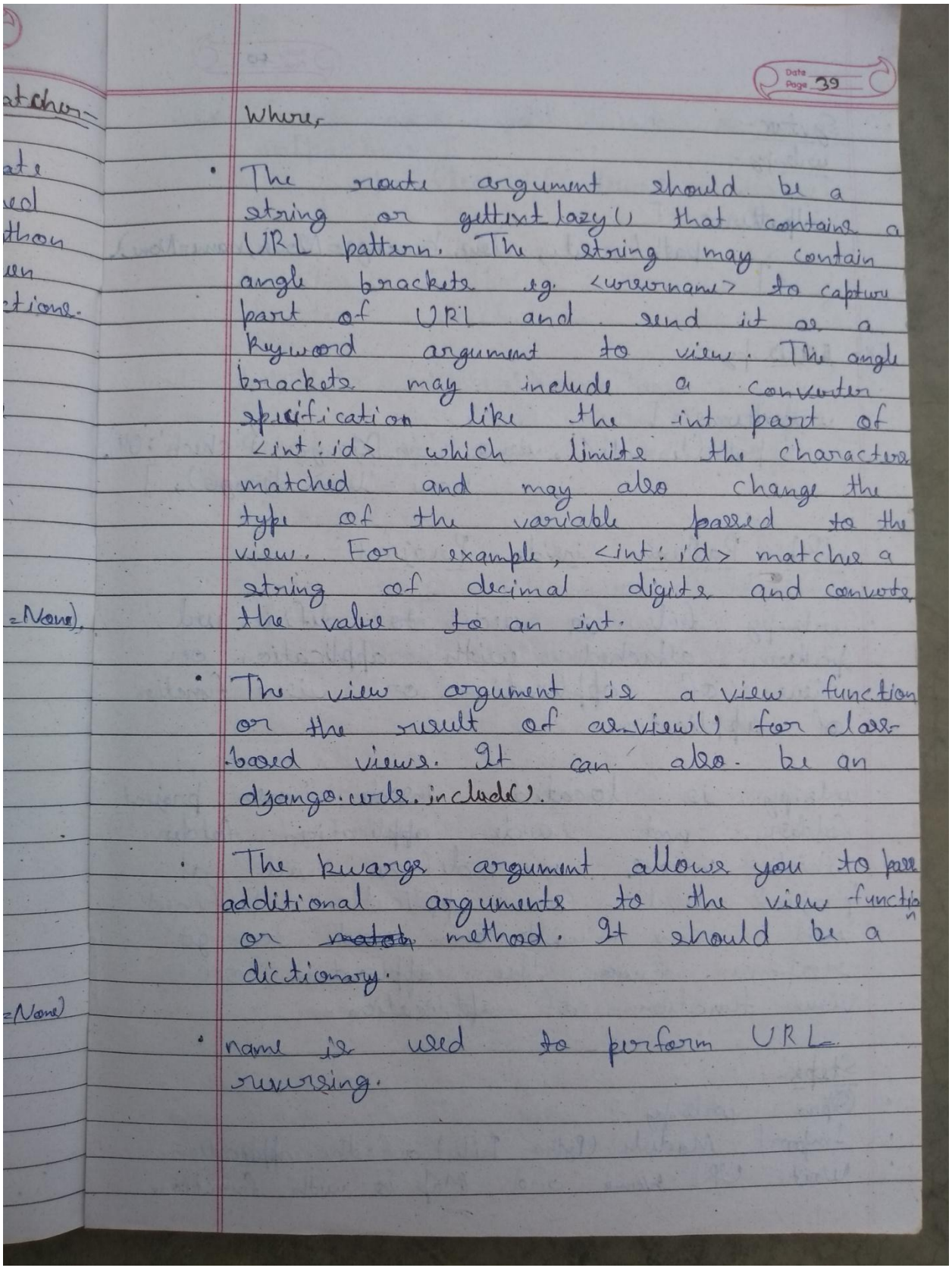
Ex- urls.py-

```
urlpatterns = [
        path('learndj/', views.learndjango),
    ]
```

route — view

- path() - It returns an element for inclusion in urlpatterns.

Syntax- path(route, view, kwargs=None, name=None)



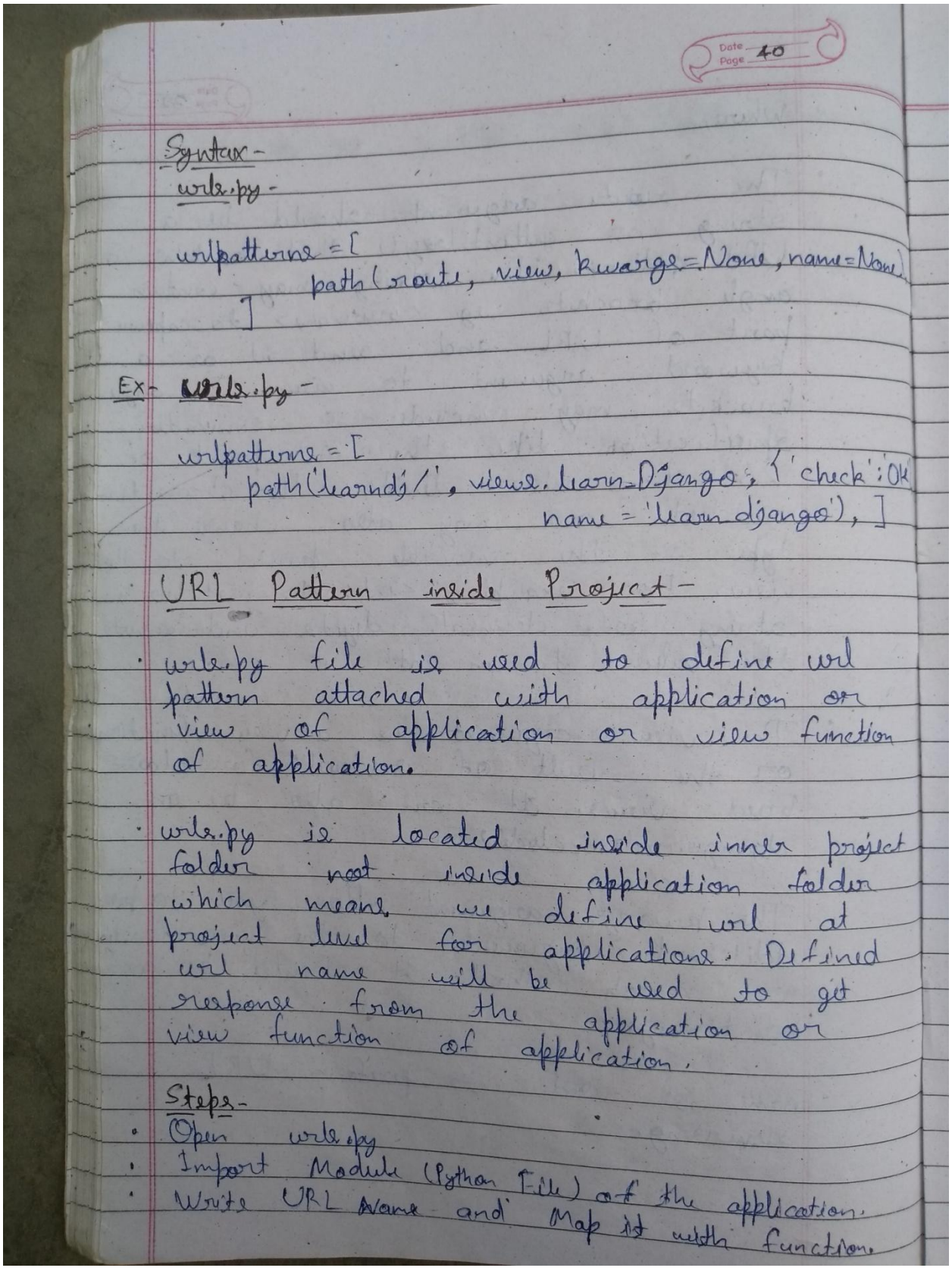
Where,

- The route argument should be a string or gettext_lazy() that contains a URL pattern. The string may contain angle brackets e.g. <username> to capture part of URL and send it as a keyword argument to view. The angle brackets may include a converter specification like the int part of <int:id> which limits the characters matched and may also change the type of the variable passed to the view. For example, <int:id> matches a string of decimal digits and converts the value to an int.

- The view argument is a view function or the result of as_view() for class-based views. It can also be an django.urls.include().

- The kwargs argument allows you to pass additional arguments to the view function or ~~match~~ method. It should be a dictionary.

- name is used to perform URL reversing.



Date Page 40

Syntax -

urls.py -

```
urlpatterns = [
    path(route, view, kwargs=None, name=None),
]
```

Ex- urls.py -

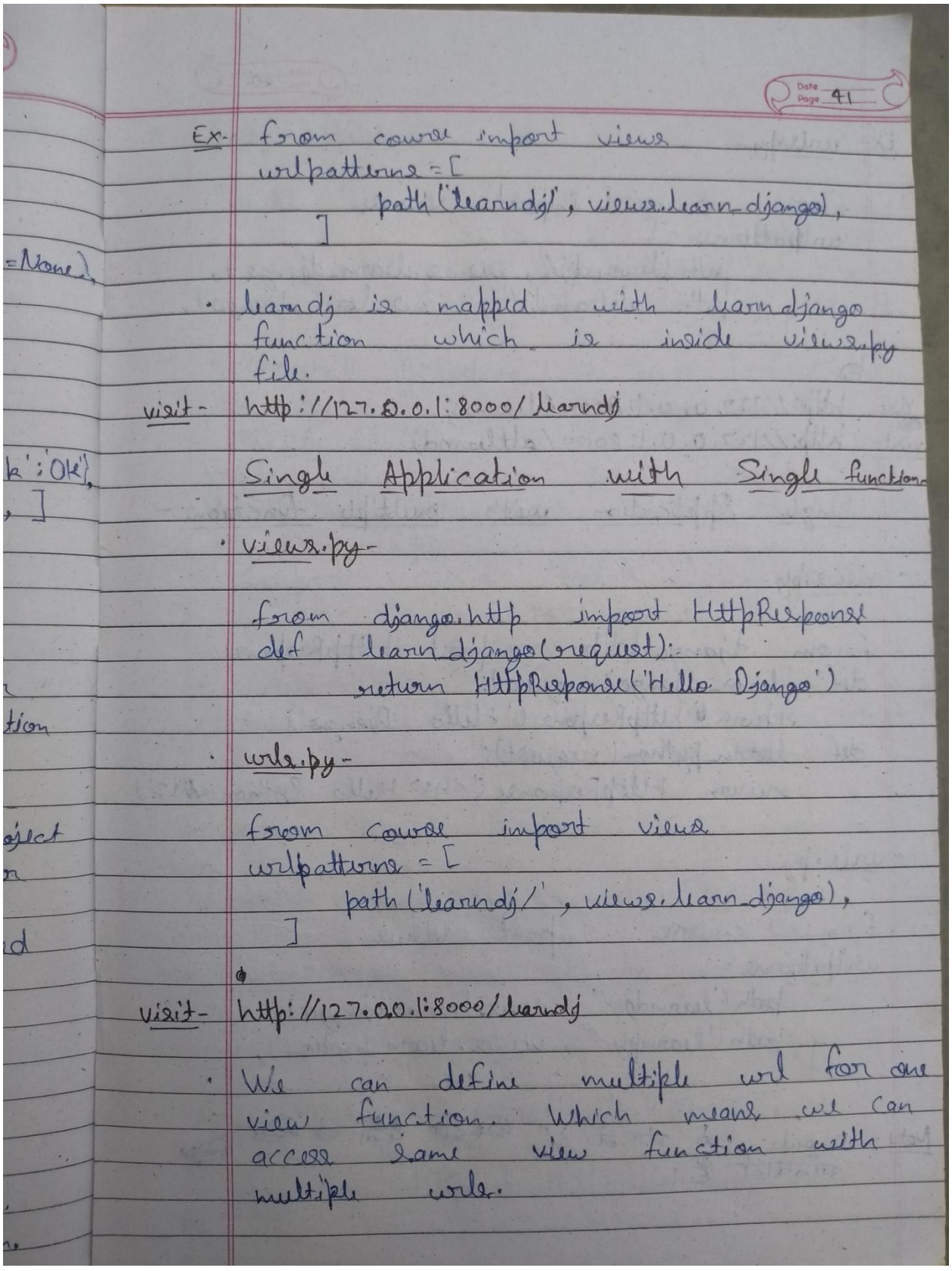
```
urlpatterns = [
    path('learndj/', views.learn_Django, {'check':'OK'},
         name = 'learn_django'), ]
```

URL Pattern inside Project -

- urls.py file is used to define url pattern attached with application or view of application or view function of application.
- urls.py is located inside inner project folder not inside application folder which means we define url at project level for applications. Defined url name will be used to get response from the application or view function of application.

Steps -

- Open urls.py
- Import Module (Python File) of the application.
- Write URL Name and Map it with function.



Date Page 41

Ex- from course import views
urlpatterns = [
path('learndj', views.learn_django),
]

- learndj is mapped with learn_django function which is inside views.py file.

visit- http://127.0.0.1:8000/learndj

Single Application with Single function

- views.py-

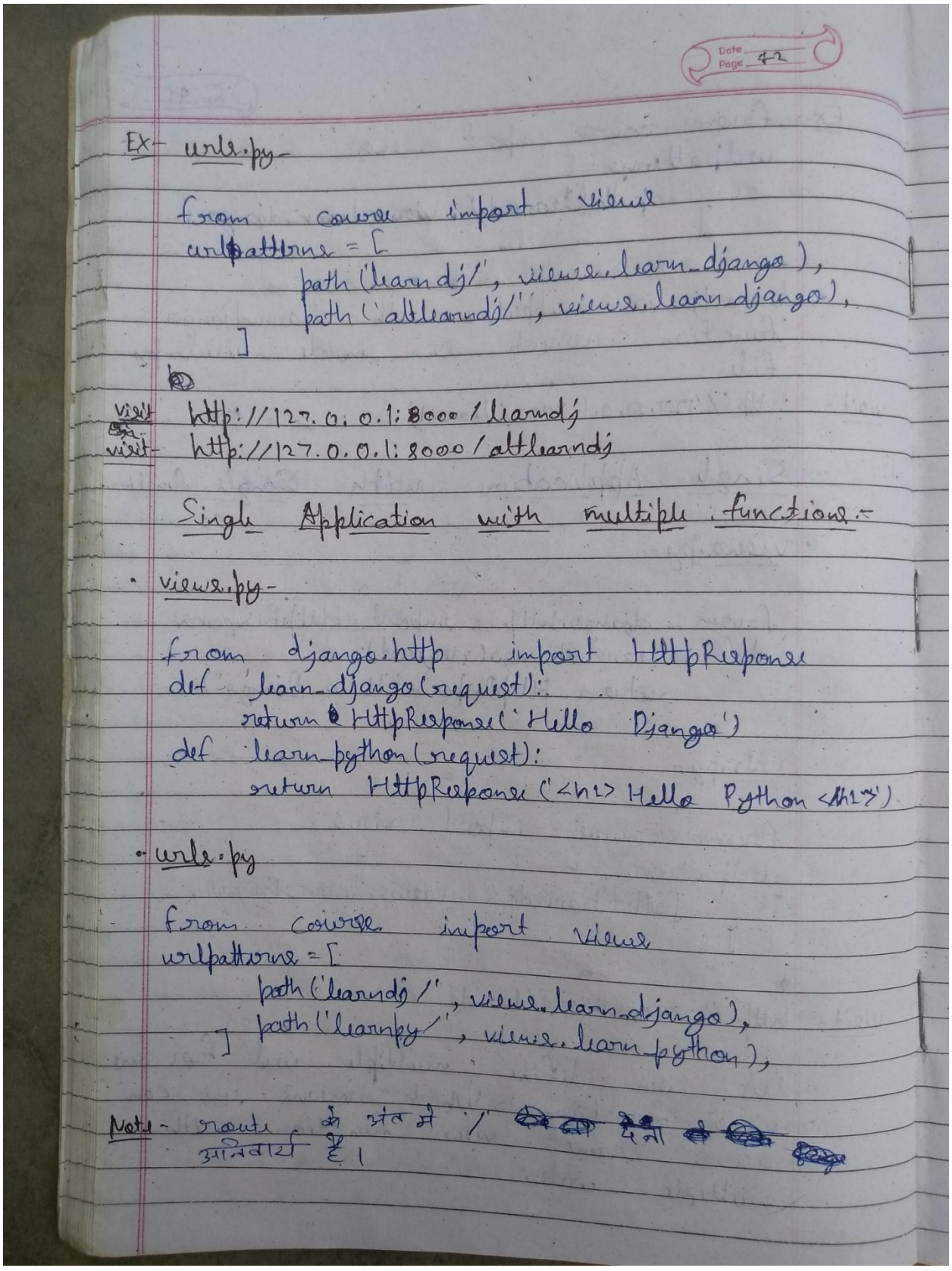
from django.http import HttpResponse
def learn_django(request):
return HttpResponse('Hello Django')

- urls.py-

from course import views
urlpatterns = [
path('learndj/', views.learn_django),
]

visit- http://127.0.0.1:8000/learndj

- We can define multiple url for one view function. Which means we can access same view function with multiple urls.



Ex- urls.py-

```
from course import views
urlpatterns = [
        path('learndj/', views.learn_django),
        path('altlearndj/', views.learn_django),
    ]
```

visit - http://127.0.0.1:8000/learndj
or
visit - http://127.0.0.1:8000/altlearndj

Single Application with multiple functions :-

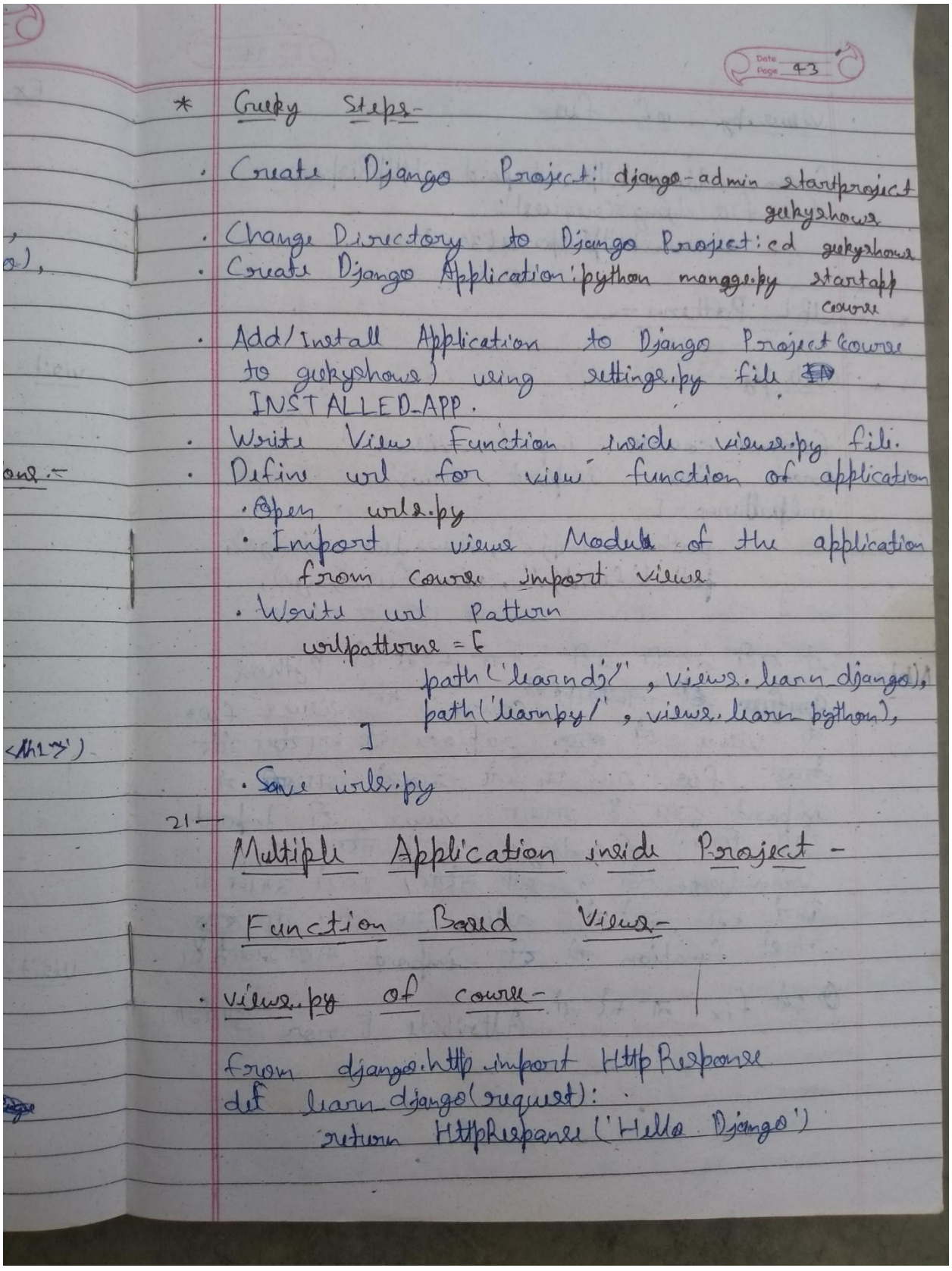
- views.py-

```
from django.http import HttpResponse
def learn_django(request):
    return HttpResponse('Hello Django')
def learn_python(request):
    return HttpResponse('<h1> Hello Python </h1>')
```

- urls.py

```
from course import views
urlpatterns = [
        path('learndj/', views.learn_django),
    ] path('learnpy/', views.learn_python),
```

Note - route के अंत में / देना अनिवार्य है।



* Geeky Steps-

- Create Django Project: django-admin startproject geekyshows
- Change Directory to Django Project: cd geekyshows
- Create Django Application: python manage.py startapp course
- Add/Install Application to Django Project (course to geekyshows) using settings.py file in INSTALLED-APP.
- Write View Function inside views.py file.
- Define url for view function of application
  - Open urls.py
  - Import views Module of the application
    from course import views
  - Write url Pattern

```
urlpatterns = [
        path('learndj/', views.learn_django),
        path('learnpy/', views.learn_python),
]
```

  - Save urls.py

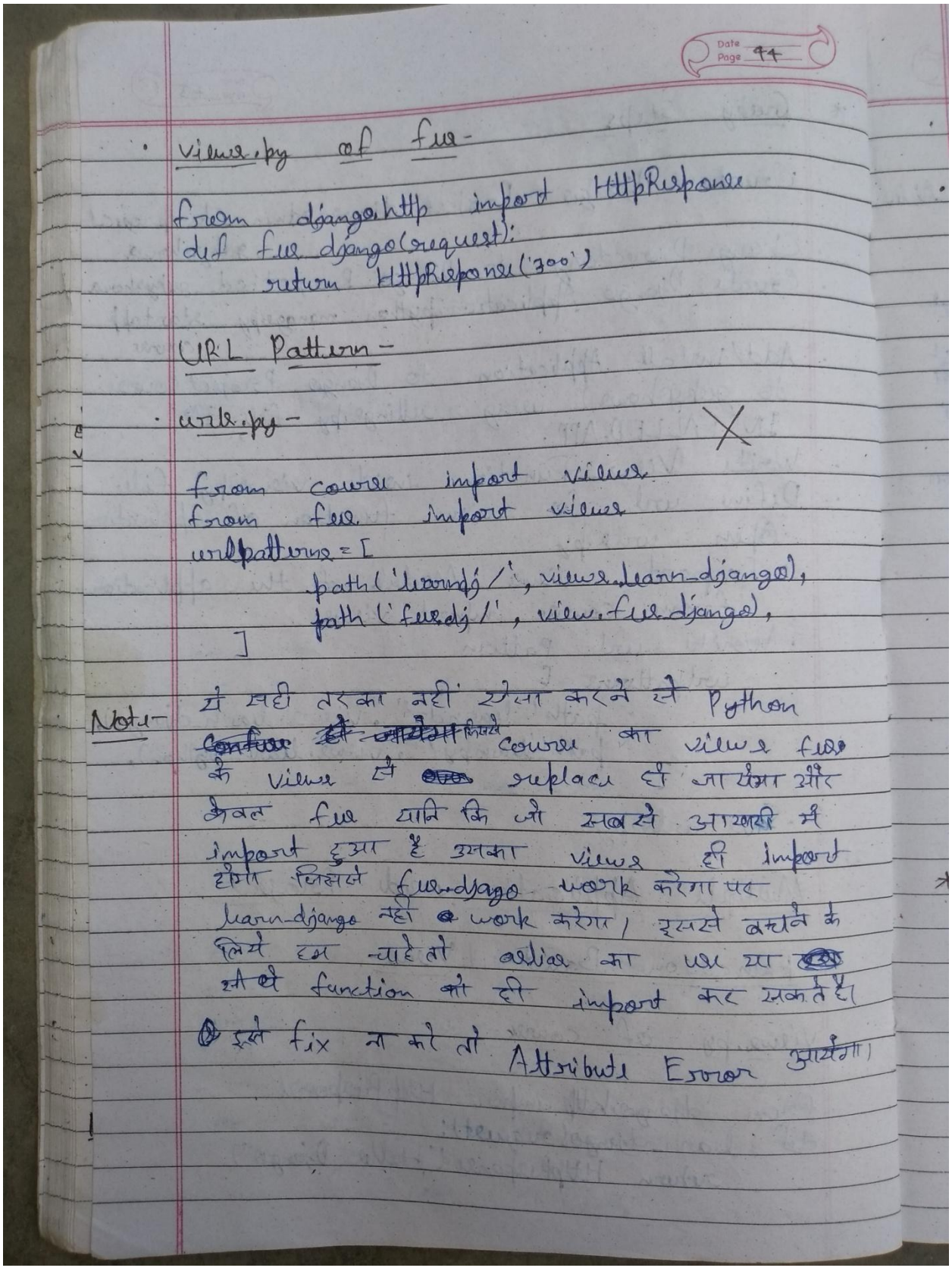
21—

Multiple Application inside Project -

- Function Based Views-

- views.py of course-

```
from django.http import HttpResponse
def learn_django(request):
    return HttpResponse('Hello Django')
```



Date Page 44

- views.py of fun -

```
from django.http import HttpResponse
def fun_django(request):
    return HttpResponse('300')
```

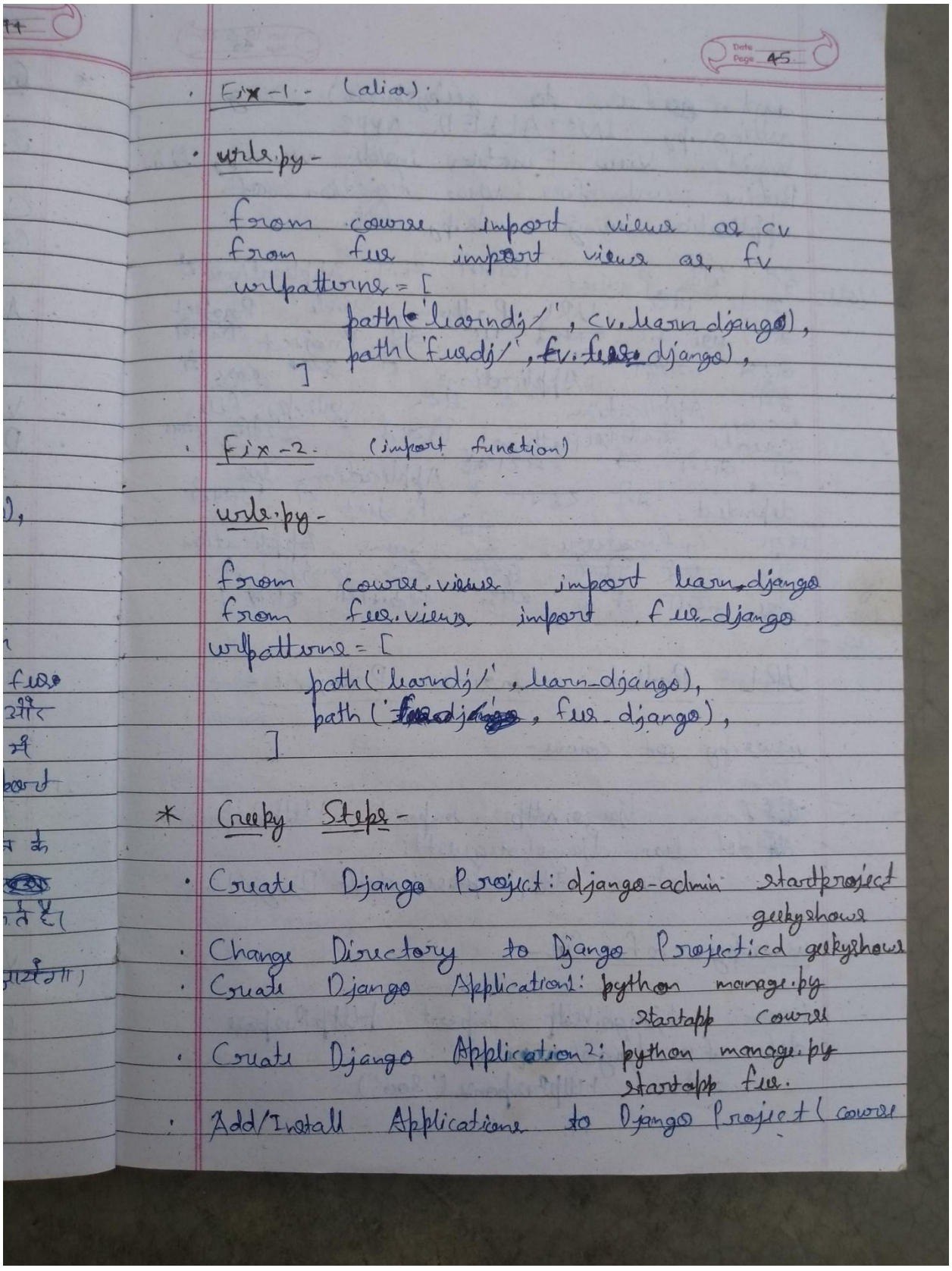
URL Pattern -

- urls.py -

```
from course import views
from fun import views
urlpatterns = [
    path('learndj/', views.learn_django),
    path('fundj/', view.fun_django),
]
```

Note- यह सही तरका नहीं होता करने से Python ~~confuse हो जायेगा~~ किसके course का views fun के views से replace हो जायेगा और केवल fun यानि कि जो सबसे आखरी में import हुआ है उसका views ही import होगा जिसके fun_django work करेगा पर learn_django नहीं work करेगा। इससे बचने के लिये हम चाहे तो alias का use या ~~use~~ से function को ही import कर सकते है।

इसे fix न करे तो Attribute Error आयेगा।



Date Page 45

- Fix-1 - (alias):

- urls.py -

```
from course import views as cv
from fee import views as fv
urlpatterns = [
        path('learndj/', cv.learn_django),
        path('feedj/', fv.fee_django),
    ]
```

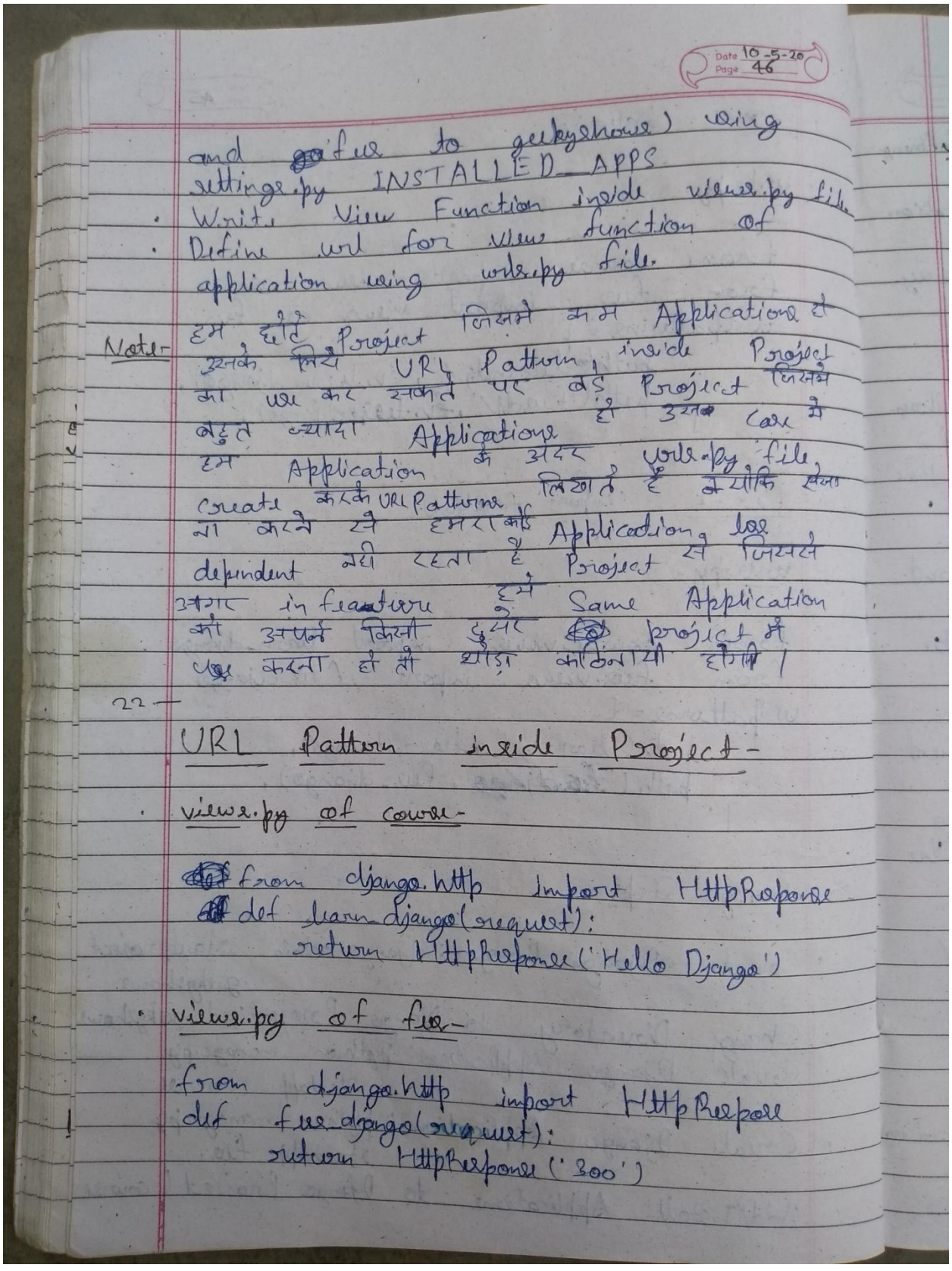
- Fix-2 (import function)

urls.py -

```
from course.views import learn_django
from fee.views import fee_django
urlpatterns = [
        path('learndj/', learn_django),
        path('feedj/', fee_django),
    ]
```

* Geeky Steps -

- Create Django Project: django-admin startproject geekyshows
- Change Directory to Django Project: cd geekyshows
- Create Django Application1: python manage.py startapp course
- Create Django Application2: python manage.py startapp fee.
- Add/Install Applications to Django Project (course



Date 10-5-20
Page 46

and for to geekyshows) using settings.py INSTALLED_APPS
- Write View Function inside views.py file.
- Define url for view function of application using urls.py file.

Note- हम, दूसरे Project जिसमे कम Applications है उनके लिए URL Pattern, inside Project का use कर सकते पर बड़े Project जिसमे बहुत ज्यादा Applications है उनके case मे हम Application के अंदर urls.py file, create करके URL Patterns लिखते है क्योंकि ऐसा ना करने से हमारा Applications loose dependent नही रहता है Project से जिससे अगर in feature हमे Same Application को अपने किसी दूसरे project मे use करना हो तो थोड़ा कठिनाई होगी।

22 —

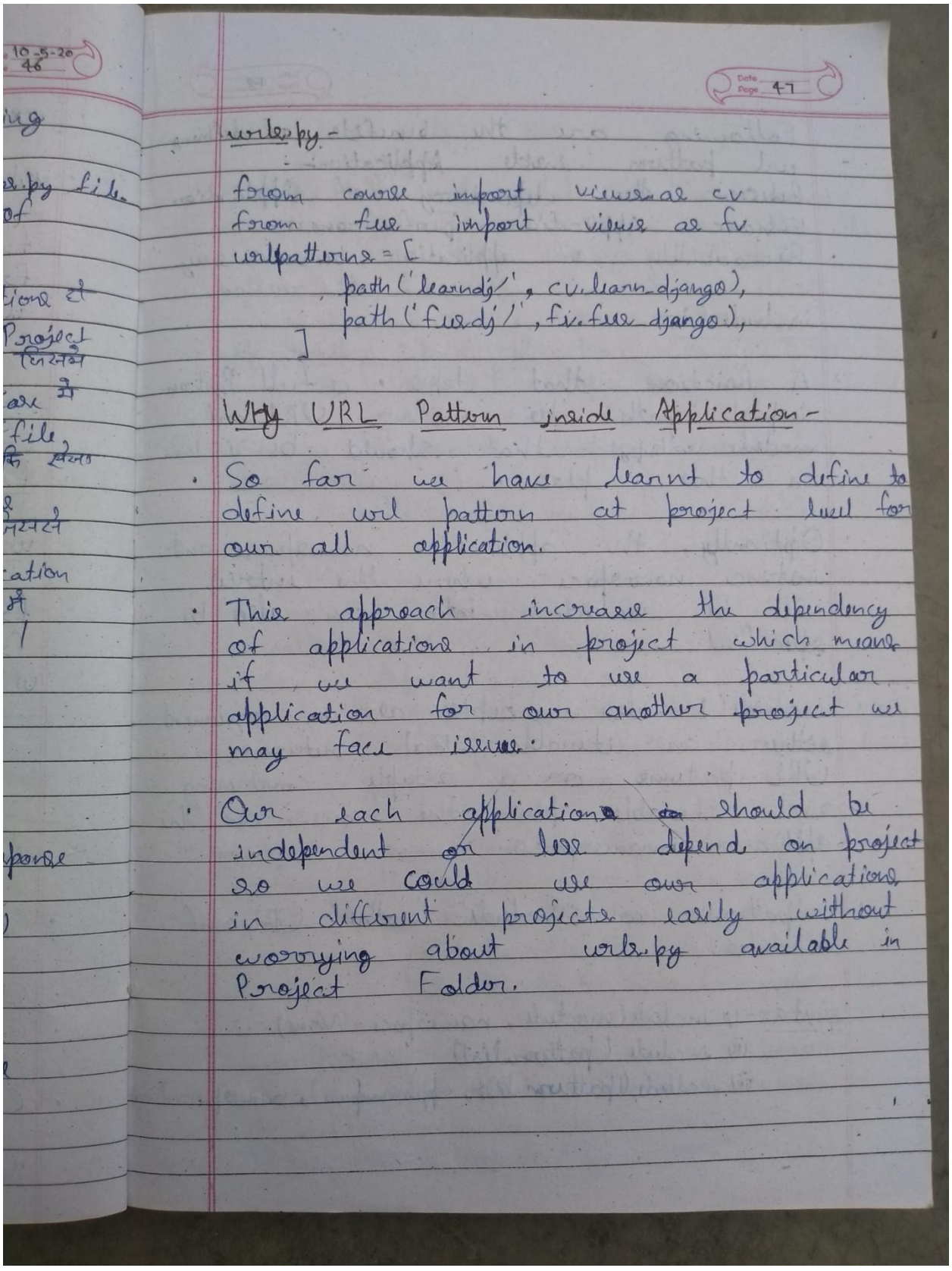
URL Pattern inside Project -

- views.py of course -

```
from django.http import HttpResponse
def learn_django(request):
    return HttpResponse('Hello Django')
```

- views.py of fun -

```
from django.http import HttpResponse
def fun_django(request):
    return HttpResponse('300')
```

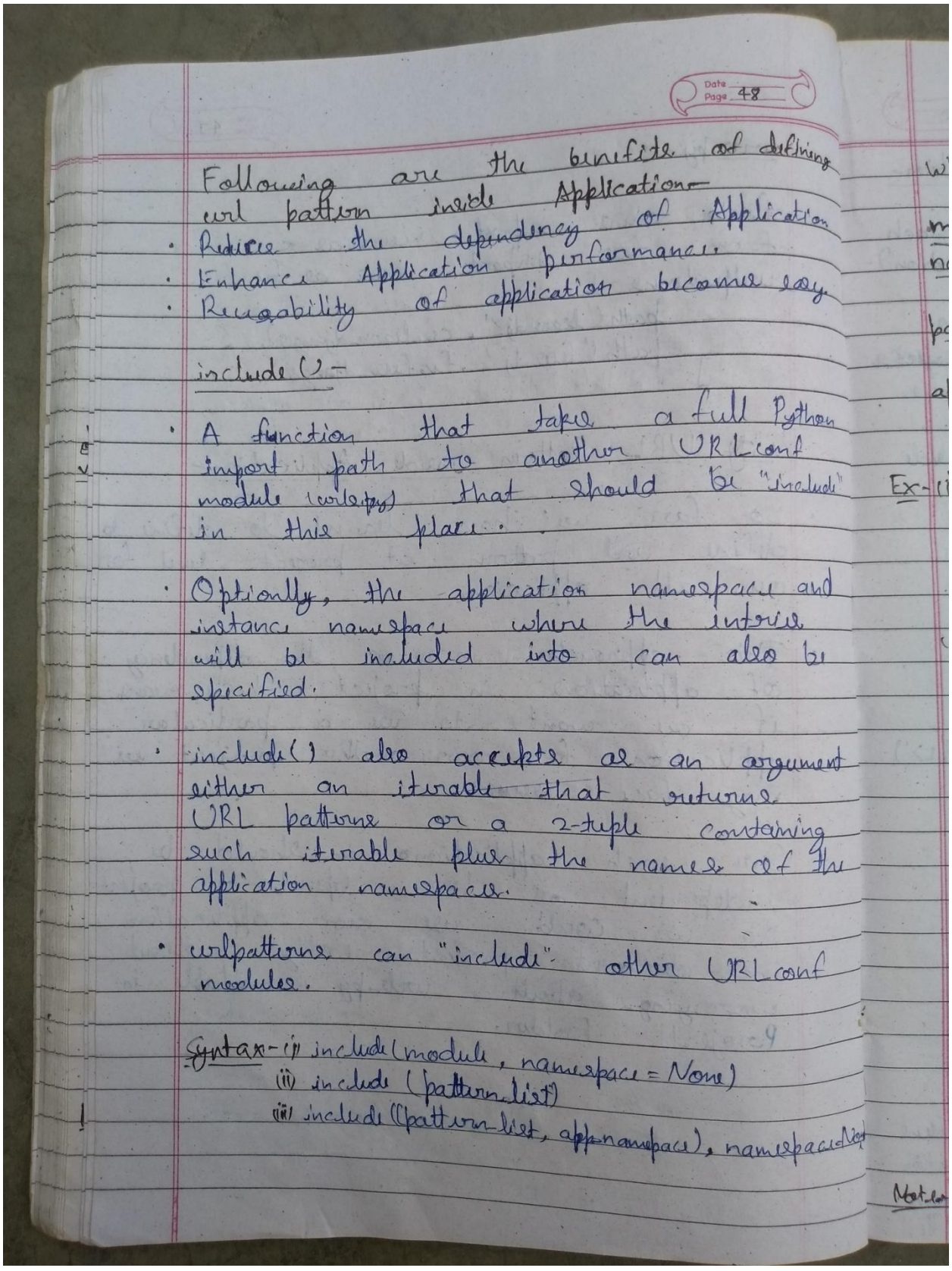


urls.py -

```
from course import views as cv
from fun import views as fv
urlpatterns = [
    path('learndj/', cv.learn_django),
    path('fundj/', fv.fun_django),
]
```

Why URL Pattern inside Application -

- So far we have learnt to define to define url pattern at project level for our all application.
- This approach increase the dependency of applications in project which means if we want to use a particular application for our another project we may face issue.
- Our each applications should be independent or less depend on project so we could use our applications in different projects easily without worrying about urls.py available in Project Folder.



Date
Page 48

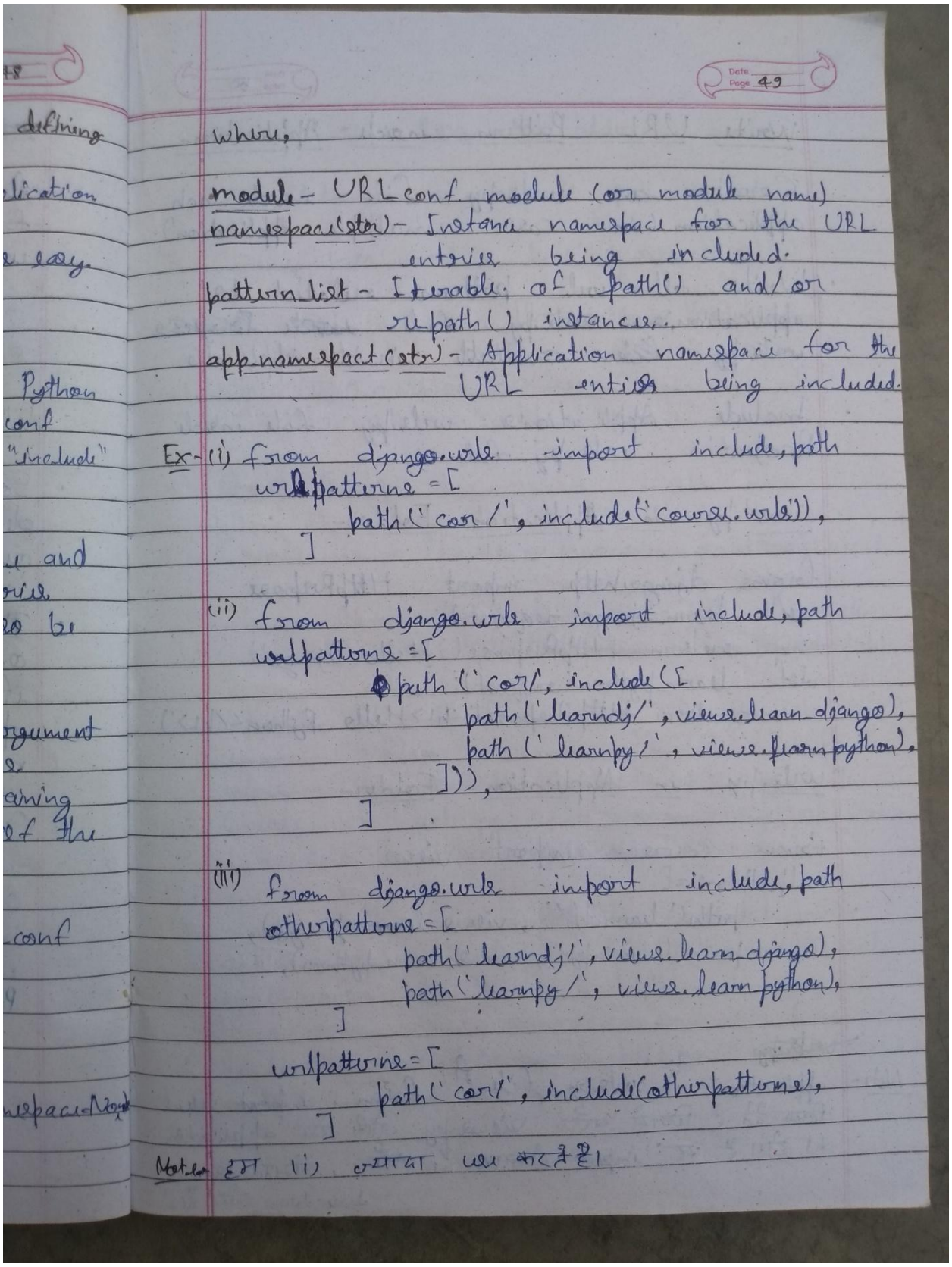
Following are the benifits of defining url pattern inside Application-

- Reduce the dependency of Application
- Enhance Application performance.
- Reusability of application become easy.

include() -

- A function that takes a full Python import path to another URLconf module (urlpatterns) that should be "included" in this place.
- Optionally, the application namespace and instance namespace where the entries will be included into can also be specified.
- include() also accepts as an argument either an iterable that returns URL patterns or a 2-tuple containing such iterable plus the names of the application namespaces.
- urlpatterns can "include" other URLconf modules.

Syntax- (i) include(module, namespace=None)
(ii) include(pattern_list)
(iii) include((pattern_list, app_namespace), namespace)



where,

module - URL conf. module (or module name)
namespace(str) - Instance namespace for the URL entries being included.
pattern_list - Iterable of path() and/or re_path() instances.
app_namespace(str) - Application namespace for the URL entries being included.

Ex- (i)
```
from django.urls import include, path
urlpatterns = [
    path('coor/', include('course.urls')),
]
```

(ii)
```
from django.urls import include, path
urlpatterns = [
    path('coor/', include([
        path('learndj/', views.learn_django),
        path('learnpy/', views.learnpython),
    ])),
]
```

(iii)
```
from django.urls import include, path
otherpatterns = [
    path('learndj/', views.learn_django),
    path('learnpy/', views.learn_python),
]
urlpatterns = [
    path('coor/', include(otherpatterns),
]
```

Note- हम (i) ज्यादा use करते हैं।

Date Page 50

Write URL Pattern inside Application-

- Create an urls.py file inside each application (in case multiple application).
- Write all url pattern related to application, in urls.py file ~~inside Projects urls.py file~~ available inside application.
- Include Application's urls.py file inside Project's urls.py file.

visit=

- views.py in Application Folder-

```
from django.http import HttpResponse
def learn_django(request):
    return HttpResponse('Hello Django')
def learn_python(request):
    return HttpResponse('<h1>Hello Python</h1>')
```

- urls.py in Application Folder-

```
from course import views
urlpatterns=[
    path('learndj/', views.learn_django),
    path('learnpy', views.learn_python),
]
```

- ~~urls.py~~ के application के urls.py file में from . import views लिख देते हैं जिससे वहाँ views.py जोकि उस application का होता है वह import हो जाता है। जहाँ . का मतलब same directory होता है।

. urls.py in Project Folder-

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

↑ Package Name ↑ Module Name

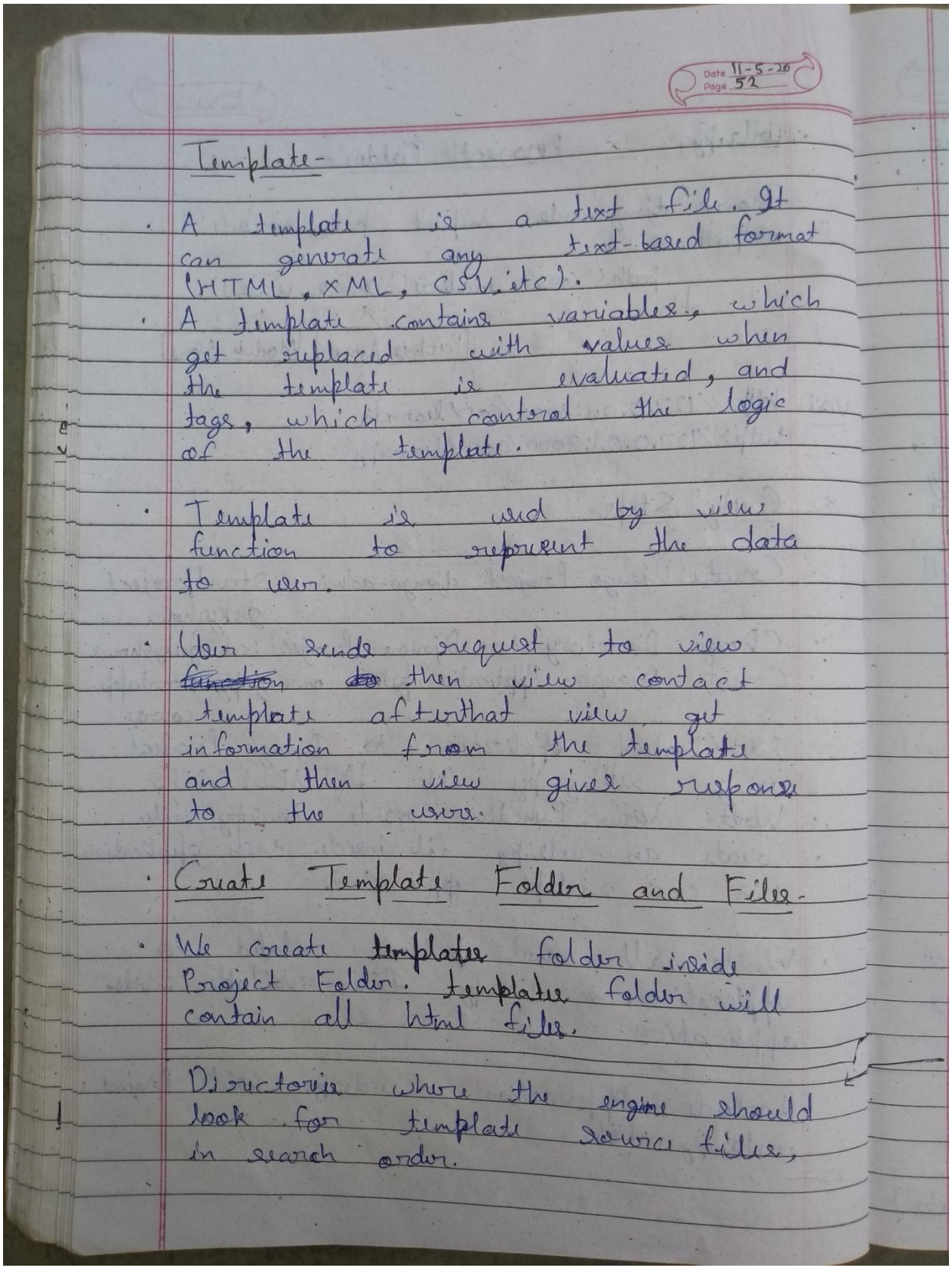
Visit: http://127.0.0.1:8000/cor/learndj

http://127.0.0.1:8000/cor/learnpy

* Geeky Steps-

- Create Django Project: django-admin startproject geekyshows
- Change Directory to Django Project: cd geekyshows
- Create Django Application: python manage.py startapp course
- Add/Install Application to Django Project using settings.py file INSTALLED APPS
- Write View Function inside views.py file.
- Create an urls.py file inside each application (in case multiple application).
- Write all url pattern related to application, in urls.py file available inside application.
- Include Application's urls.py inside Project urls.py file.

23—



Date 11-5-20
Page 52

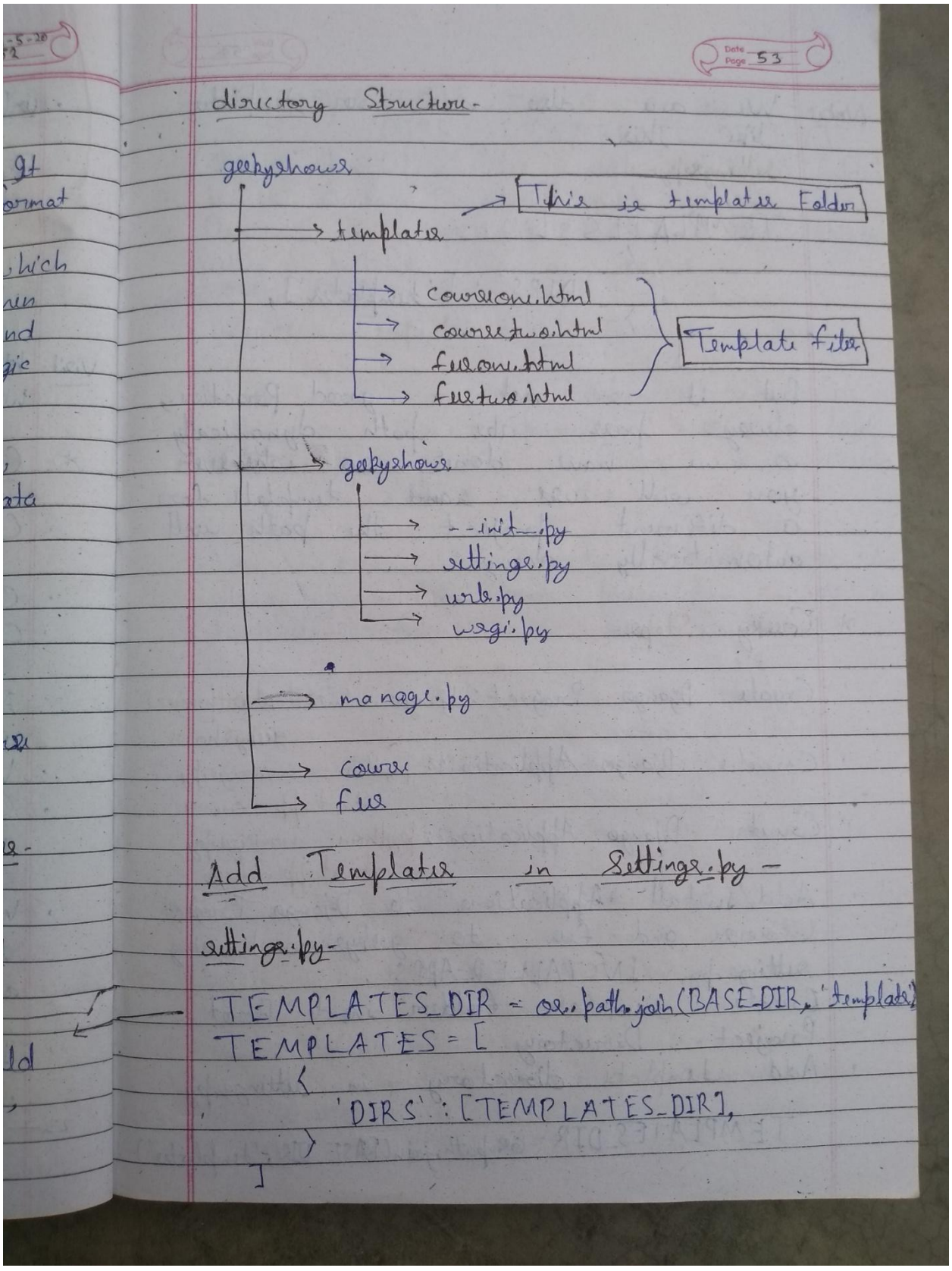
Template-

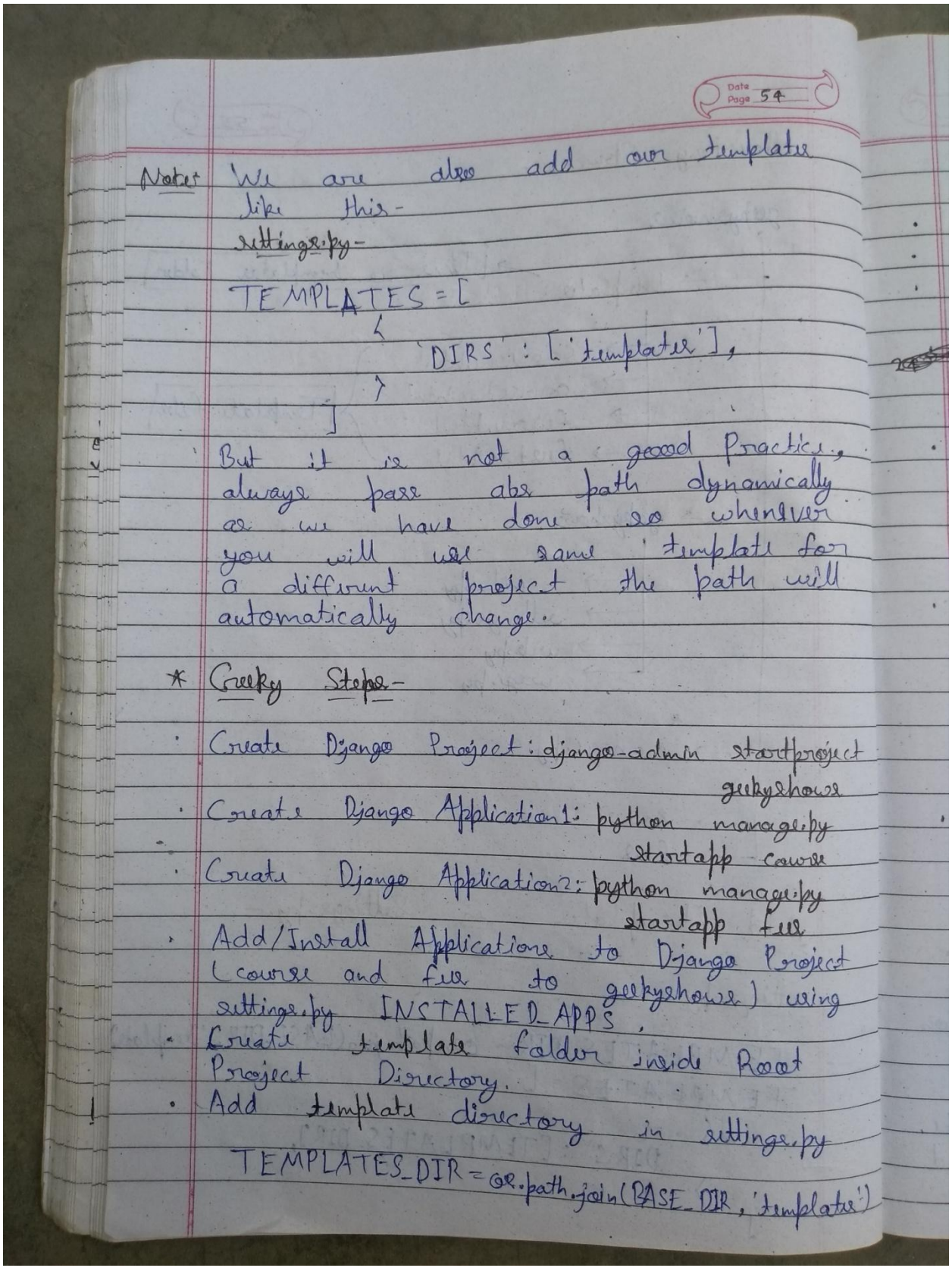
- A template is a text file. It can generate any text-based format (HTML, XML, CSV, etc).
- A template contains variables, which get replaced with values when the template is evaluated, and tags, which control the logic of the template.
- Template is used by view function to represent the data to user.
- User sends request to view ~~function~~ ~~do~~ then view contact template after that view get information from the template and then view give response to the user.

Create Template Folder and Files.

- We create ~~template~~ folder inside Project Folder. ~~template~~ folder will contain all html files.

Directories where the engine should look for template source files, in search order.


Date Page 53
directory Structure -
geekyshows
This is template Folder
templates
courseone.html
coursetwo.html
funone.html
funtwo.html
Template files
geekyshows
__init__.py
settings.py
urls.py
wsgi.py
manage.py
course
fun
Add Templates in Settings.py -
settings.py -
TEMPLATES_DIR = os.path.join(BASE_DIR, 'templates')
TEMPLATES = [
{
'DIRS' : [TEMPLATES_DIR],
}
]



Date
Page 54

Note: We are also add our template like this-

settings.py-

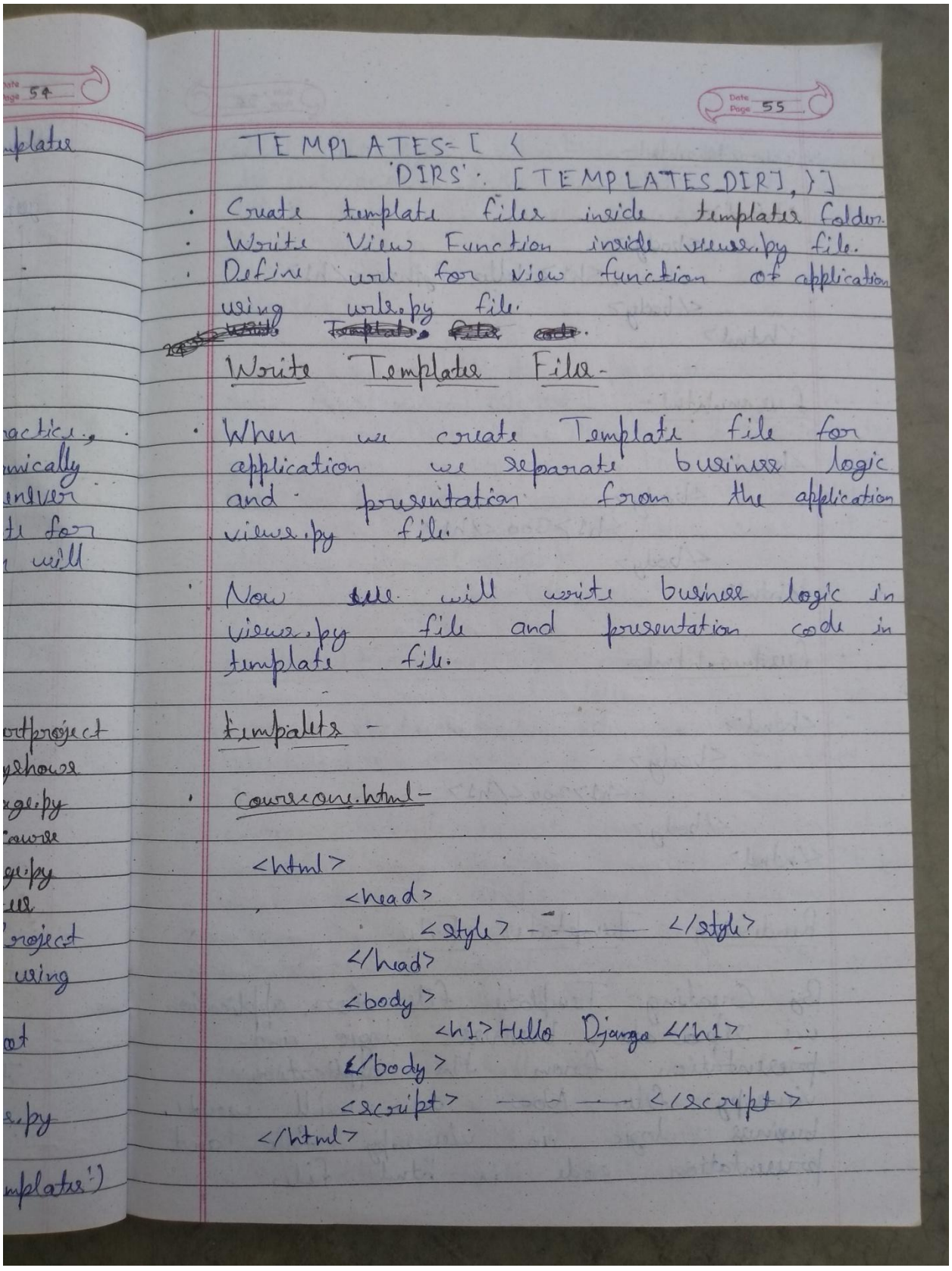
```
TEMPLATES = [
    {
        'DIRS' : ['template'],
    }
]
```

But it is not a good Practice, always pass abs path dynamically as we have done so whenever you will use same template for a different project the path will automatically change.

* Geeky Steps-

- Create Django Project: django-admin startproject geekyshows
- Create Django Application1: python manage.py startapp course
- Create Django Application2: python manage.py startapp fee
- Add/Install Applications to Django Project (course and fee to geekyshows) using settings.py INSTALLED_APPS.
- Create template folder inside Root Project Directory.
- Add template directory in settings.py

```
TEMPLATES_DIR = os.path.join(BASE_DIR, 'template')
```



TEMPLATES=[{
'DIRS': [TEMPLATES_DIR], }]

- Create template files inside templates folder.
- Write View Function inside views.py file.
- Define url for view function of application using urls.py file.

~~Write Templates Files.~~

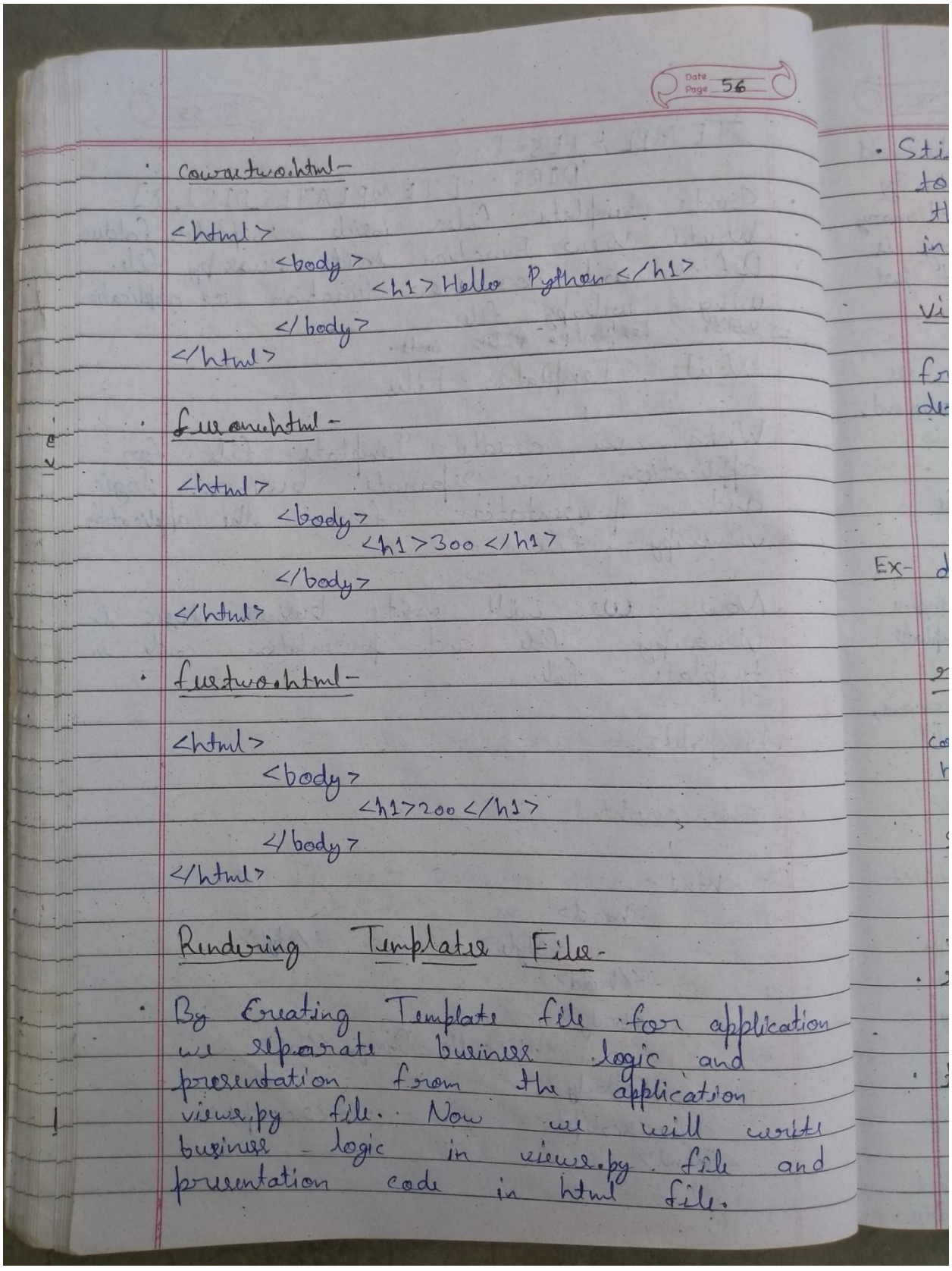
Write Template File-

- When we create Template file for application we separate business logic and presentation from the application views.py file.
- Now we will write business logic in views.py file and presentation code in template file.

Templates -

- courseone.html-

```
<html>
    <head>
        <style> ------ </style>
    </head>
    <body>
        <h1> Hello Django </h1>
    </body>
    <script> ------ </script>
</html>
```



Date
Page 56

courses/two.html -

```
<html>
    <body>
        <h1> Hello Python </h1>
    </body>
</html>
```

fees/one.html -

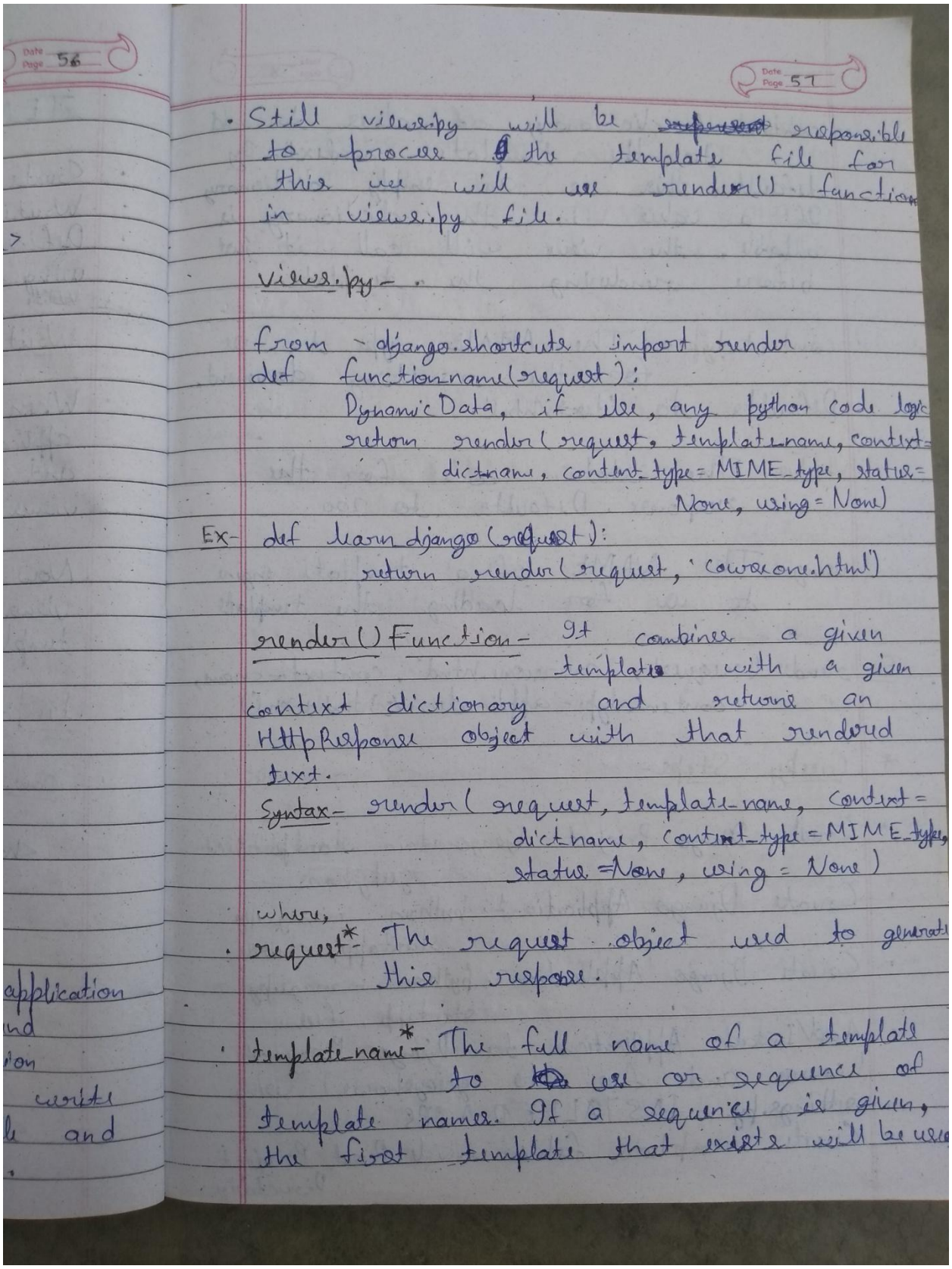
```
<html>
    <body>
        <h1> 300 </h1>
    </body>
</html>
```

fees/two.html -

```
<html>
    <body>
        <h1> 200 </h1>
    </body>
</html>
```

Rendering Template File -

- By Creating Template file for application we separate business logic and presentation from the application views.py file. Now we will write business logic in views.py file and presentation code in html file.



Date Page 56

Date Page 57

- Still views.py will be ~~responsible~~ responsible to process the template file for this we will use render() function in views.py file.

views.py -

```
from - django.shortcuts import render
def function_name(request):
    Dynamic Data, if use, any python code logic
    return render(request, template_name, context=
        dict_name, content_type = MIME_type, status=
        None, using = None)
```

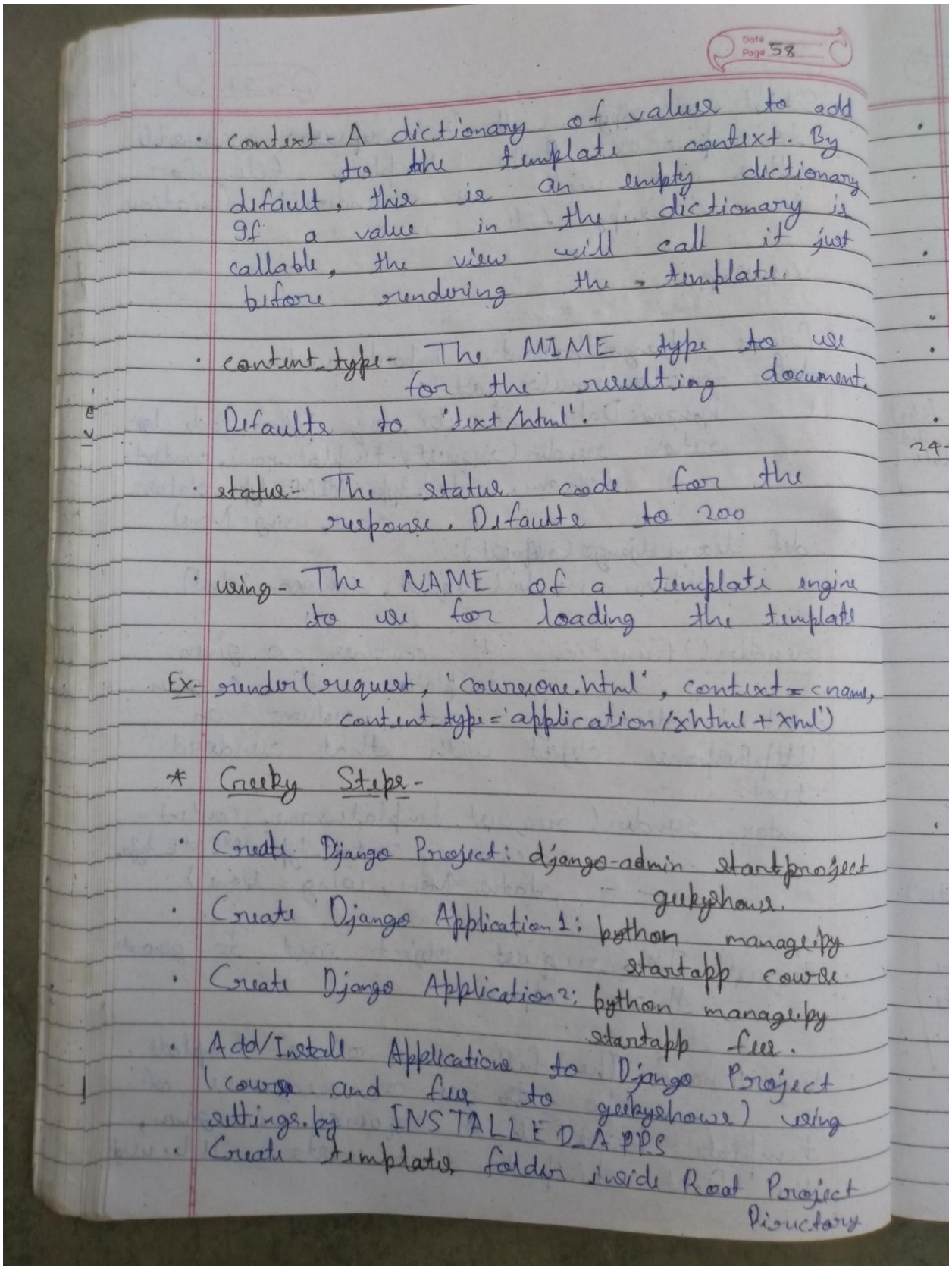
Ex- def learn_django(request):
    return render(request, 'courseone.html')

render() Function - It combines a given template with a given context dictionary and returns an HttpResponse object with that rendered text.

Syntax - render(request, template_name, context = dict_name, content_type = MIME_type, status = None, using = None)

where,

- request* - The request object used to generate this response.
- template_name* - The full name of a template to use or sequence of template names. If a sequence is given, the first template that exists will be used



Date Page 58

- context - A dictionary of values to add to the template context. By default, this is an empty dictionary. If a value in the dictionary is callable, the view will call it just before rendering the template.

- content_type - The MIME type to use for the resulting document. Defaults to 'text/html'.

- status - The status code for the response. Defaults to 200

- using - The NAME of a template engine to use for loading the template

Ex- render(request, 'courseone.html', context=cname, content_type='application/xhtml+xml')

* Geeky Steps -

- Create Django Project: django-admin startproject geekyshows.
- Create Django Application 1: python manage.py startapp courses
- Create Django Application 2: python manage.py startapp fees.
- Add/Install Applications to Django Project (courses and fees to geekyshows) using settings.py INSTALLED_APPS
- Create templates folder inside Root Project Directory

58

Date Page 59

- Add template directory in settings.py
  TEMPLATES_DIR = os.path.join(BASE_DIR, 'templates')
  TEMPLATES=[{
  'DIRS': [TEMPLATES_DIR], }]
- Create template file inside template folder.
- Write View Function inside views.py file.
- Define url for view function of application using urls.py file.
- Write Template file code.

24—

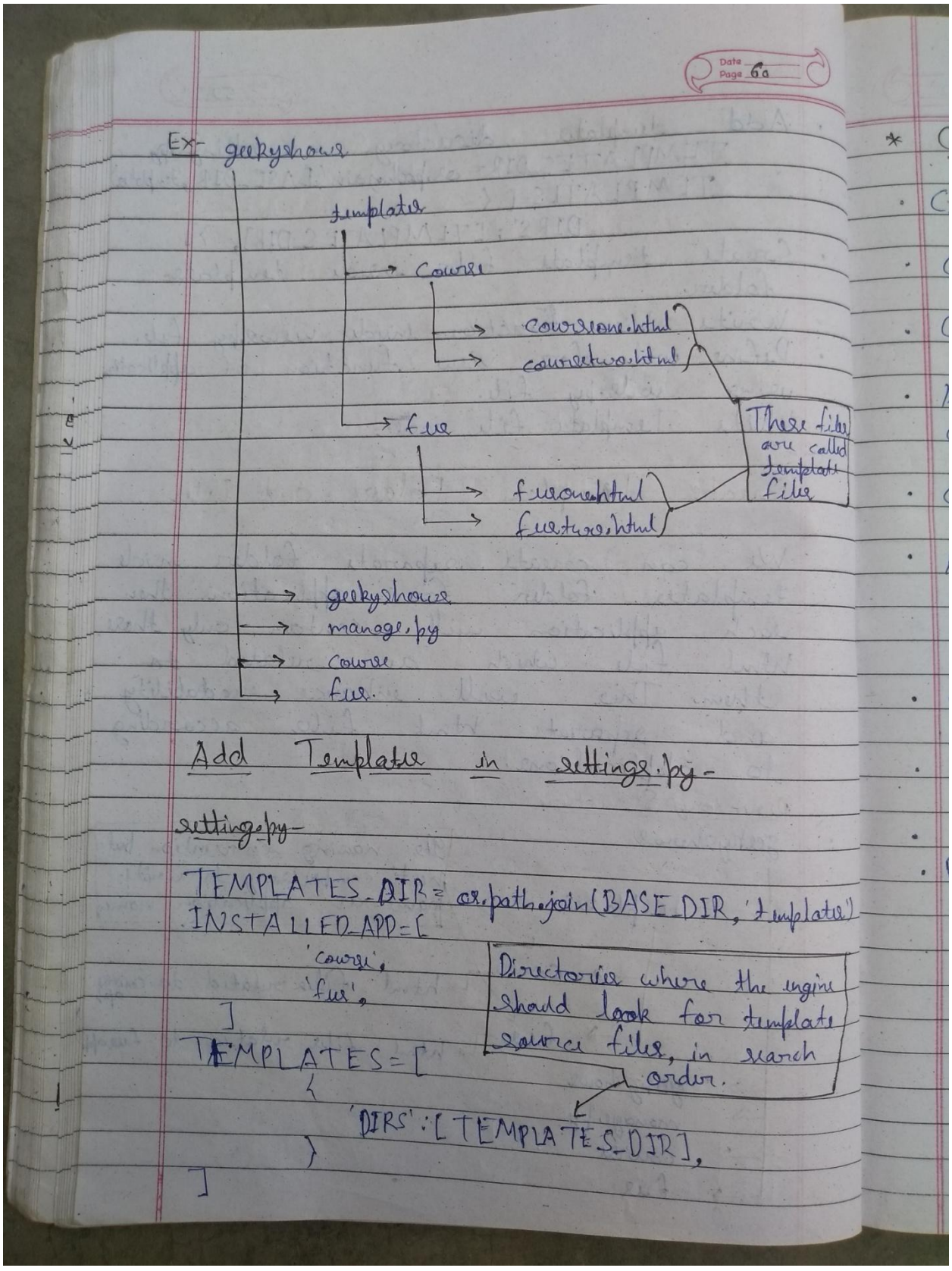
Create Template Folder and Files-

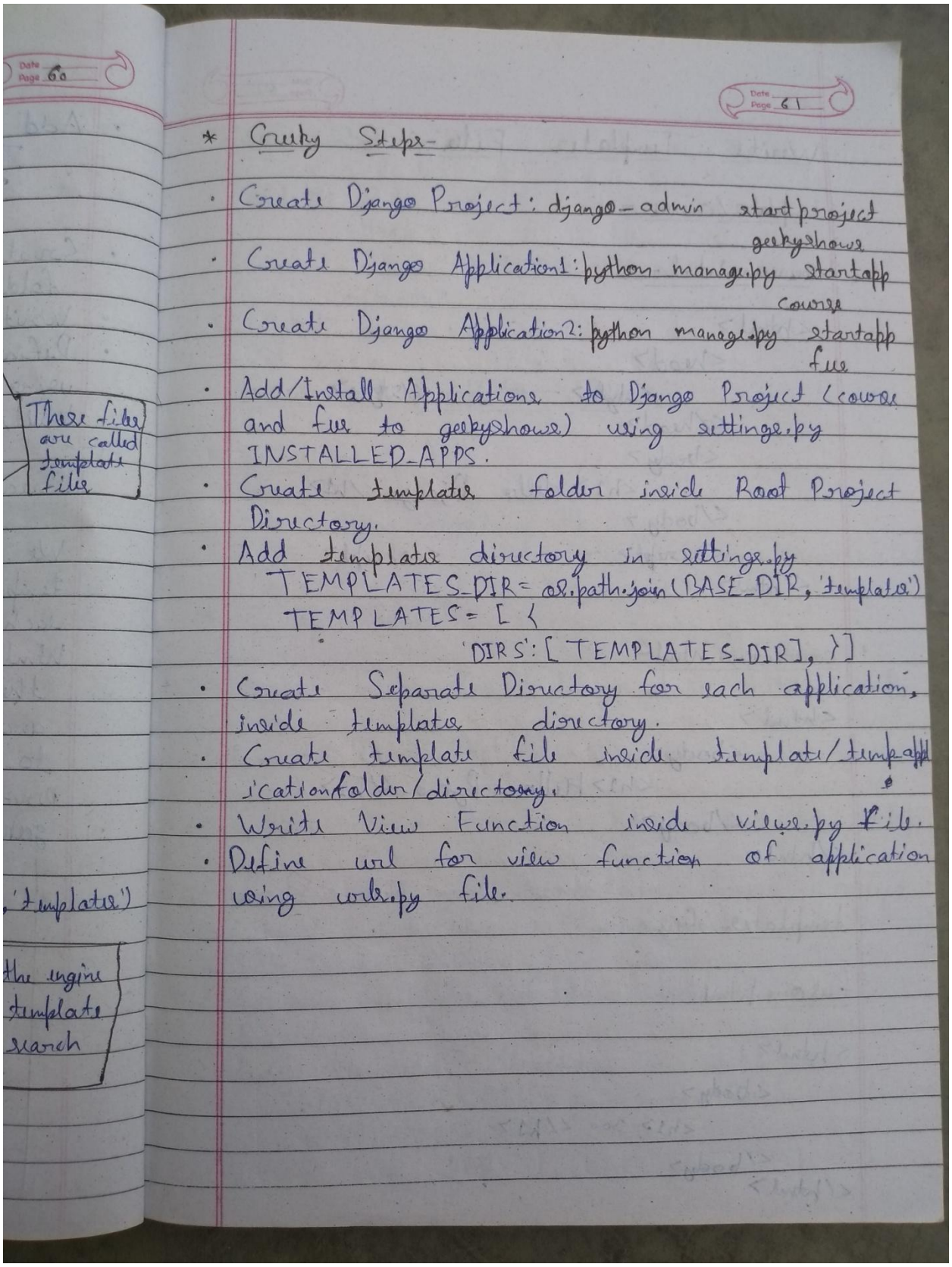
We can create separate folder inside templates folder for applications then each application will contain only those html file which are related to them. This will enhance readability and separate html files according to applications.

Directory Structure -

- geekyshows
  - templates
    - course → html files related to course app
    - fee → html file related to fee app
  - geekyshows
  - manage.py
  - course
  - fee

Its naming convention but not compulsory to write name as application name.


Date
Page 60
Ex- geekyshows
templates
course
courseone.html
coursetwo.html
fue
fueone.html
fuetwo.html
These files are called template files
geekyshows
manage.py
course
fue
Add Template in settings.py -
setting.py -
TEMPLATES_DIR = os.path.join(BASE_DIR, 'templates')
INSTALLED_APP=[
'course',
'fue',
]
TEMPLATES=[
{
'DIRS':[TEMPLATES_DIR],
}
]
Directories where the engine should look for template source files, in search order.



Date Page 60

Date Page 61

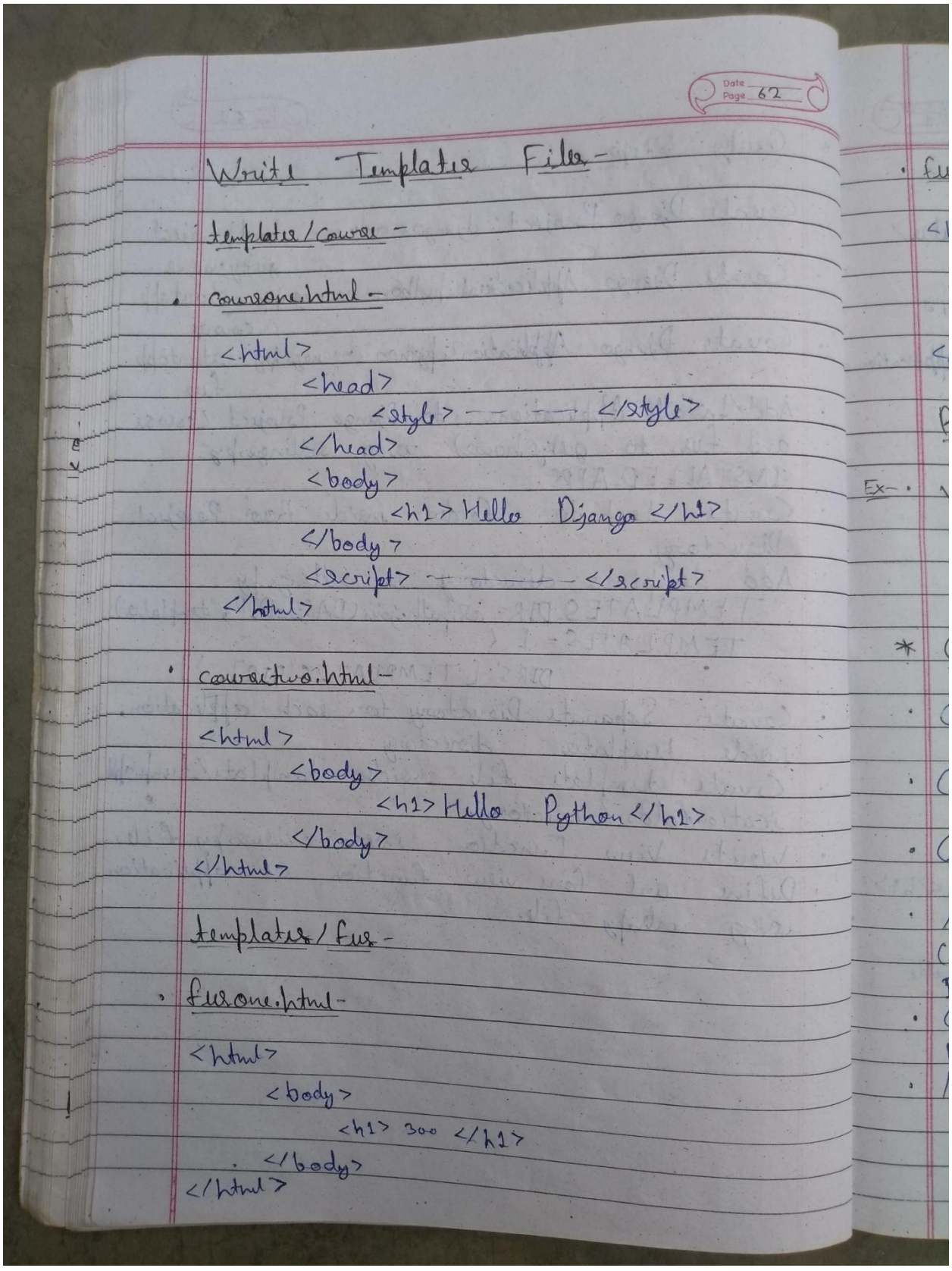
* Quicky Steps-

- Create Django Project: django-admin startproject geekyshows
- Create Django Application1: python manage.py startapp course
- Create Django Application2: python manage.py startapp fee
- Add/Install Applications to Django Project (course and fee to geekyshows) using settings.py INSTALLED_APPS.
- Create templates folder inside Root Project Directory.
- Add template directory in settings.py
  TEMPLATES_DIR = os.path.join(BASE_DIR, 'templates')
  TEMPLATES = [{
  'DIRS': [TEMPLATES_DIR], }]
- Create Separate Directory for each application, inside templates directory.
- Create template file inside templates/templ-application folder/directory.
- Write View Function inside views.py file.
- Define url for view function of application using urls.py file.

These files are called template files

'templates')

the engine template search



Date
Page 62

Write Template File -

templates/course -

- courseone.html -

```
<html>
    <head>
        <style> ---- ---- </style>
    </head>
    <body>
        <h1> Hello Django </h1>
    </body>
    <script> ---- ---- </script>
</html>
```

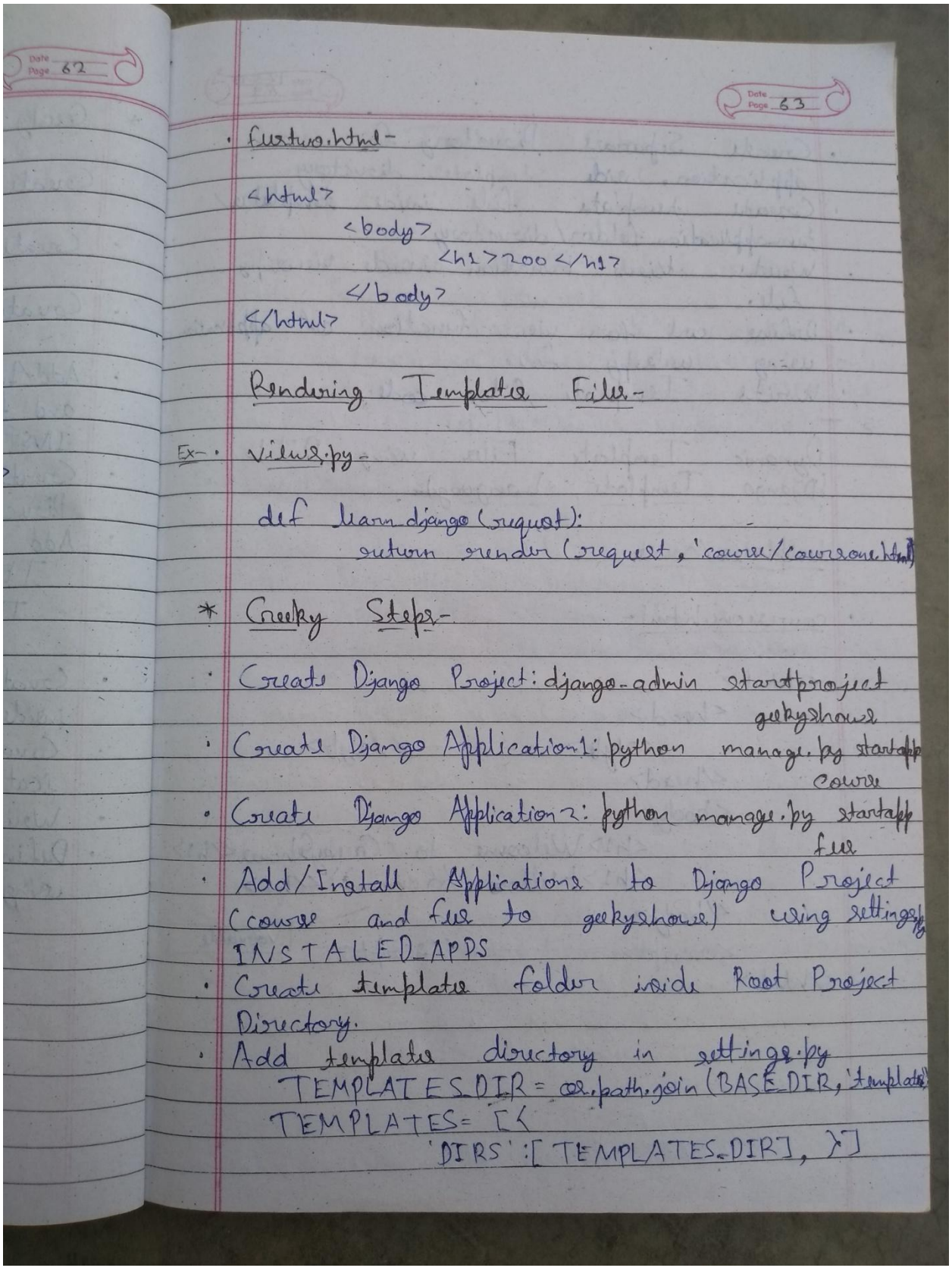
- courseitwo.html -

```
<html>
    <body>
        <h1> Hello Python </h1>
    </body>
</html>
```

templates/fee -

- feeone.html -

```
<html>
    <body>
        <h1> 300 </h1>
    </body>
</html>
```



Date Page 62

Date Page 63

- first.html -

```
<html>
    <body>
        <h1>200</h1>
    </body>
</html>
```

Rendering Template File -

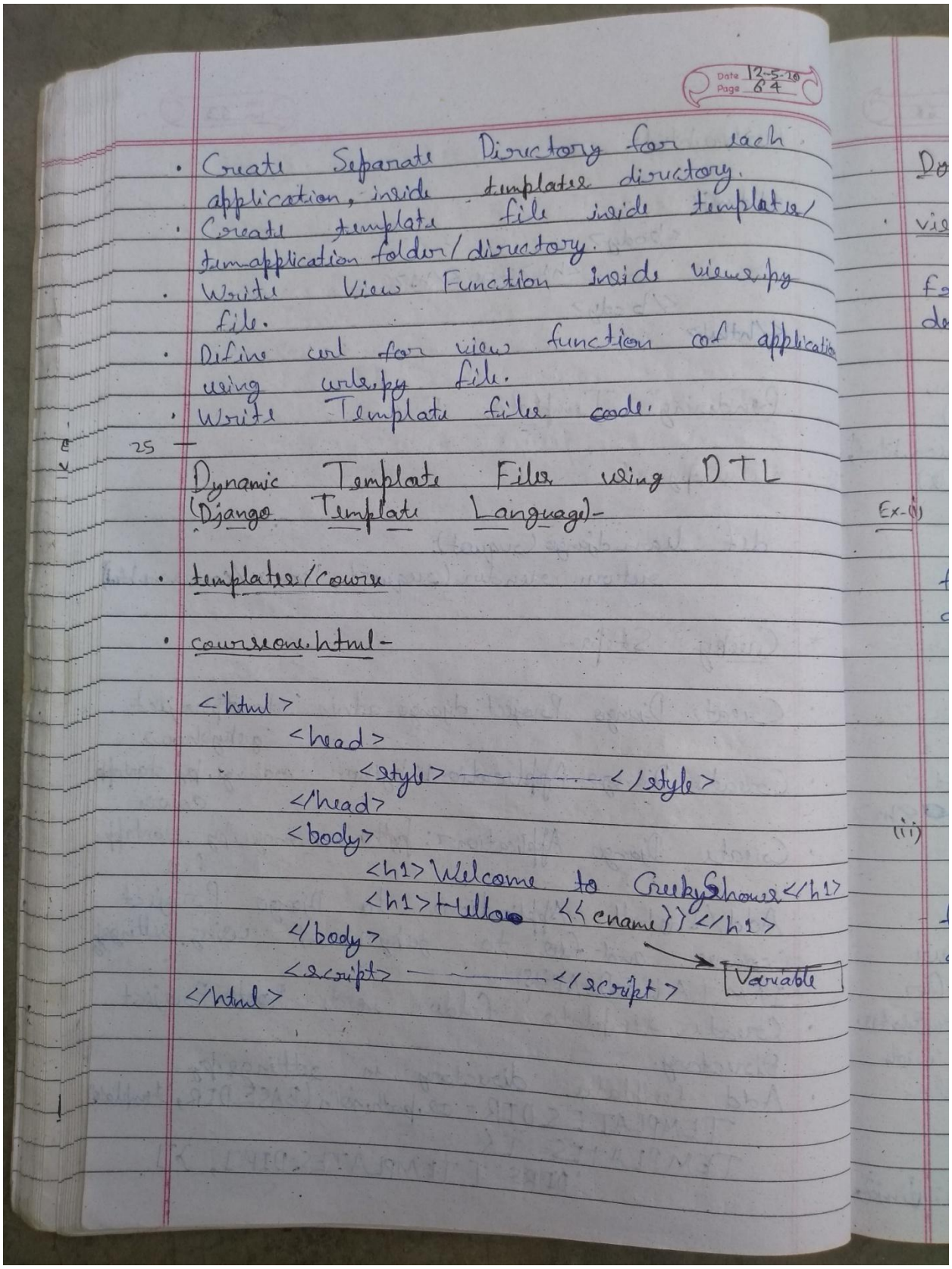
Ex- • views.py -

```
def learn_django(request):
    return render(request, 'course/course.html')
```

* Geeky Steps -

- Create Django Project: django-admin startproject geekyshows
- Create Django Application1: python manage.py startapp course
- Create Django Application2: python manage.py startapp fee
- Add/Install Applications to Django Project (course and fee to geekyshows) using settings.py INSTALLED_APPS
- Create template folder inside Root Project Directory.
- Add template directory in settings.py

```
TEMPLATES_DIR = os.path.join(BASE_DIR, 'templates')
TEMPLATES = [{
        'DIRS': [TEMPLATES_DIR], }]
```



Date 12-5-20
Page 64

- Create Separate Directory for each application, inside templates directory.
- Create template file inside templates/tem-application folder/directory.
- Write View Function inside views.py file.
- Difine url for view function of application using urls.py file.
- Write Template files code.

25 —

Dynamic Template Files using DTL (Django Template Language)- Ex-(i)

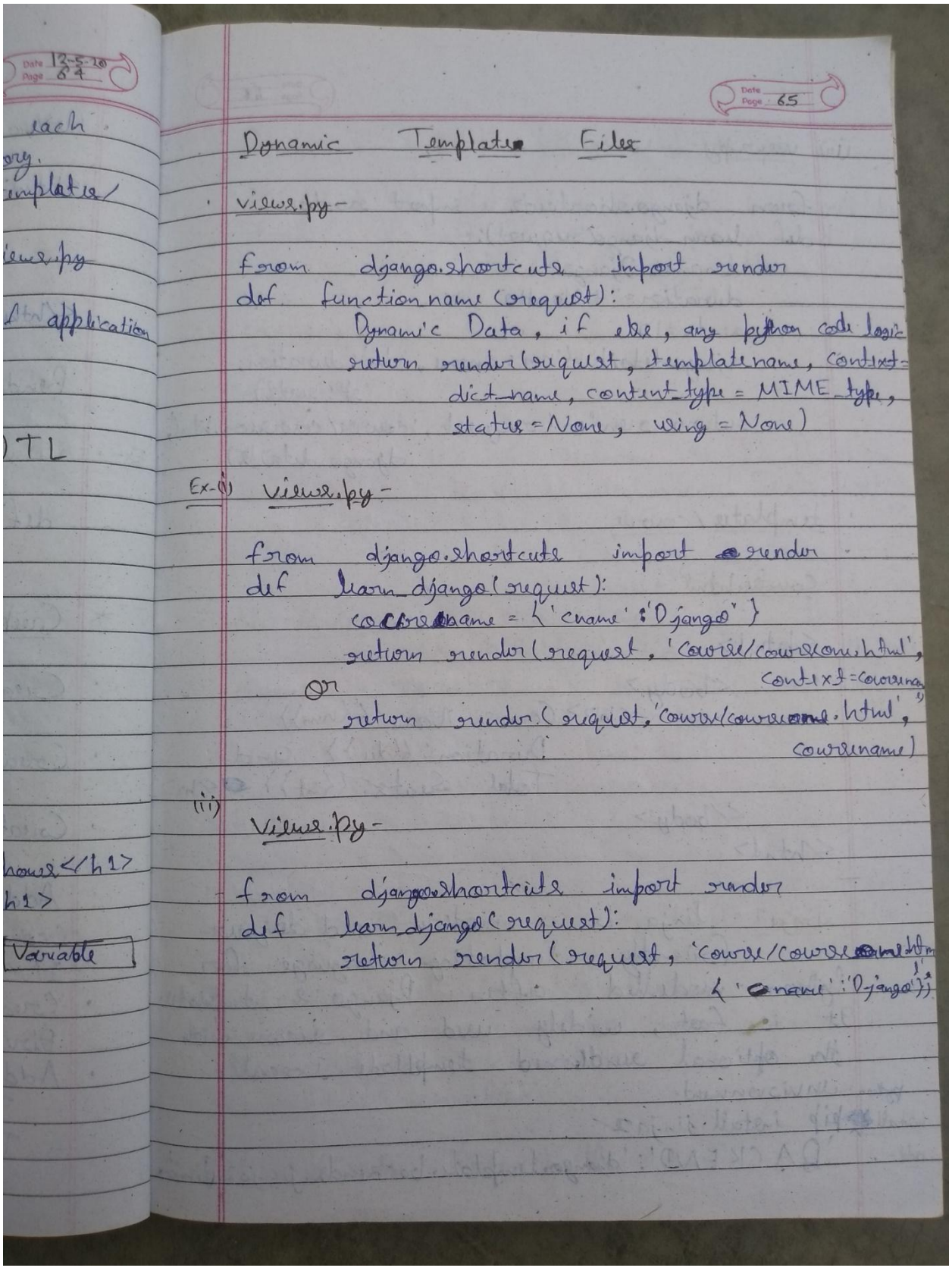
- templates/courses

- courseone.html-

```
<html>
    <head>
        <style> ---------- </style>
    </head>
    <body>
        <h1> Welcome to GeekyShows </h1>
        <h1> Hello {{ cname }} </h1>
    </body>
    <script> ---------- </script>
</html>
```

Variable

(ii)



Dynamic Template Files

views.py -

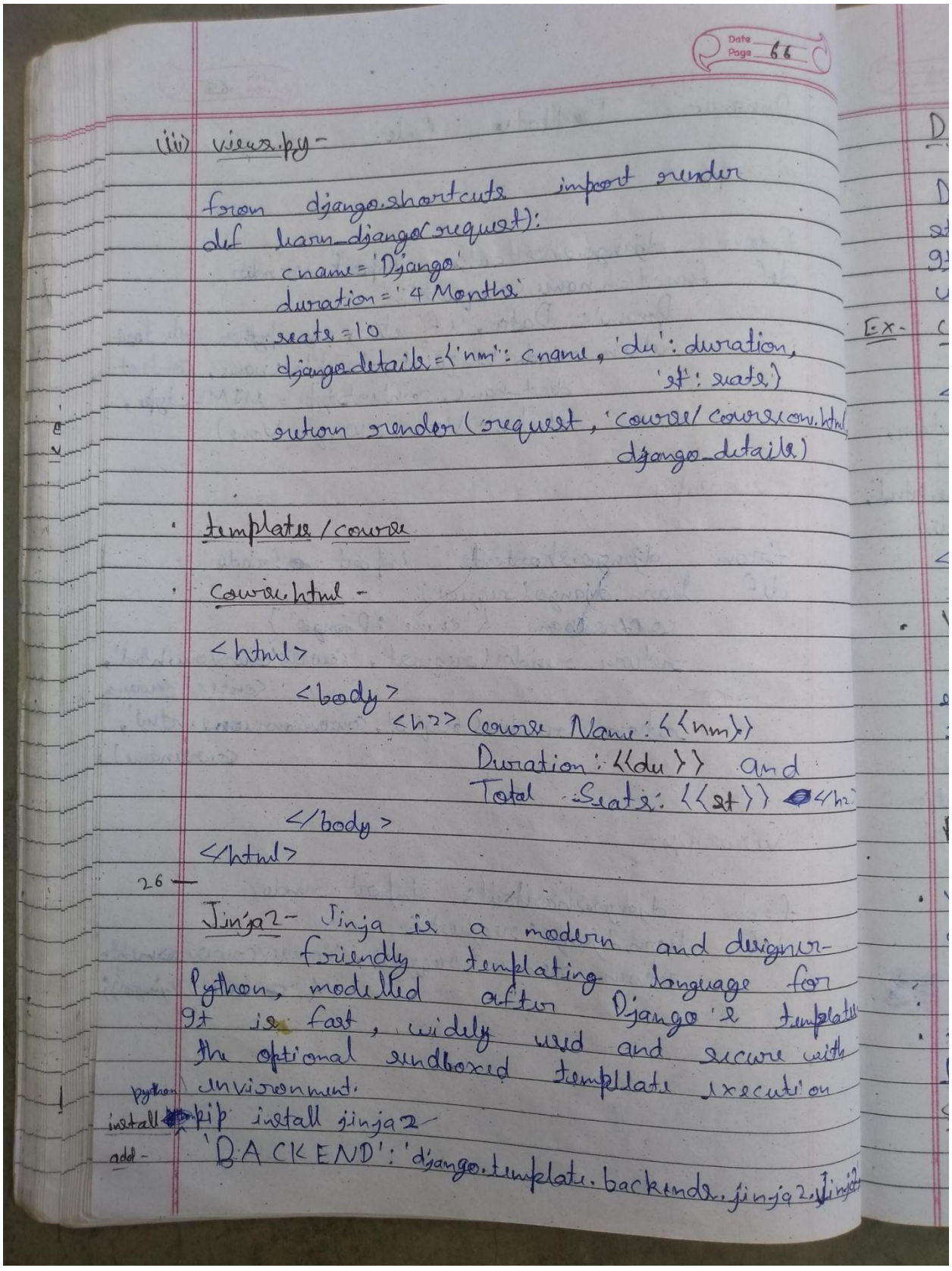
```
from django.shortcuts import render
def functionname(request):
    Dynamic Data, if else, any python code logic
    return render(request, templatename, context=
                  dict_name, content_type = MIME_type,
                  status = None, using = None)
```

Ex-(i) views.py -

```
from django.shortcuts import render
def learn_django(request):
    coursename = {'cname': 'Django'}
    return render(request, 'course/courseone.html',
                  context=coursename)
    or
    return render(request, 'course/coursename.html',
                  coursename)
```

(ii) Views.py -

```
from django.shortcuts import render
def learn_django(request):
    return render(request, 'course/coursename.html',
                  {'cname': 'Django'})
```



(iii) views.py -

```
from django.shortcuts import render
def learn_django(request):
    cname = 'Django'
    duration = '4 Months'
    seats = 10
    django_details = {'nm': cname, 'du': duration,
                      'st': seats}
    return render(request, 'course/courseconv.html',
                  django_details)
```

Ex -

- templates / course
- course.html -

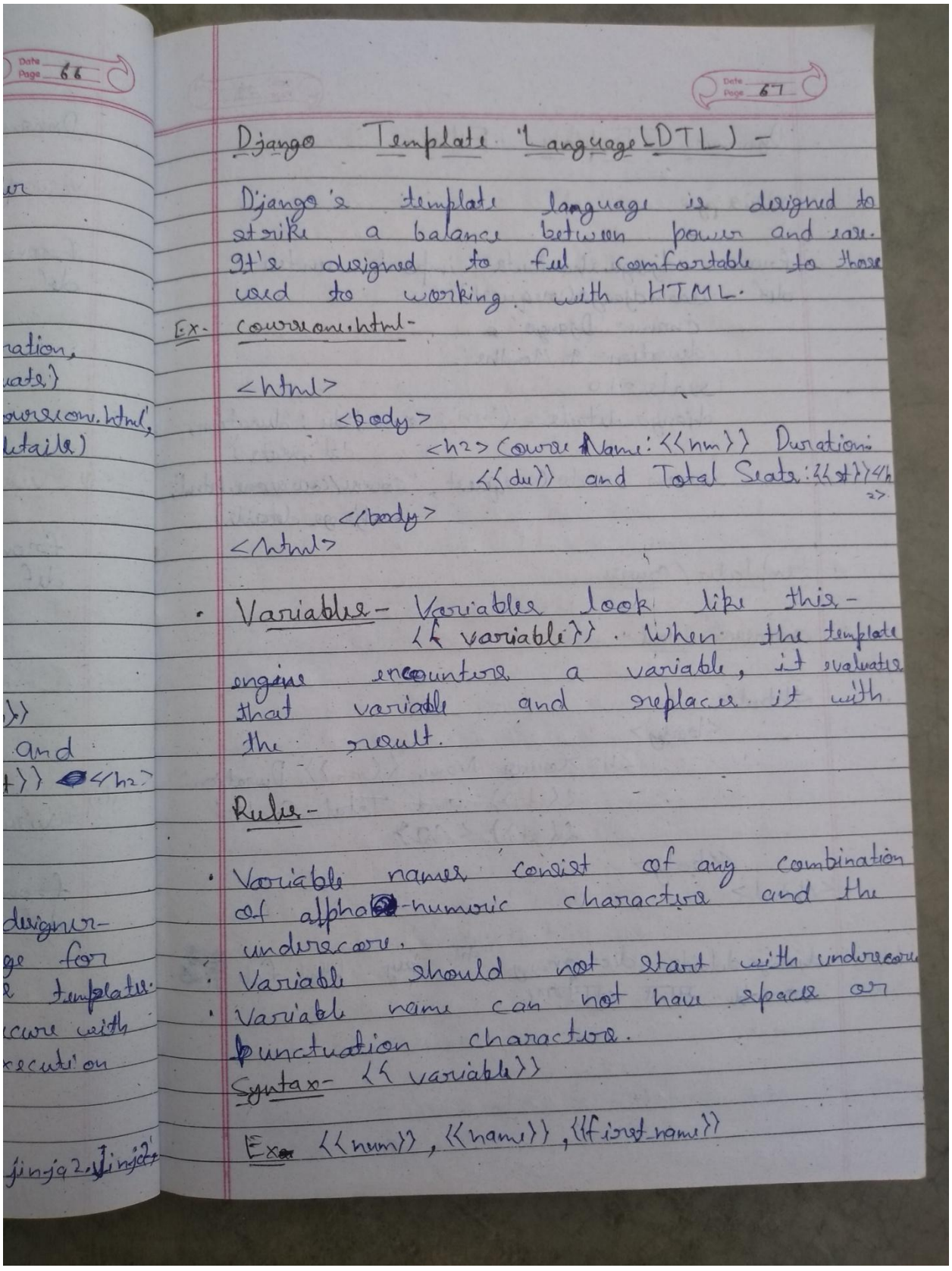
```
<html>
    <body>
        <h2> Course Name : {{nm}}
             Duration : {{du}} and
             Total Seats : {{st}} </h2>
    </body>
</html>
```

26 —

Jinja2 - Jinja is a modern and designer-friendly templating language for Python, modelled after Django's templates. It is fast, widely used and secure with the optional sandboxed template execution environment.

python install - pip install jinja2

add - 'BACKEND': 'django.template.backends.jinja2.Jinja2',



Date Page 66

Date Page 67

Django Template Language (DTL) -

Django's template language is designed to strike a balance between power and ease. It's designed to feel comfortable to those used to working with HTML.

Ex- courseone.html-

```
<html>
    <body>
        <h2> Course Name: {{nm}} Duration:
        {{du}} and Total Seats: {{st}} </h2>
    </body>
</html>
```

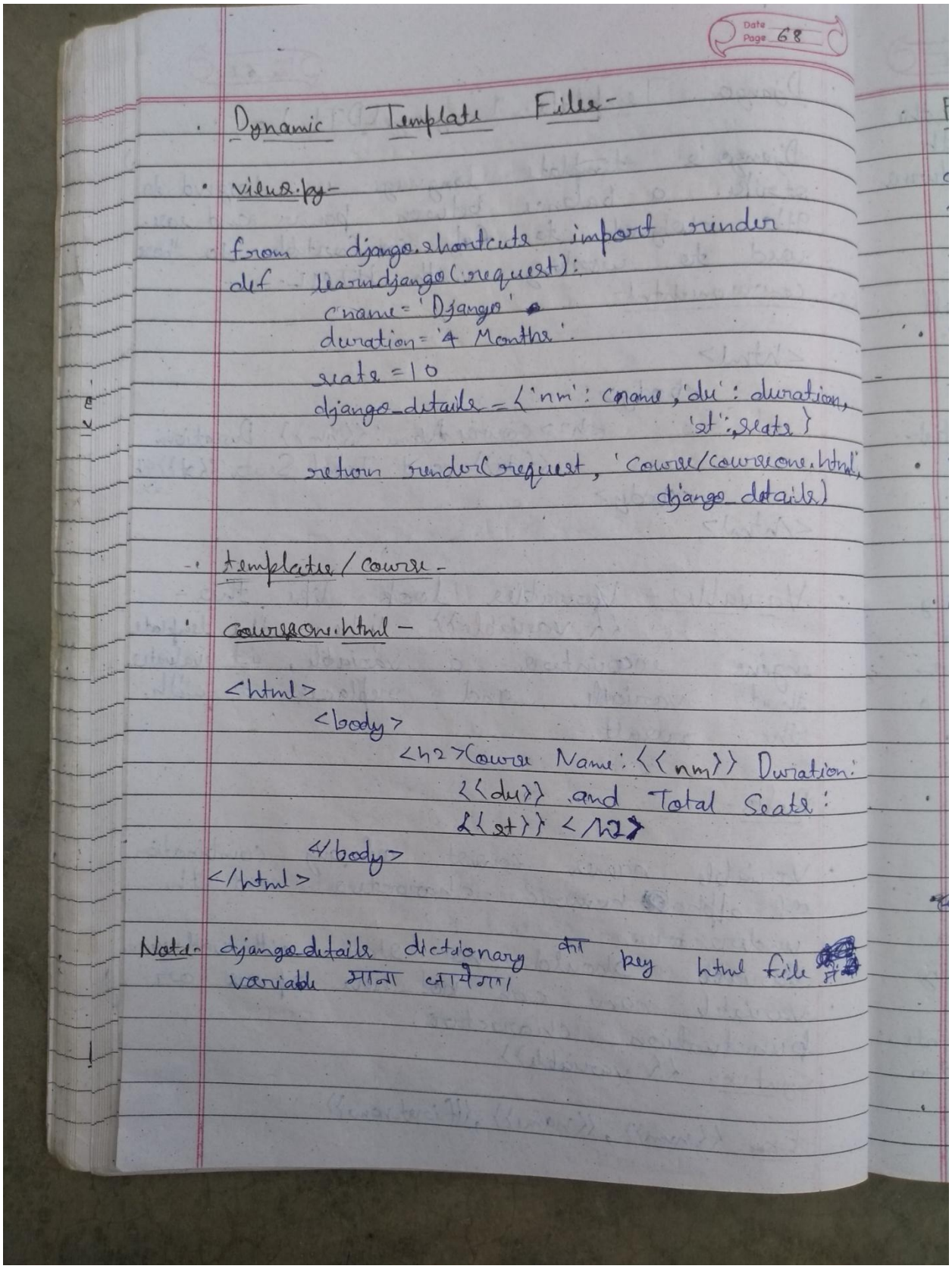
- **Variables** - Variables look like this- {{ variable }}. When the template engine encounters a variable, it evaluates that variable and replaces it with the result.

Rules -

- Variable names consist of any combination of alpha-numeric characters and the underscore.
- Variable should not start with underscore.
- Variable name can not have spaces or punctuation characters.

Syntax- {{ variable }}

Ex- {{num}}, {{name}}, {{first_name}}



Date
Page 68

- Dynamic Template Files-

- views.py-

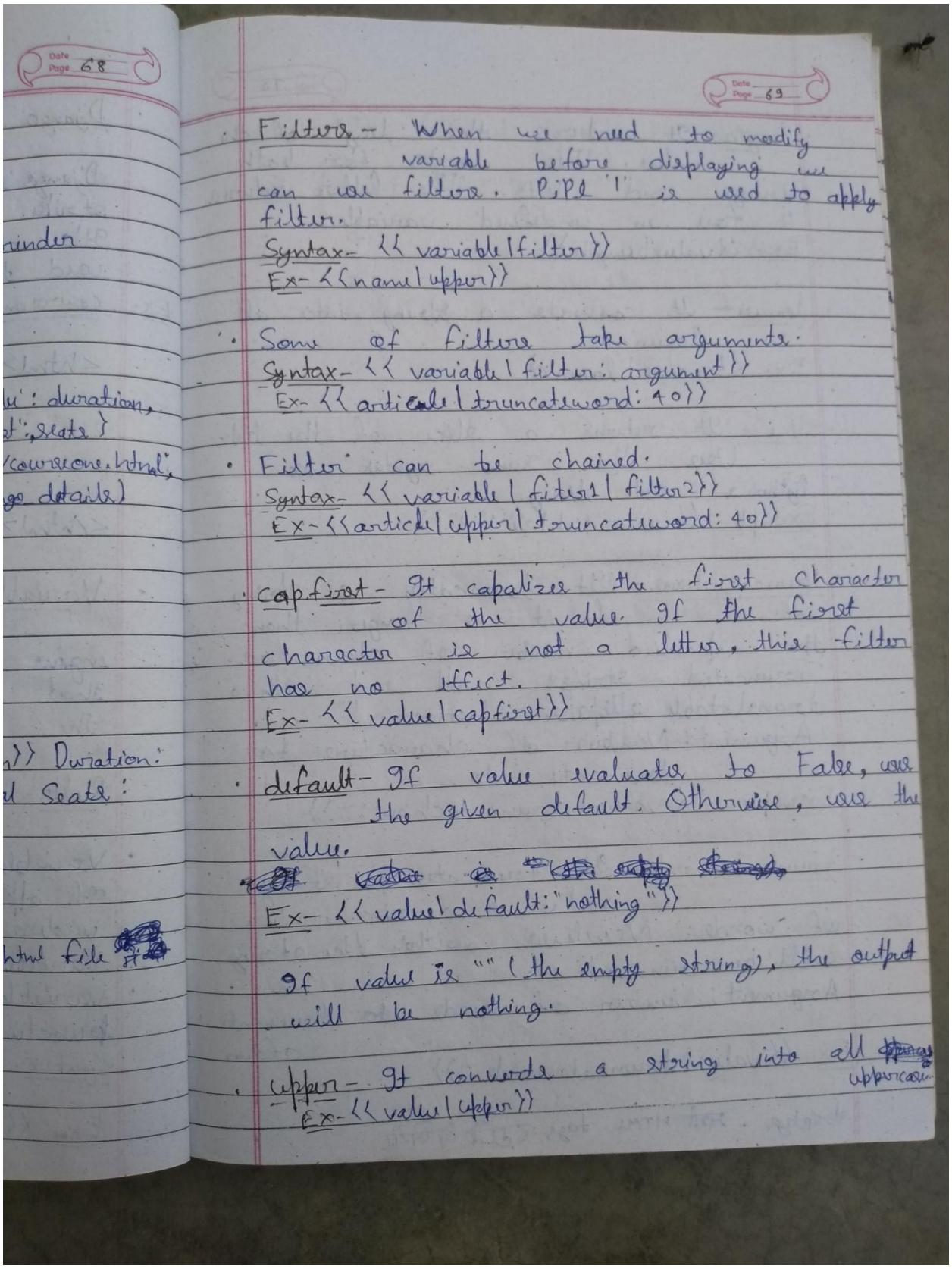
```
from django.shortcuts import render
def learndjango(request):
    cname='Django'
    duration='4 Months'
    seats=10
    django_details={'nm':cname,'du':duration,
                    'st':seats}
    return render(request,'course/courseone.html',
                  django_details)
```

- template/course -

- courseone.html -

```
<html>
    <body>
        <h2>Course Name:{{nm}} Duration:
        {{du}} and Total Seats:
        {{st}}</h2>
    </body>
</html>
```

Note- django_details dictionary का key html file में variable जैसा use होगा।



Filters - When we need to modify variable before displaying we can use filters. Pipe '|' is used to apply filter.

Syntax- {{ variable|filter }}

Ex- {{ name|upper }}

- Some of filters take arguments.

Syntax- {{ variable|filter:argument }}

Ex- {{ article|truncateword:40 }}

- Filters can be chained.

Syntax- {{ variable|filter1|filter2 }}

Ex- {{ article|upper|truncateword:40 }}

- capfirst - It capalizes the first character of the value. If the first character is not a letter, this filter has no effect.

Ex- {{ value|capfirst }}

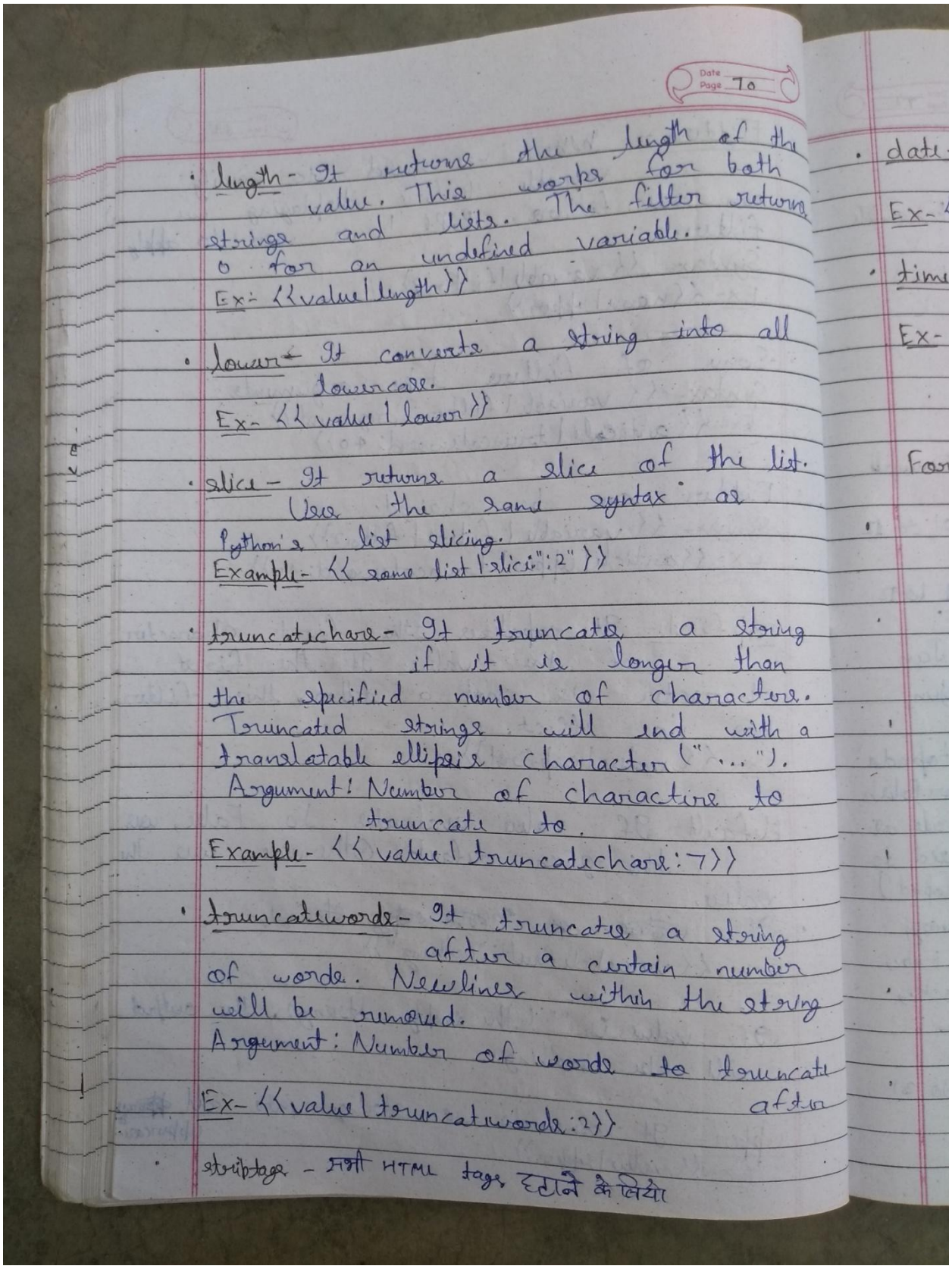
- default - If value evaluate to False, use the given default. Otherwise, use the value.

Ex- {{ value|default:"nothing" }}

If value is "" (the empty string), the output will be nothing.

- upper - It converts a string into all uppercase.

Ex- {{ value|upper }}



- length - It returns the length of the value. This works for both strings and lists. The filter returns 0 for an undefined variable.
  Ex- {{value|length}}

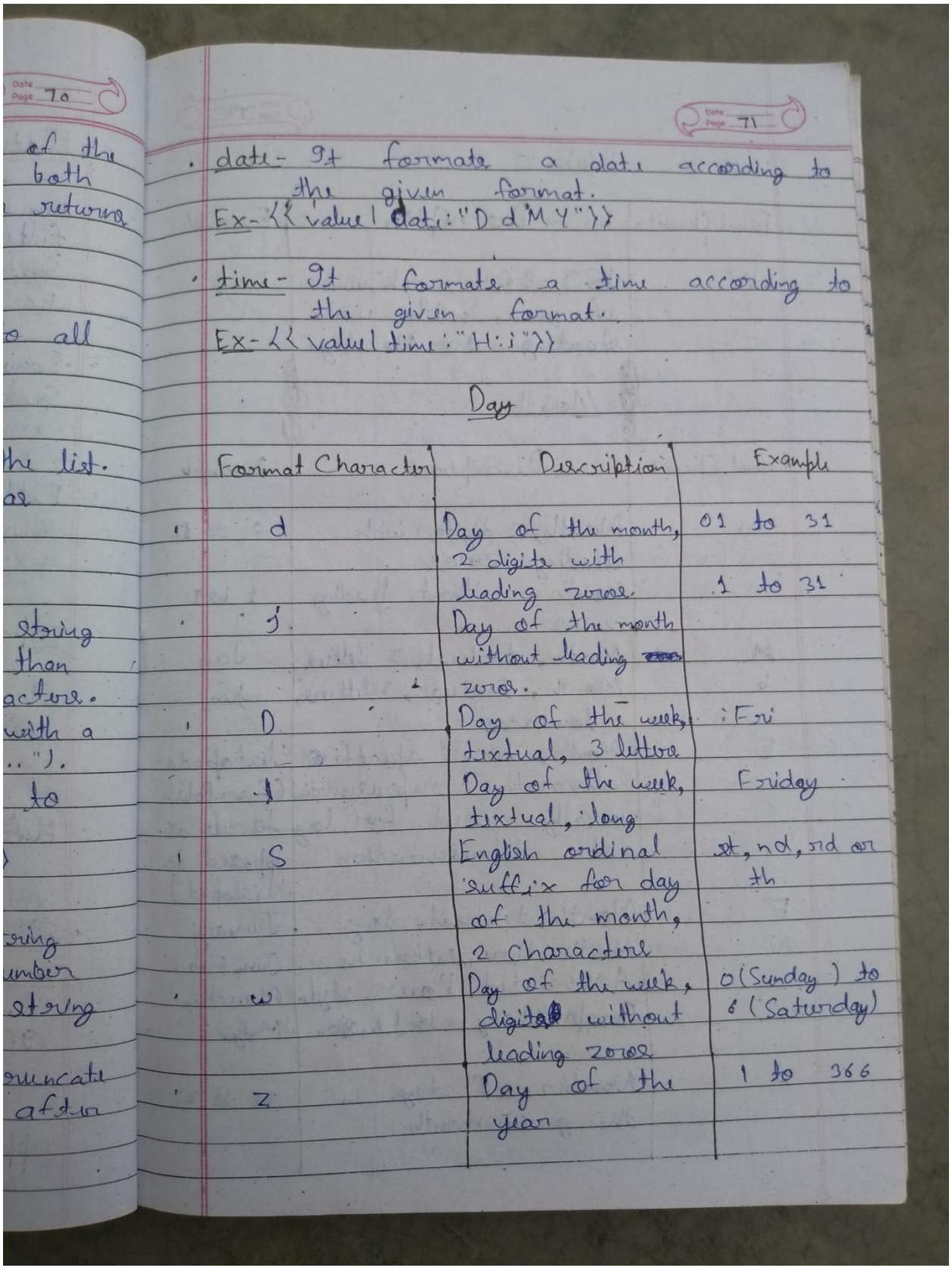
- lower - It converts a string into all lowercase.
  Ex- {{value|lower}}

- slice - It returns a slice of the list. Uses the same syntax as Python's list slicing.
  Example- {{some_list|slice":2"}}

- truncatechars - It truncates a string if it is longer than the specified number of characters. Truncated strings will end with a translatable ellipsis character ("..."). Argument: Number of charactire to truncate to.
  Example- {{value|truncatechars:7}}

- truncatewords - It truncates a string after a certain number of words. Newlines within the string will be removed.
  Argument: Number of words to truncate after
  Ex- {{value|truncatewords:2}}

- striptags - सभी HTML tags हटाने के लिये

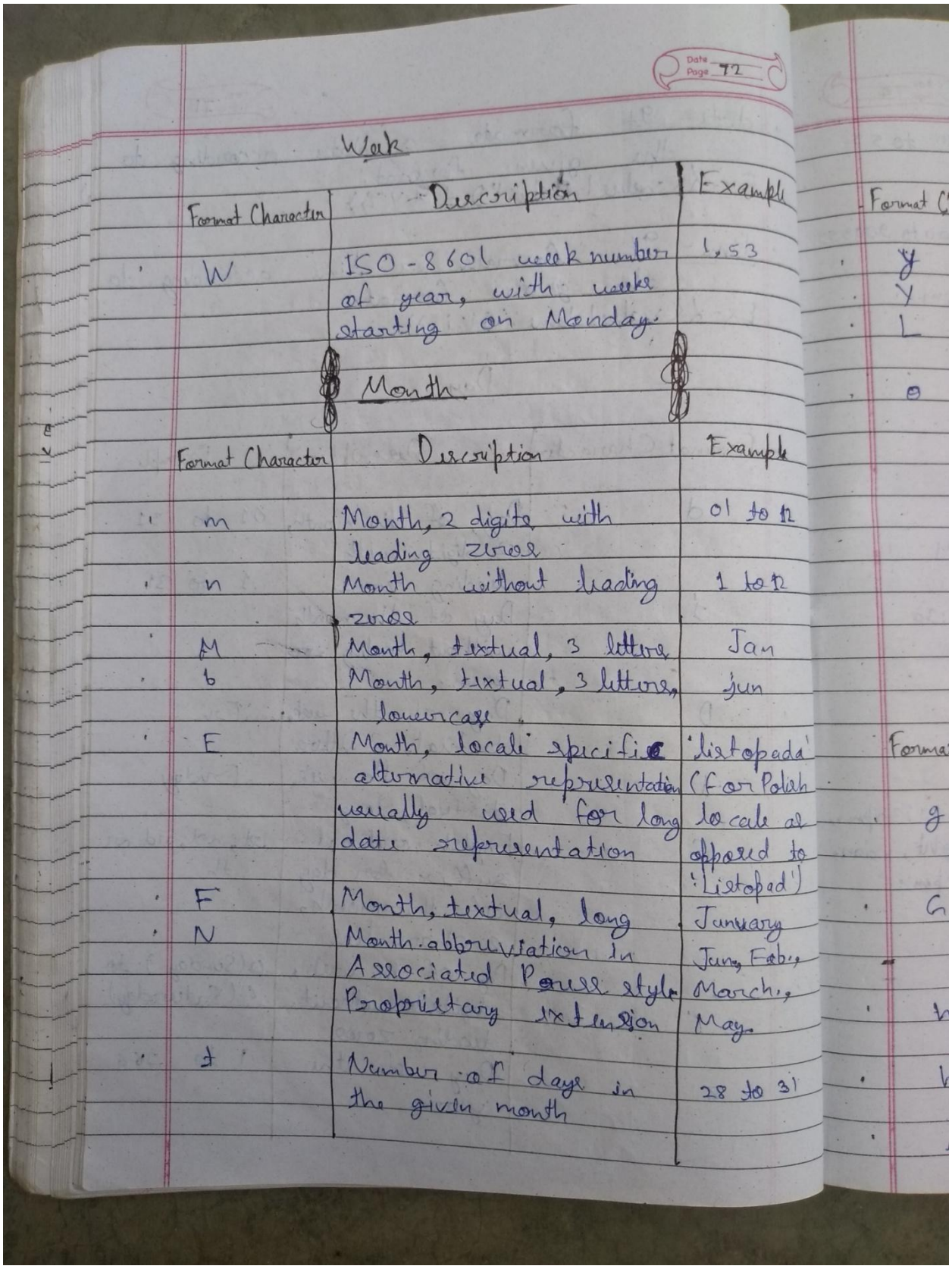


- **date** - It formats a date according to the given format.
  Ex- {{ value | date:"D d M Y" }}

- **time** - It formats a time according to the given format.
  Ex- {{ value | time:"H:i" }}

Day

Format Character	Description	Example
d	Day of the month, 2 digits with leading zeros.	01 to 31
j	Day of the month without leading zeros.	1 to 31
D	Day of the week, textual, 3 letters	Fri
l	Day of the week, textual, long	Friday
S	English ordinal suffix for day of the month, 2 characters	st, nd, rd or th
w	Day of the week, digital without leading zeros	0 (Sunday) to 6 (Saturday)
z	Day of the year	1 to 366

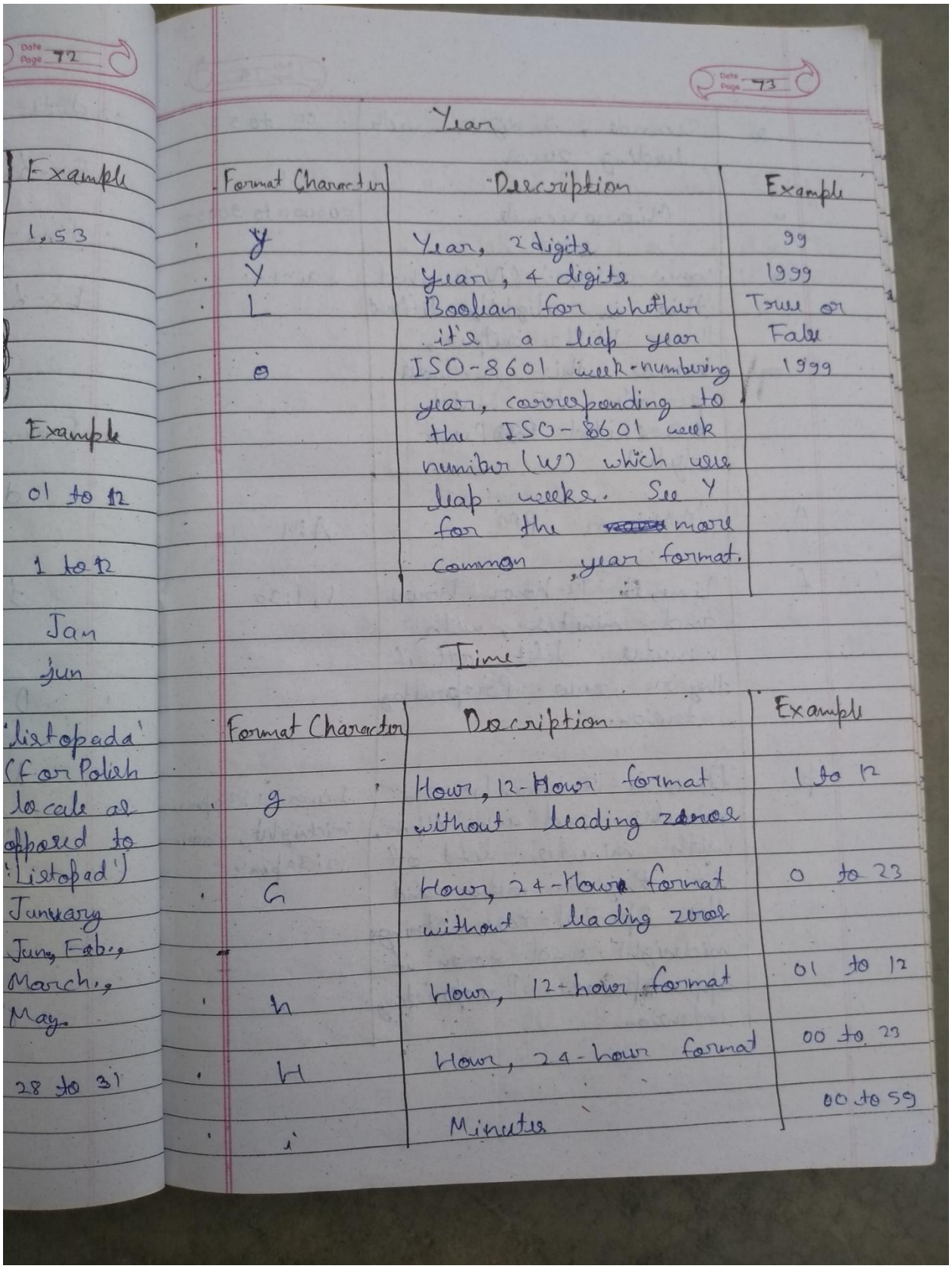


Week

Format Character	Description	Example
W	ISO-8601 week number of year, with weeks starting on Monday.	1, 53

Month

Format Character	Description	Example
m	Month, 2 digits with leading zeros	01 to 12
n	Month without leading zeros	1 to 12
M	Month, textual, 3 letters	Jan
b	Month, textual, 3 letters, lowercase	jun
E	Month, locale specific alternative representation usually used for long date representation	'listopada' (for Polish locale as opposed to 'Listopad')
F	Month, textual, long	January
N	Month abbreviation in Associated Press style. Proprietary extension	Jan., Feb., March., May.
t	Number of days in the given month	28 to 31



Date Page 72

Example

1, 53

Example

01 to 12

1 to 12

Jan

Jun

'listopada' (For Polish locale as opposed to 'Listopad')

January

Jun, Feb., March, May.

28 to 31

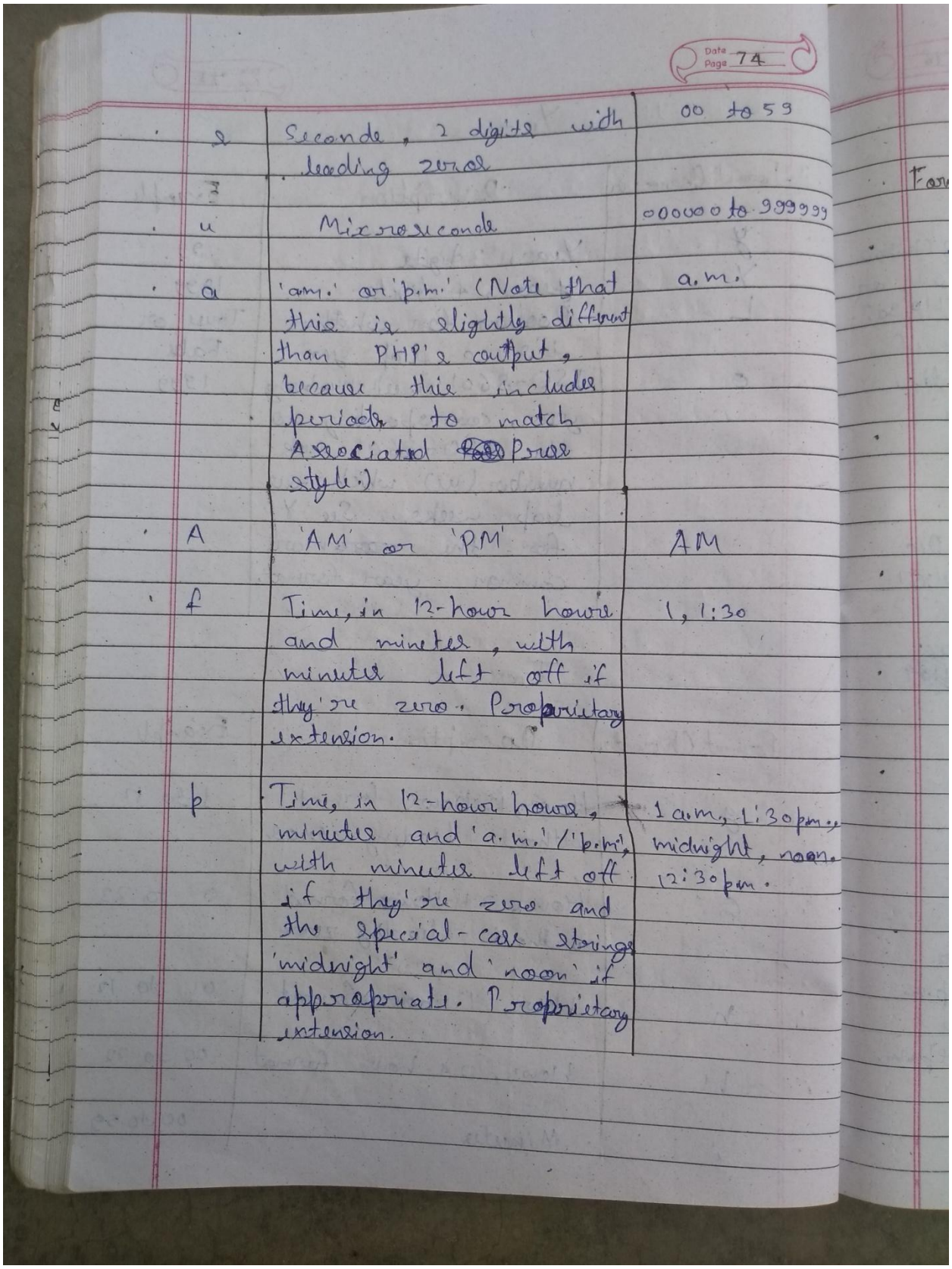
Date Page 73

Year

Format Character	Description	Example
y	Year, 2 digits	99
Y	Year, 4 digits	1999
L	Boolean for whether it's a leap year	True or False
o	ISO-8601 week-numbering year, corresponding to the ISO-8601 week number (W) which uses leap weeks. See Y for the more common year format.	1999

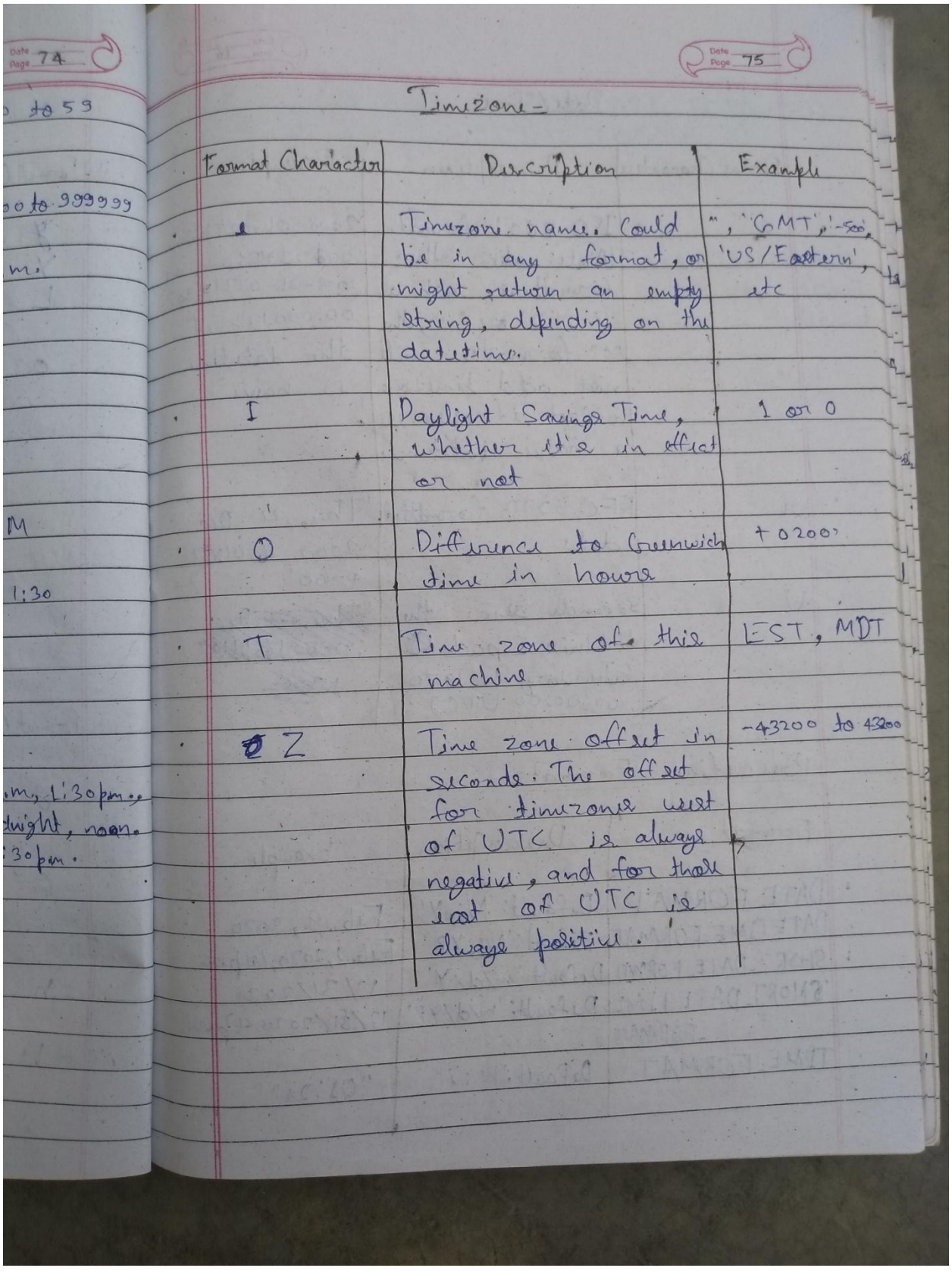
Time

Format Character	Description	Example
g	Hour, 12-Hour format without leading zeros	1 to 12
G	Hour, 24-Hour format without leading zeros	0 to 23
h	Hour, 12-hour format	01 to 12
H	Hour, 24-hour format	00 to 23
i	Minutes	00 to 59



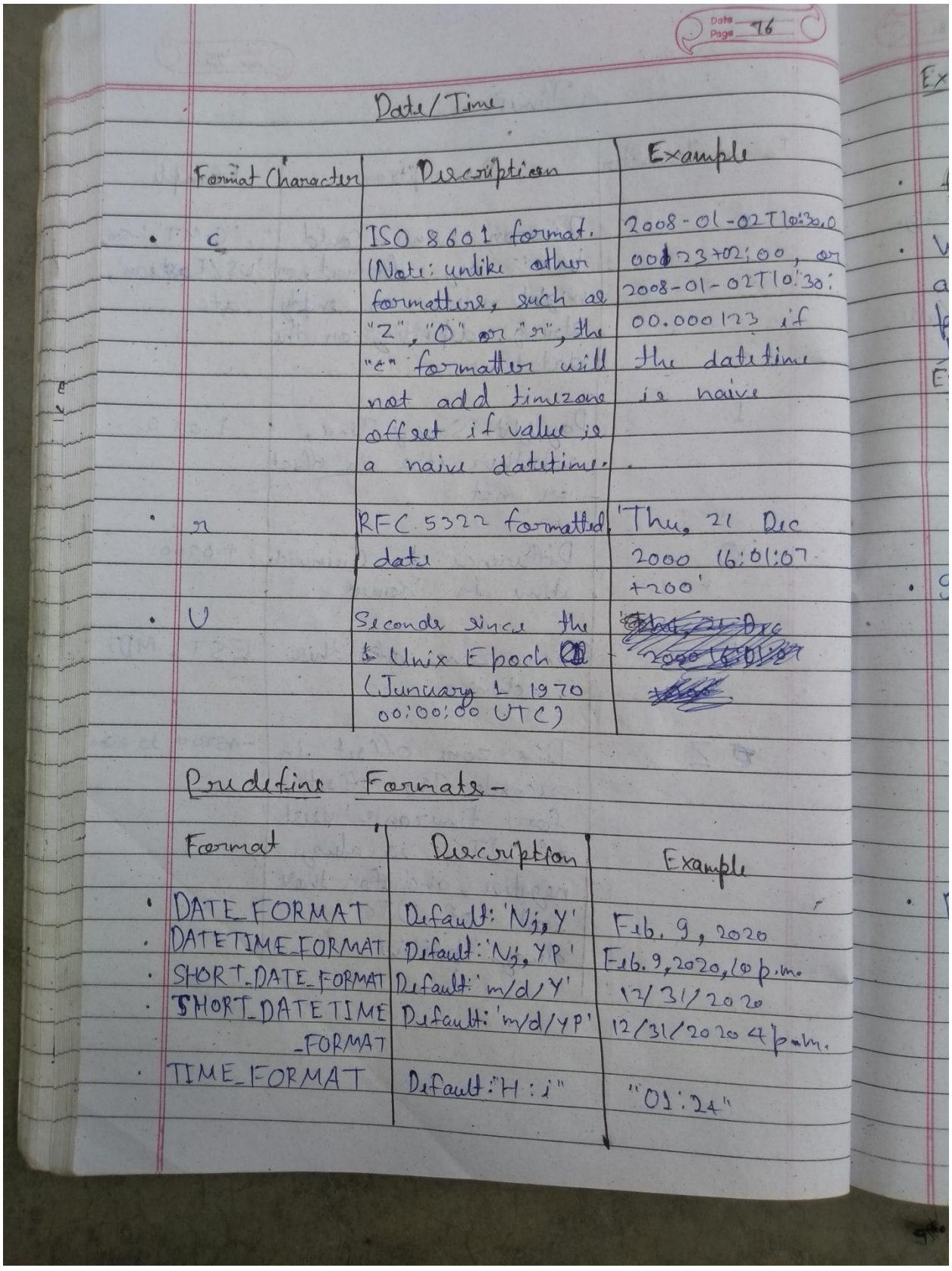
Date Page 74

s	Seconds, 2 digits with leading zeros	00 to 59
u	Microseconds	000000 to 999999
a	'am.' or 'p.m.' (Note that this is slightly different than PHP's output, because this includes periods to match Associated Press style.)	a.m.
A	'AM' or 'PM'	AM
f	Time, in 12-hour hours and minutes, with minutes left off if they're zero. Proprietary extension.	1, 1:30
P	Time, in 12-hour hours, minutes and 'a.m.'/'p.m.', with minutes left off if they're zero and the special-case strings 'midnight' and 'noon' if appropriate. Proprietary extension.	1 a.m., 1:30 p.m., midnight, noon, 12:30 p.m.



Timezone -

Format Character	Description	Example
e	Timezone name. Could be in any format, or might return an empty string, depending on the datetime.	'', 'GMT', 'US/Eastern', etc
I	Daylight Savings Time, whether it's in effect or not	1 or 0
O	Difference to Greenwich time in hours	+0200
T	Time zone of this machine	EST, MDT
Z	Time zone offset in seconds. The offset for timezones west of UTC is always negative, and for those east of UTC is always positive.	-43200 to 43200



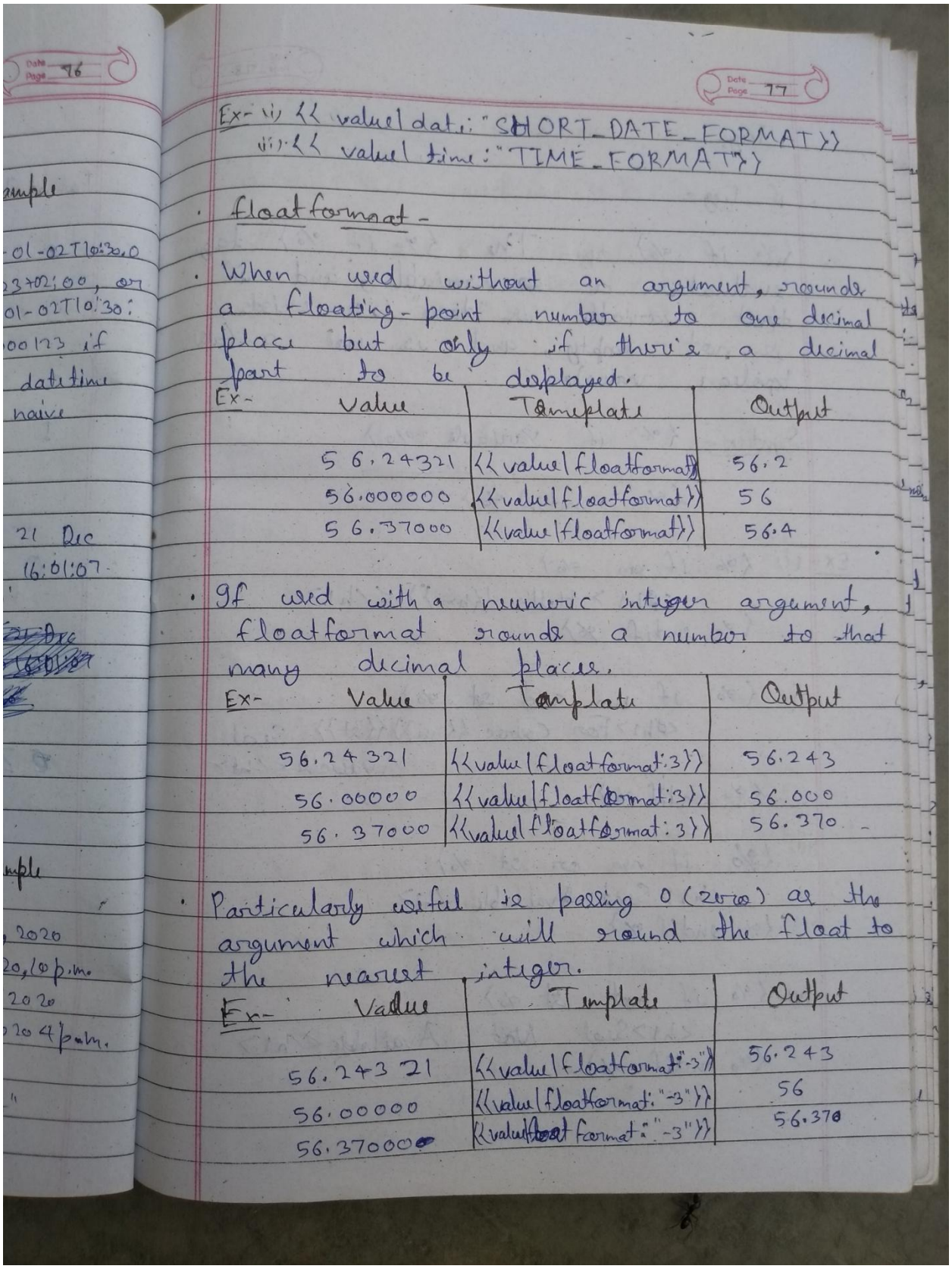
Date / Page 76

Date/Time

Format Character	Description	Example
c	ISO 8601 format. (Note: unlike other formatters, such as "Z", "O" or "r", the "c" formatter will not add timezone offset if value is a naive datetime.	2008-01-02T10:30:0 0.000123+02:00, or 2008-01-02T10:30: 00.000123 if the datetime is naive
r	RFC 5322 formatted date	'Thu, 21 Dec 2000 16:01:07 +200'
U	Seconds since the Unix Epoch (January 1 1970 00:00:00 UTC)	

Predefine Formats -

Format	Description	Example
DATE_FORMAT	Default: 'N j, Y'	Feb. 9, 2020
DATETIME_FORMAT	Default: 'N j, Y P'	Feb. 9, 2020, 10 p.m.
SHORT_DATE_FORMAT	Default: 'm/d/Y'	12/31/2020
SHORT_DATETIME_FORMAT	Default: 'm/d/Y P'	12/31/2020 4 p.m.
TIME_FORMAT	Default: 'H:i'	"01:24"



Ex- i) {{ value|date:"SHORT_DATE_FORMAT" }}
ii) {{ value|time:"TIME_FORMAT" }}

- floatformat -

- When used without an argument, rounds a floating-point number to one decimal place but only if there's a decimal part to be displayed.

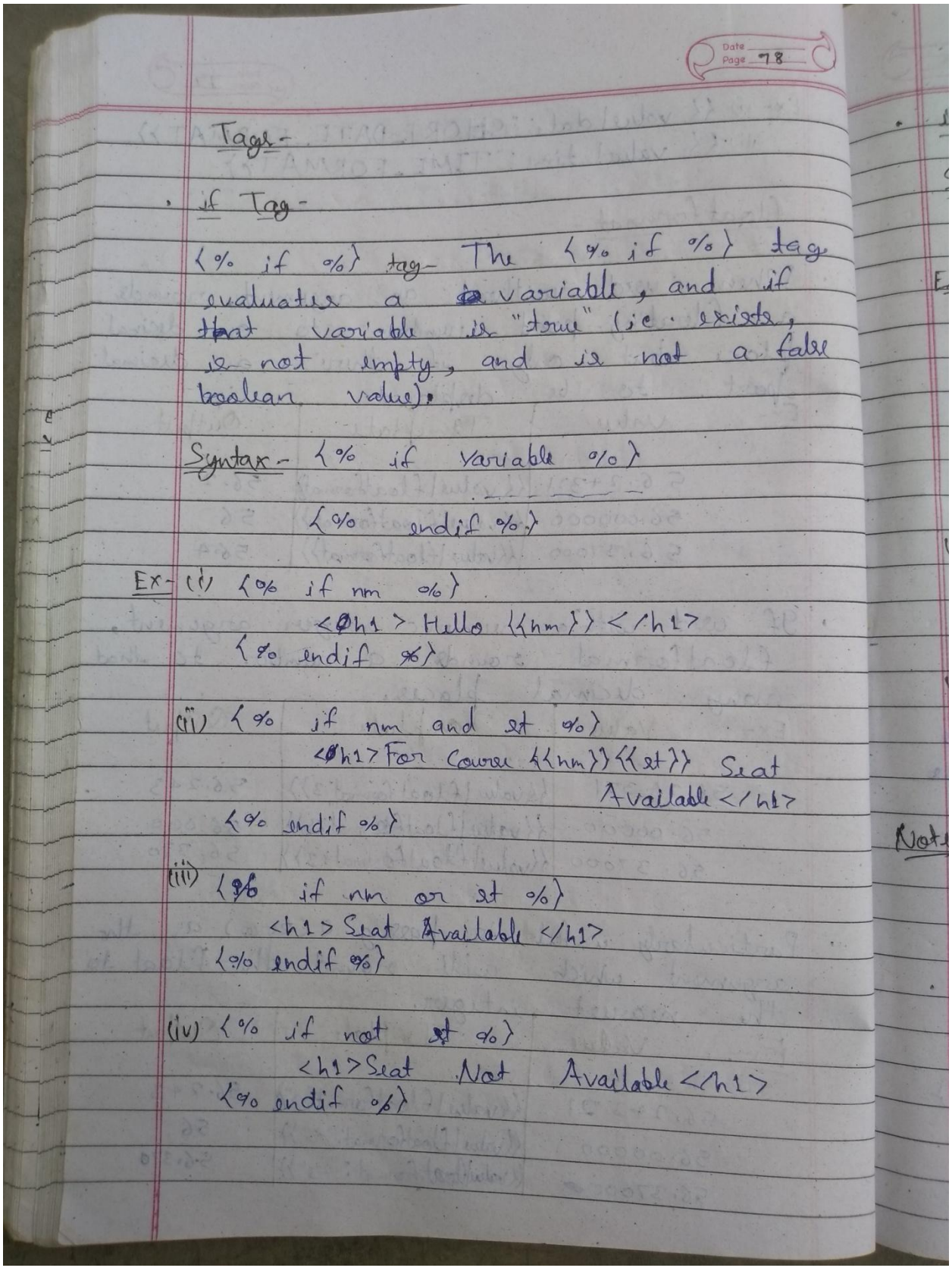
Ex- Value	Template	Output
56.24321	{{ value|floatformat }}	56.2
56.000000	{{ value|floatformat }}	56
56.37000	{{ value|floatformat }}	56.4

- If used with a numeric integer argument, floatformat rounds a number to that many decimal places.

Ex- Value	Template	Output
56.24321	{{ value|floatformat:3 }}	56.243
56.00000	{{ value|floatformat:3 }}	56.000
56.37000	{{ value|floatformat:3 }}	56.370

- Particularly useful is passing 0 (zero) as the argument which will round the float to the nearest integer.

Ex- Value	Template	Output
56.24321	{{ value|floatformat:"-3" }}	56.243
56.00000	{{ value|floatformat:"-3" }}	56
56.37000	{{ value|floatformat:"-3" }}	56.370



Date
Page 78

Tags -

. if Tag -

{% if %} tag - The {% if %} tag evaluates a variable, and if that variable is "true" (i.e. exists, is not empty, and is not a false boolean value).

Syntax - {% if variable %}

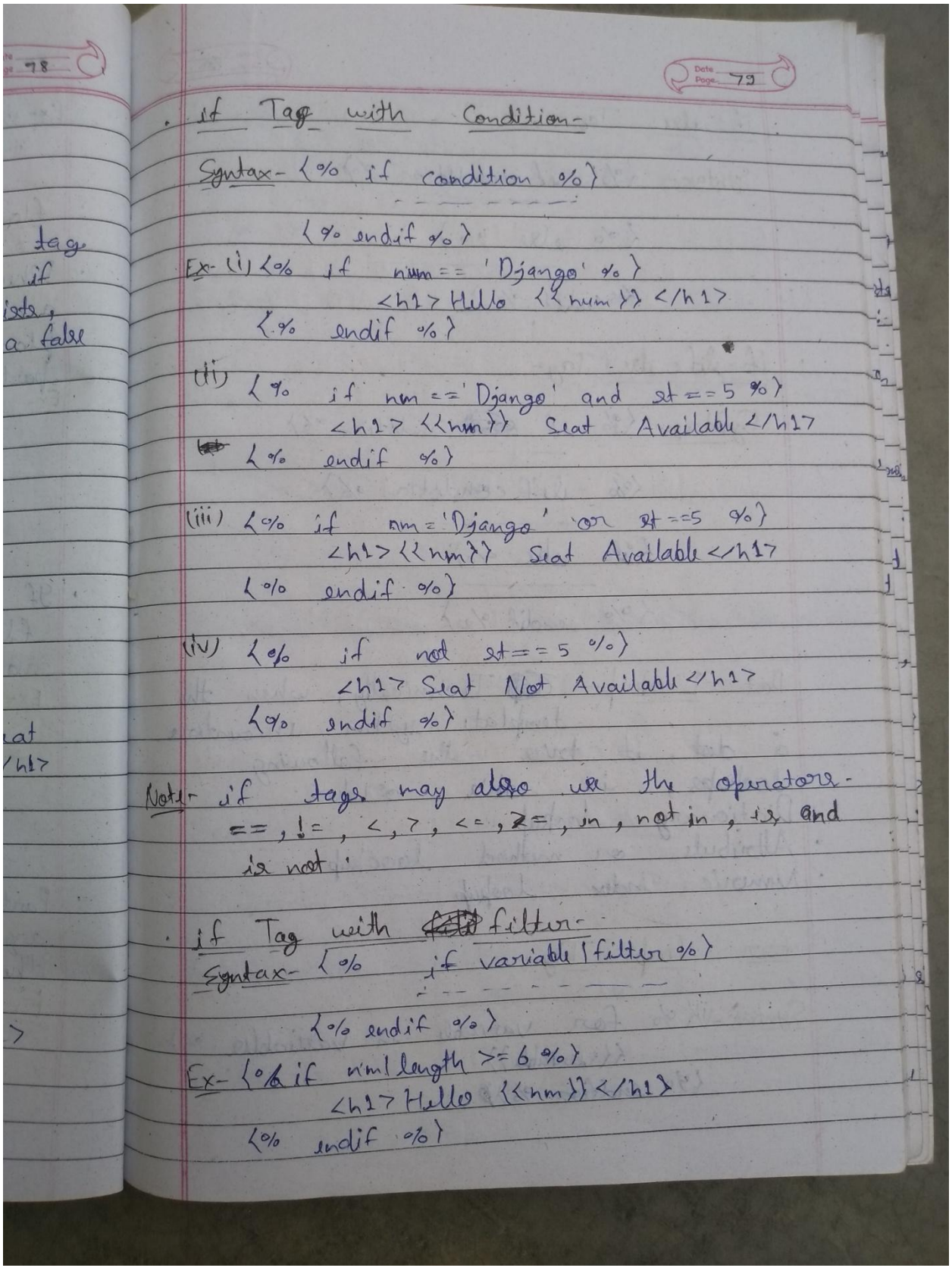
{% endif %}

Ex - (i) {% if nm %}
<h1> Hello {{nm}} </h1>
{% endif %}

(ii) {% if nm and st %}
<h1> For Course {{nm}} {{st}} Seat Available </h1>
{% endif %}

(iii) {% if nm or st %}
<h1> Seat Available </h1>
{% endif %}

(iv) {% if not st %}
<h1> Seat Not Available </h1>
{% endif %}



. if Tag with Condition-

Syntax- {% if condition %}
.....
{% endif %}

Ex- (i) {% if nm == 'Django' %}
<h1> Hello {{ nm }} </h1>
{% endif %}

(ii) {% if nm == 'Django' and st == 5 %}
<h1> {{ nm }} Seat Available </h1>
{% endif %}

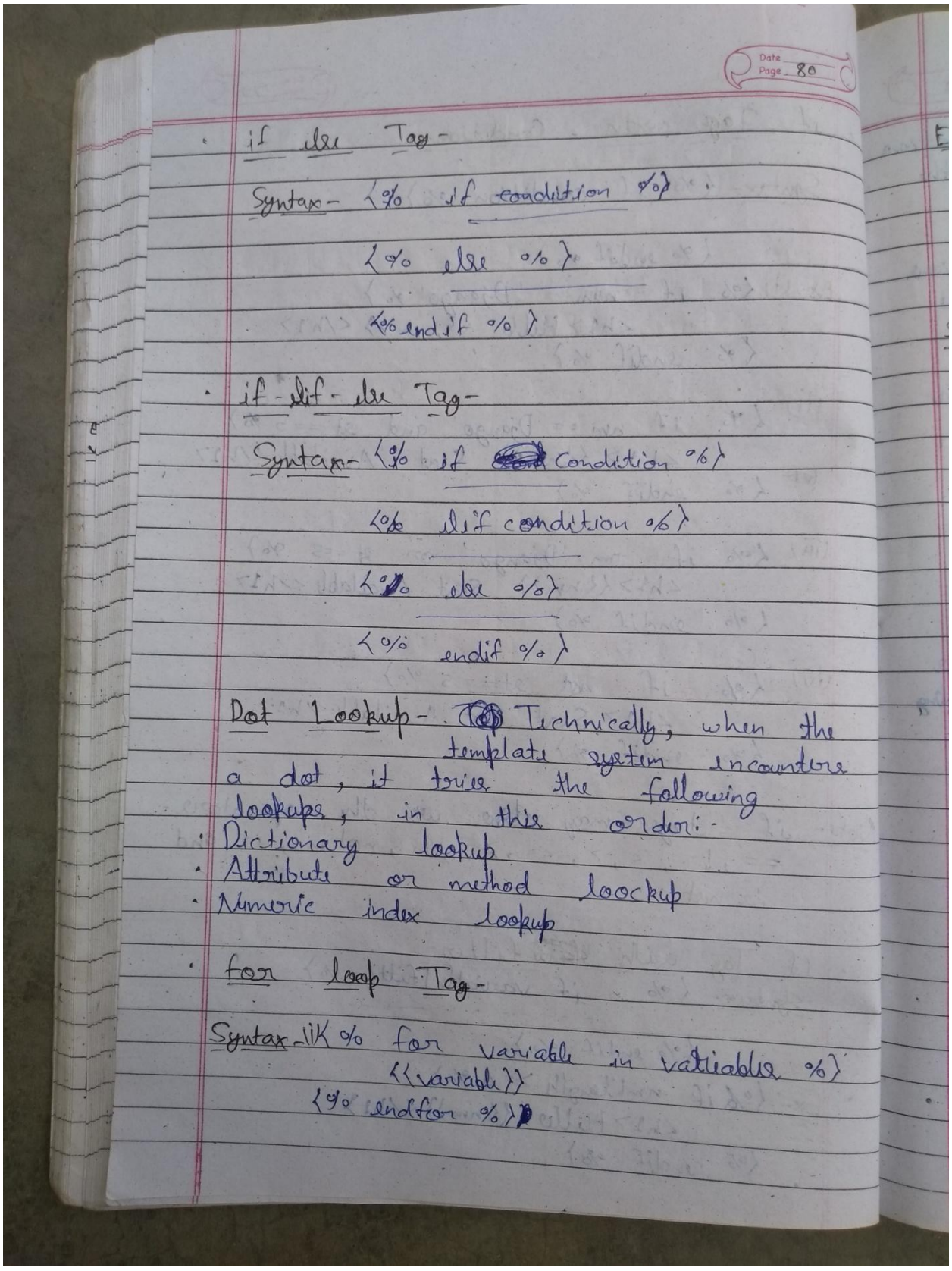
(iii) {% if nm = 'Django' or st == 5 %}
<h1> {{ nm }} Seat Available </h1>
{% endif %}

(iv) {% if not st == 5 %}
<h1> Seat Not Available </h1>
{% endif %}

Note- if tags may also use the operators-
==, !=, <, >, <=, >=, in, not in, is and is not.

. if Tag with filter-
Syntax- {% if variable | filter %}
.....
{% endif %}

Ex- {% if nm | length >= 6 %}
<h1> Hello {{ nm }} </h1>
{% endif %}



Date
Page 80

- if else Tag -

Syntax - {% if condition %}

{% else %}

{% endif %}

- if - elif - else Tag -

Syntax - {% if condition %}

{% elif condition %}

{% else %}

{% endif %}

Dot Lookup - Technically, when the template system encounters a dot, it tries the following lookups, in this order:

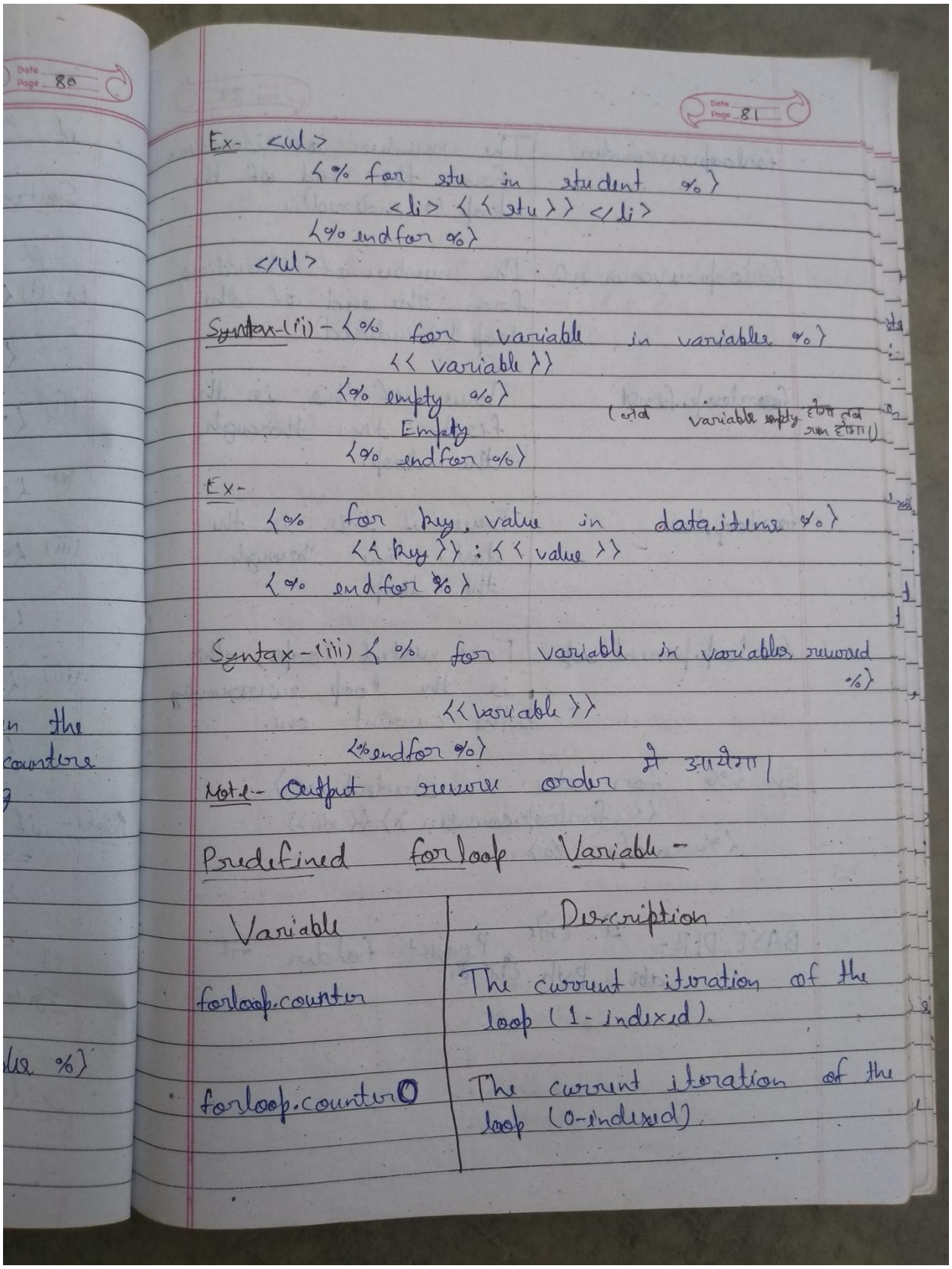
- Dictionary lookup
- Attribute or method loockup
- Numeric index lookup

- for loop Tag -

Syntax - {% for variable in variable %}

{{variable}}

{% endfor %}



Ex-
{% for stu in student %}
 {{ stu }}
{% endfor %}

Syntax-(i) - {% for variable in variables %}
{{ variable }}
{% empty %}
Empty
{% endfor %}

(जब variable empty होगा तब run होगा।)

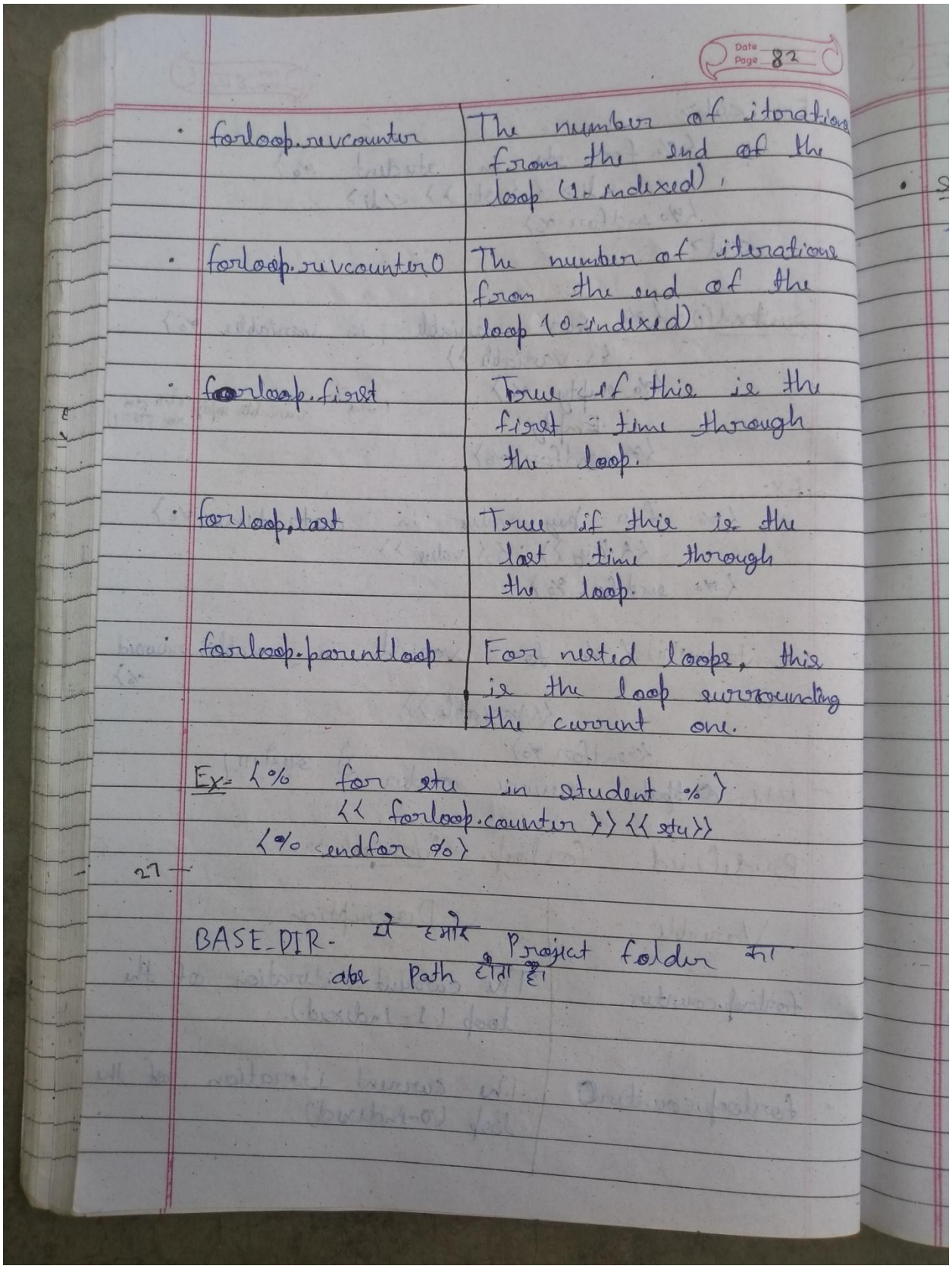
Ex-
{% for key, value in data.items %}
{{ key }} : {{ value }}
{% endfor %}

Syntax-(iii) {% for variable in variables reversed %}
{{ variable }}
{% endfor %}

Note:- Output reverse order में आयेगा।

Predefined forloop Variable -

Variable	Description
forloop.counter	The current iteration of the loop (1-indexed).
forloop.counter0	The current iteration of the loop (0-indexed).



Date Page 82

forloop.revcounter	The number of iterations from the end of the loop (1-indexed).
forloop.revcounter0	The number of iterations from the end of the loop (0-indexed).
forloop.first	True if this is the first time through the loop.
forloop.last	True if this is the last time through the loop.
forloop.parentloop	For nested loops, this is the loop surrounding the current one.

Ex- {% for stu in student %}
{{ forloop.counter }} {{ stu }}
{% endfor %}

27 -

BASE_DIR- यह हमारे Project folder का abs Path होता है।