PART-2

django Web Framework

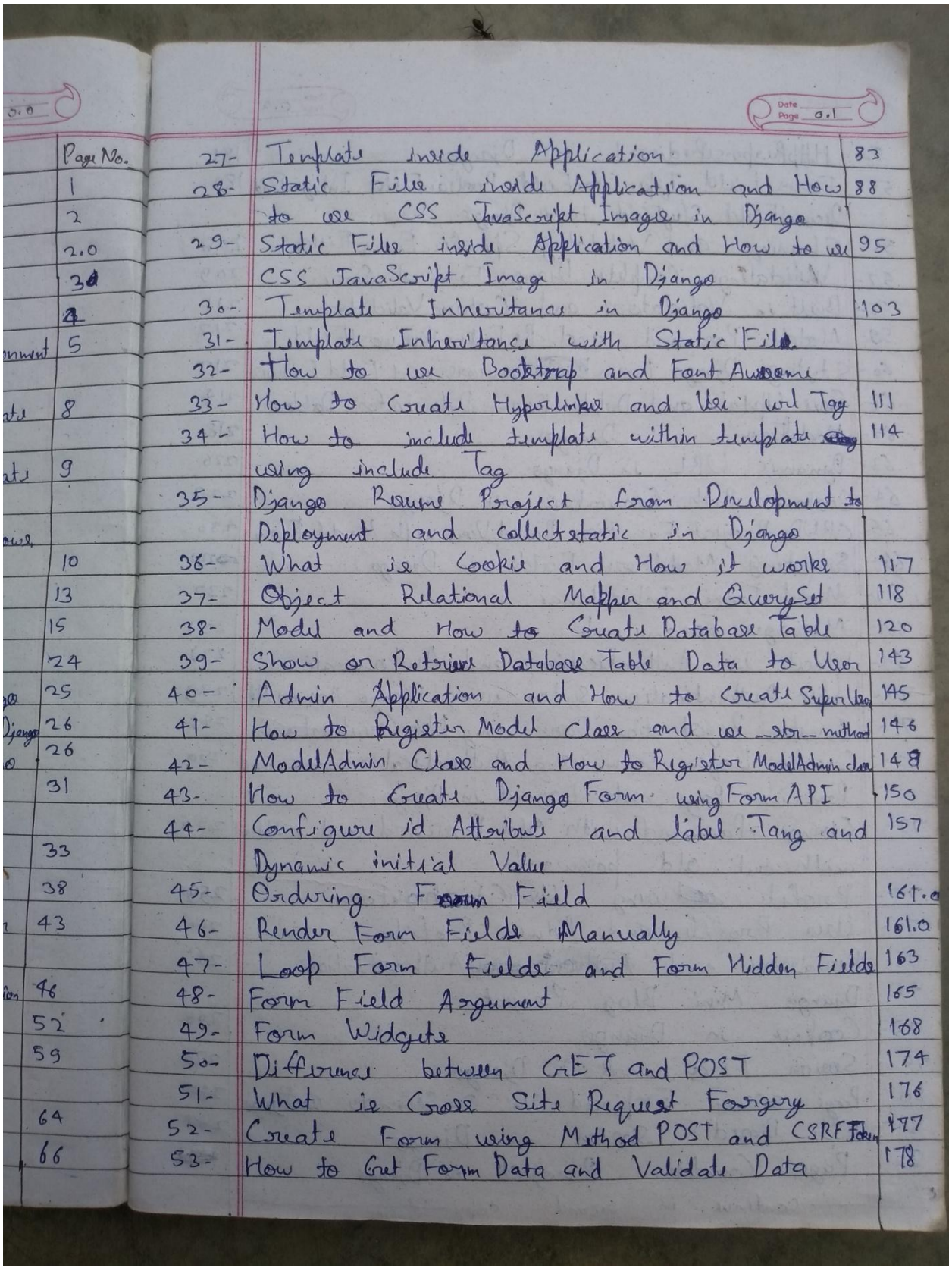
Geeky Shows YouTube Channel Learning Notes

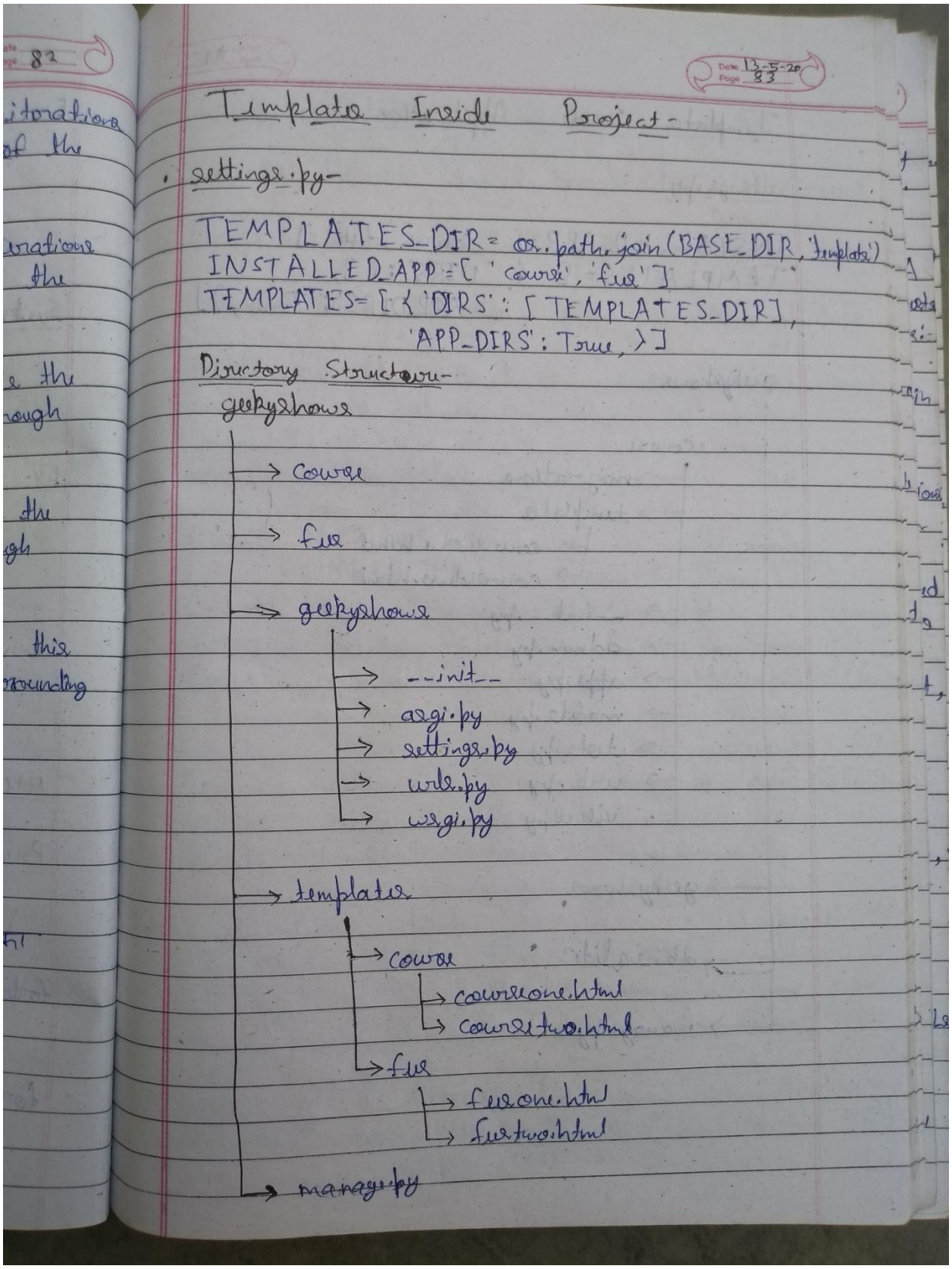
SATYAM SETH

14/09/2020

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

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





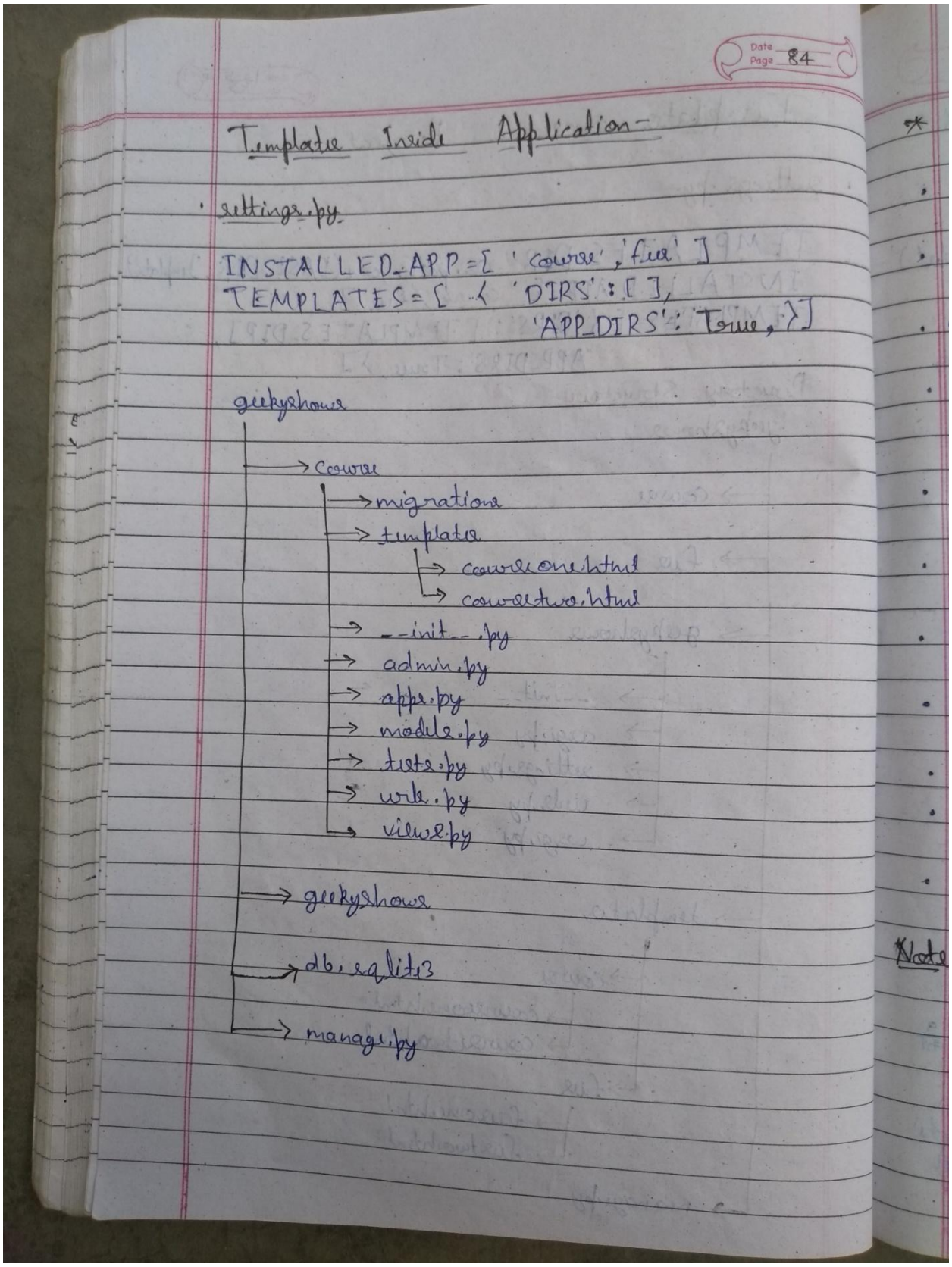
Template Inside Project -

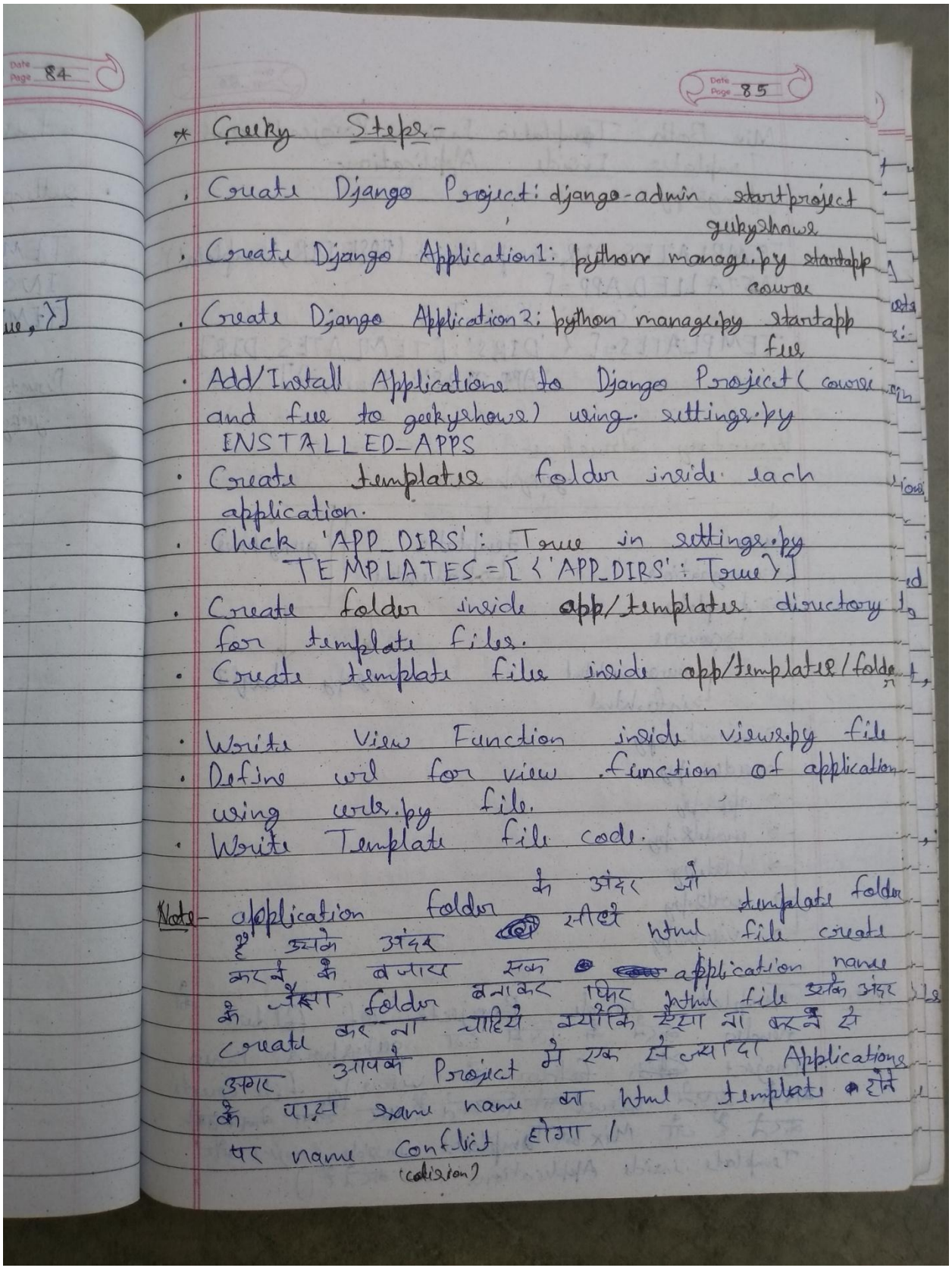
- settings.py -

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

Directory Structure -

```
geekyshows
    → course
    → fee
    → geekyshows
        → __init__
        → asgi.py
        → settings.py
        → urls.py
        → wsgi.py
    → templates
        → course
            → courseone.html
            → coursetwo.html
        → fee
            → feeone.html
            → feetwo.html
    → manage.py
```


Date
Page 84
Template Inside Application-
settings.py
INSTALLED_APP=['course',]
TEMPLATES=[{ 'DIRS':[ ],
'APP_DIRS': True, }]
geekyshows
course
migrations
template
courseone.html
coursetwo.html
__init__.py
admin.py
apps.py
models.py
tests.py
urls.py
views.py
geekyshows
db.sqlite3
manage.py
Note



Date Page 84

Date Page 85

* 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 templates folder inside each application.
. Check 'APP_DIRS': True in settings.py TEMPLATES = [{'APP_DIRS': True}]
. Create folder inside app/templates directory for template files.
. Create template file inside app/templates/folder
. Write View Function inside views.py file
. Define url for view function of application using urls.py file.
. Write Template file code.

Note - application folder के अंदर जो templates folder हम उसके अंदर सीधे html file create करने के बजाय एक application name के जैसा folder बनाकर फिर html file उसके अंदर create कर ना चाहिये क्योंकि ऐसा ना करने से आगे आपके Project में एक से ज्यादा Applications के पास same name का html template होने पर name Conflict होगा / (collision)

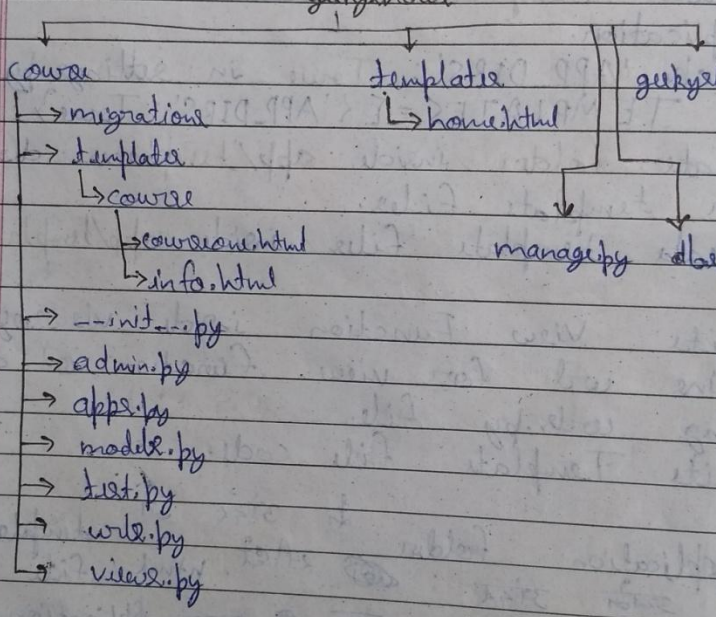
Date Page 86

Mix Both Templates Inside Project and Templates Inside Application

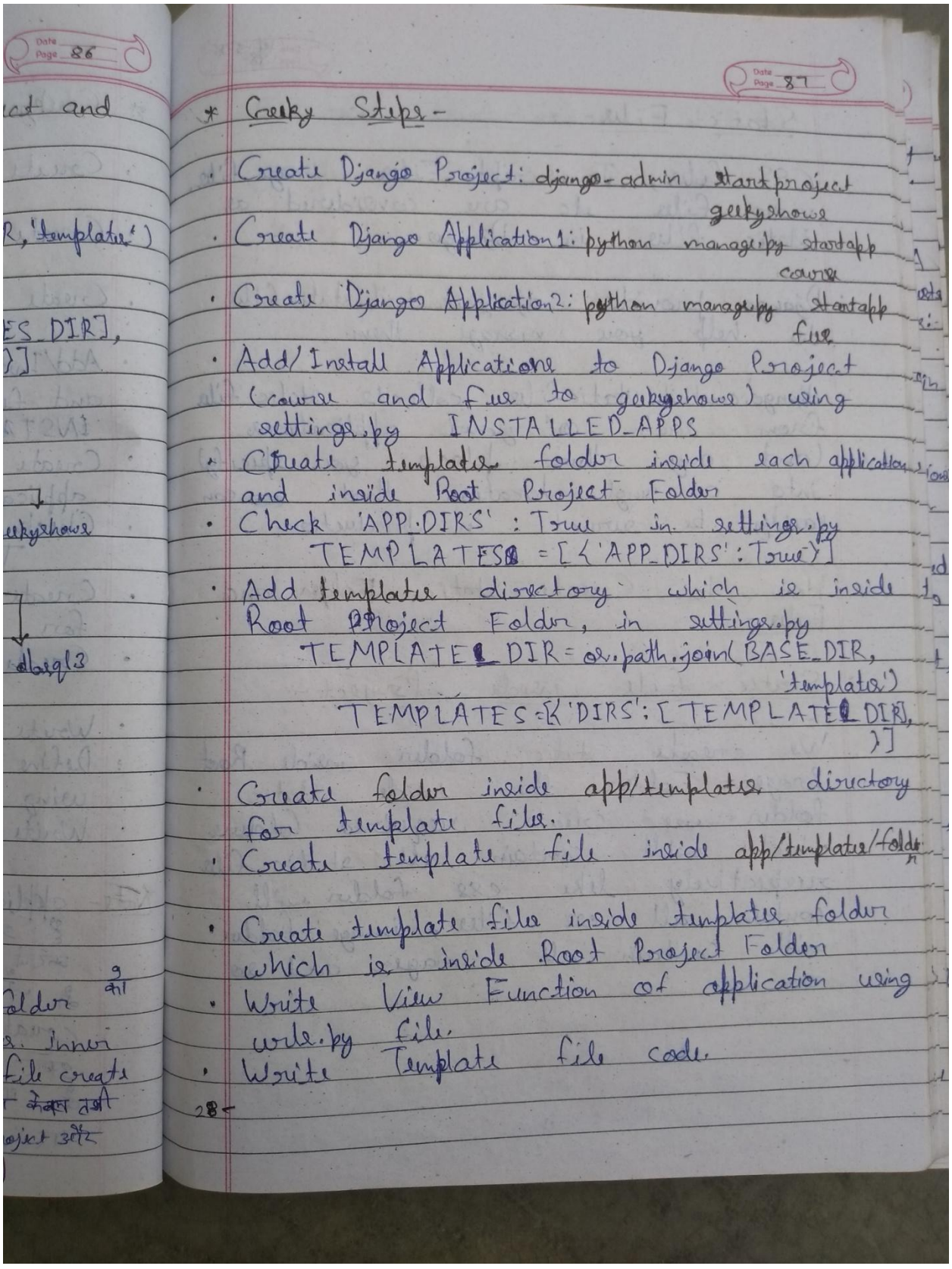
settings.py

```
TEMPLATES_DIR = os.path.join(BASE_DIR, 'templates')
INSTALLED_APP = [
        'course', 'fun' ]
TEMPLATES = [ { 'DIRS': [TEMLATES_DIR],
        'APP_DIRS': True, } ]
```

Directory Structure-



Note:- Template inside Projects of folder को render करने के लिये हम geekyshows inner Project folder में views.py file create करके अपने views को लिखते हैं। पर ऐसा केवल तभी करते हैं जो Mix both Template inside Project और Template inside Applications use करते हैं।



Date Page 86

ct and

R, 'template')

ES_DIR],
}]

ekyshows

dbsql3

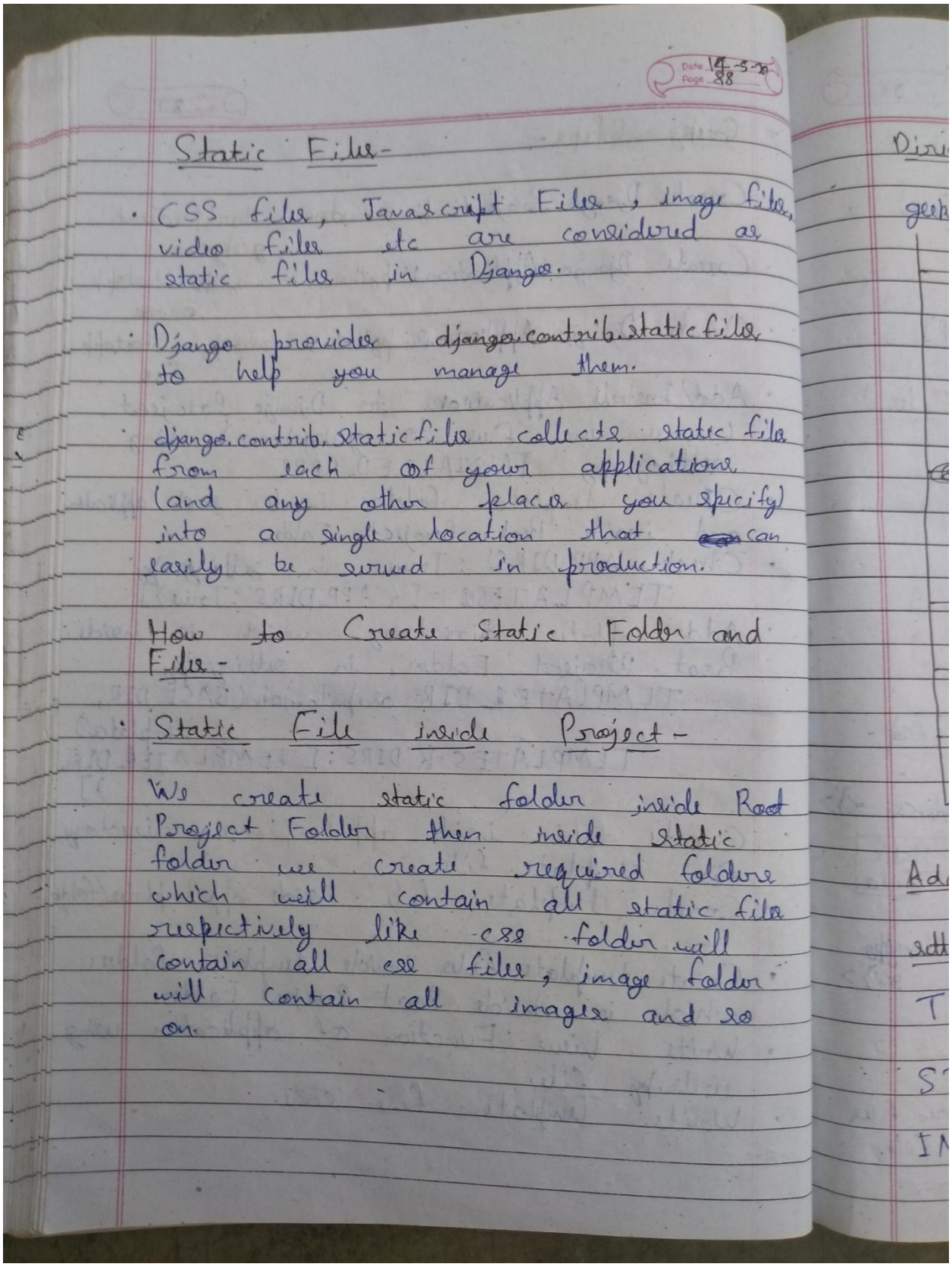
Folder

Date Page 87

* Geeky Steps -

- Create Django Project: django-admin startproject geekyshows
- Create Django Application 1: python manage.py startapp course
- Create Django Application 2: 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 each application and inside Root Project Folder
- Check 'APP_DIRS' : True in settings.py
  TEMPLATES = [{'APP_DIRS': True}]
- Add template directory which is inside Root Project Folder, in settings.py
  TEMPLATE_DIR = os.path.join(BASE_DIR, 'templates')
  TEMPLATES = [{'DIRS': [TEMPLATE_DIR],}]
- Create folder inside app/templates directory for template file.
- Create template file inside app/templates/folder
- Create template file inside templates folder which is inside Root Project Folder
- Write View Function of application using urls.py file.
- Write Template file code.

28-



Date 14-5-20
Page 88

Static Files-

- CSS files, Javascript Files, image files, video files etc are considered as static files in Django.

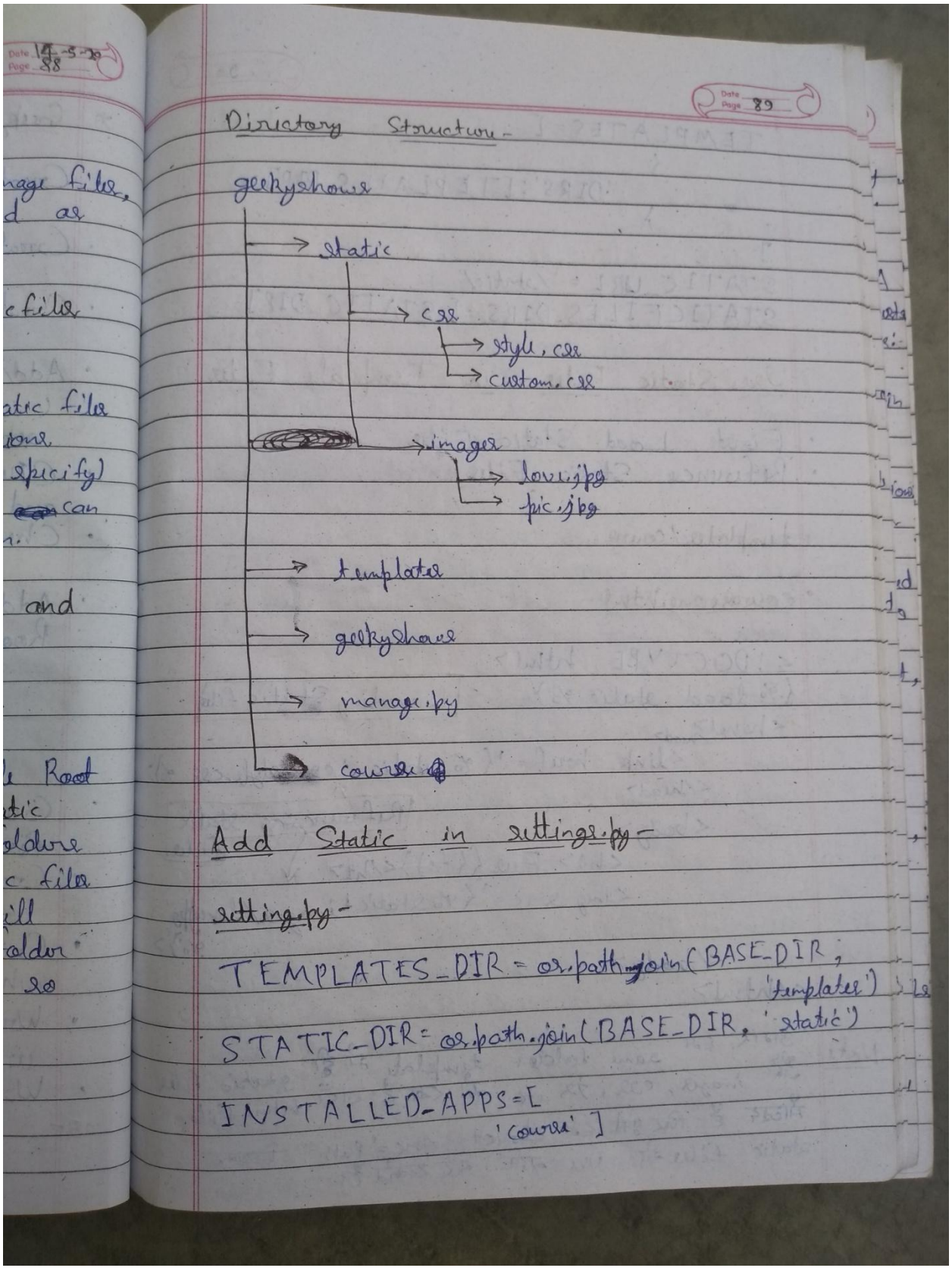
- Django provides django.contrib.staticfiles to help you manage them.

- django.contrib.staticfiles collects static files from each of your applications (and any other places you specify) into a single location that can easily be served in production.

How to Create Static Folder and Files-

- **Static File inside Project -**

We create static folder inside Root Project Folder then inside static folder we create required foldors which will contain all static files respectively like css folder will contain all css files, image folder will contain all images and so on.



Diractory Structure -

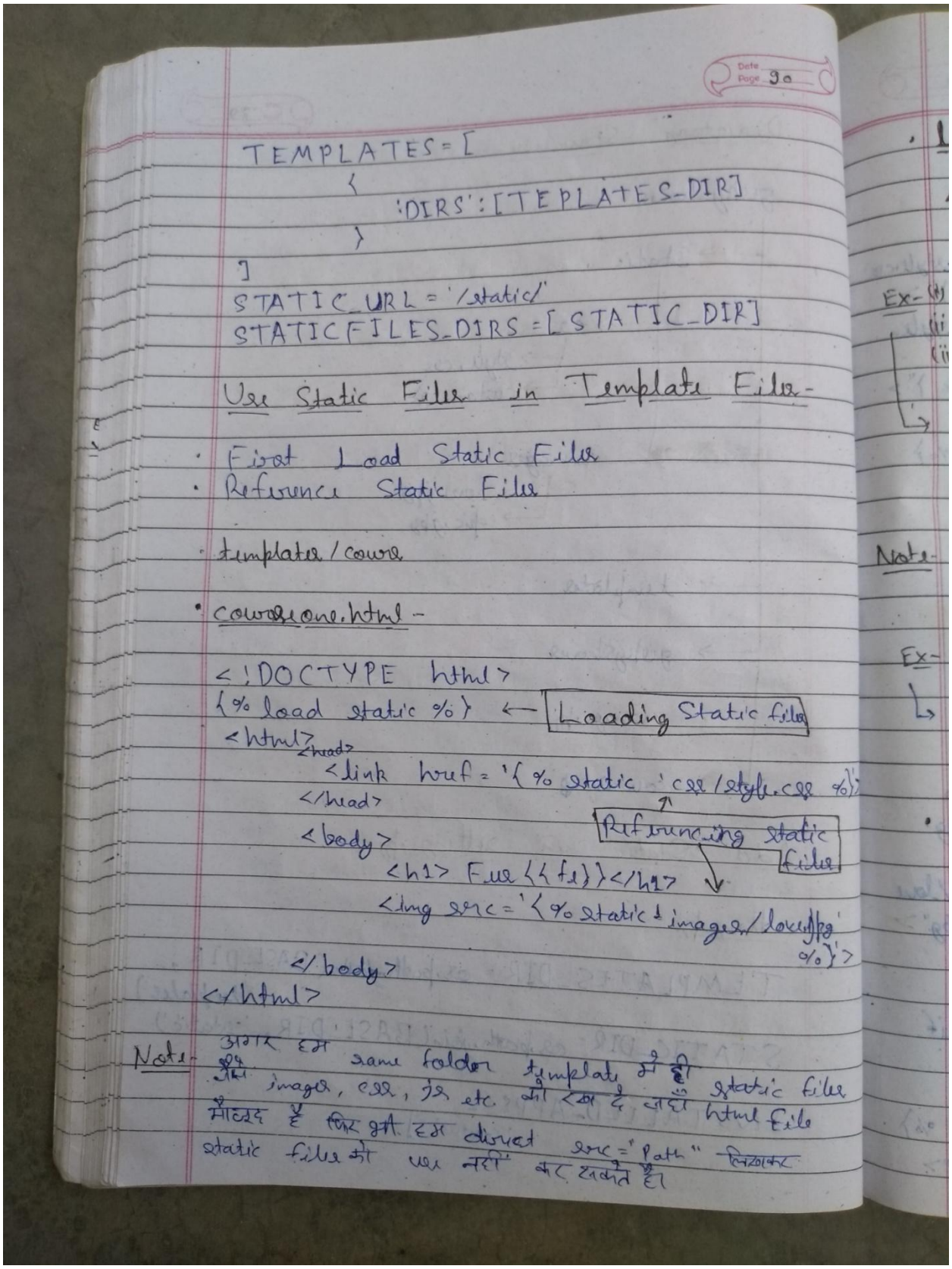
geekyshows

- static
  - css
    - style.css
    - custom.css
  - images
    - love.jpg
    - pic.jpg
- templates
- geekyshows
- manage.py
- course

Add Static in settings.py -

setting.py -

```
TEMPLATES_DIR = os.path.join(BASE_DIR, 'templates')
STATIC_DIR = os.path.join(BASE_DIR, 'static')
INSTALLED_APPS=[
    'course' ]
```



```
TEMPLATES = [
        {
            'DIRS': [TEPLATES_DIR]
        }
]
STATIC_URL = '/static/'
STATICFILES_DIRS = [STATIC_DIR]
```

Use Static Files in Template File-

- First Load Static Files
- Reference Static Files

- templates / course

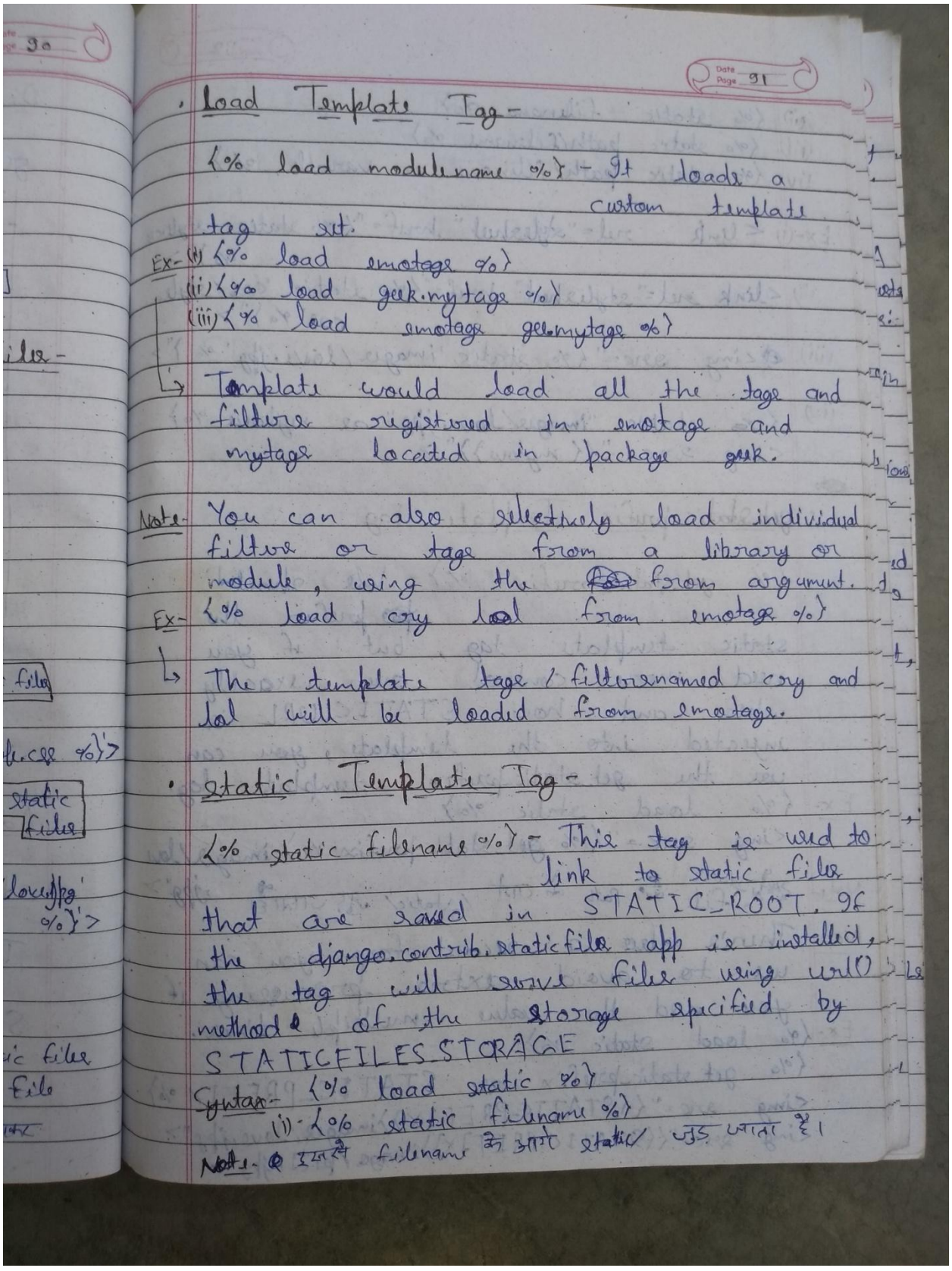
- courseone.html -

```
<!DOCTYPE html>
{% load static %}
<html>
    <head>
        <link href='{% static 'css/style.css' %}'>
    </head>
    <body>
        <h1> Fee {{ fee }} </h1>
        <img src='{% static 'images/love.jpg' %}'>
    </body>
</html>
```

Loading Static files

Referencing static files

Note- अगर हम same folder template में ही static files जैसे images, css, js etc. को रखा है जहाँ html file मौजूद है फिर भी हम direct src="Path" लिखकर static files को use नहीं कर सकते है



• Load Template Tag -

{% load modulename %} - It loads a custom template tag set.

Ex- (i) {% load emotags %}
(ii) {% load geek.mytags %}
(iii) {% load emotags geek.mytags %}

↳ Template would load all the tags and filters registered in emotags and mytags located in package geek.

Note- You can also selectively load individual filter or tags from a library or module, using the from argument.

Ex- {% load cry lol from emotags %}

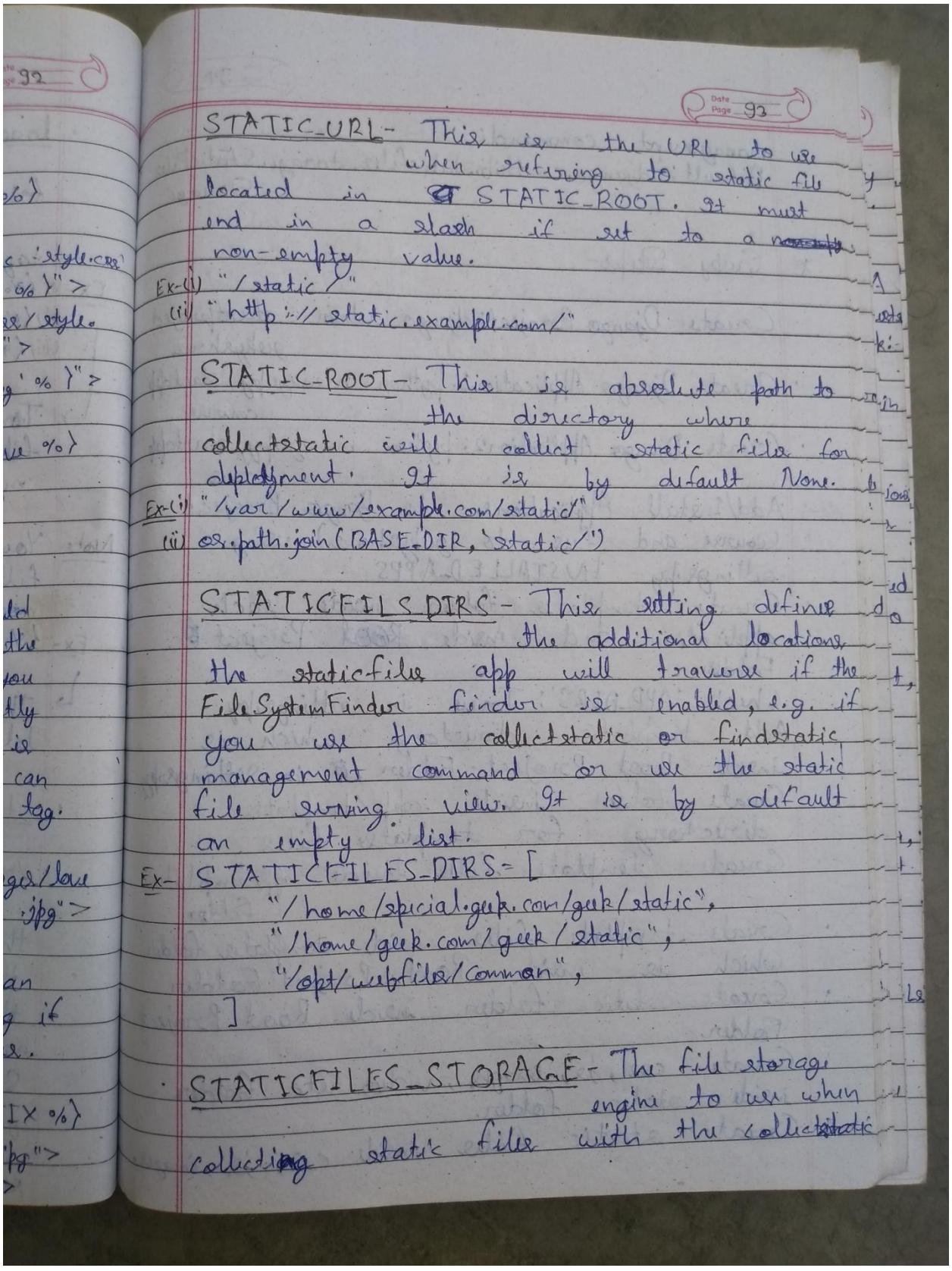
↳ The template tags / filters named cry and lol will be loaded from emotags.

• static Template Tag -

{% static filename %} - This tag is used to link to static file that are saved in STATIC_ROOT. If the django.contrib.staticfiles app is installed, the tag will serve files using url() method of the storage specified by STATICFILES_STORAGE

Syntax- {% load static %}
(i) - {% static filename %}

Note- @ इसमें filename के आगे static/ जुड़ जाता है।



STATIC_URL - This is the URL to use when referring to static file located in STATIC_ROOT. It must end in a slash if set to a non-empty value.

Ex-(i) "/static/"

(ii) "http://static.example.com/"

STATIC_ROOT - This is absolute path to the directory where collectstatic will collect static files for deployment. It is by default None.

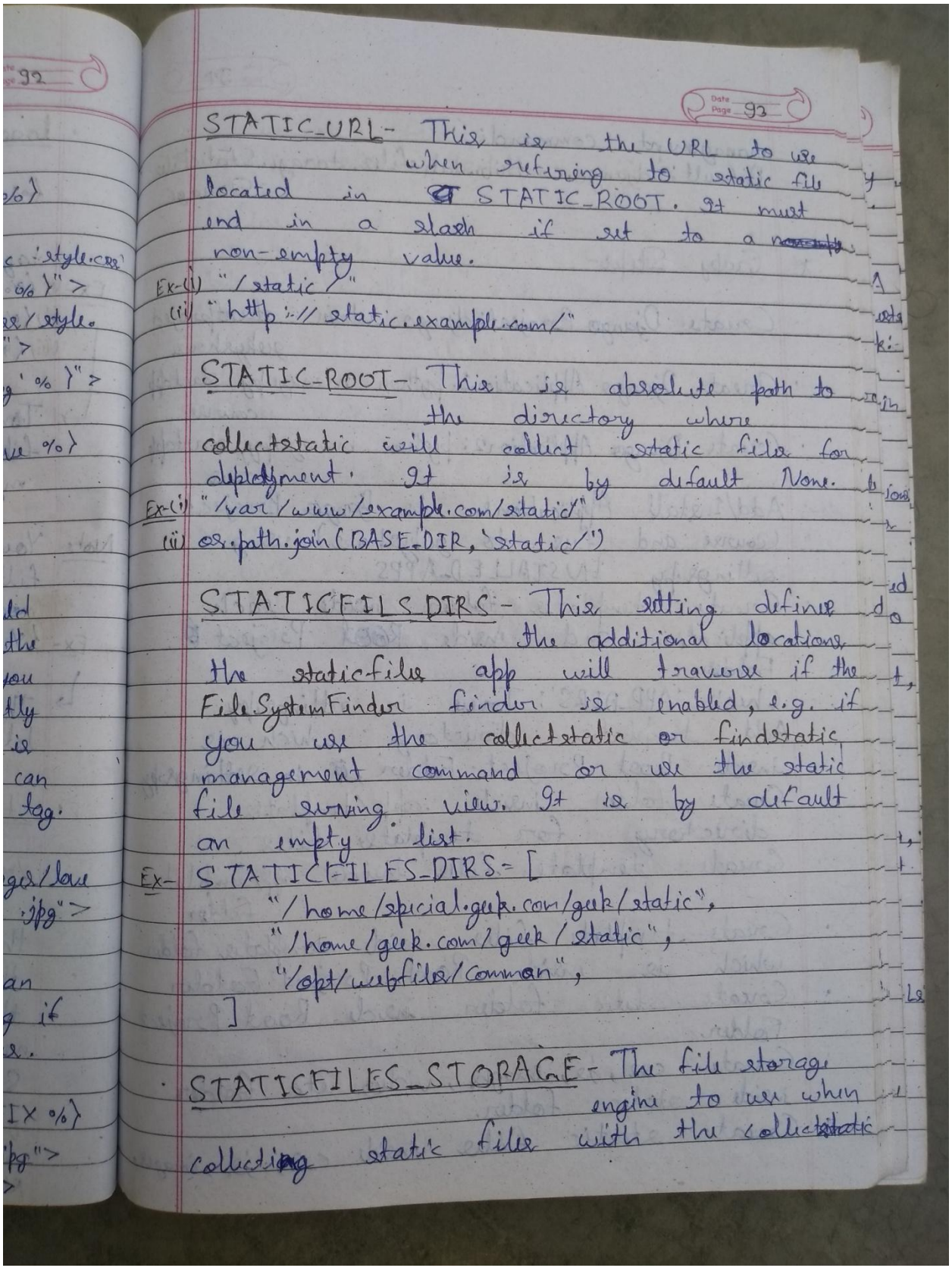
Ex-(i) "/var/www/example.com/static/"

(ii) os.path.join(BASE_DIR, 'static/')

STATICFILES_DIRS - This setting define the additional locations the staticfiles app will traverse if the FileSystemFinder finder is enabled, e.g. if you use the collectstatic or findstatic management command or use the static file serving view. It is by default an empty list.

Ex- STATICFILES_DIRS = [
"/home/special.geek.com/geek/static",
"/home/geek.com/geek/static",
"/opt/webfiles/common",
]

STATICFILES_STORAGE - The file storage engine to use when collecting static files with the collectstatic



STATIC_URL - This is the URL to use when referring to static file located in STATIC_ROOT. It must end in a slash if set to a non-empty value.

Ex-(i) "/static/"

(ii) "http://static.example.com/"

STATIC_ROOT - This is absolute path to the directory where collectstatic will collect static files for deployment. It is by default None.

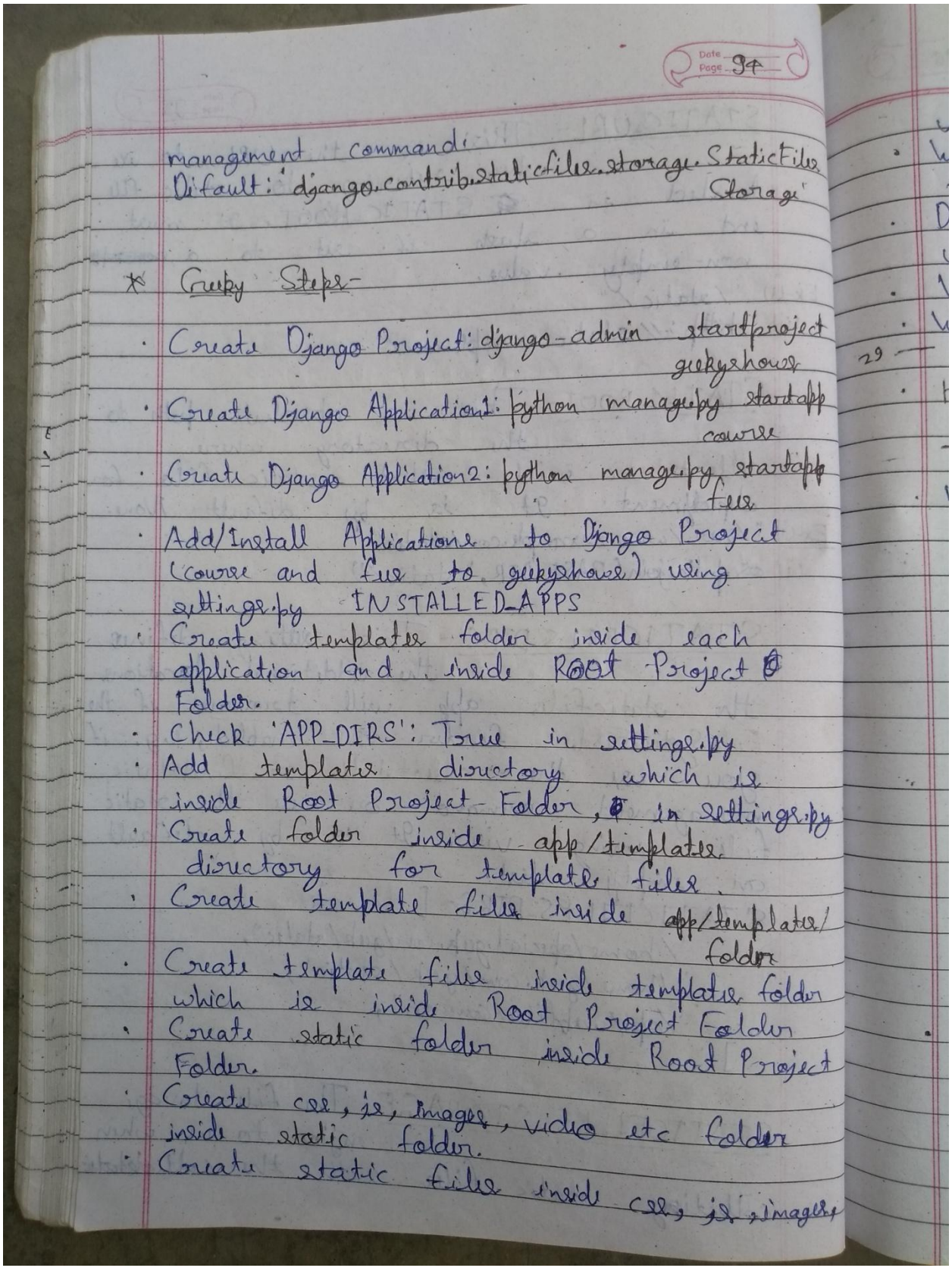
Ex-(i) "/var/www/example.com/static/"

(ii) os.path.join(BASE_DIR, 'static/')

STATICFILES_DIRS - This setting defines the additional locations the staticfiles app will traverse if the FileSystemFinder finder is enabled, e.g. if you use the collectstatic or findstatic management command or use the static file serving view. It is by default an empty list.

Ex- STATICFILES_DIRS = [
"/home/special.geek.com/geek/static",
"/home/geek.com/geek/static",
"/opt/webfiles/common",
]

STATICFILES_STORAGE - The file storage engine to use when collecting static files with the collectstatic



management command.
Default: django.contrib.staticfiles.storage.StaticFilesStorage

* 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 templates folder inside each application and inside Root Project Folder.
- Check 'APP_DIRS': True in settings.py
- Add template directory which is inside Root Project Folder, in settings.py
- Create folder inside app/templates directory for template files.
- Create template files inside app/templates/ folder
- Create template files inside templates folder which is inside Root Project Folder
- Create static folder inside Root Project Folder.
- Create css, js, images, video etc folder inside static folder.
- Create static files inside css, js, images

video etc folder.

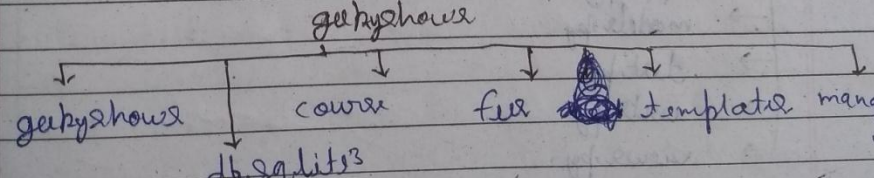
- Write View Function inside views.py file.
- Define url view function of application using urls.py file.
- Write Template file code.
- Write Static file code.

29 —

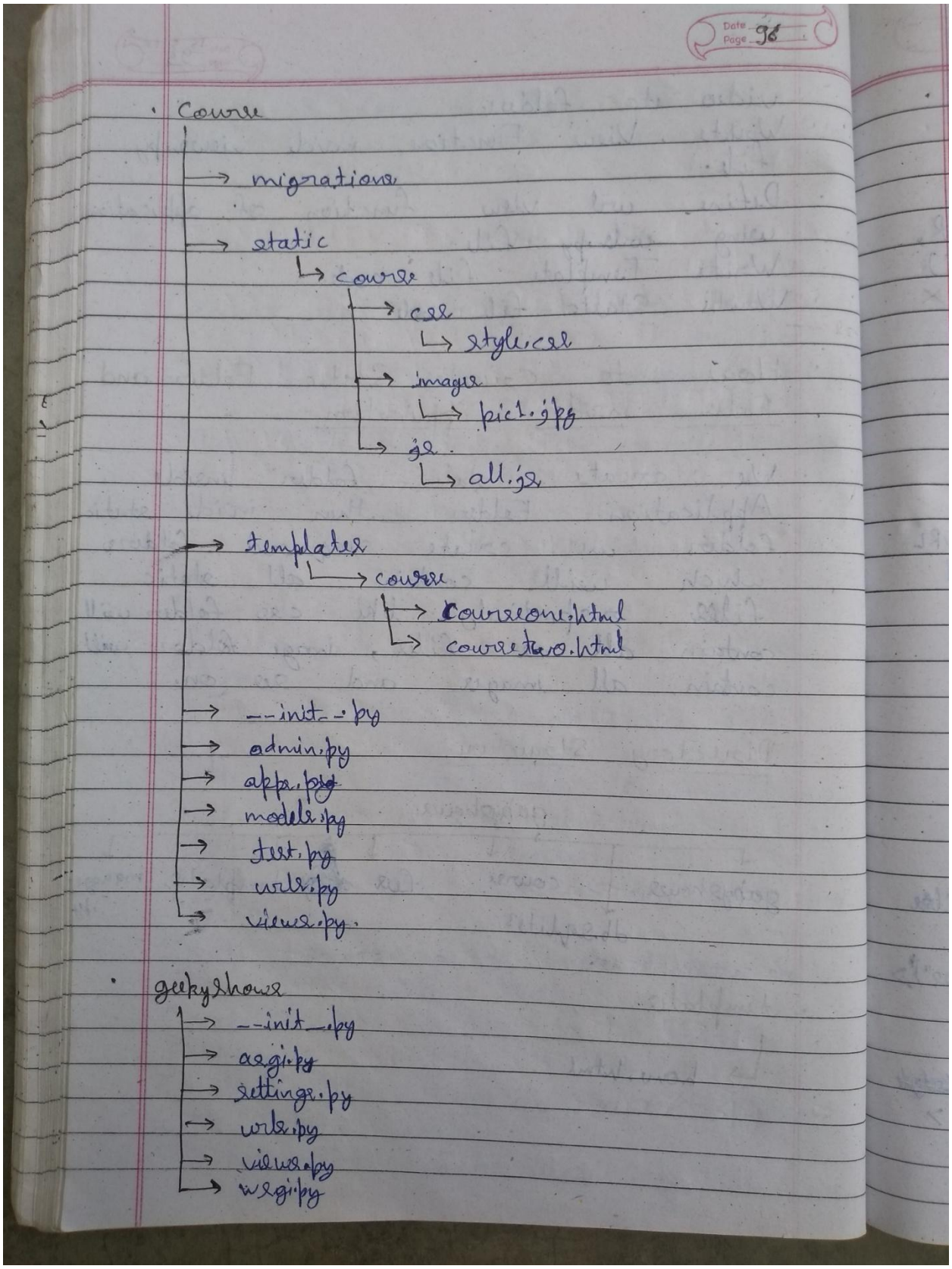
- How to Create Static Folder and Files inside Application -

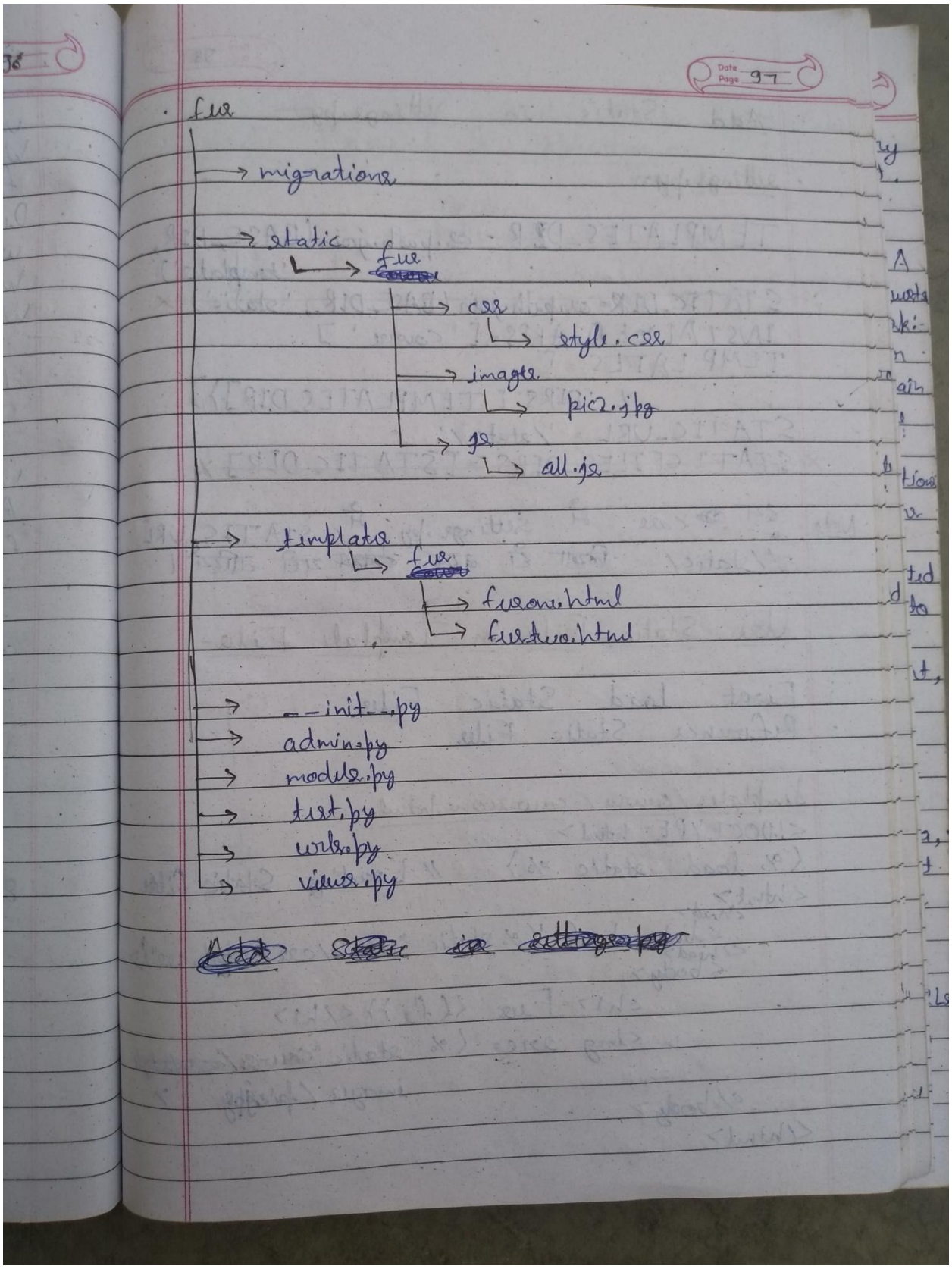
We create static folder inside Application Folder then inside static folder we create required folders which will contain all static files respectively like css folder will contain all css files, image folder will contain all images and so on.

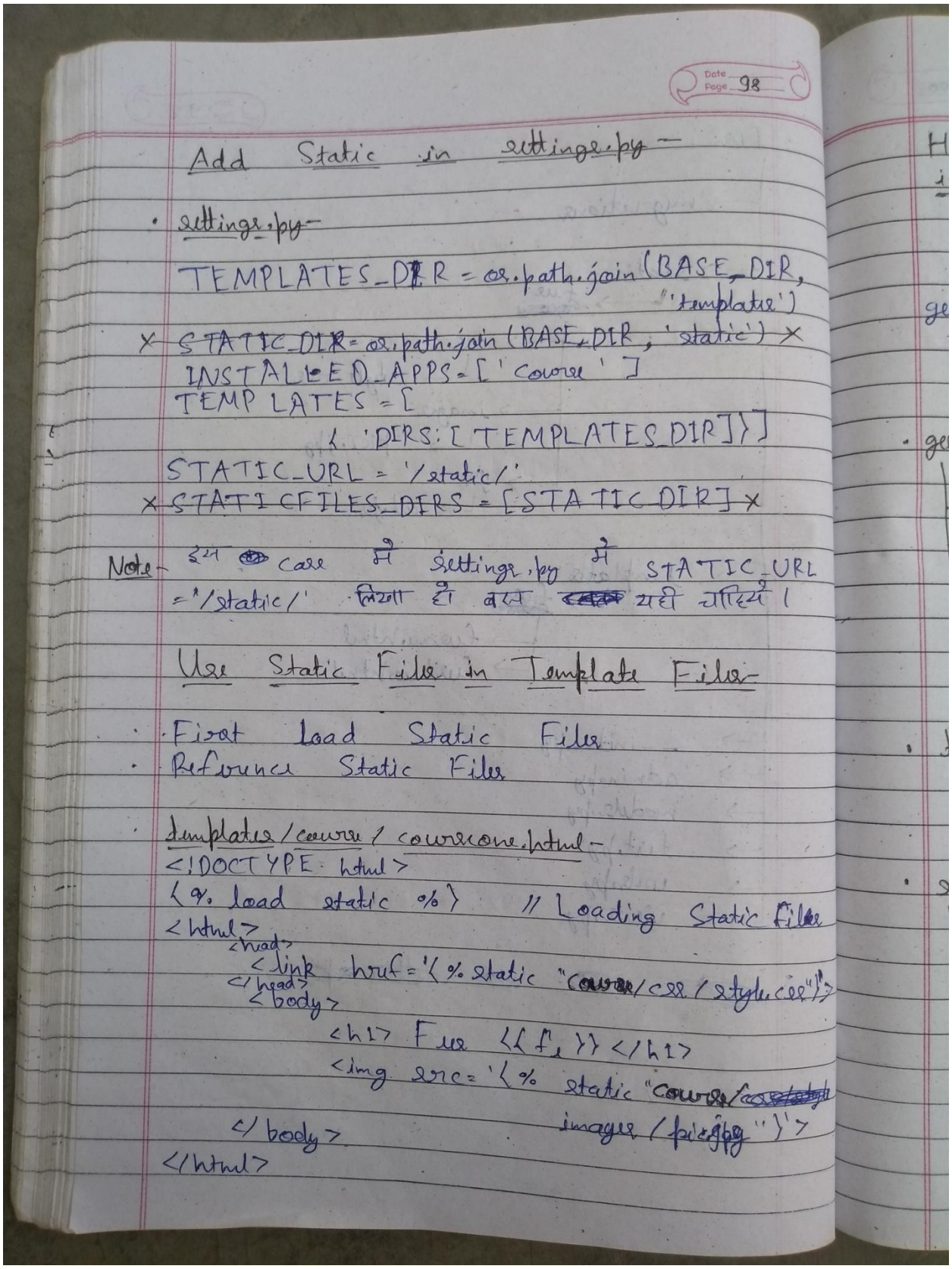
- Directory Structure -



- templates
  - home.html


Date
Page 96
Course
migrations
static
course
css
style.css
images
pic1.jpg
js
all.js
templates
course
courseone.html
coursetwo.html
__init__.py
admin.py
apps.py
models.py
test.py
urls.py
views.py
geekyshows
__init__.py
asgi.py
settings.py
urls.py
views.py
wsgi.py


fun
migrations
static
fun
css
style.css
images
pic2.jpg
js
all.js
templates
fun
fun_one.html
fun_two.html
__init__.py
admin.py
models.py
test.py
urls.py
views.py



Add Static in settings.py –

- settings.py –

```
TEMPLATES_DIR = os.path.join(BASE_DIR, 'template')
STATIC_DIR = os.path.join(BASE_DIR, 'static')
INSTALLED_APPS = ['course']
TEMPLATES = [
        {'DIRS': [TEMPLATES_DIR]}]
STATIC_URL = '/static/'
STATICFILES_DIRS = [STATIC_DIR]
```

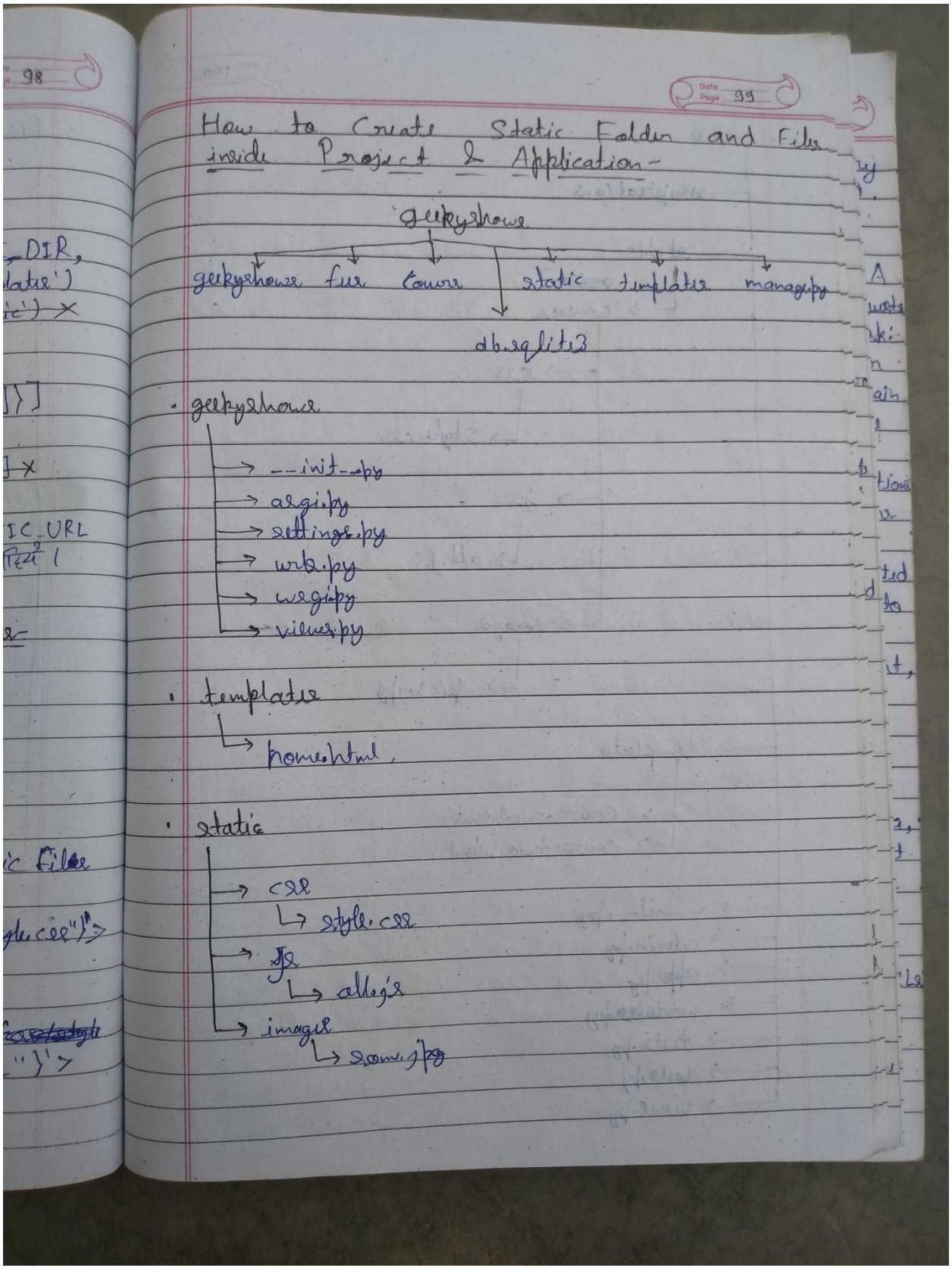
Note – इस case में settings.py में STATIC_URL = '/static/' लिखा हो तब यही चाहिये।

Use Static File in Template File –

- First load Static File
- Refrence Static File

- templates/course/courseone.html –

```
<!DOCTYPE html>
{% load static %}  // Loading Static File
<html>
    <head>
        <link href='{% static "course/css/style.css" %}'>
    </head>
    <body>
        <h1> Fee {{ fee }} </h1>
        <img src='{% static "course/images/pic.jpg" %}'>
    </body>
</html>
```

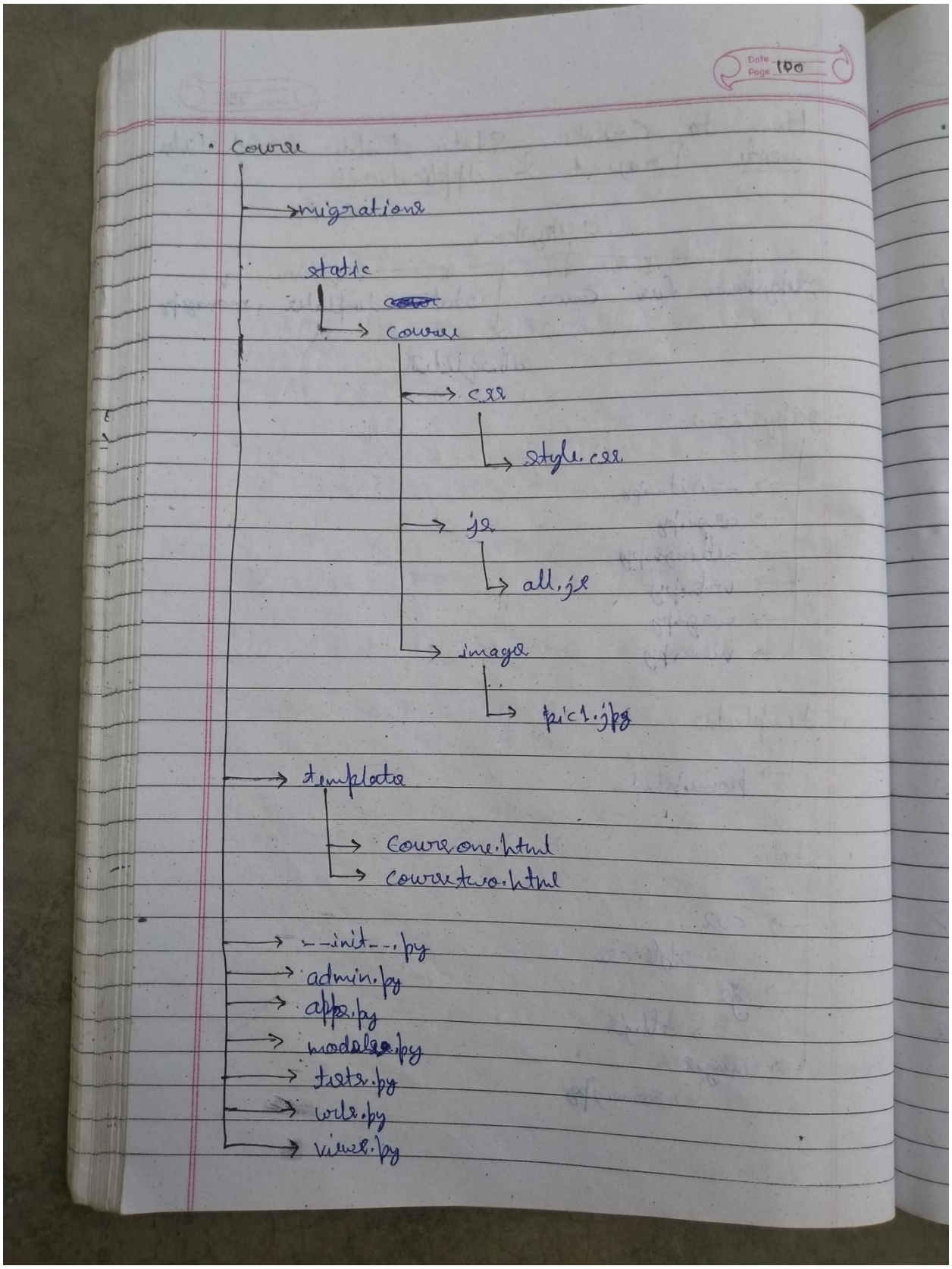


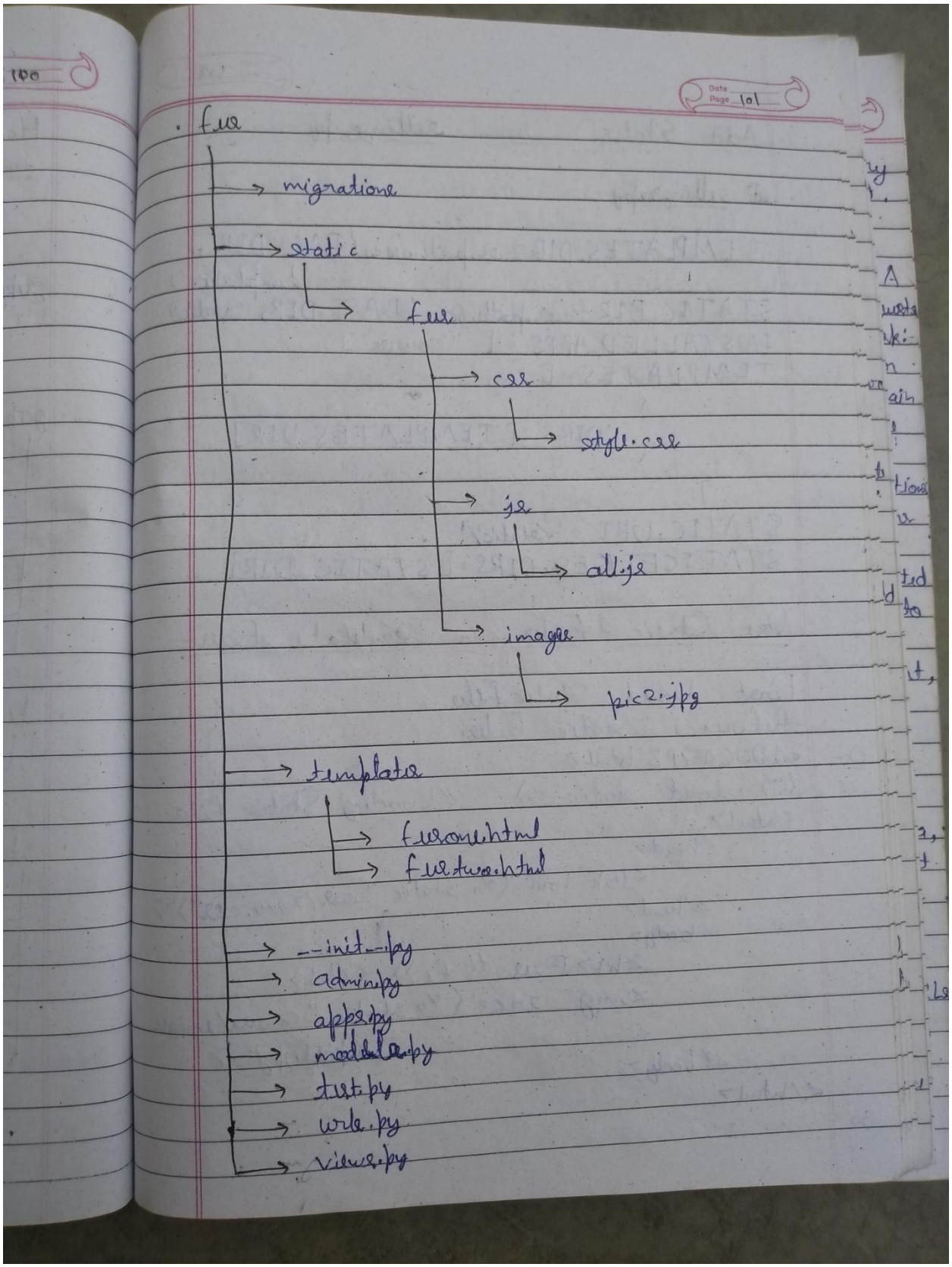
How to Create Static Folder and File inside Project & Application -

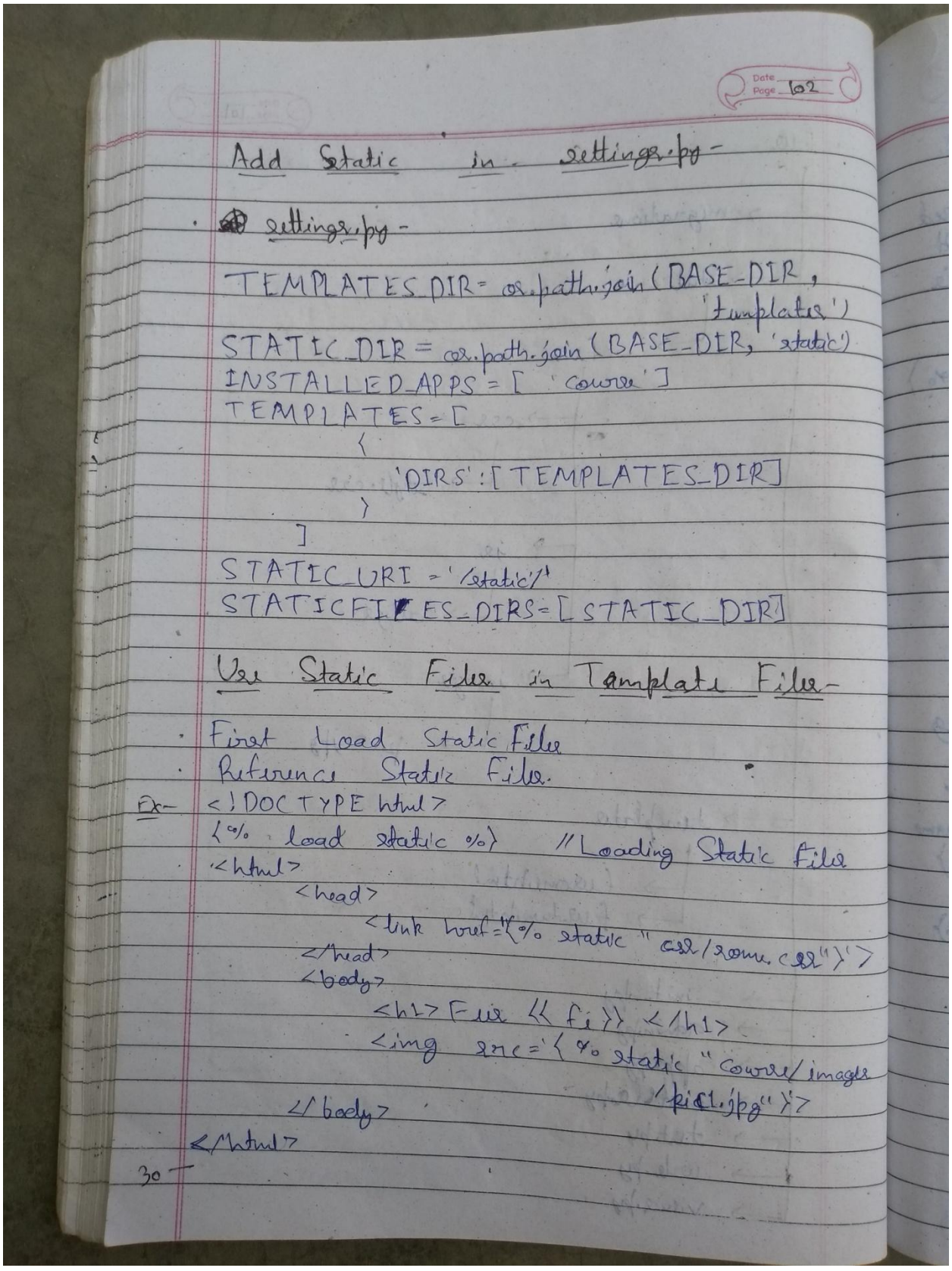
geekyshows

geekyshows | fun | Course | db.sqlite3 | static | template | manage.py

- geekyshows
  - → __init__.py
  - → asgi.py
  - → settings.py
  - → urls.py
  - → wsgi.py
  - → views.py
- template
  - → home.html
- static
  - → css
    - → style.css
  - → js
    - → alljs
  - → image
    - → snow.jpg


Date
Page 100
Course
migrations
static
Course
css
style.css
js
all.js
image
pic1.jpg
template
Course.one.html
Course.two.html
__init__.py
admin.py
apps.py
models.py
tests.py
urls.py
views.py


fun
migrations
static
fun
css
style.css
js
all.js
images
pic2.jpg
templates
funone.html
funtwo.html
__init__.py
admin.py
apps.py
models.py
test.py
urls.py
views.py



Add Static in settings.py -

- settings.py -

```
TEMPLATES_DIR = os.path.join(BASE_DIR, 'templates')
STATIC_DIR = os.path.join(BASE_DIR, 'static')
INSTALLED_APPS = [ 'course' ]
TEMPLATES = [
        {
            'DIRS': [TEMPLATES_DIR]
        }
    ]
STATIC_URI = '/static/'
STATICFILES_DIRS = [STATIC_DIR]
```

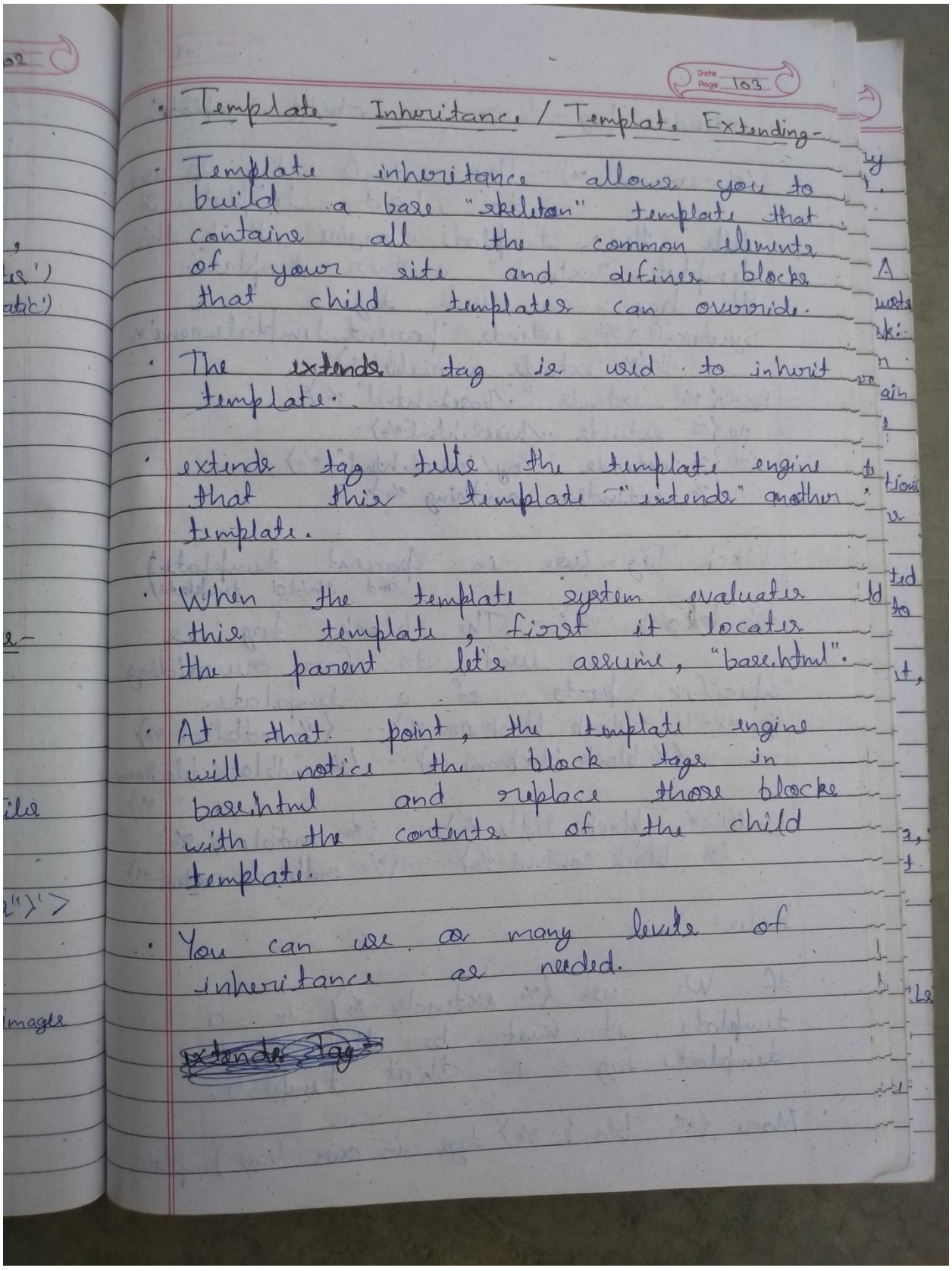
Use Static Files in Template File -

- First Load Static File
- Reference Static File.

Ex-

```
<!DOCTYPE html>
{% load static %}   // Loading Static File
<html>
    <head>
        <link href="{% static "css/course.css" %}">
    </head>
    <body>
        <h1> File {{ fi }} </h1>
        <img src="{% static "course/images/pic1.jpg" %}">
    </body>
</html>
```

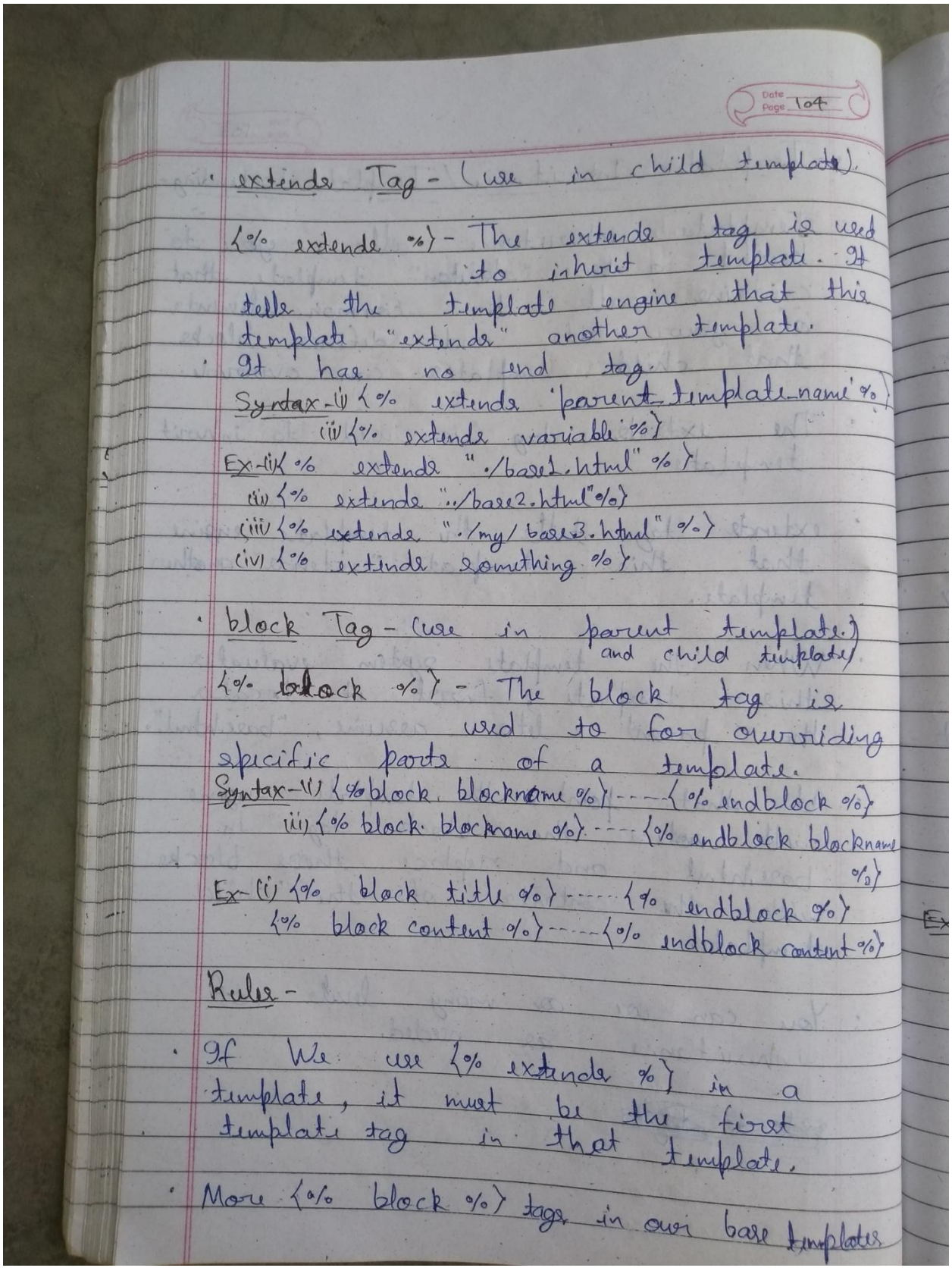
30 —



Template Inheritance / Template Extending -

- Template inheritance allows you to build a base "skeleton" template that contains all the common elements of your site and defines blocks that child templates can override.
- The extends tag is used to inherit template.
- extends tag tells the template engine that this template "extends" another template.
- When the template system evaluates this template, first it locates the parent let's assume, "base.html".
- At that point, the template engine will notice the block tags in base.html and replace those blocks with the contents of the child template.
- You can use as many levels of inheritance as needed.

extends tag



Date Page 104

- extends Tag - (use in child template).

{% extends %} - The extends tag is used to inherit template. It tells the template engine that this template "extends" another template.

- It has no end tag.

Syntax - (i) {% extends 'parent_template_name' %}
(ii) {% extends variable %}

Ex - (i) {% extends "./base1.html" %}
(ii) {% extends "../base2.html" %}
(iii) {% extends "./my/base3.html" %}
(iv) {% extends something %}

- block Tag - (use in parent template and child template)

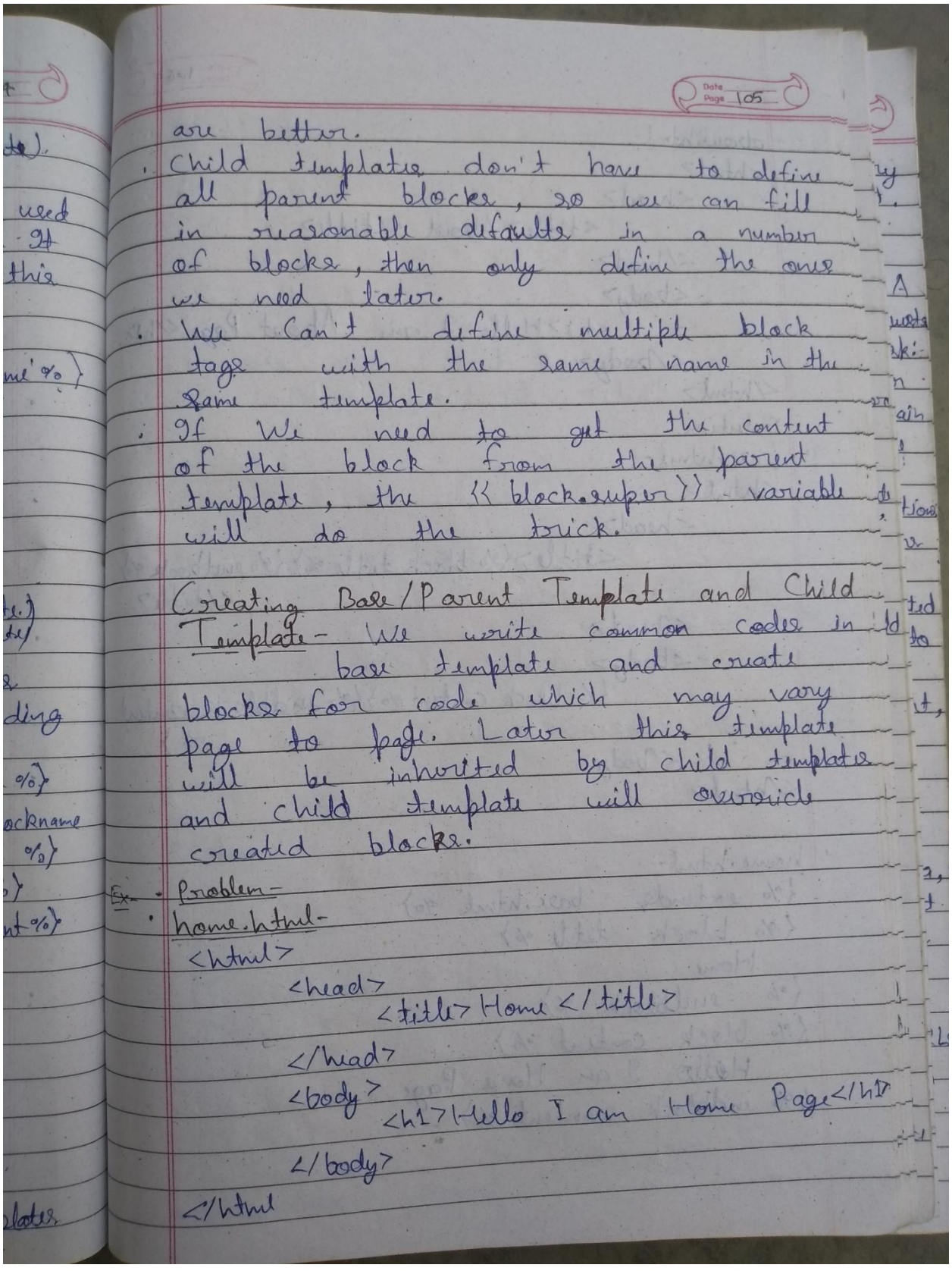
{% block %} - The block tag is used to for overriding specific parts of a template.

Syntax - (i) {% block blockname %} ---- {% endblock %}
(ii) {% block blockname %} ---- {% endblock blockname %}

Ex - (i) {% block title %} ---- {% endblock %}
{% block content %} ---- {% endblock content %}

Rules -

- If We use {% extends %} in a template, it must be the first template tag in that template.
- More {% block %} tags in our base templates



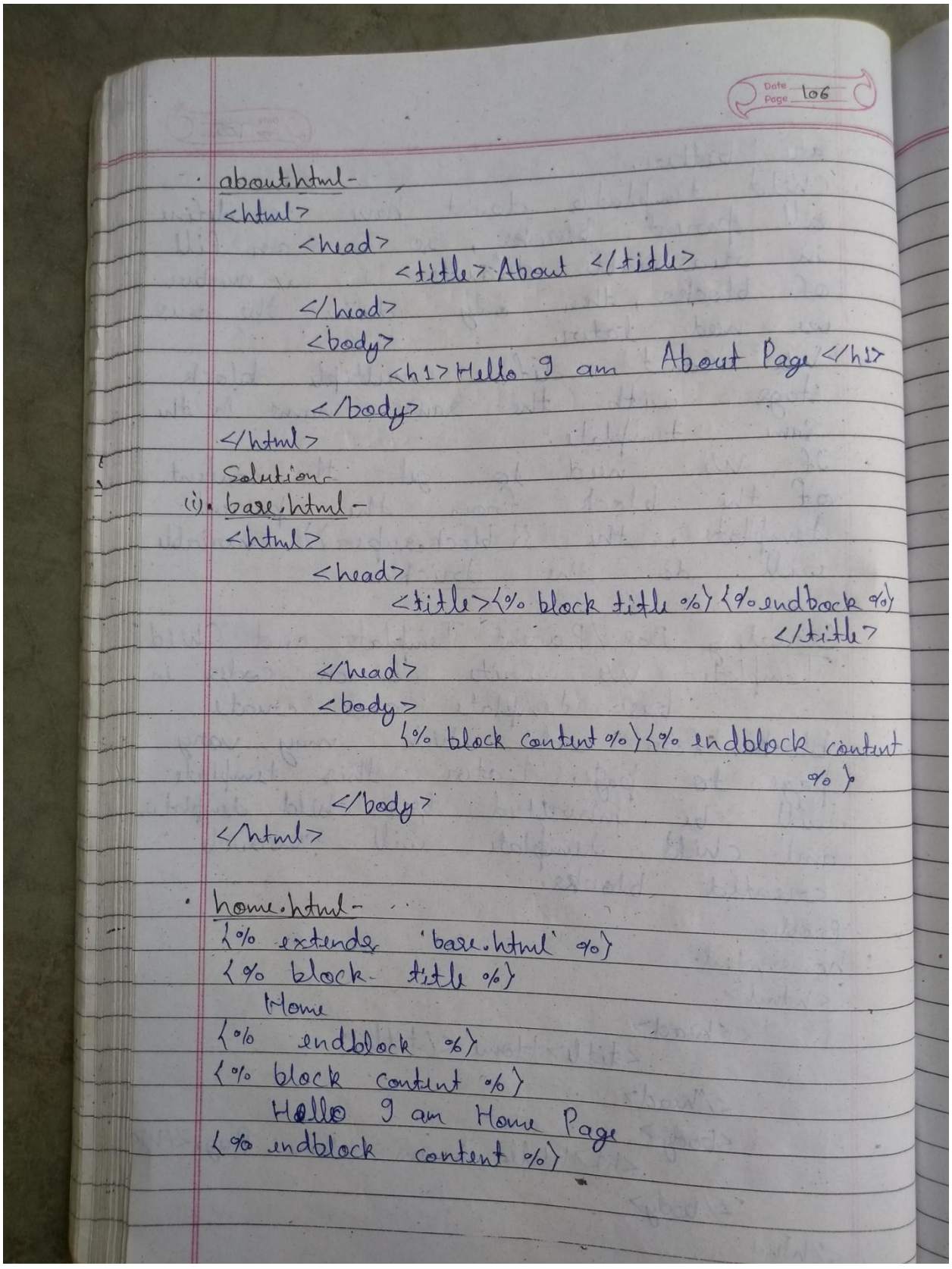
are better.
- Child templates don't have to define all parent blocks, so we can fill in reasonable defaults in a number of blocks, then only define the ones we need later.
- We can't define multiple block tags with the same name in the same template.
- If We need to get the content of the block from the parent template, the {{ block.super }} variable will do the trick.

Creating Base/Parent Template and Child Template - We write common codes in base template and create blocks for code which may vary page to page. Later this template will be inherited by child templates and child template will override created blocks.

Ex. - Problem -

home.html -

```
<html>
    <head>
        <title> Home </title>
    </head>
    <body>
        <h1> Hello I am Home Page </h1>
    </body>
</html>
```



Date
Page 106

- about.html -

```
<html>
    <head>
        <title> About </title>
    </head>
    <body>
        <h1> Hello I am About Page </h1>
    </body>
</html>
```

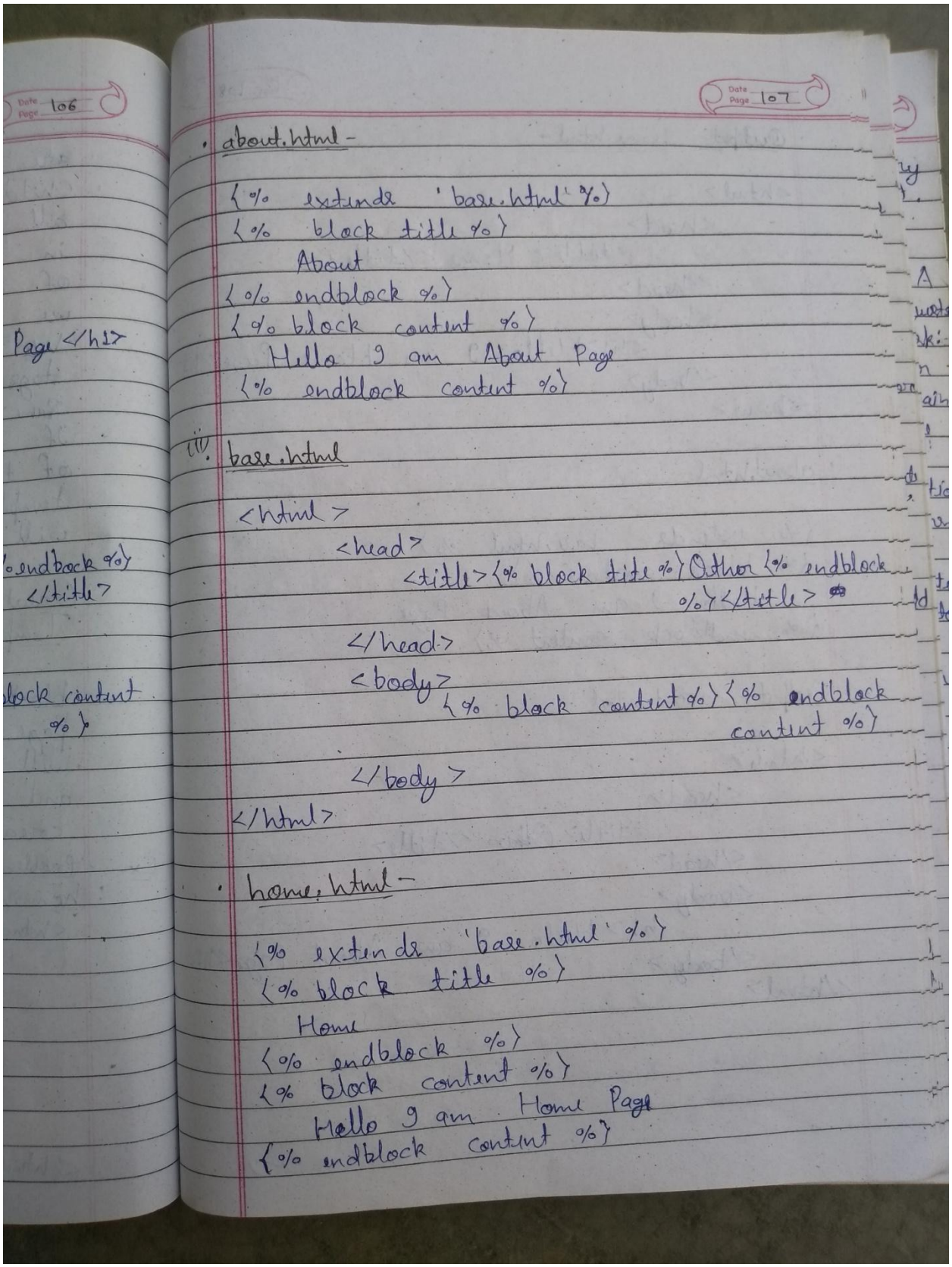
Solution -

(i). base.html -

```
<html>
    <head>
        <title>{% block title %}{% endblock %}
        </title>
    </head>
    <body>
        {% block content %}{% endblock content
        %}
    </body>
</html>
```

- home.html -

```
{% extends 'base.html' %}
{% block title %}
  Home
{% endblock %}
{% block content %}
   Hello I am Home Page
{% endblock content %}
```



• about.html -

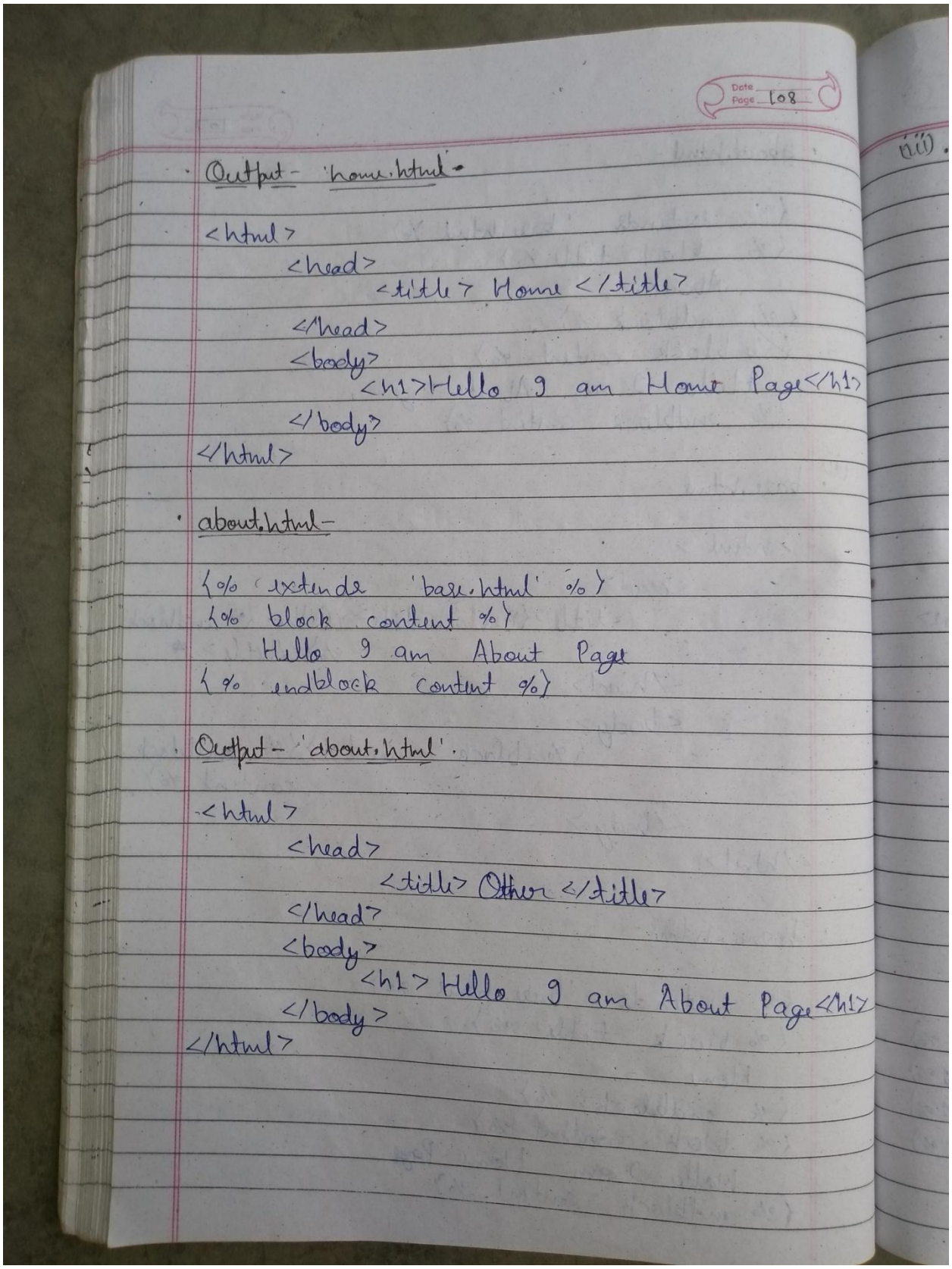
```
{% extends 'base.html' %}
{% block title %}
    About
{% endblock %}
{% block content %}
  Hello I am About Page
{% endblock content %}
```

iii. base.html

```
<html>
    <head>
        <title>{% block title %}Othon{% endblock %}</title>
    </head>
    <body>
        {% block content %}{% endblock content %}
    </body>
</html>
```

• home.html -

```
{% extends 'base.html' %}
{% block title %}
  Home
{% endblock %}
{% block content %}
  Hello I am Home Page
{% endblock content %}
```



Date
Page 108

- Output - 'home.html'-

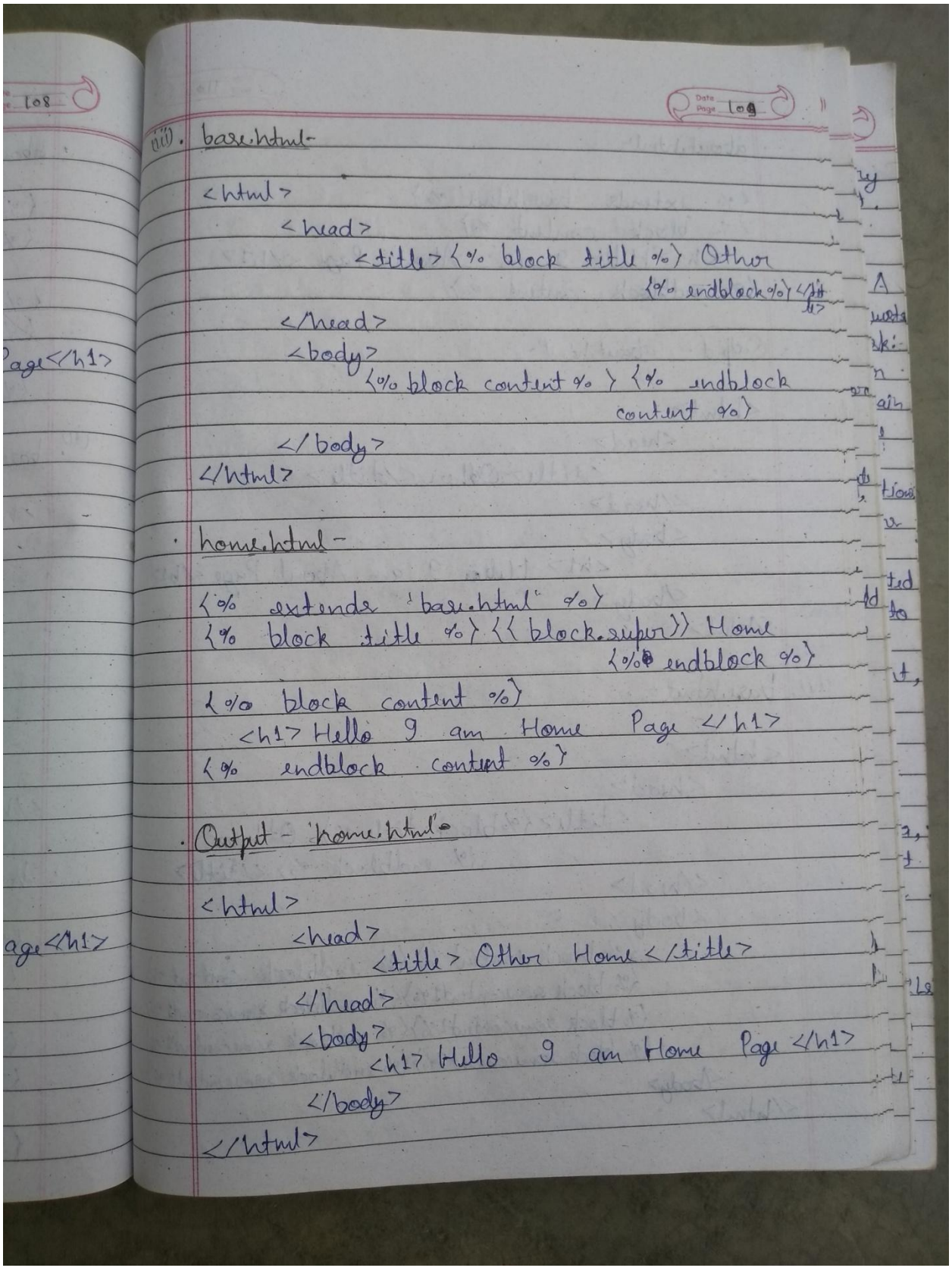
```
<html>
    <head>
        <title> Home </title>
    </head>
    <body>
        <h1>Hello I am Home Page</h1>
    </body>
</html>
```

- about.html-

```
{% extends 'base.html' %}
{% block content %}
    Hello I am About Page
{% endblock content %}
```

- Output - 'about.html'.

```
<html>
    <head>
        <title> Other </title>
    </head>
    <body>
        <h1> Hello I am About Page</h1>
    </body>
</html>
```



(iii). base.html-

```
<html>
    <head>
        <title>{% block title %} Other
                                {% endblock %}</title>
    </head>
    <body>
        {% block content %} {% endblock
                                content %}
    </body>
</html>
```

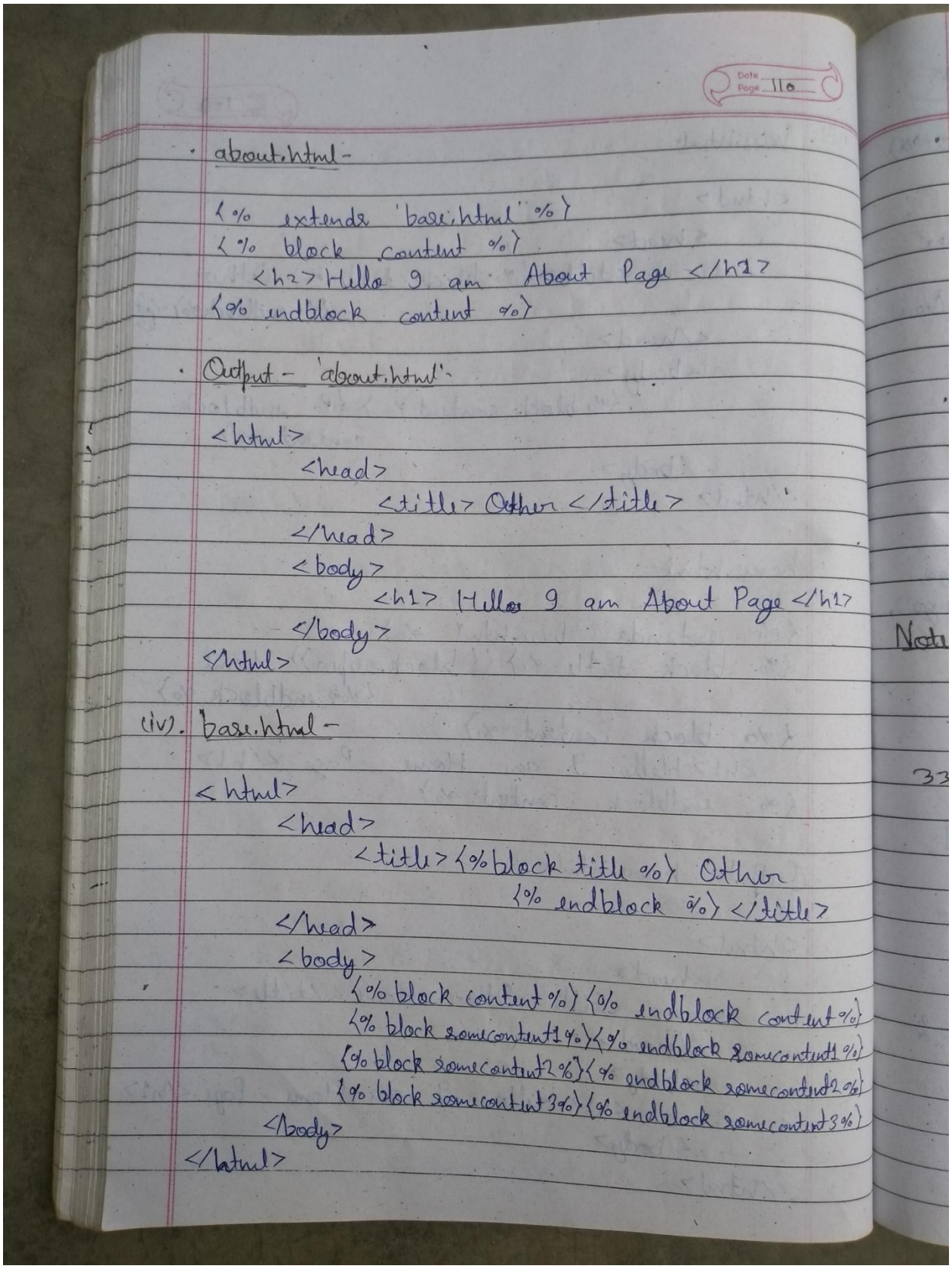
- home.html -

```
{% extends 'base.html' %}
{% block title %} {{ block.super }} Home
                                {% endblock %}

{% block content %}
  <h1> Hello I am Home Page </h1>
{% endblock content %}
```

- Output - 'home.html'

```
<html>
    <head>
        <title> Other Home </title>
    </head>
    <body>
        <h1> Hello I am Home Page </h1>
    </body>
</html>
```



- about.html -

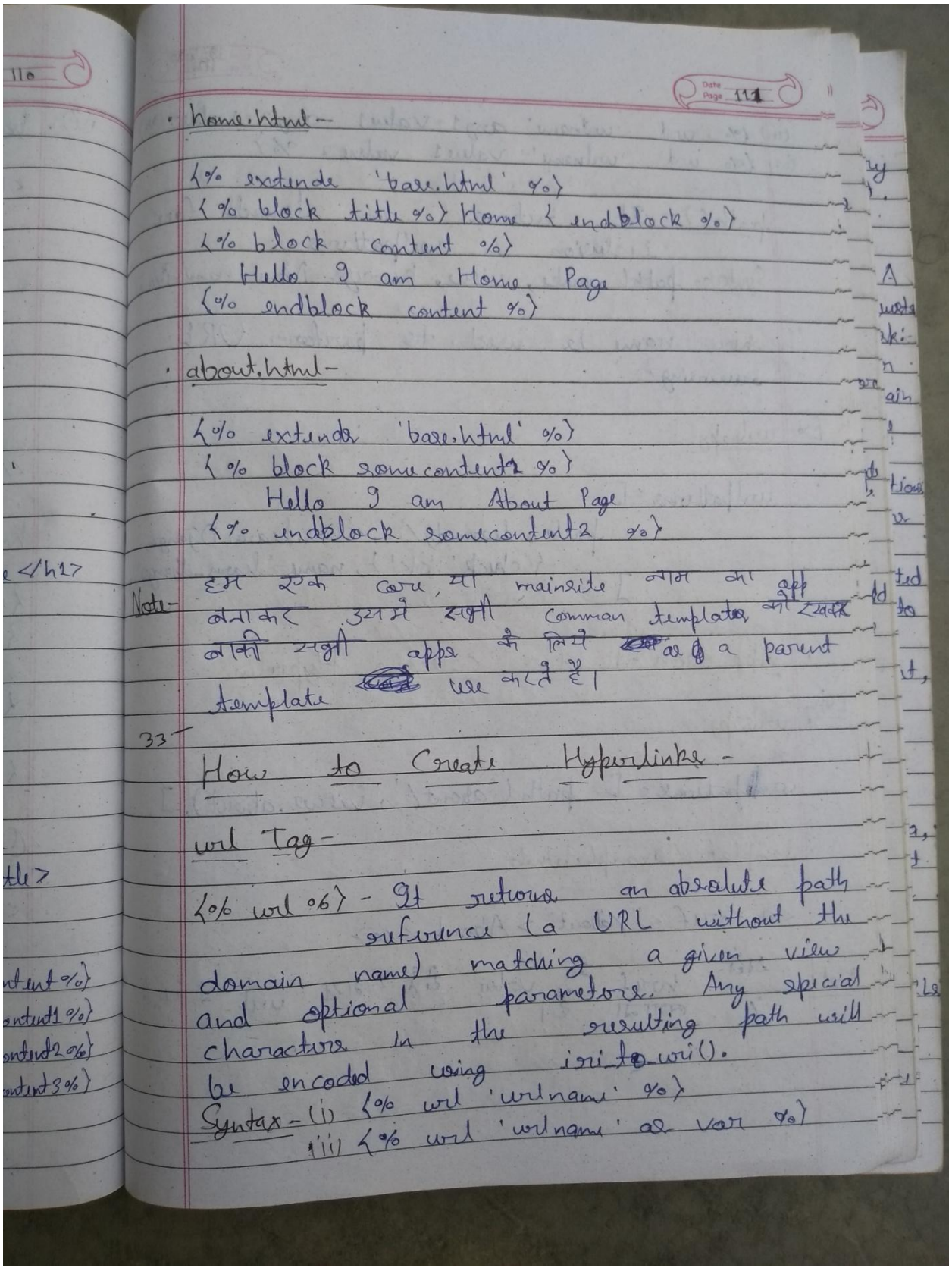
```
{% extends 'base.html' %}
{% block content %}
  <h2> Hello I am About Page </h2>
{% endblock content %}
```

- Output - 'about.html'.

```
<html>
    <head>
        <title> Other </title>
    </head>
    <body>
        <h1> Hello I am About Page </h1>
    </body>
</html>
```

(iv). base.html -

```
<html>
    <head>
        <title> {% block title %} Other
                {% endblock %} </title>
    </head>
    <body>
        {% block content %} {% endblock content %}
        {% block somecontent1 %} {% endblock somecontent1 %}
        {% block somecontent2 %} {% endblock somecontent2 %}
        {% block somecontent3 %} {% endblock somecontent3 %}
    </body>
</html>
```



· home.html -

{% extends 'base.html' %}
{% block title %} Home {% endblock %}
{% block content %}
Hello I am Home Page
{% endblock content %}

· about.html -

{% extends 'base.html' %}
{% block somecontent2 %}
Hello I am About Page
{% endblock somecontent2 %}

Note- हम एक core, या mainsite नाम का एक बनाकर उसमें सभी Comman templates को रखकर बाकी सभी apps के लिए as a parent template use करते है।

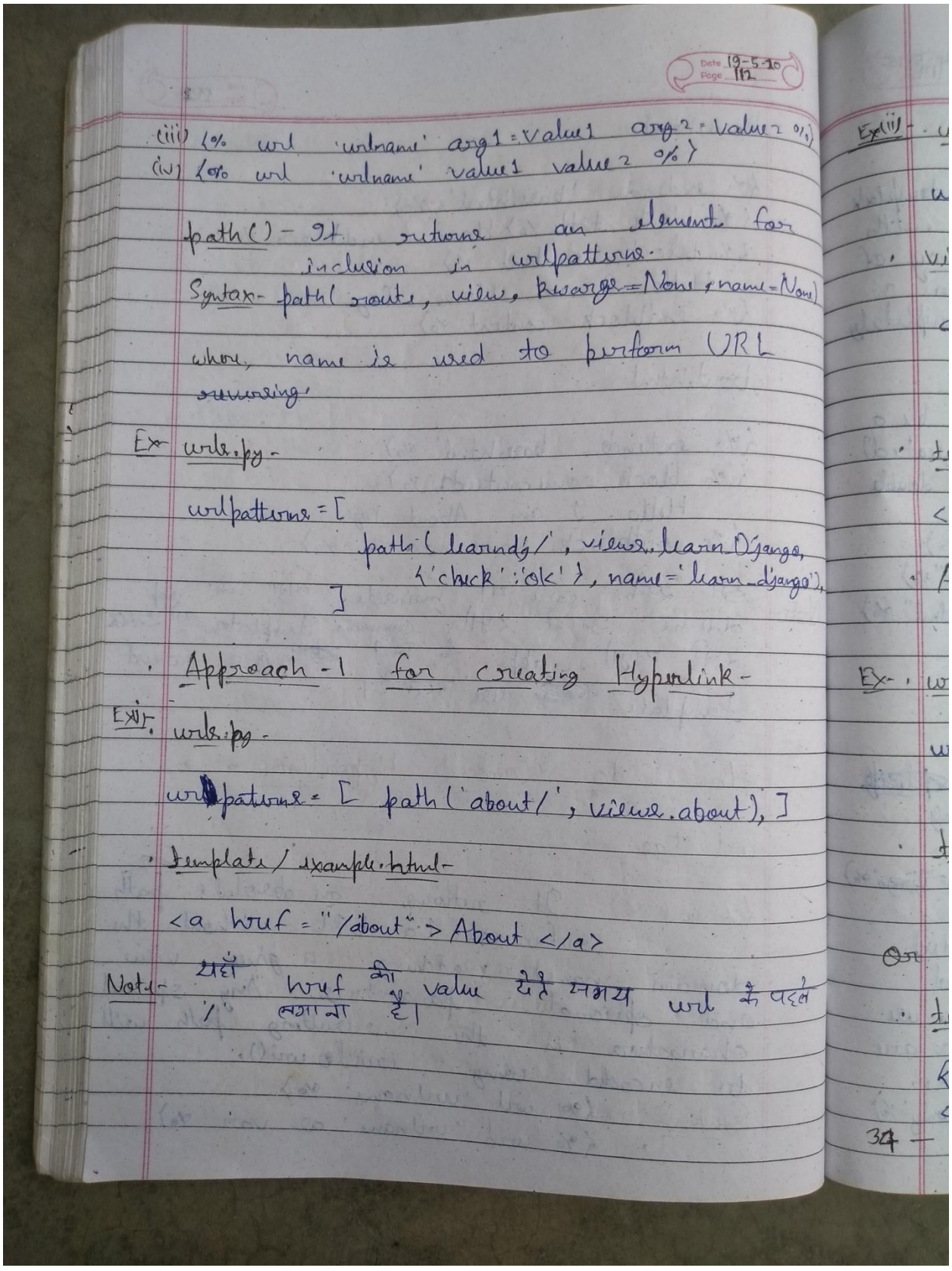
33

How to Create Hyperlinks -

url Tag -

{% url %} - It returns an absolute path reference (a URL without the domain name) matching a given view and optional parameters. Any special characters in the resulting path will be encoded using iri_to_uri().

Syntax - (i) {% url 'urlname' %}
(ii) {% url 'urlname' as var %}



(iii) {% url 'urlname' arg1=value1 arg2=value2 %}
(iv) {% url 'urlname' value1 value2 %}

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

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

where, name is used to perform URL reversing.

Ex urls.py -

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

- Approach - 1 for creating Hyperlink -

Ex: urls.py -

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

- template/example.html -

```
<a href="/about"> About </a>
```

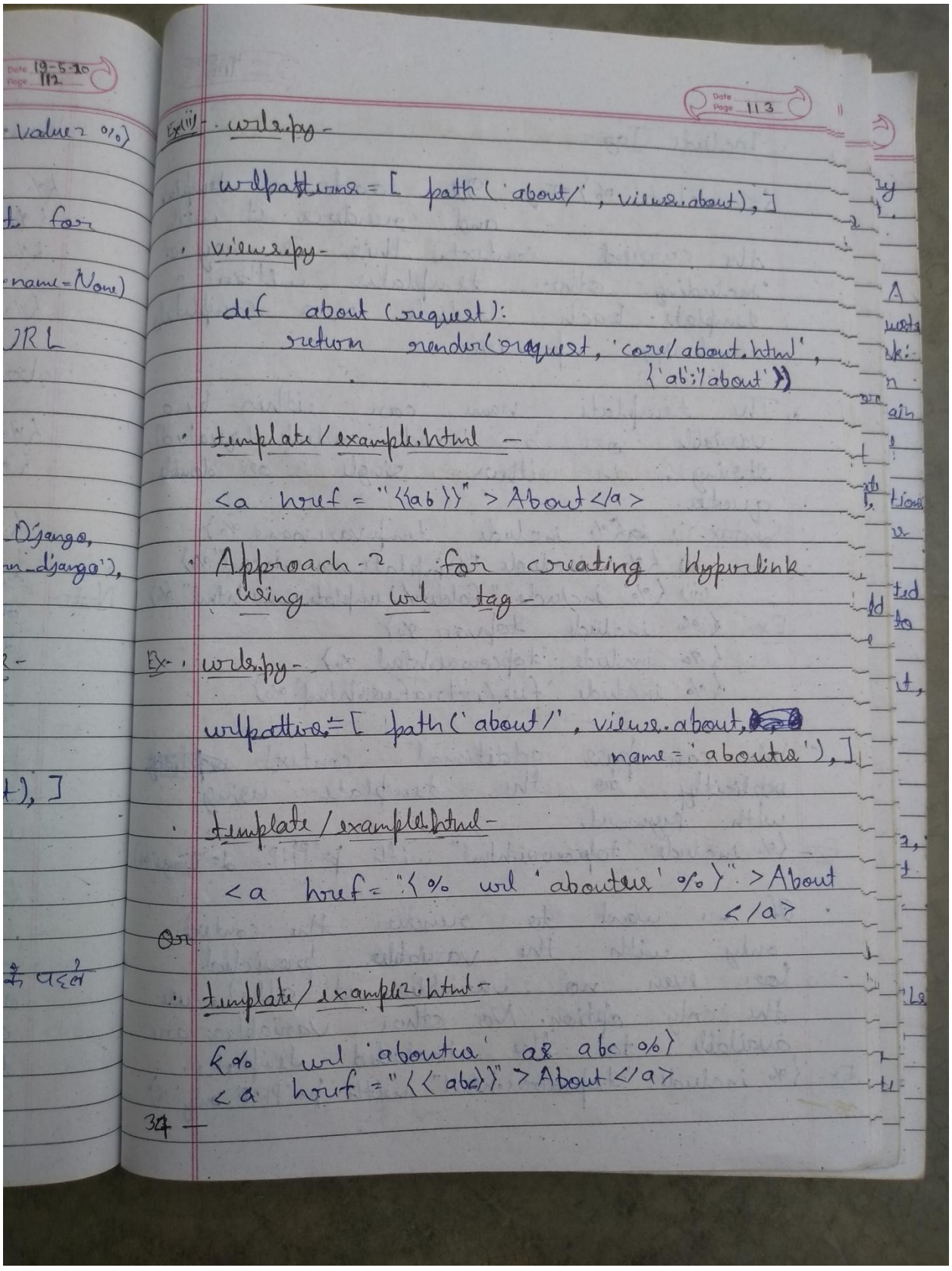
Note- यहाँ href की value देते समय url के पहले / लगाना है।

Ex(iii) -

Ex -

Or

34 -



Ex(iii) - urls.py -

urlpatterns = [path('about/', views.about),]

- views.py -

def about(request):
    return render(request, 'core/about.html', {'ab':'about'})

- template/example.html -

 About

- Approach-2 for creating Hyperlink using url tag -

Ex - urls.py -

urlpatterns = [path('about/', views.about, name = 'aboutus'),]

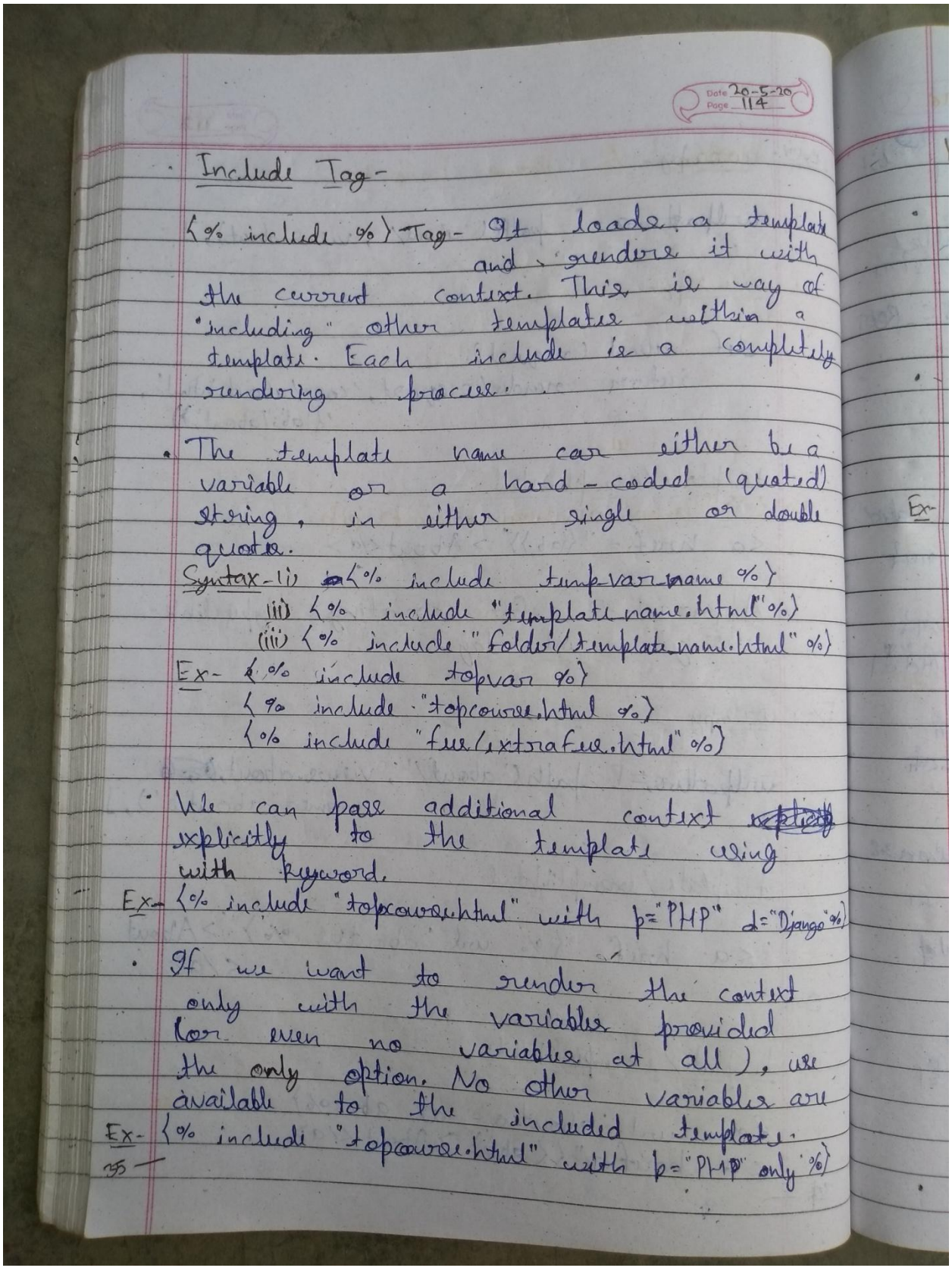
- template/example1.html -

 About

Or

- template/example2.html -

{% url 'aboutus' as abc %}
 About



Date 20-5-20
Page 114

Include Tag-

{% include %} Tag- It loads a template and renders it with the current context. This is way of "including" other templates within a template. Each include is a completely rendering process.

- The template name can either be a variable or a hard-coded (quoted) string, in either single or double quotes.

Syntax- (i) {% include temp_var_name %}
(ii) {% include "template_name.html" %}
(iii) {% include "folder/template_name.html" %}

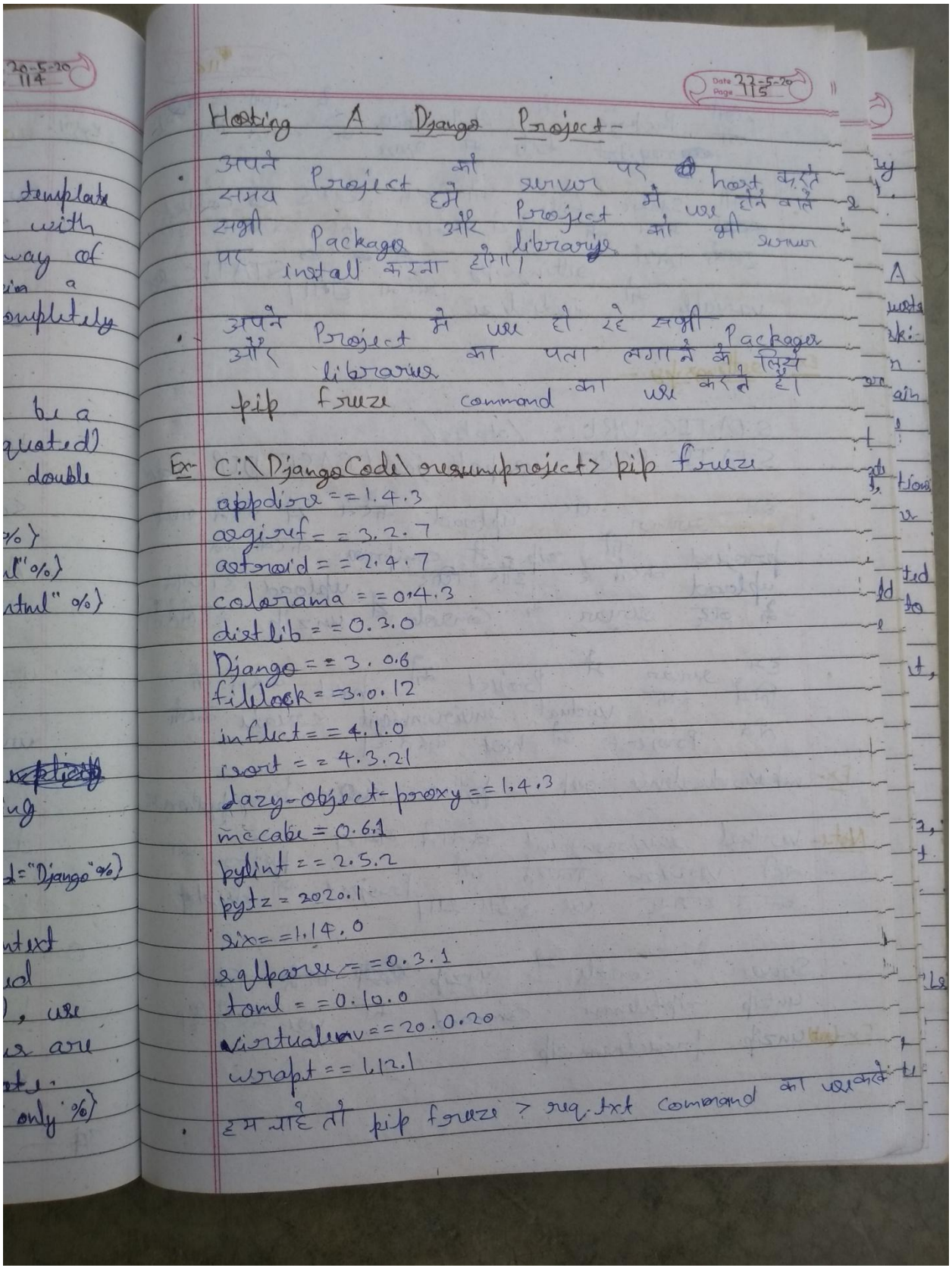
Ex- {% include topvar %}
{% include "topcourse.html %}
{% include "fee/extrafee.html" %}

- We can pass additional context explicitly to the template using with keyword.

Ex- {% include "topcourse.html" with p="PHP" d="Django" %}

- If we want to render the context only with the variables provided (or even no variables at all), use the only option. No other variables are available to the included templates.

Ex- {% include "topcourse.html" with p="PHP" only %}



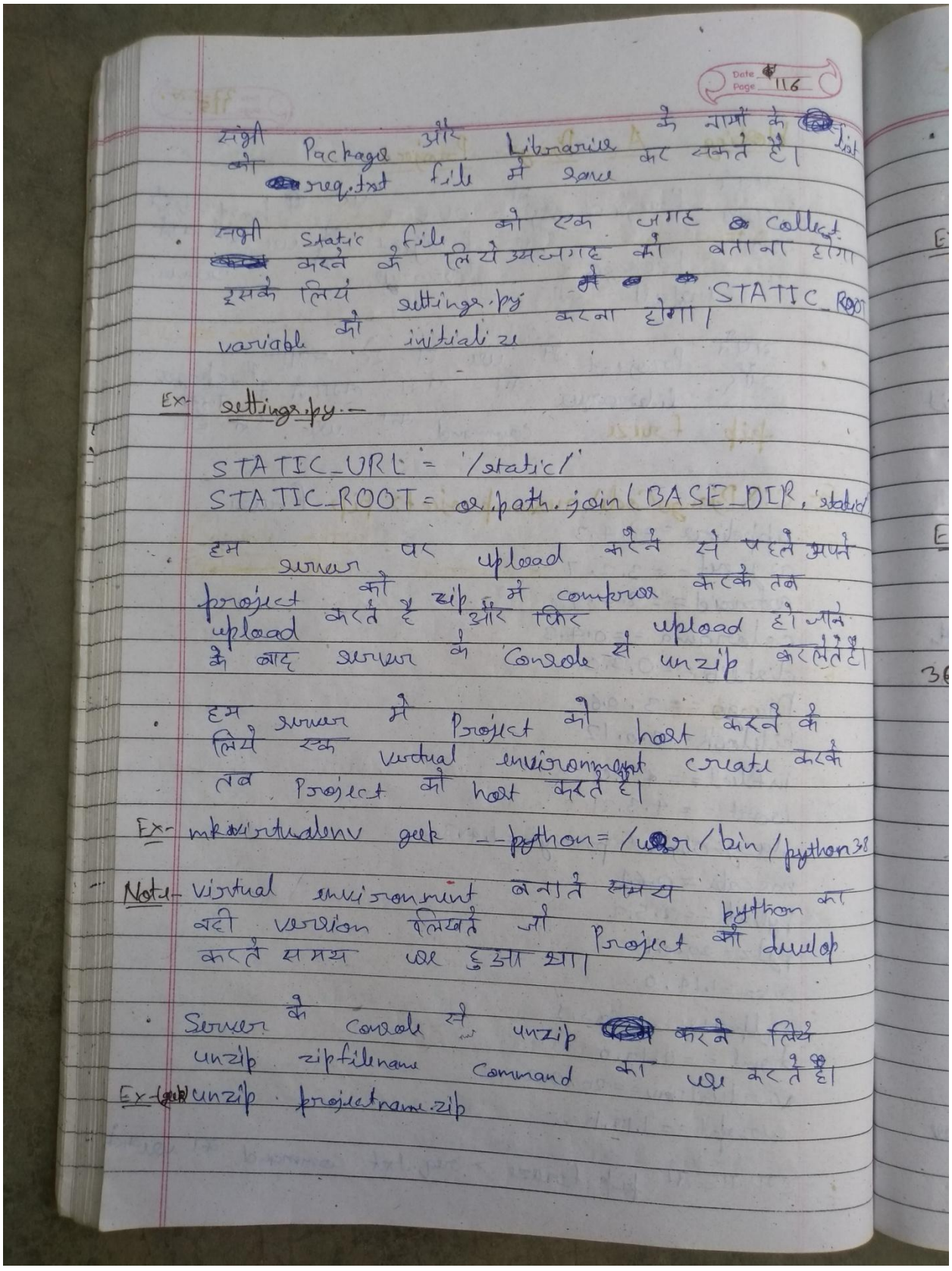
Hosting A Django Project -

- अपने Project को server पर host करते समय हमें Project में use होने वाले सभी Packages और libraries को भी server पर install करना होगा।
- अपने Project में use हो रहे सभी Packages और libraries का पता लगाने के लिए pip freeze command का use करते है।

Ex- C:\DjangoCode\resumproject> pip freeze

```
appdirs==1.4.3
asgiref==3.2.7
astroid==2.4.1
colorama==0.4.3
distlib==0.3.0
Django==3.0.6
filelock==3.0.12
inflect==4.1.0
isort==4.3.21
lazy-object-proxy==1.4.3
mccabe==0.6.1
pylint==2.5.2
pytz==2020.1
six==1.14.0
sqlparse==0.3.1
toml==0.10.0
virtualenv==20.0.20
wrapt==1.12.1
```

- हम चाहें तो pip freeze > req.txt command का use करके इ



Date
Page 116

सभी Package और Libraries के नामों के list को req.txt file में save कर सकते हैं।

- सभी Static file को एक जगह collect करने के लिये उस जगह को बताना होगा। इसके लिये settings.py में STATIC_ROOT variable को initialize करना होगा।

Ex- settings.py --

```
STATIC_URL = '/static/'
STATIC_ROOT = os.path.join(BASE_DIR, 'static')
```

- हम server पर upload करने से पहले अपने project को zip में compress करके तब upload करते हैं और फिर upload हो जाने के बाद server के Console से unzip कर लेते हैं।

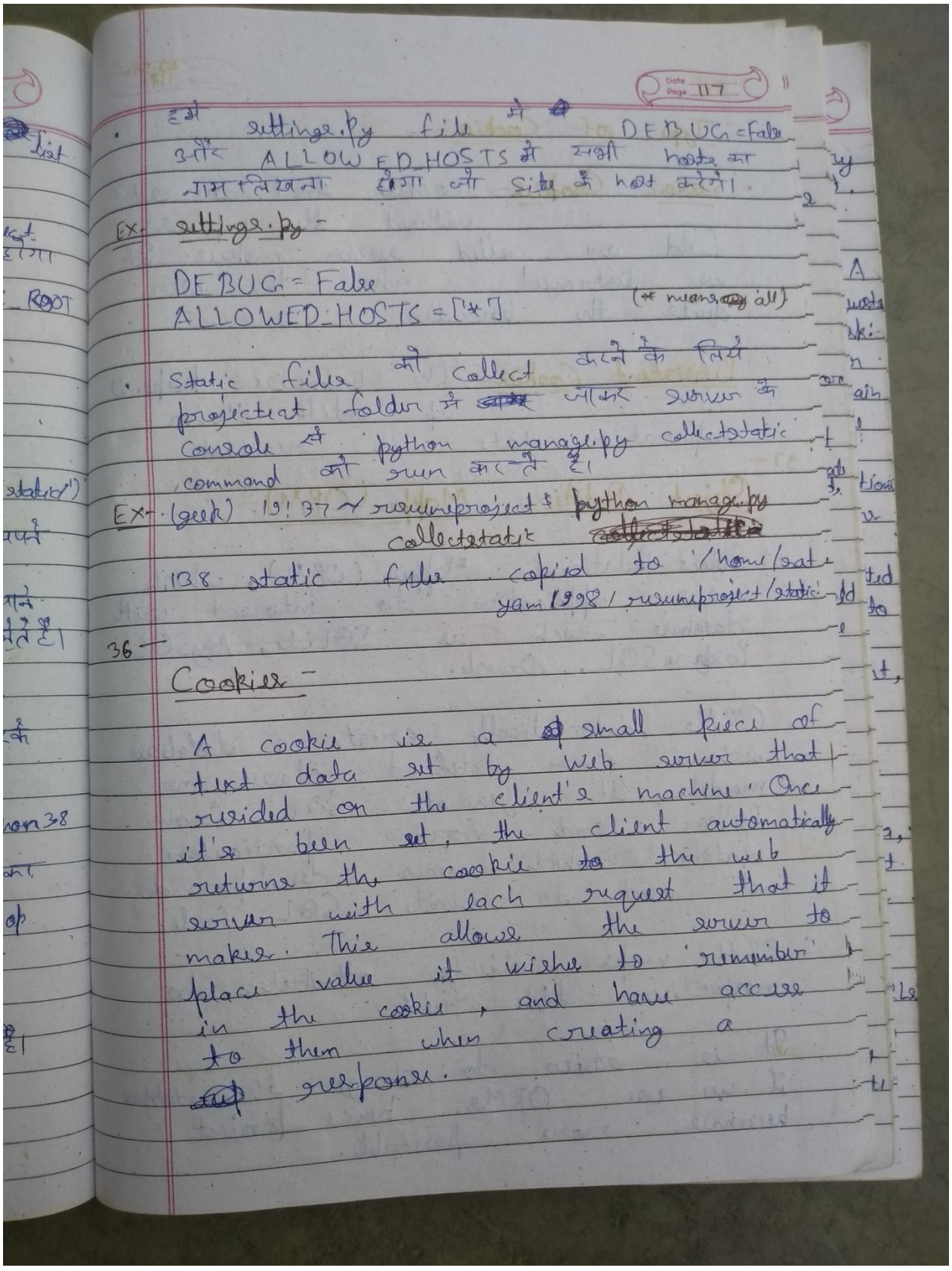
- हम server में Project को host करने के लिये एक virtual environment create करके तब Project को host करते हैं।

Ex- mkvirtualenv geek --python=/usr/bin/python38

Note- virtual environment बनाते समय python का वही version लिखते जो Project को develop करते समय use हुआ था।

- Server के Console से unzip करने लिये unzip zipfilename command का use करते हैं।

Ex- unzip projectname.zip



इसे settings.py file में DEBUG=False और ALLOWED_HOSTS में सभी hosts का नाम लिखना होगा जो Site को host करेंगे।

Ex. settings.py -

DEBUG = False
ALLOWED_HOSTS = ['*'] (* means all)

Static files को collect करने के लिये project folder में जाकर server के console में python manage.py collectstatic command को run करते है।

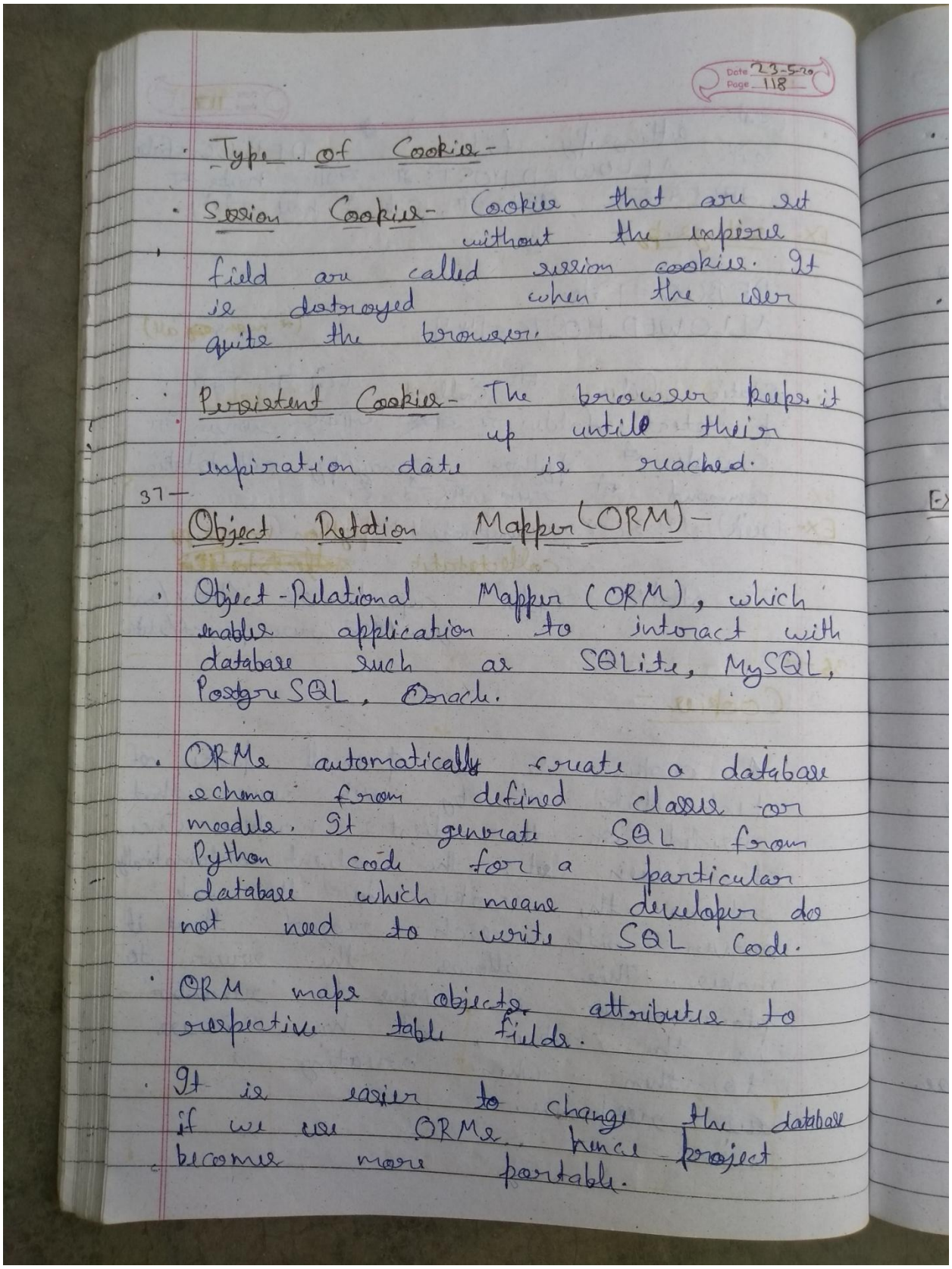
Ex. (geek) 19:37 ~ rusumproject $ python manage.py collectstatic

138 static files copied to '/home/satyam1998/rusumproject/static'.

36 —

Cookies -

A cookie is a small piece of text data set by web server that resided on the client's machine. Once it's been set, the client automatically returns the cookie to the web server with each request that it makes. This allows the server to place value it wishes to 'remember' in the cookie, and have access to them when creating a response.



Date 23-5-20
Page 118

- Type of Cookie -
- Session Cookie - Cookie that are set without the expire field are called session cookie. It is destroyed when the user quite the browser.
- Persistent Cookie - The browser keeps it up until their expiration date is reached.

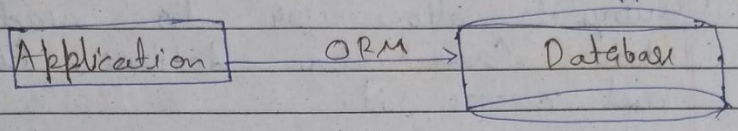
37 -

Object Relation Mapper (ORM) -

- Object-Relational Mapper (ORM), which enable application to interact with database such as SQLite, MySQL, PostgreSQL, Oracle.
- ORMs automatically create a database schema from defined class or module. It generate SQL from Python code for a particular database which means developer do not need to write SQL code.
- ORM maps objects attributes to respective table fields.
- It is easier to change the database if we use ORMs. hence project becomes more portable.

Date 24-5-20
Page 119

- Django's ORM is just a way to create SQL to query and manipulate your database and get result in a pythonic fashion.
- ORMs use connectors to connect databases with a web applications.

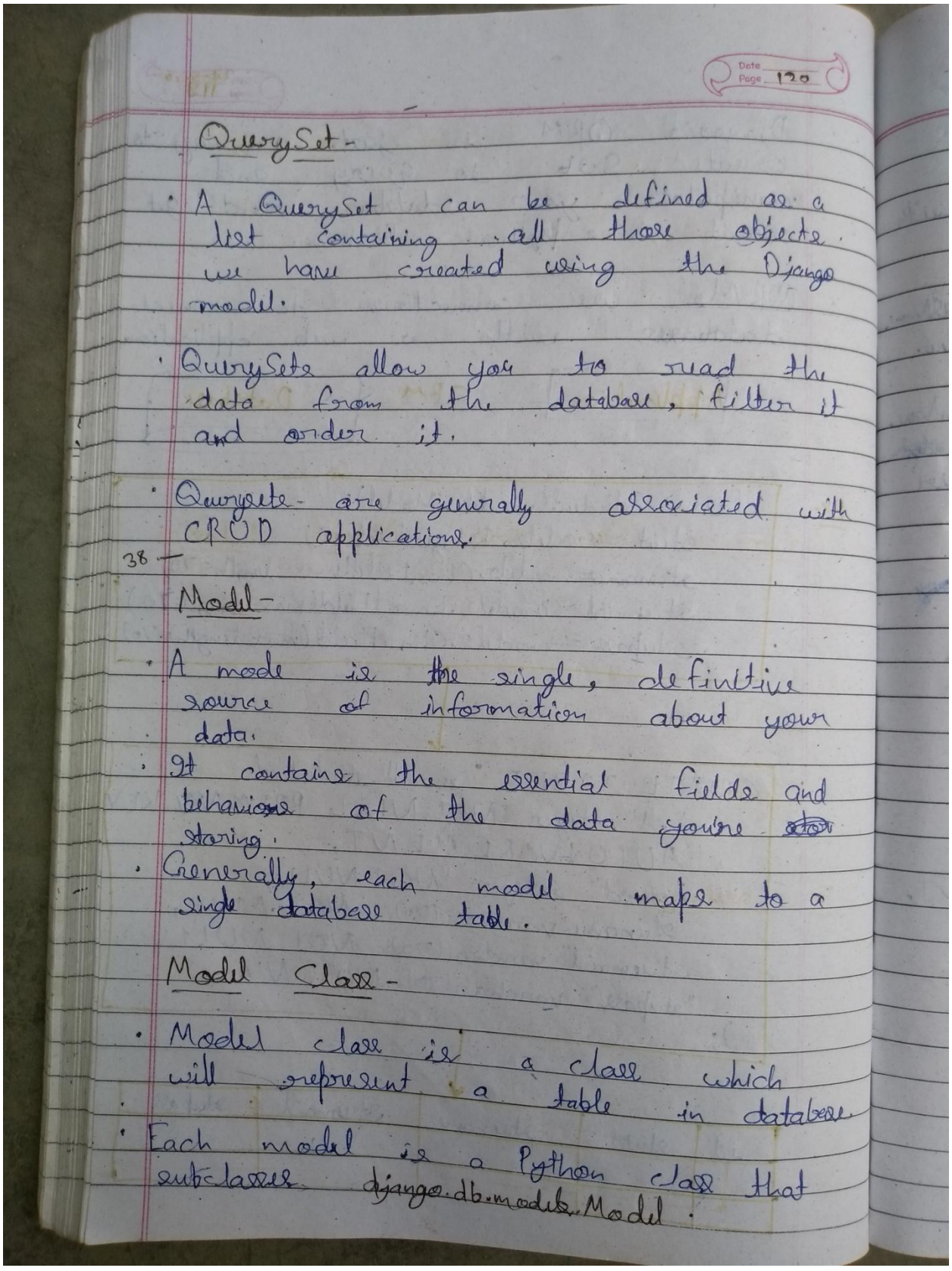


Ex-

```
class Student(models.Model):
    stuid = models.IntegerField()
    stuname = models.CharField(max_length=70)
    stumail = models.EmailField(max_length=70)
    stupass = models.CharField(max_length=70)
```

```
CREATE TABLE "enroll_student" (
    "id" integer NOT NULL PRIMARY KEY
    AUTOINCREMENT,
    "stuid" integer NOT NULL,
    "stuname" varchar(70) NOT NULL,
    "stumail" varchar(70) NOT NULL,
    "stupass" varchar(70) NOT NULL
);
```

id	stuid	stuname	stumail	stupass



Date
Page 120

QuerySet -

- A QuerySet can be defined as a list containing all those objects we have created using the Django model.
- QuerySets allow you to read the data from the database, filter it and order it.
- QuerySets are generally associated with CRUD applications.

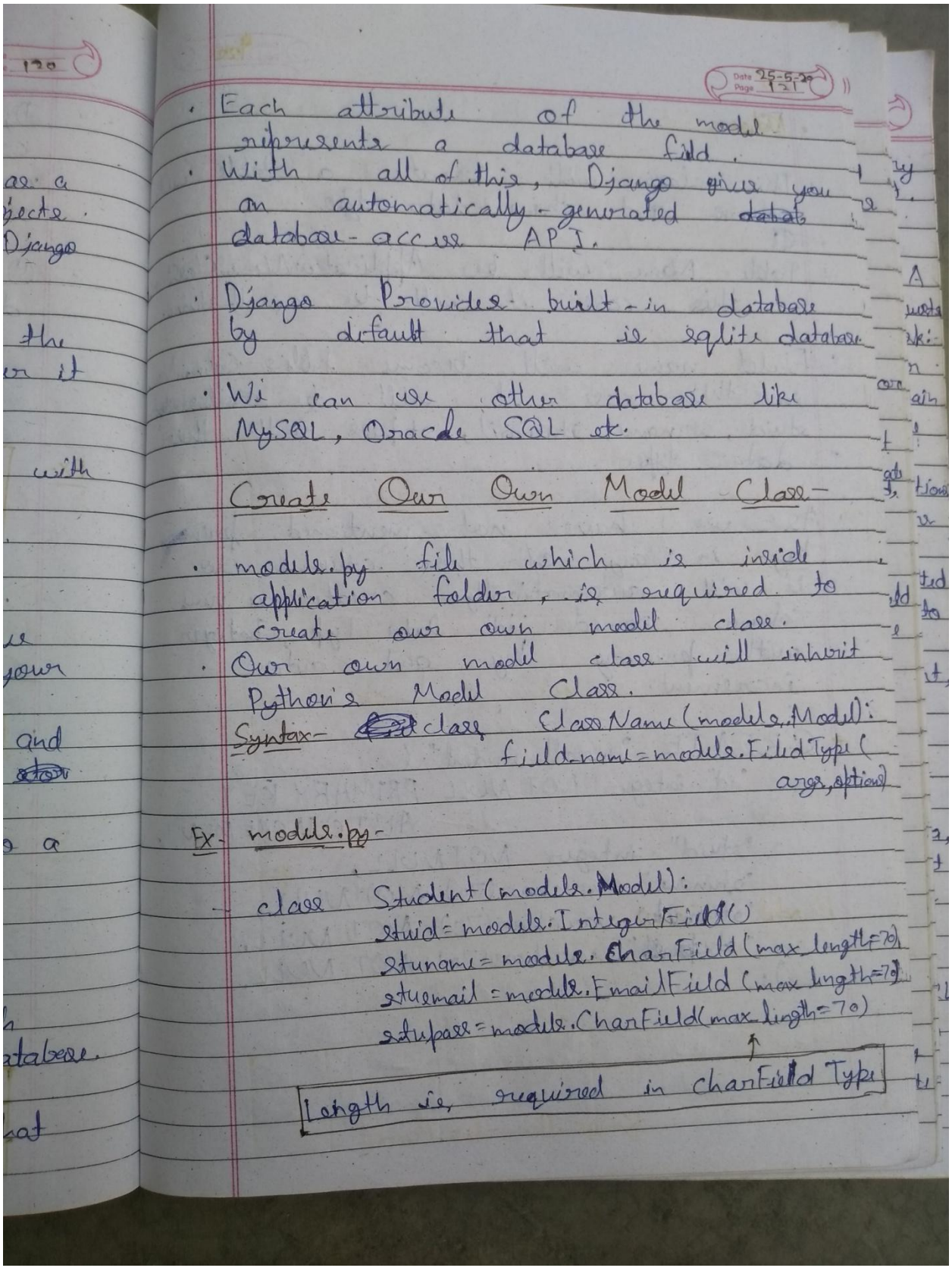
38 —

Model -

- A model is the single, definitive source of information about your data.
- It contains the essential fields and behaviors of the data you're storing.
- Generally, each model maps to a single database table.

Model Class -

- Model class is a class which will represent a table in database.
- Each model is a Python class that subclasses django.db.models.Model.



- Each attribute of the model represents a database field.
- With all of this, Django gives you an automatically-generated database-access API.
- Django Provides built-in database by default that is sqlite database.
- We can use other database like MySQL, Oracle SQL etc.

Create Our Own Model Class-

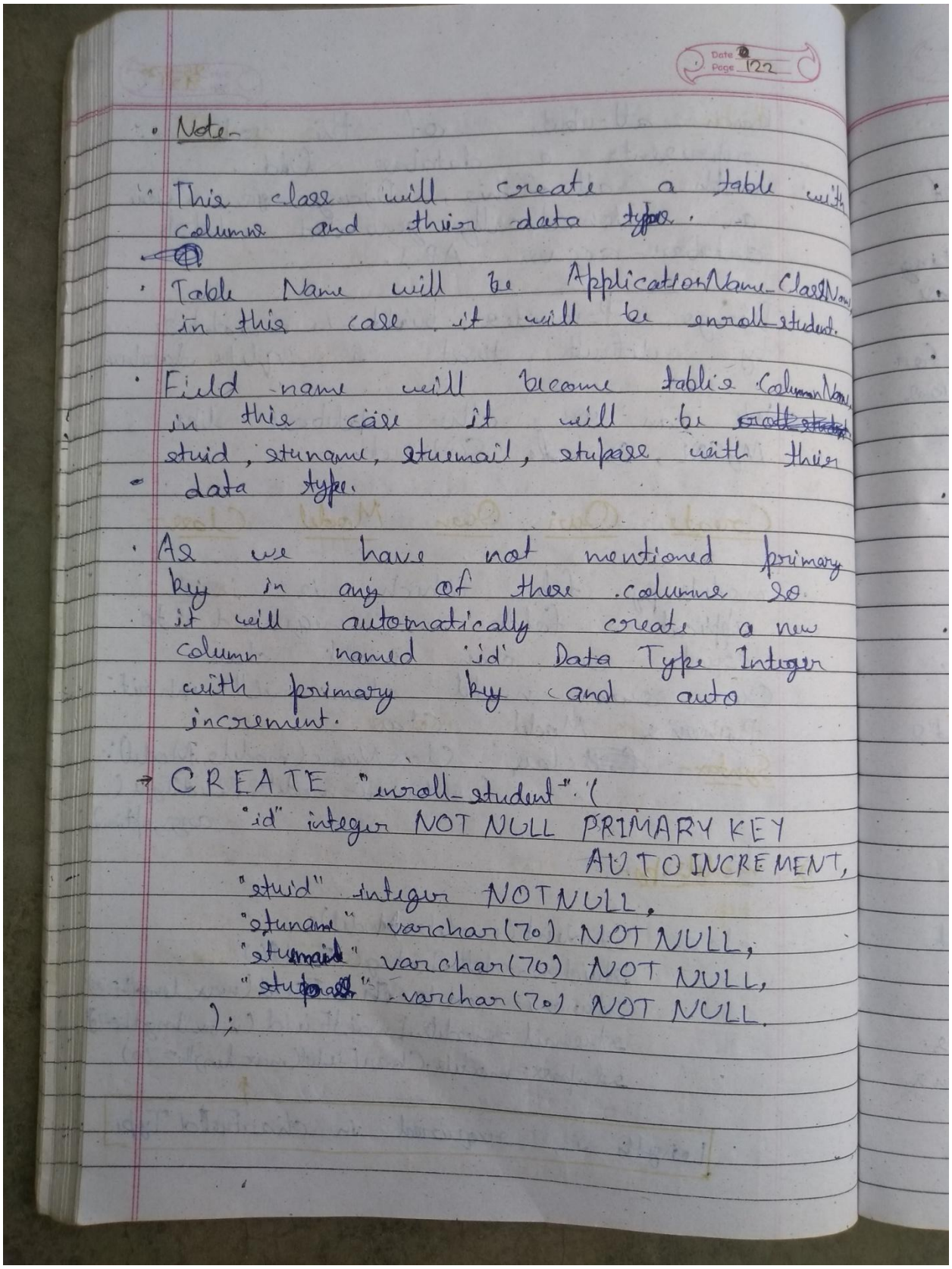
- models.py file which is inside application folder, is required to create our own model class.
- Our own model class will inherit Python's Model Class.

Syntax- class ClassName(models.Model):
    field_name = models.FieldType(arge, option)

Ex- models.py-

class Student(models.Model):
    stuid = models.IntegerField()
    stuname = models.CharField(max_length=70)
    stuemail = models.EmailField(max_length=70)
    stupass = models.CharField(max_length=70)

Length is required in CharField Type



Date
Page 122

- Note-

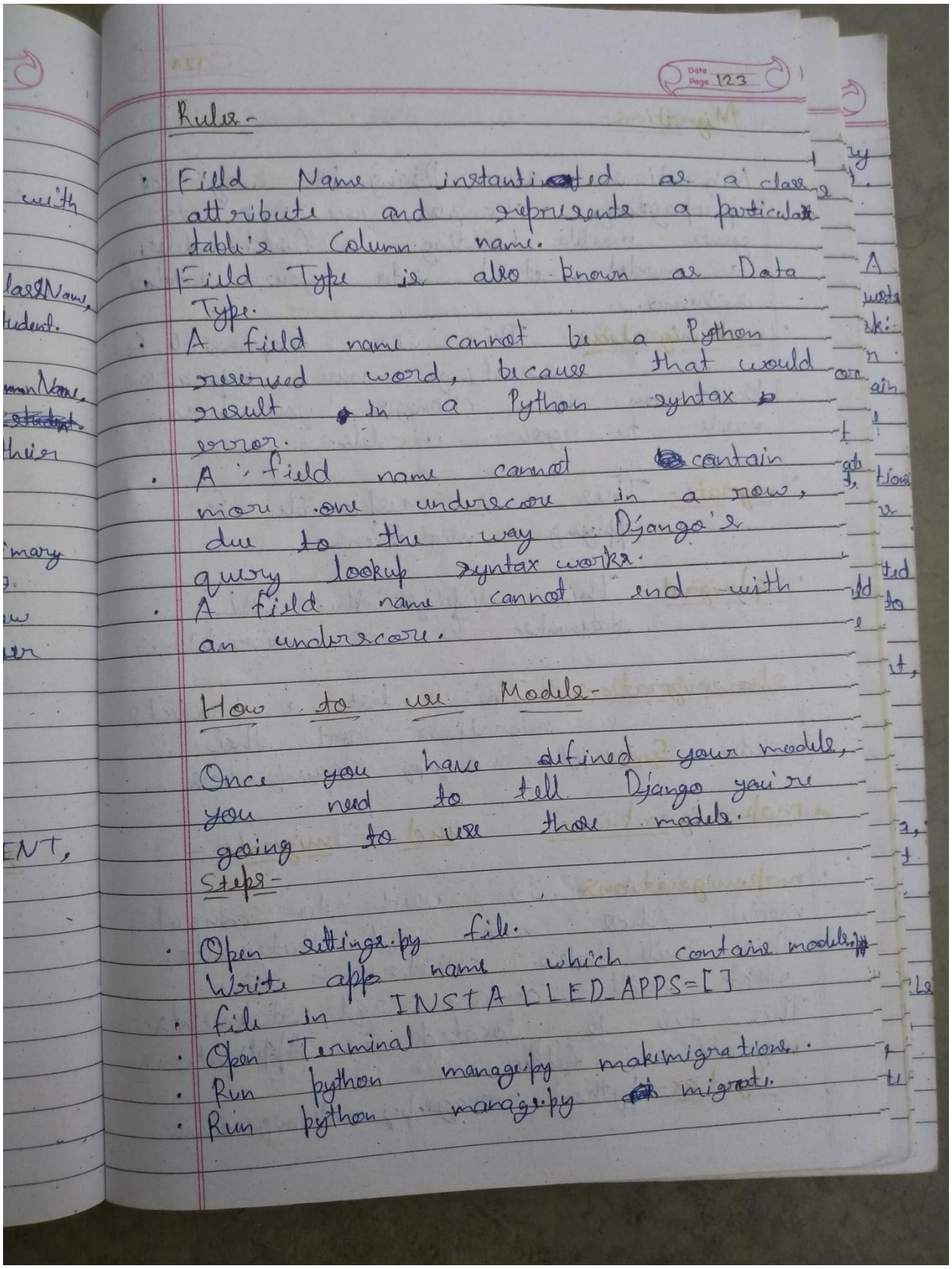
- This class will create a table with columns and their data type.

- Table Name will be ApplicationName_ClassName in this case it will be enroll_student.

- Field name will become table ColumnName in this case it will be ~~roll_student~~ stuid, stuname, stuemail, stupass with their data type.

- As we have not mentioned primary key in any of these columns so it will automatically create a new column named 'id' Data Type Integer with primary key and auto increment.

→ CREATE "enroll_student" (
"id" integer NOT NULL PRIMARY KEY AUTOINCREMENT,
"stuid" integer NOTNULL,
"stuname" varchar(70) NOT NULL,
"stuemail" varchar(70) NOT NULL,
"stupass" varchar(70) NOT NULL.
);



Rules-

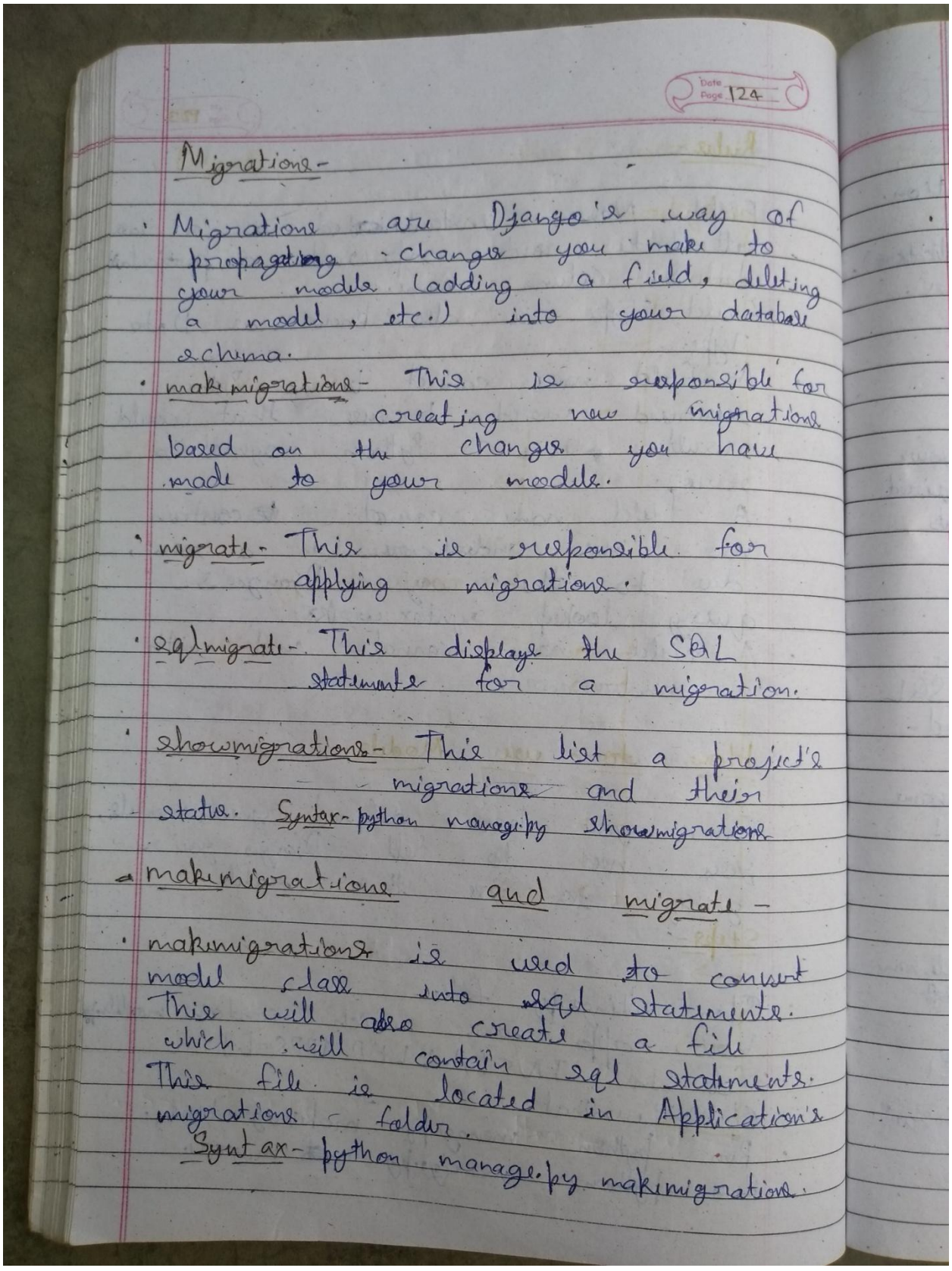
- Field Name instantiated as a class attribute and represents a particular table's Column name.
- Field Type is also known as Data Type.
- A field name cannot be a Python reserved word, because that would result in a Python syntax error.
- A field name cannot contain more one underscore in a row, due to the way Django's query lookup syntax works.
- A field name cannot end with an underscore.

How to use Module-

Once you have defined your models, you need to tell Django you're going to use those models.

Steps-

- Open settings.py file.
- Write app name which contains models file in INSTALLED_APPS=[]
- Open Terminal
- Run python manage.py makemigrations.
- Run python manage.py migrate.

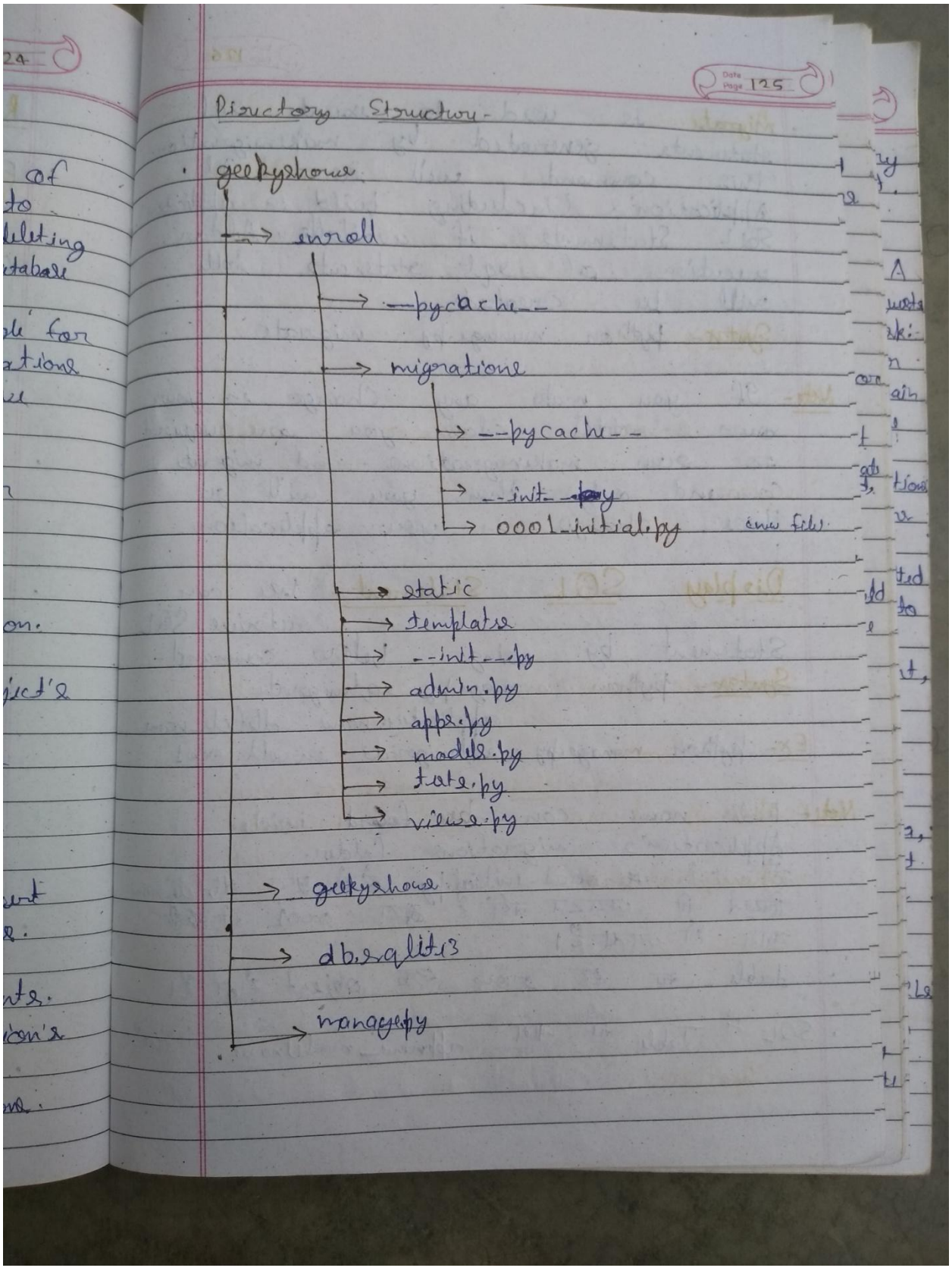

Date Page 124

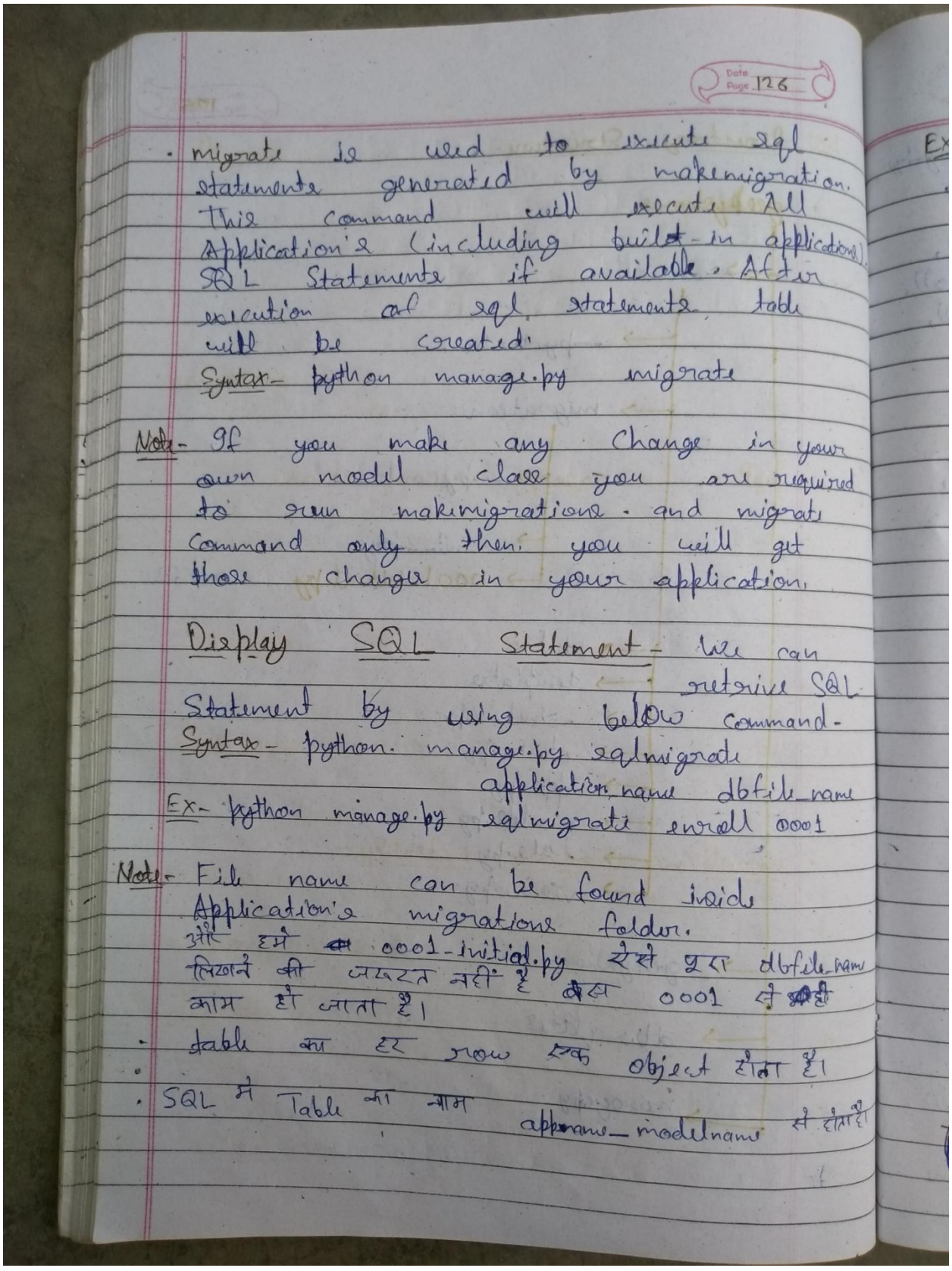
Migrations-

- Migrations are Django's way of propagating changes you make to your models (adding a field, deleting a model, etc.) into your database schema.
- makemigrations- This is responsible for creating new migrations based on the changes you have made to your models.
- migrate- This is responsible for applying migrations.
- sqlmigrate- This displays the SQL statements for a migration.
- showmigrations- This list a project's migrations and their status. Syntax- python manage.py showmigrations

makemigrations and migrate -

- makemigrations is used to convert model class into sql statements. This will also create a file which will contain sql statements. This file is located in Application's migrations folder.
  Syntax- python manage.py makemigrations.


Directory Structure-
geekyshows
enroll
__pycache__
migrations
__pycache__
__init__.py
0001_initial.py
static
templates
__init__.py
admin.py
apps.py
models.py
tests.py
views.py
geekyshows
db.sqlite3
manage.py



Date Page 126

- migrate is used to execute sql statements generated by makemigration. This command will execute All Application's (including built-in applications) SQL Statements if available. After execution of sql statements table will be created.

Syntax- python manage.py migrate

Note- If you make any change in your own model class you are required to run makemigrations. and migrate command only then you will get those changes in your application.

Display SQL Statement - We can retrive SQL Statement by using below command.

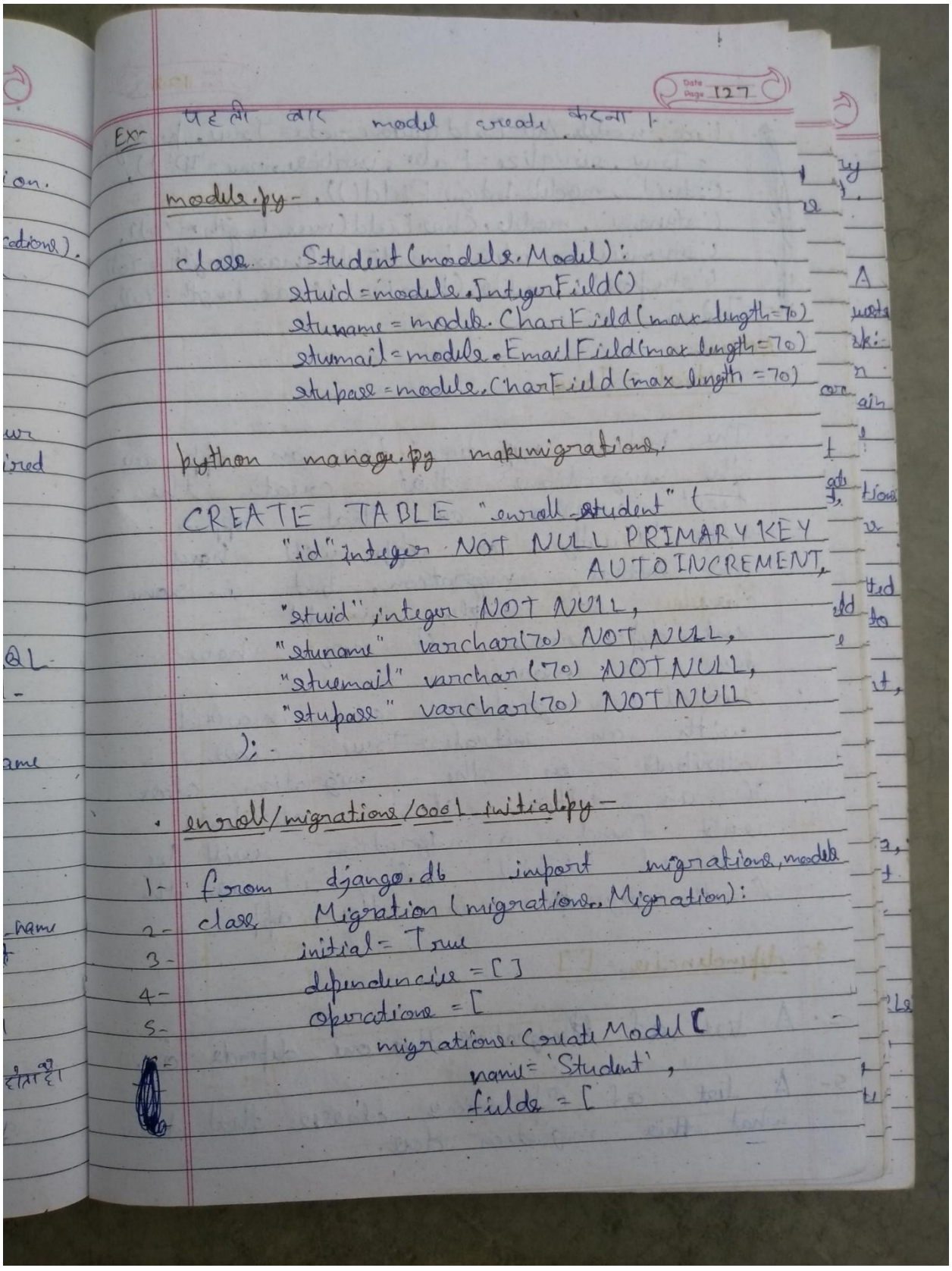
Syntax- python manage.py sqlmigrate application_name dbfile_name

Ex- python manage.py sqlmigrate enroll 0001

Note- File name can be found inside Application's migrations folder.

यहाँ हमें 0001_initial.py ऐसे पूरा dbfile_name लिखने की जरूरत नहीं है बस 0001 से ही काम हो जाता है।

- table का हर row एक object होता है।
- SQL में Table का नाम appname_modelname से होता है।



Ex- पहली बार model create करना।

module.py -

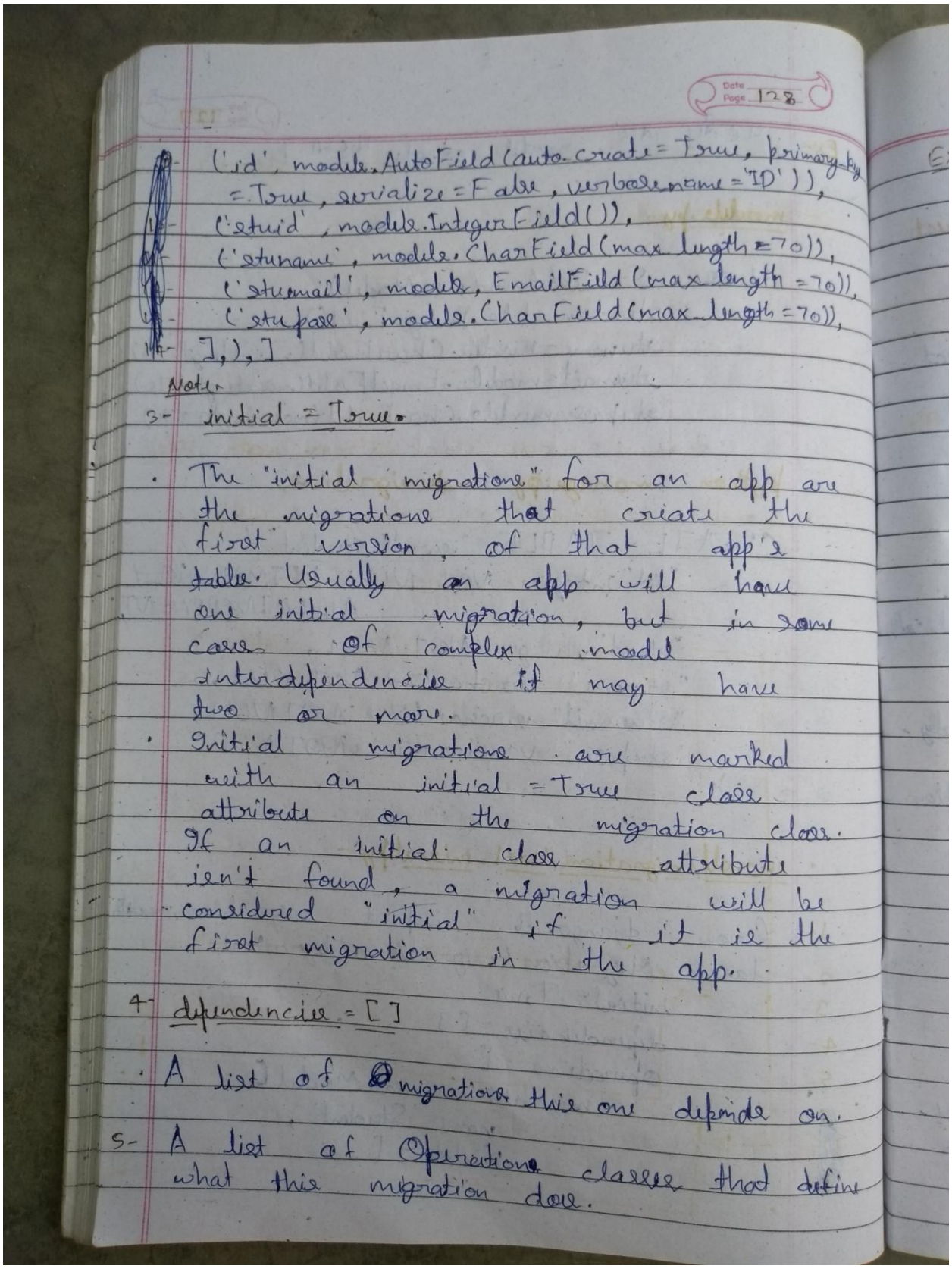
```
class Student(models.Model):
    stuid = models.IntegerField()
    stuname = models.CharField(max_length=70)
    stumail = models.EmailField(max_length=70)
    stupass = models.CharField(max_length=70)
```

python manage.py makemigrations.

```
CREATE TABLE "enroll_student" (
    "id" integer NOT NULL PRIMARY KEY
                          AUTOINCREMENT,
    "stuid" integer NOT NULL,
    "stuname" varchar(70) NOT NULL,
    "stumail" varchar(70) NOT NULL,
    "stupass" varchar(70) NOT NULL
);
```

- enroll/migrations/0001_initial.py -

```
1- from django.db import migrations, models
2- class Migration(migrations.Migration):
3-     initial = True
4-     dependencies = []
5-     operations = [
           migrations.CreateModel(
               name='Student',
               fields=[
```



('id', models.AutoField(auto_created=True, primary_key=True, serialize=False, verbose_name='ID')),
('stuid', models.IntegerField()),
('stuname', models.CharField(max_length=70)),
('stumail', models.EmailField(max_length=70)),
('stupass', models.CharField(max_length=70)),
],),]

Note:

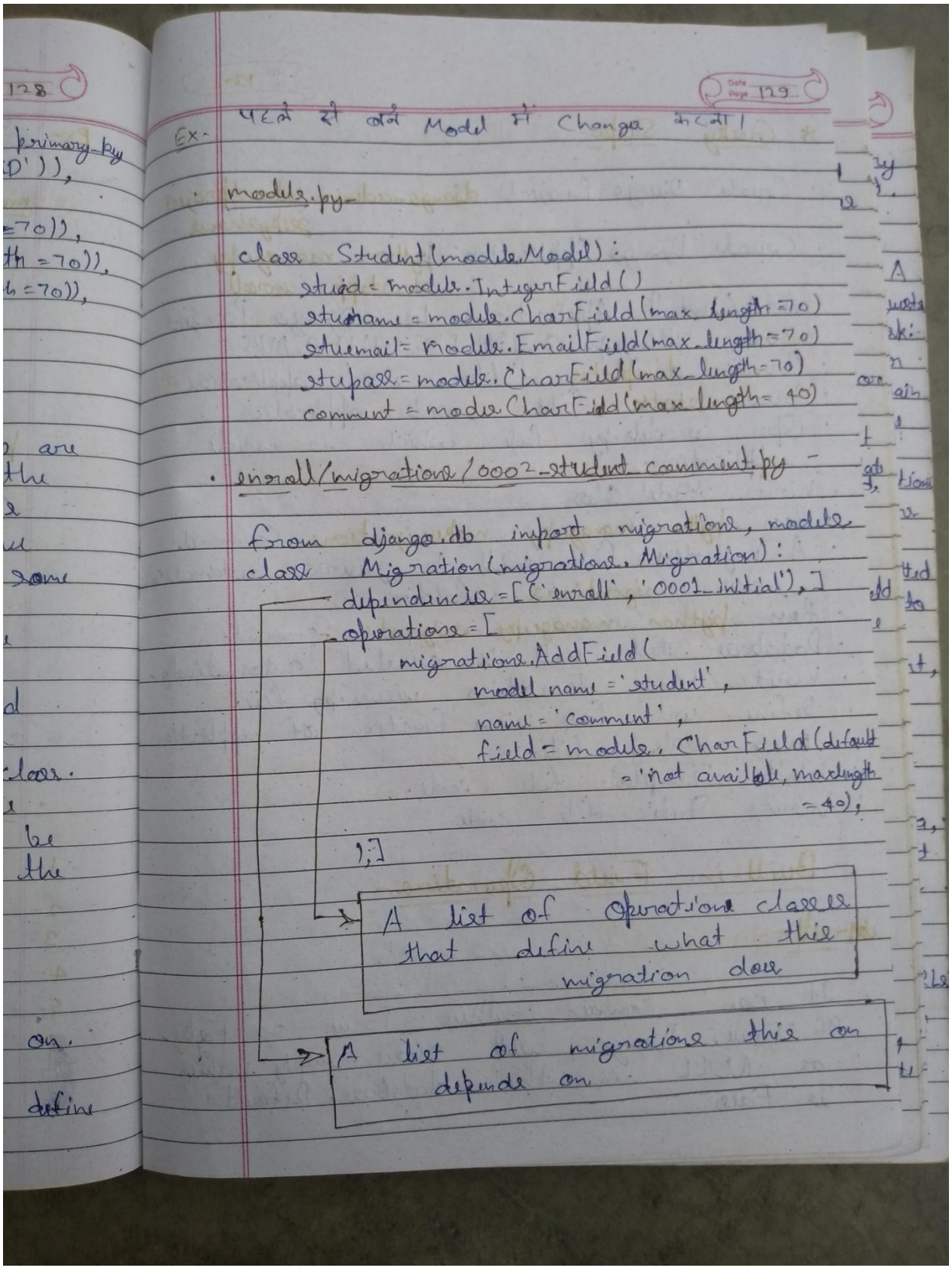
3- initial = True.

- The "initial migrations" for an app are the migrations that create the first version of that app's tables. Usually an app will have one initial migration, but in some cases of complex module interdependencies it may have two or more.
- Initial migrations are marked with an initial = True class attribute on the migration class. If an initial class attribute isn't found, a migration will be considered "initial" if it is the first migration in the app.

4- dependencies = [ ]

- A list of migrations this one depends on.

5- A list of Operations classes that define what this migration does.



Ex- पहले से बने Model में Change करना।

- models.py -

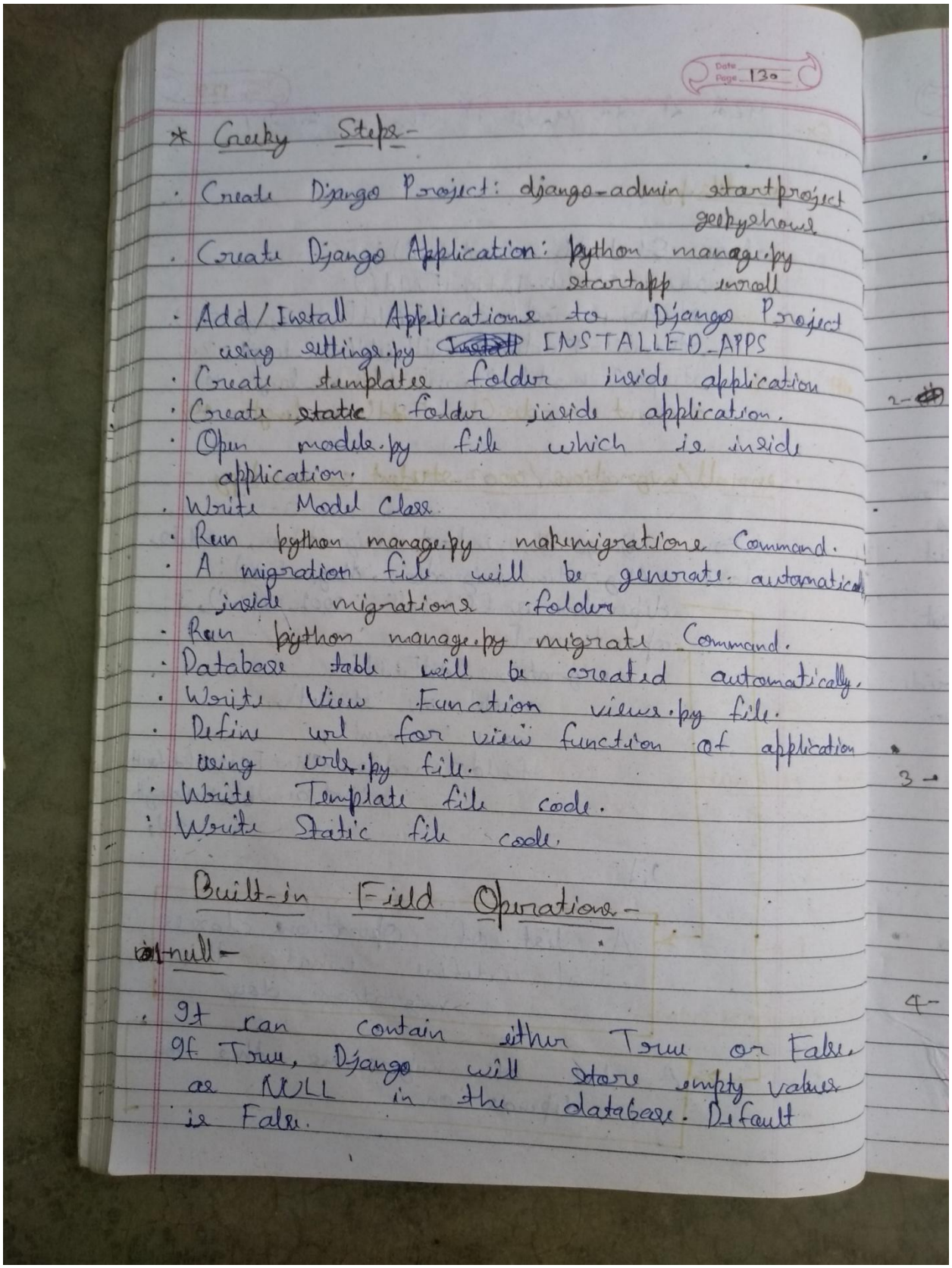
```
class Student(models.Model):
    stuid = models.IntegerField()
    stuname = models.CharField(max_length=70)
    stuemail = models.EmailField(max_length=70)
    stupass = models.CharField(max_length=70)
    comment = models.CharField(max_length=40)
```

- enroll/migrations/0002_student_comment.py -

```
from django.db import migrations, models
class Migration(migrations.Migration):
    dependencies = [('enroll', '0001_initial'),]
    operations = [
        migrations.AddField(
            model_name = 'student',
            name = 'comment',
            field = models.CharField(default
                = 'not available', max_length
                = 40),
        ),]
```

→ A list of Operations classes that define what this migration does

→ A list of migrations this one depends on



Date
Page 130

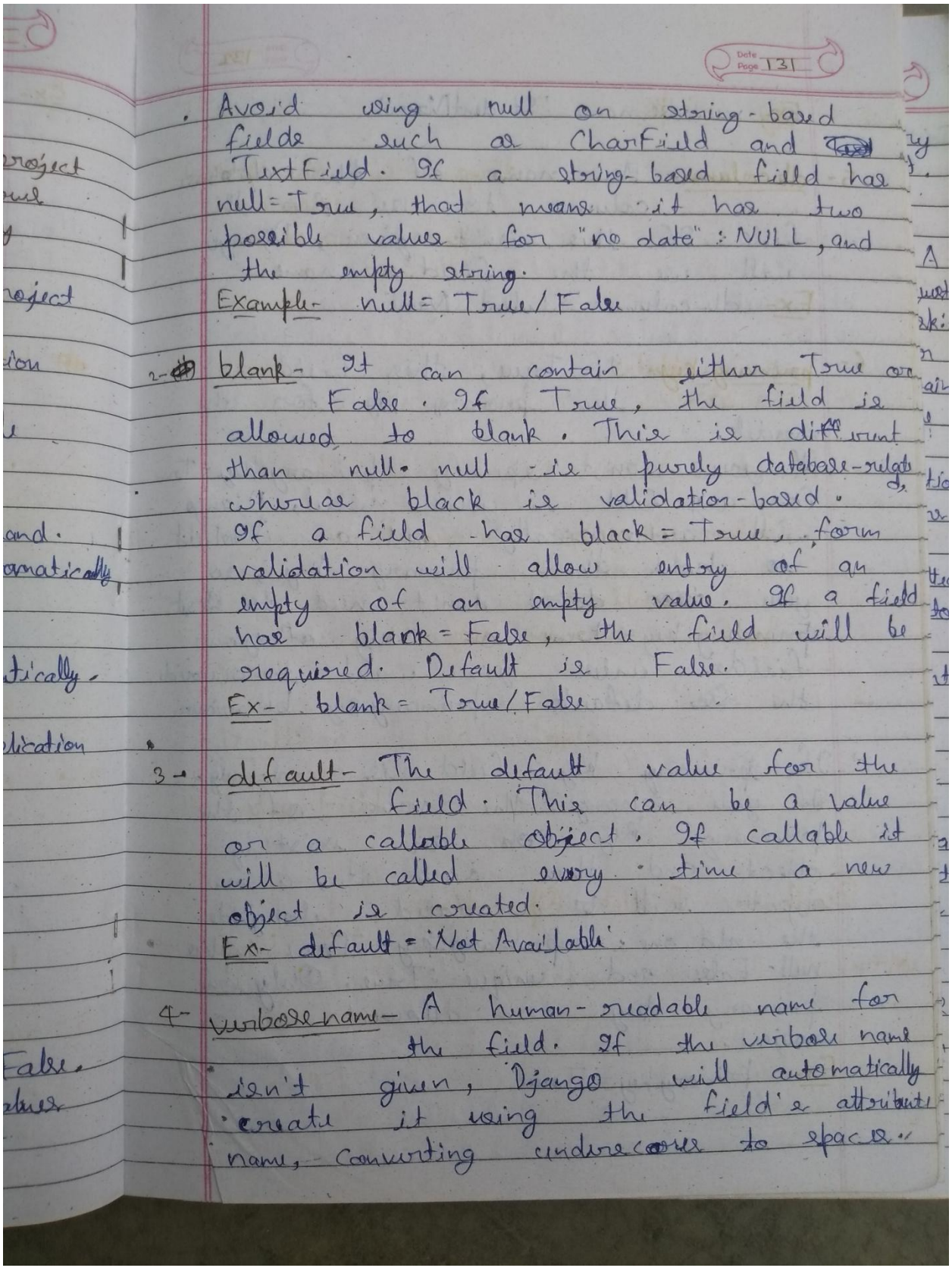
* Geeky Steps-

- Create Django Project: django-admin startproject geekyshows
- Create Django Application: python manage.py startapp myall
- Add/Install Applications to Django Project using settings.py ~~Install~~ INSTALLED_APPS
- Create templates folder inside application
- Create static folder inside application.
- Open models.py file which is inside application.
- Write Model Class
- Run python manage.py makemigrations Command.
- A migration file will be generate automatically inside migrations folder
- Run python manage.py migrate Command.
- Database table will be created automatically.
- Write View Function views.py file.
- Define url for view function of application using urls.py file.
- Write Template file code.
- Write Static file code.

Built-in Field Operations-

null-

- It can contain either True or False. If True, Django will store empty values as NULL in the database. Default is False.

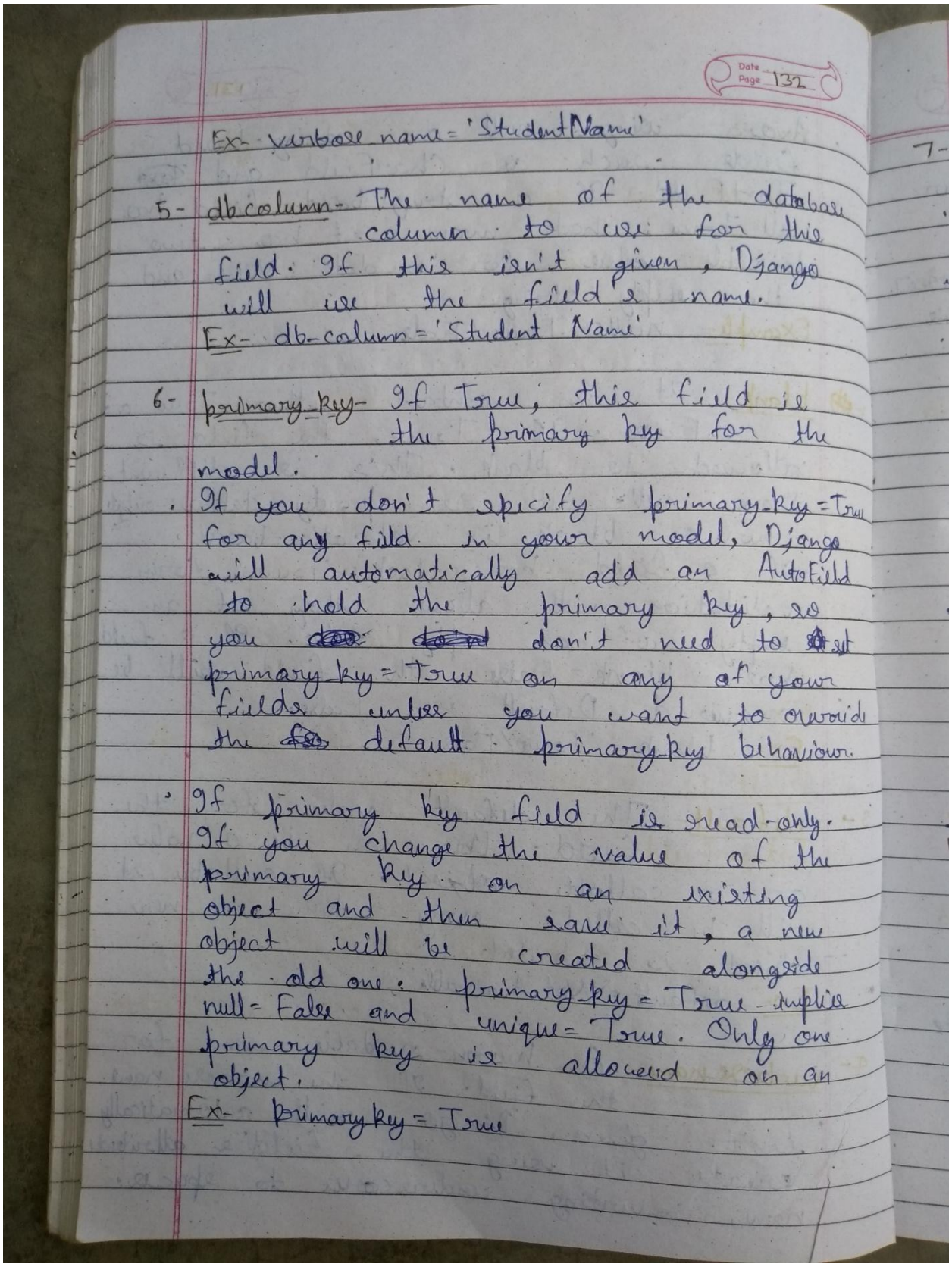


- Avoid using null on string-based fields such as CharField and TextField. If a string-based field has null=True, that means it has two possible values for "no data": NULL, and the empty string.
  Example- null=True/False

2- blank- It can contain either True or False. If True, the field is allowed to blank. This is different than null. null is purely database-related whereas black is validation-based. If a field has black=True, form validation will allow entry of an empty of an empty value. If a field has blank=False, the field will be required. Default is False.
  Ex- blank=True/False

3- default- The default value for the field. This can be a value or a callable object. If callable it will be called every time a new object is created.
  Ex- default='Not Available'

4- verbose_name- A human-readable name for the field. If the verbose name isn't given, Django will automatically create it using the field's attribute name, converting underscores to spaces.



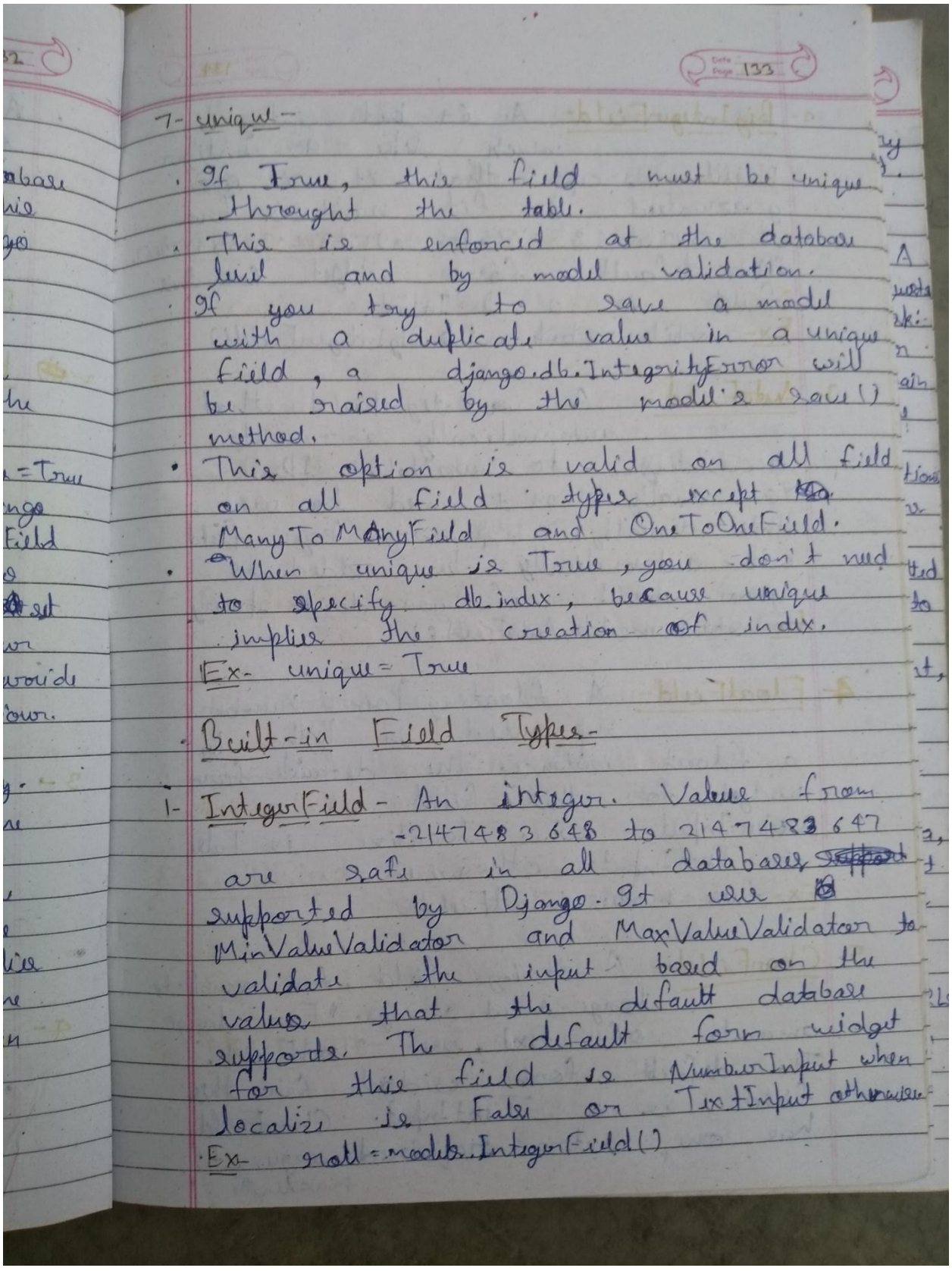
Date Page 132

Ex- verbose_name = 'Student Name'

5- db_column - The name of the database column to use for this field. If this isn't given, Django will use the field's name.
Ex- db_column = 'Student Name'

6- primary_key - If True, this field is the primary key for the model.
- If you don't specify primary_key = True for any field in your model, Django will automatically add an AutoField to hold the primary key, so you don't need to set primary_key = True on any of your fields unless you want to override the default primary key behaviour.
- If primary key field is read-only. If you change the value of the primary key on an existing object and then save it, a new object will be created alongside the old one. primary_key = True implies null = False and unique = True. Only one primary key is allowed on an object.

Ex- primary_key = True



7- unique -

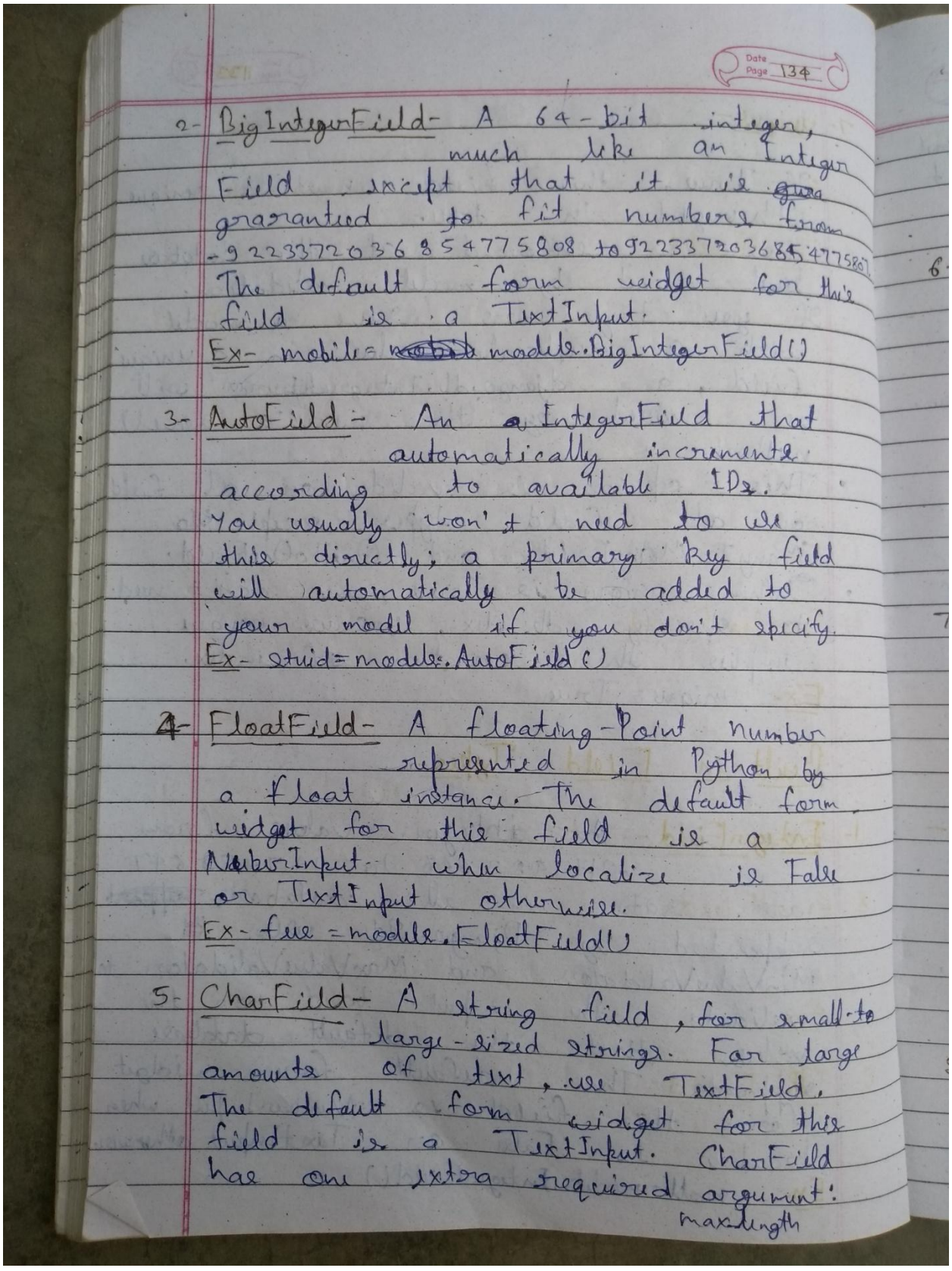
- If True, this field must be unique throughout the table.
- This is enforced at the database level and by model validation.
- If you try to save a model with a duplicate value in a unique field, a django.db.IntegrityError will be raised by the model's save() method.
- This option is valid on all field types except ManyToManyField and OneToOneField.
- When unique is True, you don't need to specify db_index, because unique implies the creation of index.

Ex- unique = True

Built-in Field Types-

1- IntegerField - An integer. Values from -2147483648 to 2147483647 are safe in all databases supported by Django. It uses MinValueValidator and MaxValueValidator to validate the input based on the values that the default database supports. The default form widget for this field is NumberInput when localize is False or TextInput otherwise.

Ex- roll = models.IntegerField()



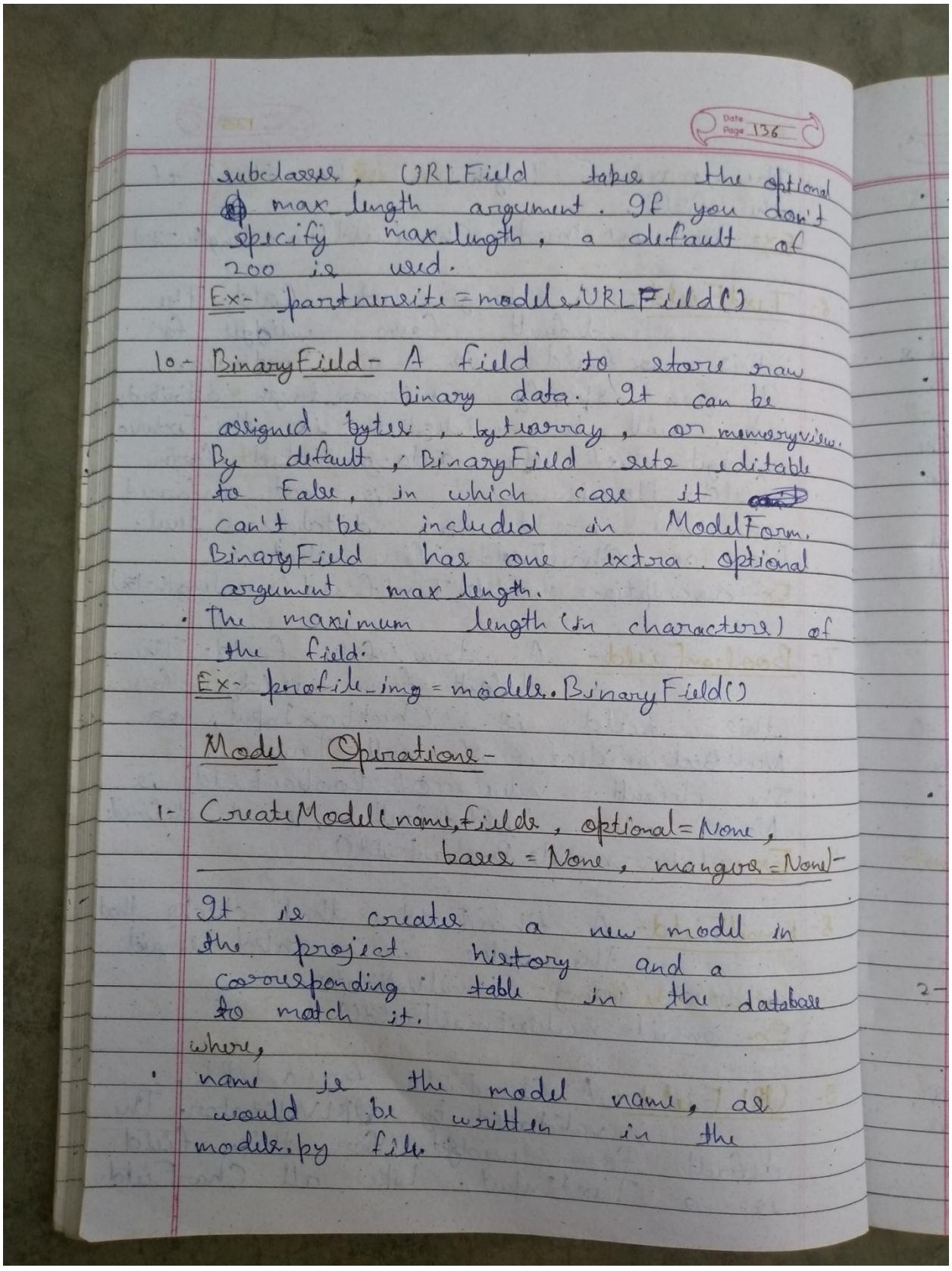
Date
Page 134

2- BigIntegerField- A 64-bit integer, much like an IntegerField except that it is guaranteed to fit numbers from -9223372036854775808 to 9223372036854775807. The default form widget for this field is a TextInput.
Ex- mobile = models.BigIntegerField()

3- AutoField - An IntegerField that automatically increments according to available IDs. You usually won't need to use this directly; a primary key field will automatically be added to your model if you don't specify.
Ex- stuid = models.AutoField()

4- FloatField- A floating-point number represented in Python by a float instance. The default form widget for this field is a NumberInput when localize is False or TextInput otherwise.
Ex- fee = models.FloatField()

5- CharField- A string field, for small-to large-sized strings. For large amounts of text, use TextField. The default form widget for this field is a TextInput. CharField has one extra required argument: max_length



subclasses. URLField takes the optional max_length argument. If you don't specify max_length, a default of 200 is used.

Ex- partnersite = models.URLField()

10- BinaryField- A field to store raw binary data. It can be assigned bytes, bytearray, or memoryview. By default, BinaryField sets editable to False, in which case it can't be included in ModelForm. BinaryField has one extra optional argument max_length.

- The maximum length (in characters) of the field.

Ex- profile_img = models.BinaryField()

Model Operations-

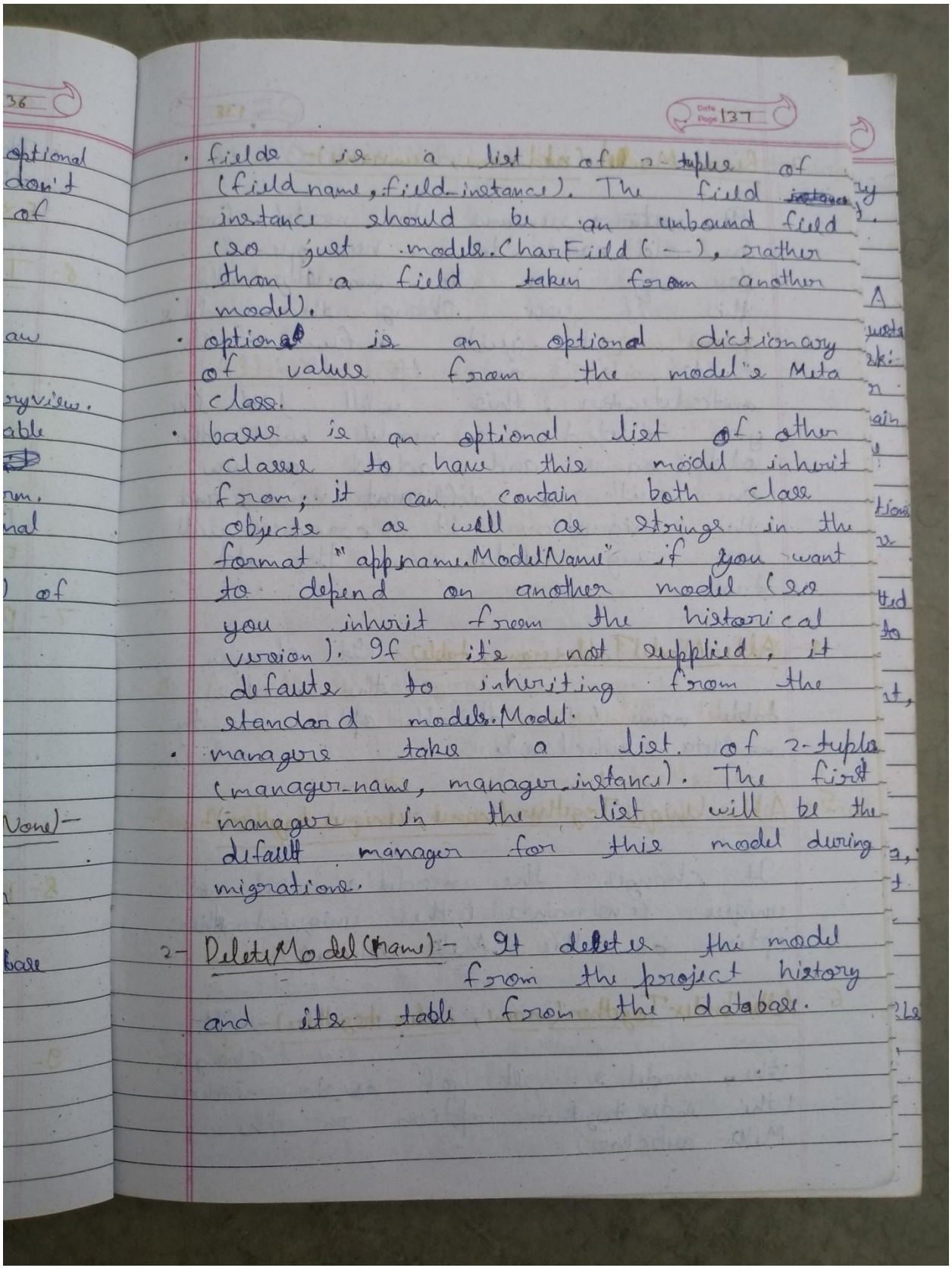
1- CreateModel(name, fields, options=None, bases=None, managers=None)-

It is create a new model in the project history and a corresponding table in the database to match it.

where,

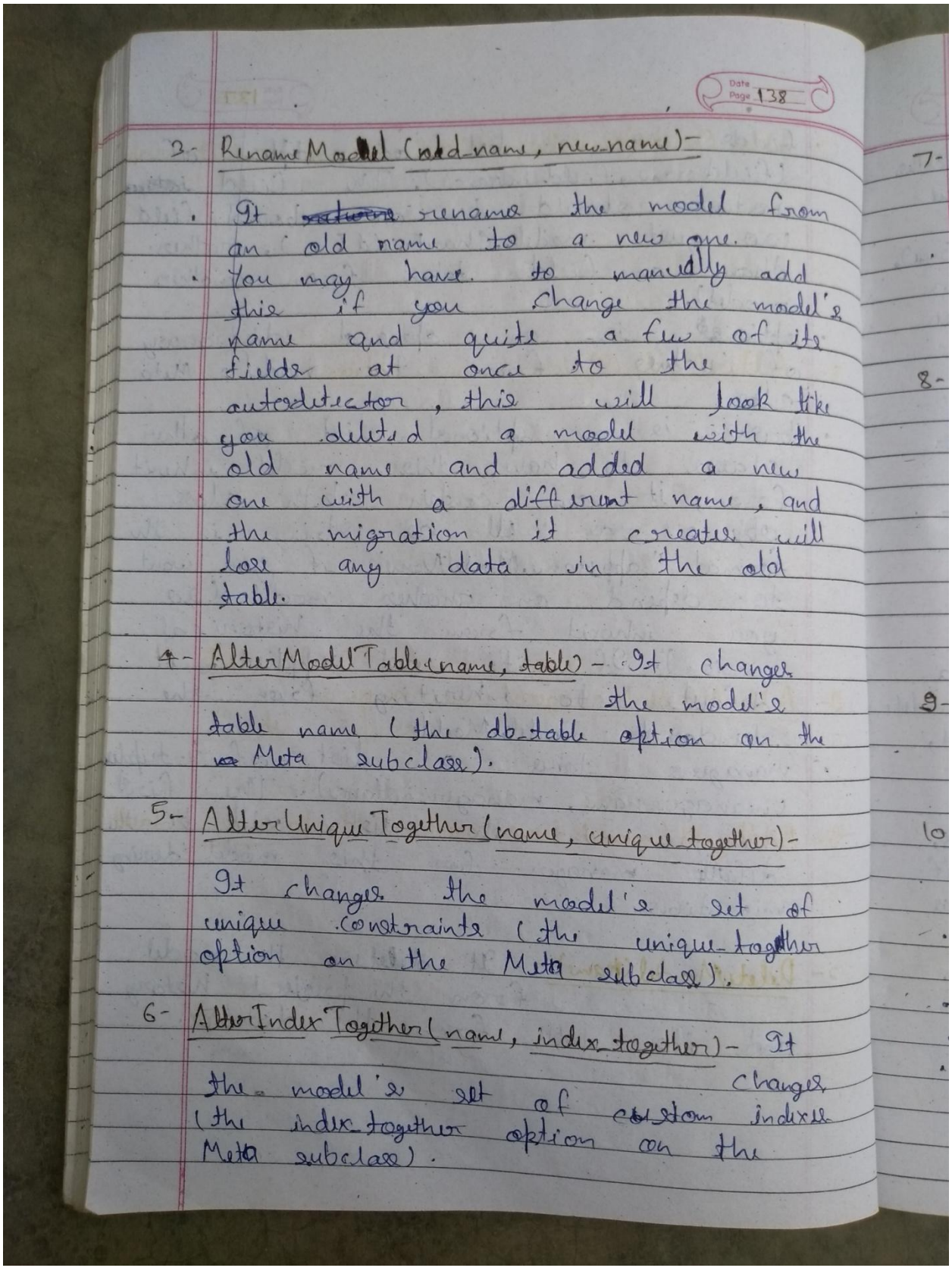
- name is the model name, as would be written in the models.py file.

2-



- fields is a list of 2-tuples of (field_name, field_instance). The field instance should be an unbound field (so just .models.CharField(...), rather than a field taken from another model).
- options is an optional dictionary of values from the model's Meta class.
- bases is an optional list of other classes to have this model inherit from; it can contain both class objects as well as strings in the format "appname.ModelName" if you want to depend on another model (so you inherit from the historical version). If it's not supplied, it defaults to inheriting from the standard models.Model.
- managers takes a list of 2-tuples (manager_name, manager_instance). The first manager in the list will be the default manager for this model during migrations.

2- DeleteModel(name) - It deletes the model from the project history and its table from the database.



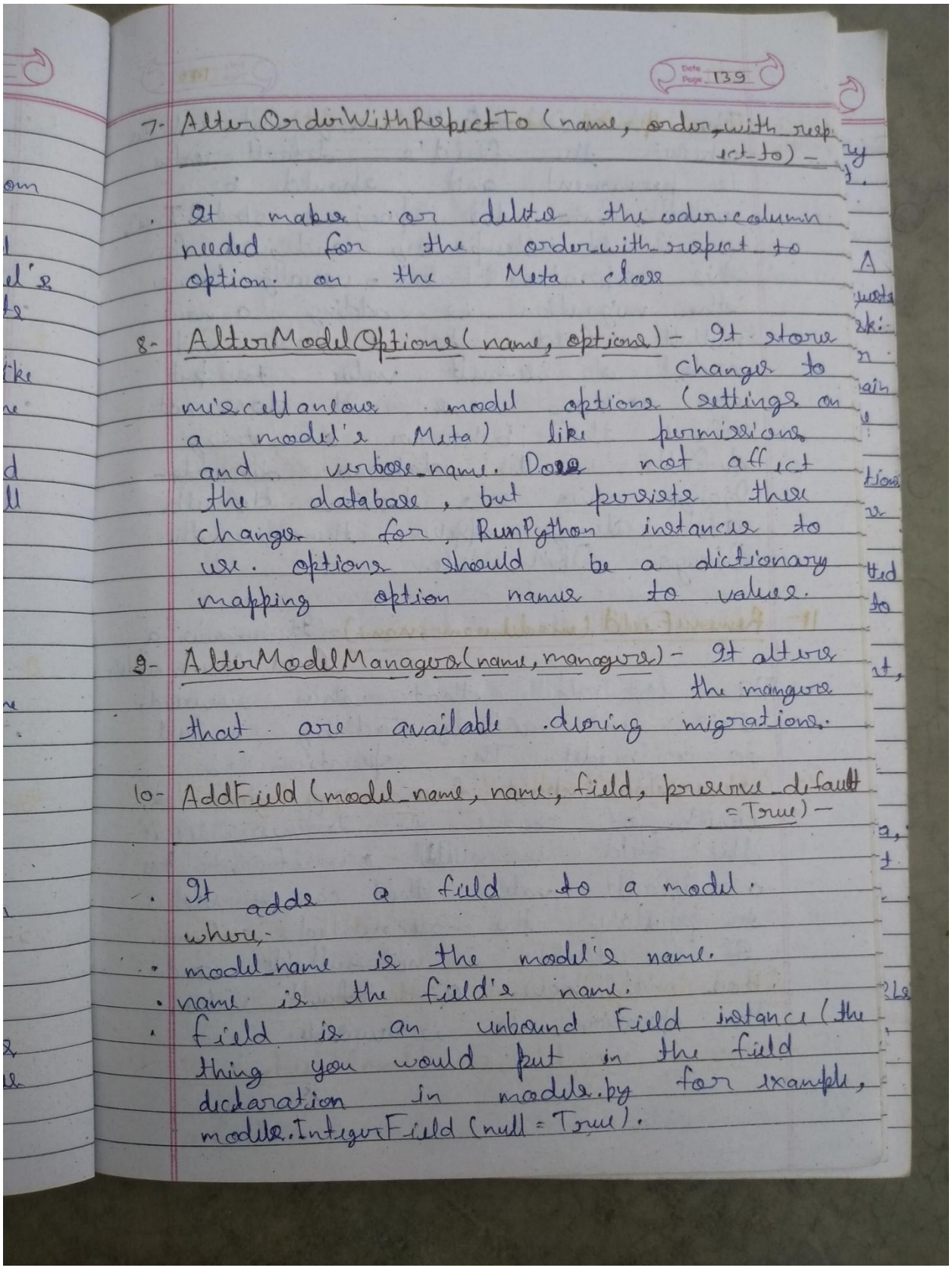
3- RenameModel (old_name, new_name)-

. It renames the model from an old name to a new one.
. You may have to manually add this if you change the model's name and quite a few of its fields at once to the autodetector, this will look like you deleted a model with the old name and added a new one with a different name, and the migration it creates will lose any data in the old table.

4- AlterModelTable(name, table) - It changes the model's table name (the db_table option on the Meta subclass).

5- AlterUniqueTogether(name, unique_together)-
It changes the model's set of unique constraints (the unique_together option on the Meta subclass).

6- AlterIndexTogether(name, index_together)- It changes the model's set of custom indexes (the index_together option on the Meta subclass).



Page 139

7- AlterOrderWithRespectTo(name, order_with_respect_to) –

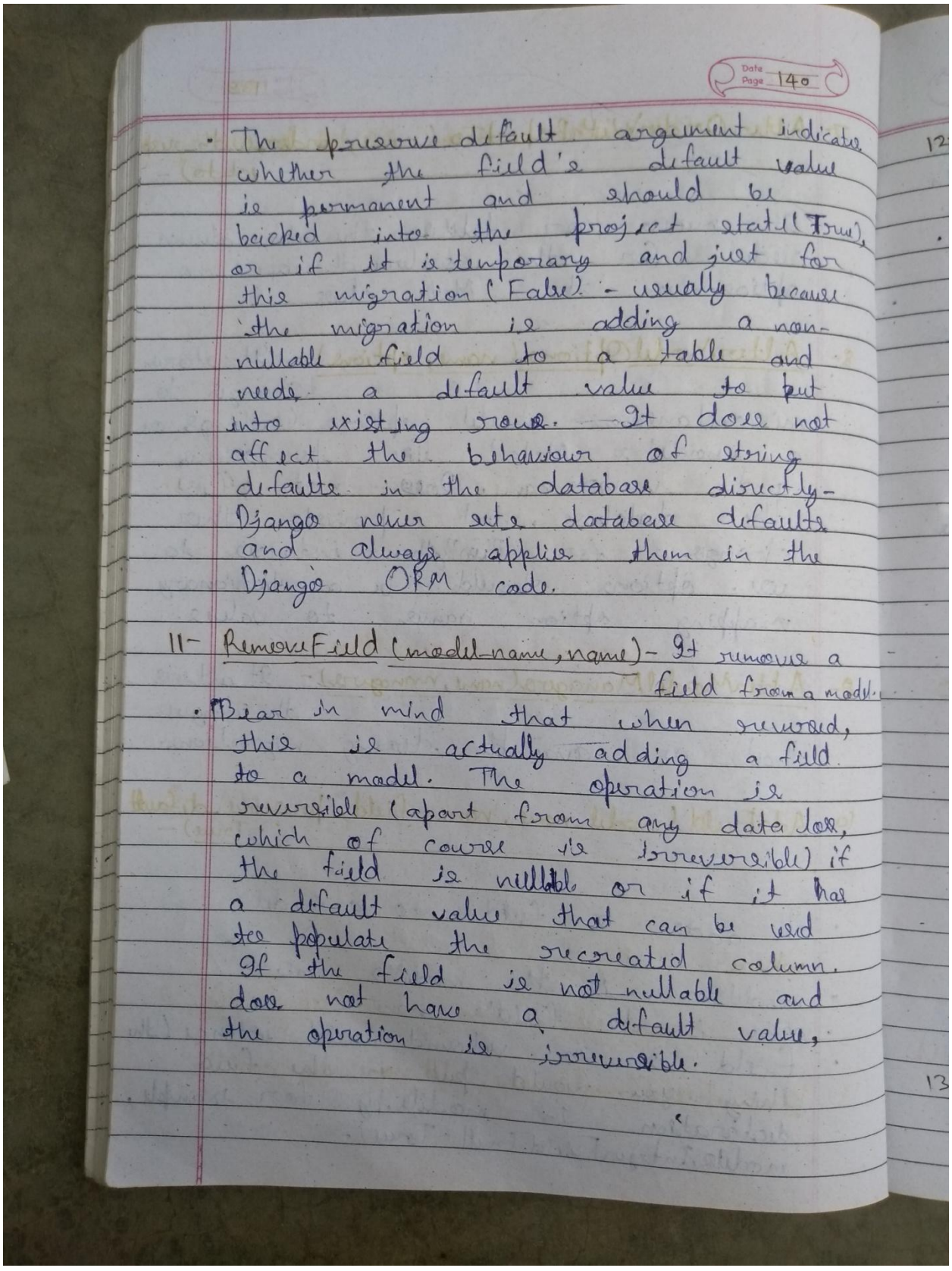
. It makes or deletes the order_column needed for the order_with_respect_to option on the Meta class

8- AlterModelOptions(name, options) – It stores changes to miscellaneous model options (settings on a model's Meta) like permissions and verbose_name. Does not affect the database, but persists these changes for RunPython instances to use. options should be a dictionary mapping option names to values.

9- AlterModelManagers(name, managers) – It alters the managers that are available during migrations.

10- AddField(model_name, name, field, preserve_default=True) –

. It adds a field to a model.
where,
. model_name is the model's name.
. name is the field's name.
. field is an unbound Field instance (the thing you would put in the field declaration in models.py for example, models.IntegerField(null=True).



Date Page 140

- The preserve_default argument indicates whether the field's default value is permanent and should be baked into the project state (True), or if it is temporary and just for this migration (False) - usually because the migration is adding a non-nullable field to a table and needs a default value to put into existing rows. It does not affect the behaviour of string defaults in the database directly - Django never sets database defaults and always applies them in the Django ORM code.

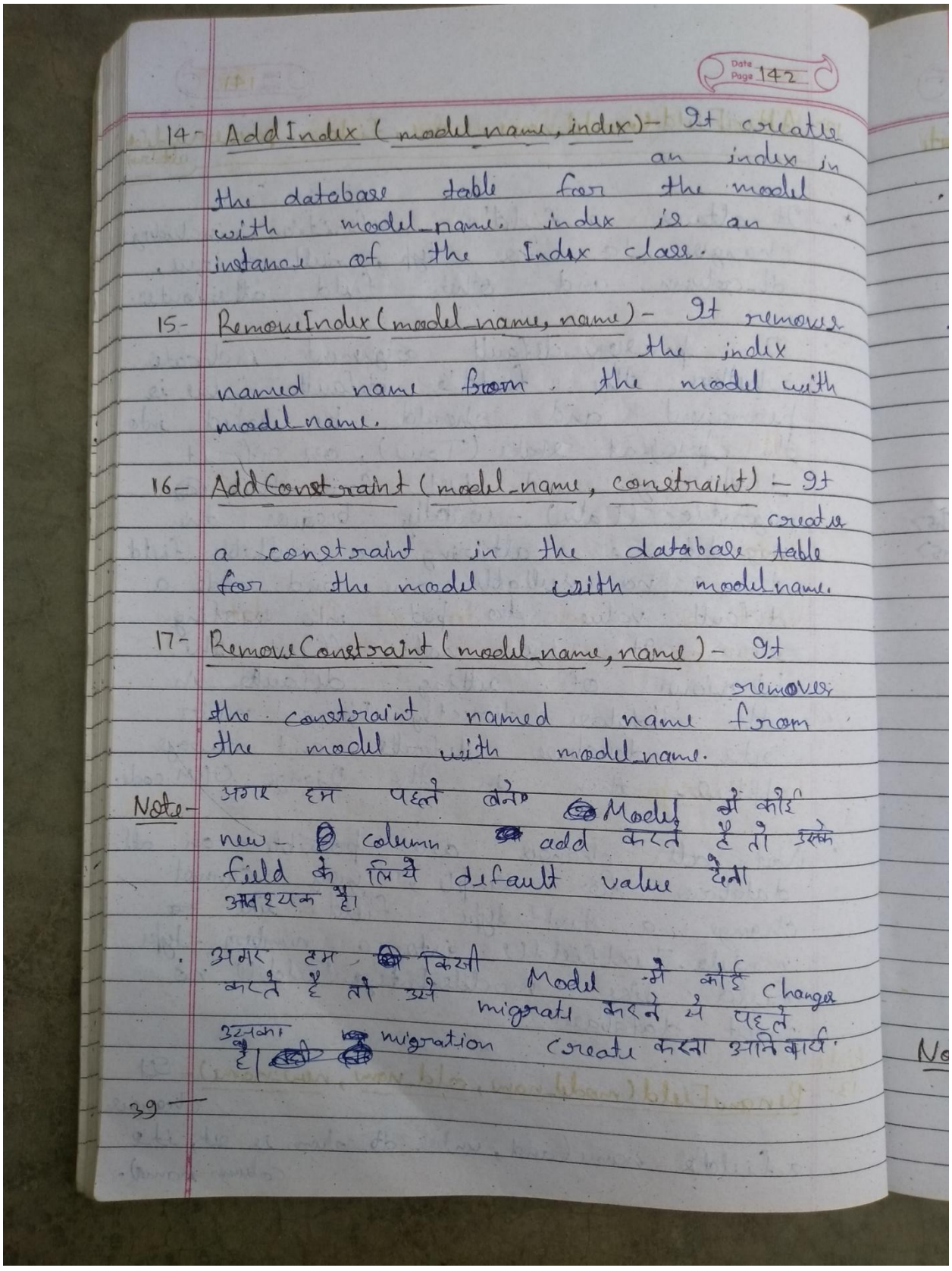
11- RemoveField (model_name, name) - It removes a field from a model.

- Bear in mind that when reversed, this is actually adding a field to a model. The operation is reversible (apart from any data loss, which of course is irreversible) if the field is nullable or if it has a default value that can be used to populate the recreated column. If the field is not nullable and does not have a default value, the operation is irreversible.

12- AlterField(model_name, name, field, preserve_default=True)

- It alters a field's definition, including changes to its type, null, unique, db_column and other field attributes.
- The preserve_default argument indicates whether the field's default value is permanent and should be baked into the project state (True), or if it temporary and just for this migration (False) - usually because the migration is altering a nullable field to a non-nullable one and needs a default value to put into existing rows. It does not affect the behaviour of setting defaults in the database directly - Django never sets database defaults and always applies them in the Django ORM code.
- Not all changes are possible on all databases - for example, you cannot change a text-type field like models.TextField() into a number-type field like models.IntegerField() no most database.

13- RenameField(model_name, old_name, new_name) - It changes a field's name (and, unless db_column is set, its column name).



Date Page 142

14- AddIndex (model_name, index) - It creates an index in the database table for the model with model_name. index is an instance of the Index class.

15- RemoveIndex (model_name, name) - It removes the index named name from the model with model_name.

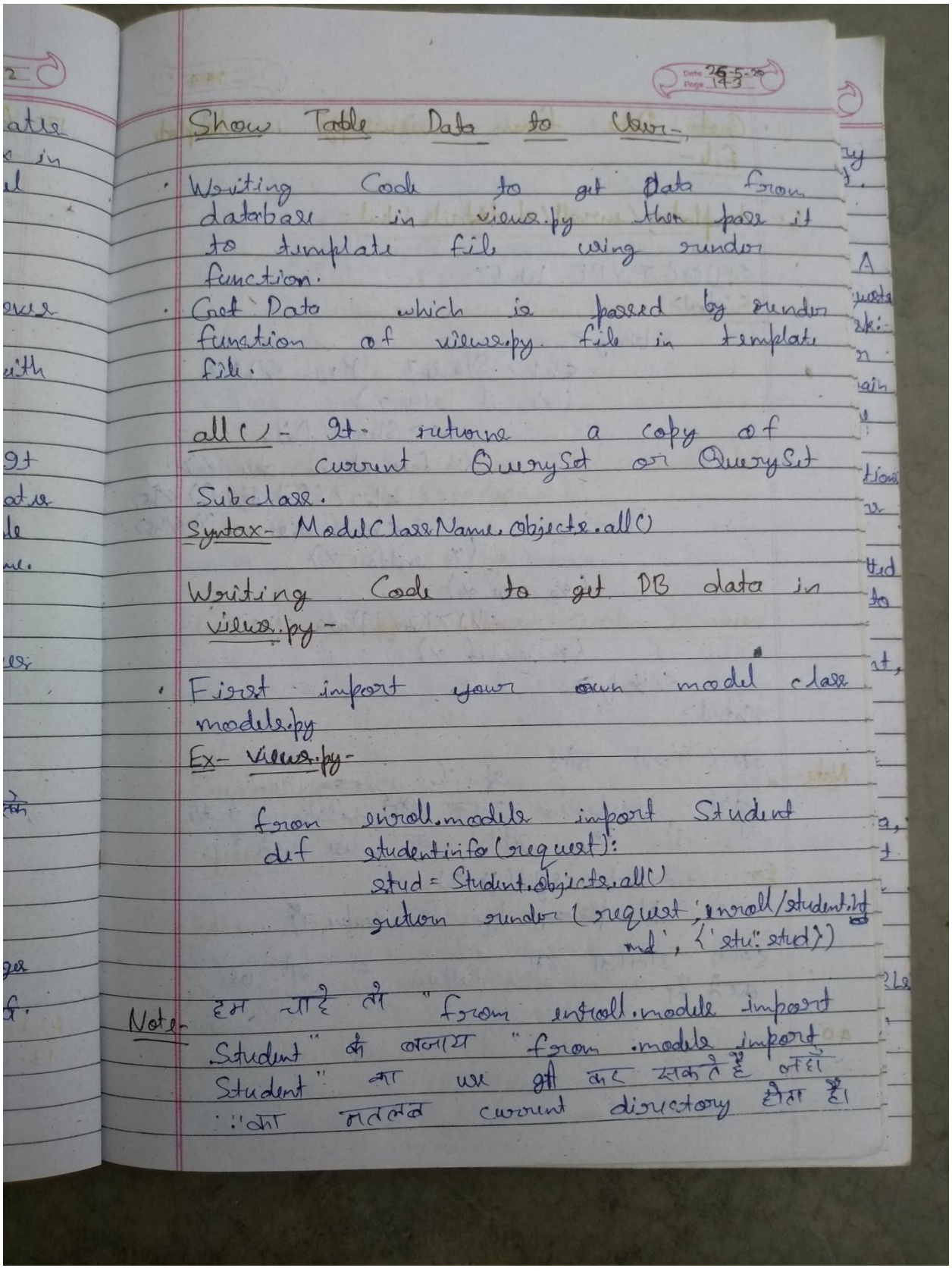
16- AddConstraint (model_name, constraint) - It creates a constraint in the database table for the model with model_name.

17- RemoveConstraint (model_name, name) - It removes the constraint named name from the model with model_name.

Note- अगर हम पहले कोई Model में कोई new column add करते है तो उसके field के लिये default value देना आवश्यक है।

अगर हम किसी Model में कोई changes करते है तो उसे migrate करने से पहले migration create करना आवश्यक है।

39 —



Show Table Data to User-

- Writing Code to get data from database in views.py then pass it to template file using render function.
- Get Data which is passed by render function of views.py file in template file.

all() - It returns a copy of current QuerySet or QuerySet Subclass.

Syntax- ModelClassName.objects.all()

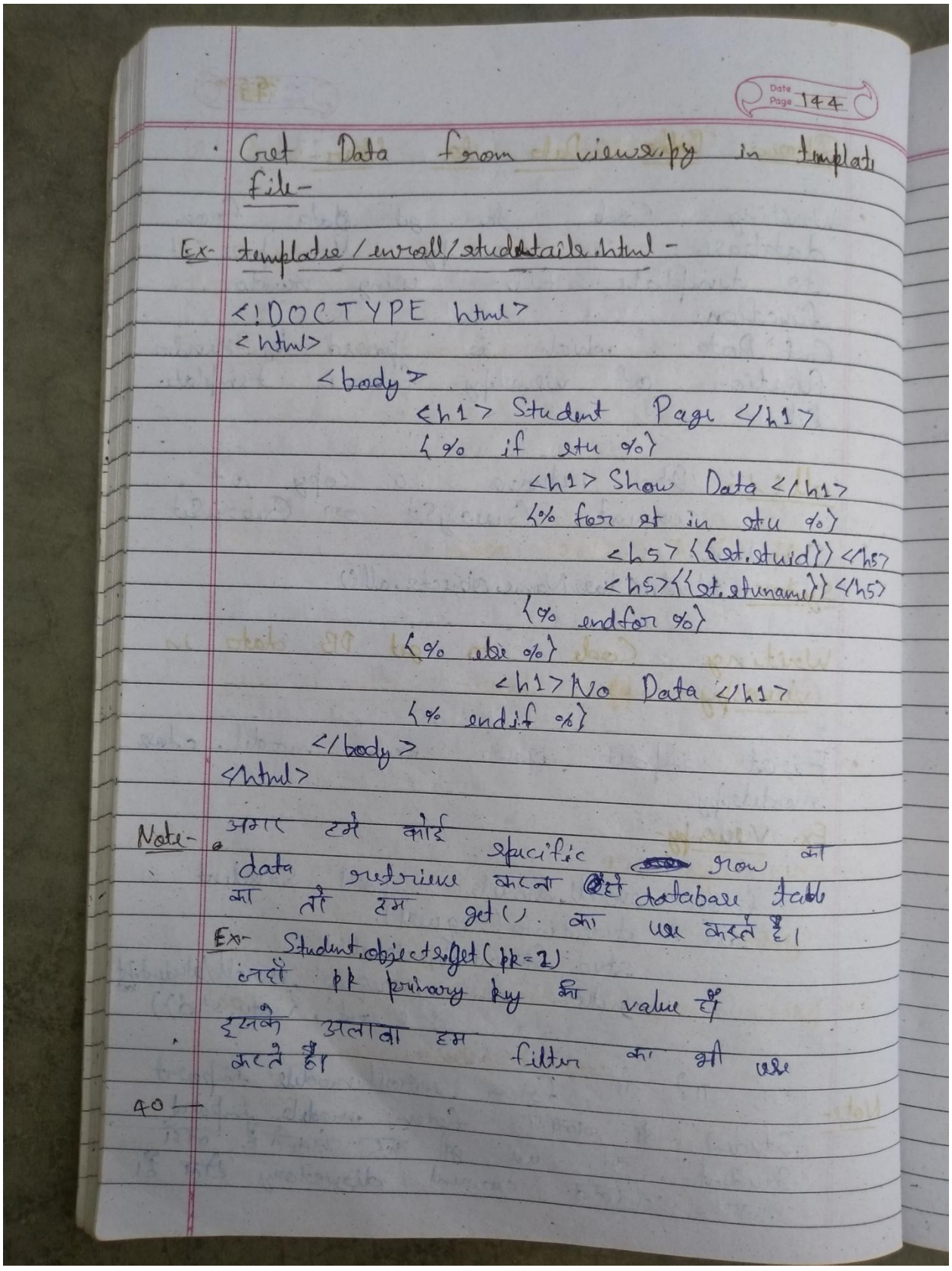
Writing Code to get DB data in views.py -

- First import your own model class models.py

Ex- views.py -

```
from enroll.models import Student
def studentinfo(request):
    stud = Student.objects.all()
    return render(request, 'enroll/student.html', {'stu': stud})
```

Note- हम चाहे तो "from enroll.models import Student" के बजाय "from .models import Student" का use भी कर सकते है यहां '.' का मतलब current directory होता है।



Date Page 144

- Get Data from views.py in template file -

Ex- templates / enroll / studdetails.html -

```
<!DOCTYPE html>
<html>
    <body>
        <h1> Student Page </h1>
        {% if stu %}
            <h1> Show Data </h1>
            {% for st in stu %}
                <h5> {{st.stuid}} </h5>
                <h5> {{st.stuname}} </h5>
            {% endfor %}
        {% else %}
            <h1> No Data </h1>
        {% endif %}
    </body>
</html>
```

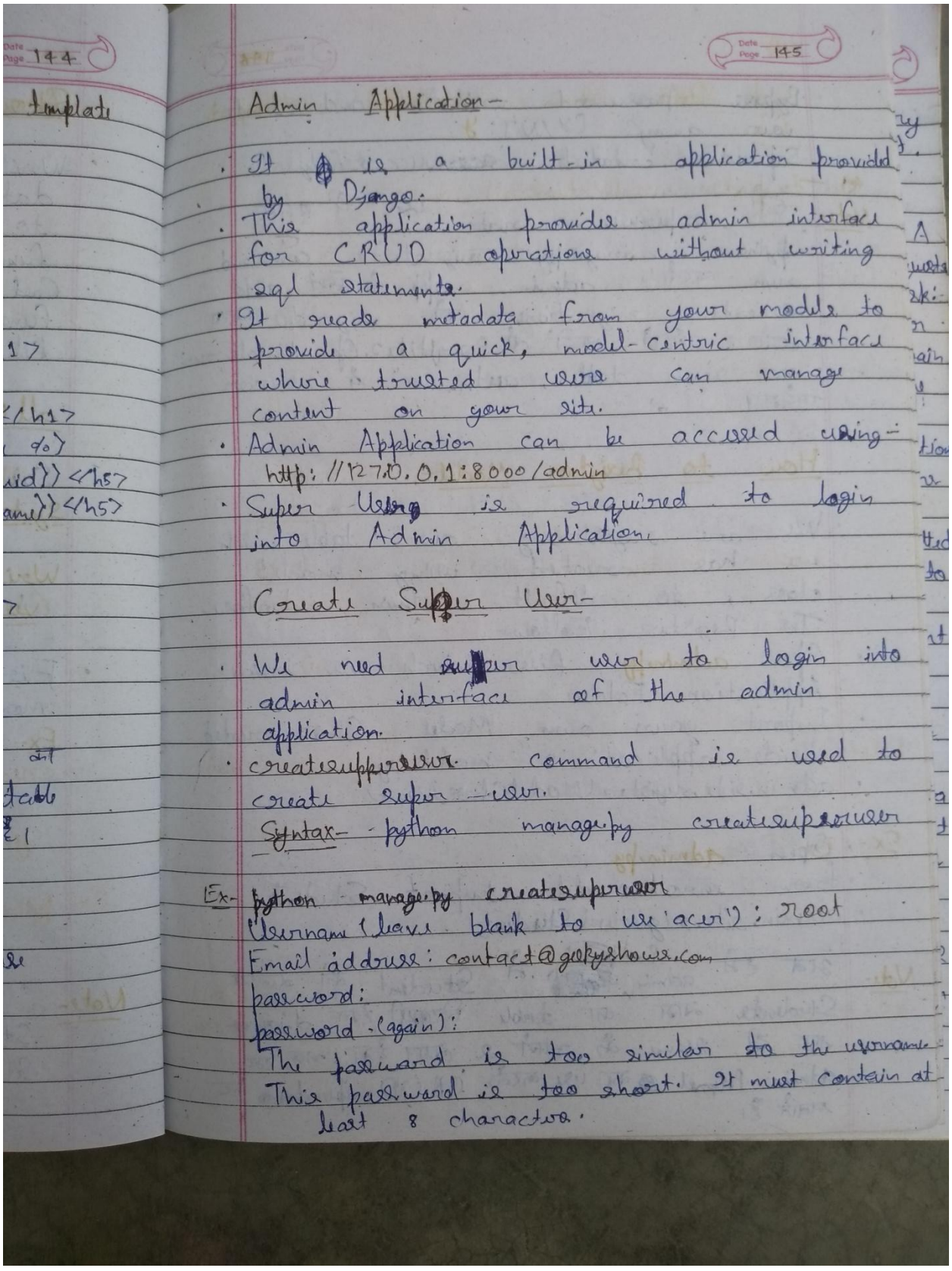
Note- अगर हमें कोई specific row का data retrieve करना है database table का तो हम get() का use करते है।

Ex- Student.objects.get(pk=2)

जहाँ pk primary key का value है

इसके अलावा हम filter का भी use करते है।

40 —



Admin Application –

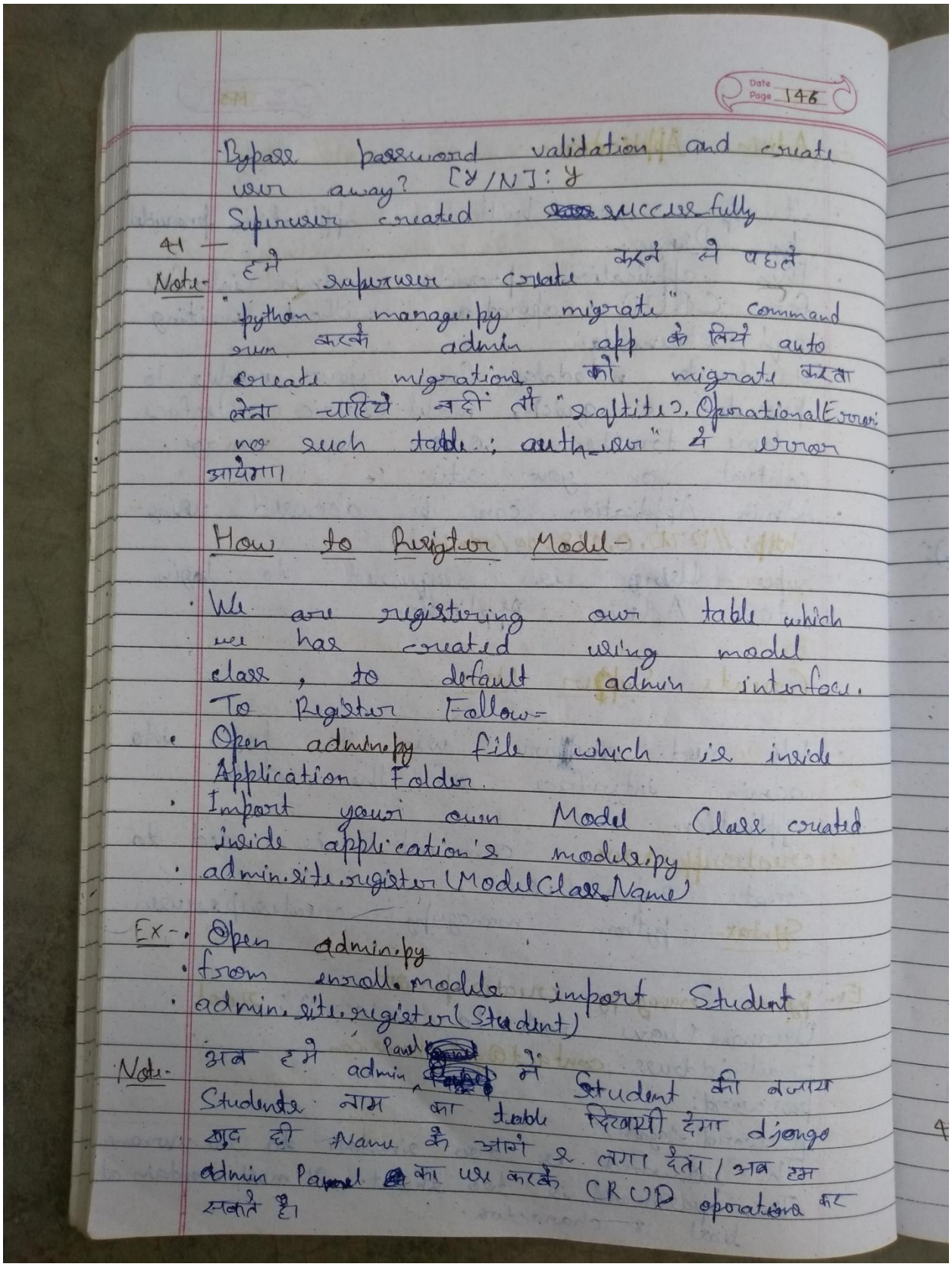
- It is a built-in application provided by Django.
- This application provides admin interface for CRUD operations without writing sql statements.
- It reads metadata from your models to provide a quick, model-centric interface where trusted users can manage content on your site.
- Admin Application can be accessed using :- http://127.0.0.1:8000/admin
- Super User is required to login into Admin Application.

Create Super User –

- We need super user to login into admin interface of the admin application.
- createsuperuser command is used to create super user.

Syntax – python manage.py createsuperuser

Ex- python manage.py createsuperuser
Username (leave blank to use 'acer'): root
Email address: contact@geekyshows.com
password:
password (again):
The password is too similar to the username.
This password is too short. It must contain at least 8 characters.



Bypass password validation and create user anyway? [Y/N]: Y
Superuser created successfully

41 —

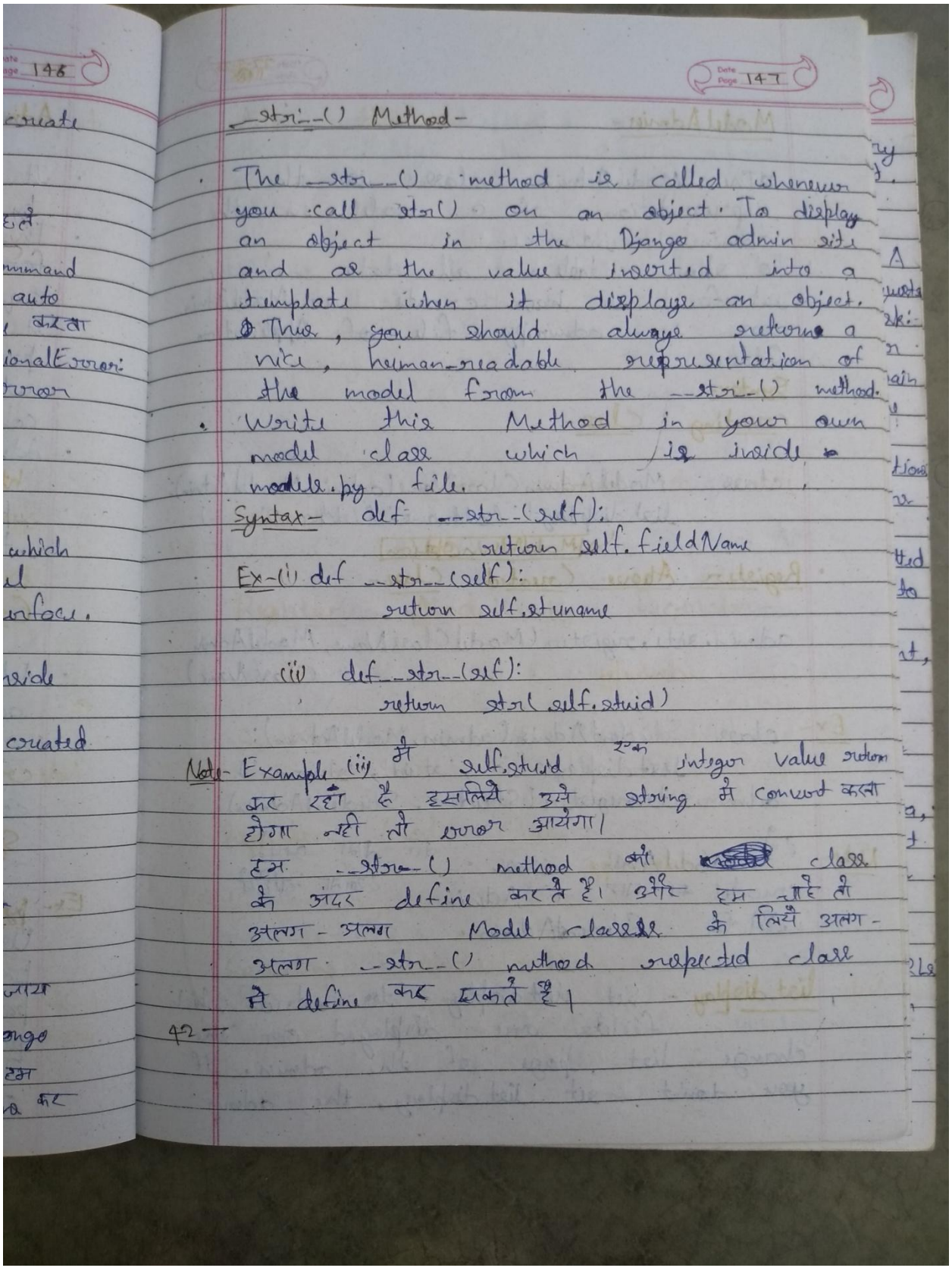
Note- हमें superuser create करने से पहले "python manage.py migrate" command run करनी admin app के लिये auto create migrations को migrate करना होना चाहिये नहीं तो "sqlite3.OperationalError: no such table: auth_user" ये error आयेगा।

How to Register Model-

- We are registering our table which we has created using model class, to default admin interface.
- To Register Follow-
- Open admin.py file which is inside Application Folder.
- Import your own Model Class created inside application's models.py
- admin.site.register(ModelClassName)

Ex- • Open admin.py
• from enroll.models import Student
• admin.site.register(Student)

Note- अब हमें admin Panel में Student की बजाय Students नाम का table दिखायी देगा django खुद ही Name के आगे s लगा देता। अब हम admin Panel का use करके CRUD operations कर सकते है।



__str__() Method -

- The __str__() method is called whenever you call str() on an object. To display an object in the Django admin site and as the value inserted into a template when it displays an object.
- Thus, you should always return a nice, human-readable representation of the model from the __str__() method.
- Write this Method in your own model class which is inside models.py file.

Syntax - def __str__(self):
    return self.fieldName

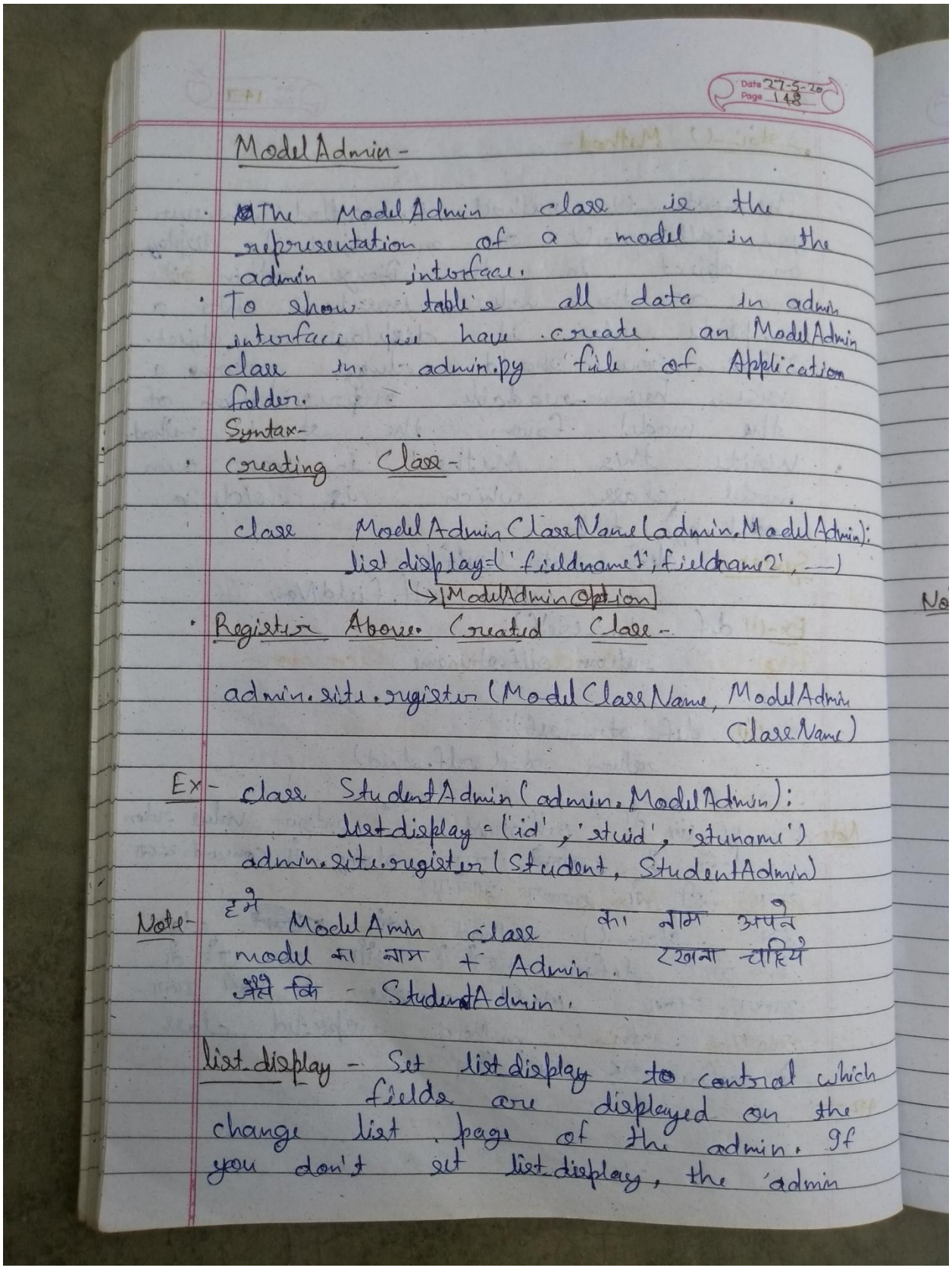
Ex-(i) def __str__(self):
    return self.stuname

(ii) def __str__(self):
    return str(self.stuid)

Note - Example (ii) में self.stuid एक integer value return कर रहा है इसलिये उसे string में convert करना होगा नहीं तो error आयेगा।

हम __str__() method को class के अंदर define करते है। और हम चाहे तो अलग-अलग Model classes के लिये अलग-अलग __str__() method respected class में define कर सकते है।

42 —



Date 27-5-20
Page 148

ModelAdmin -

- The ModelAdmin class is the representation of a model in the admin interface.
- To show table's all data in admin interface we have create an ModelAdmin class in admin.py file of Application folder.

Syntax-

- Creating Class -

```
class ModelAdminClassName(admin.ModelAdmin):
    list_display=('fieldname1','fieldname2',...)
```
↳ [ModelAdmin Option]

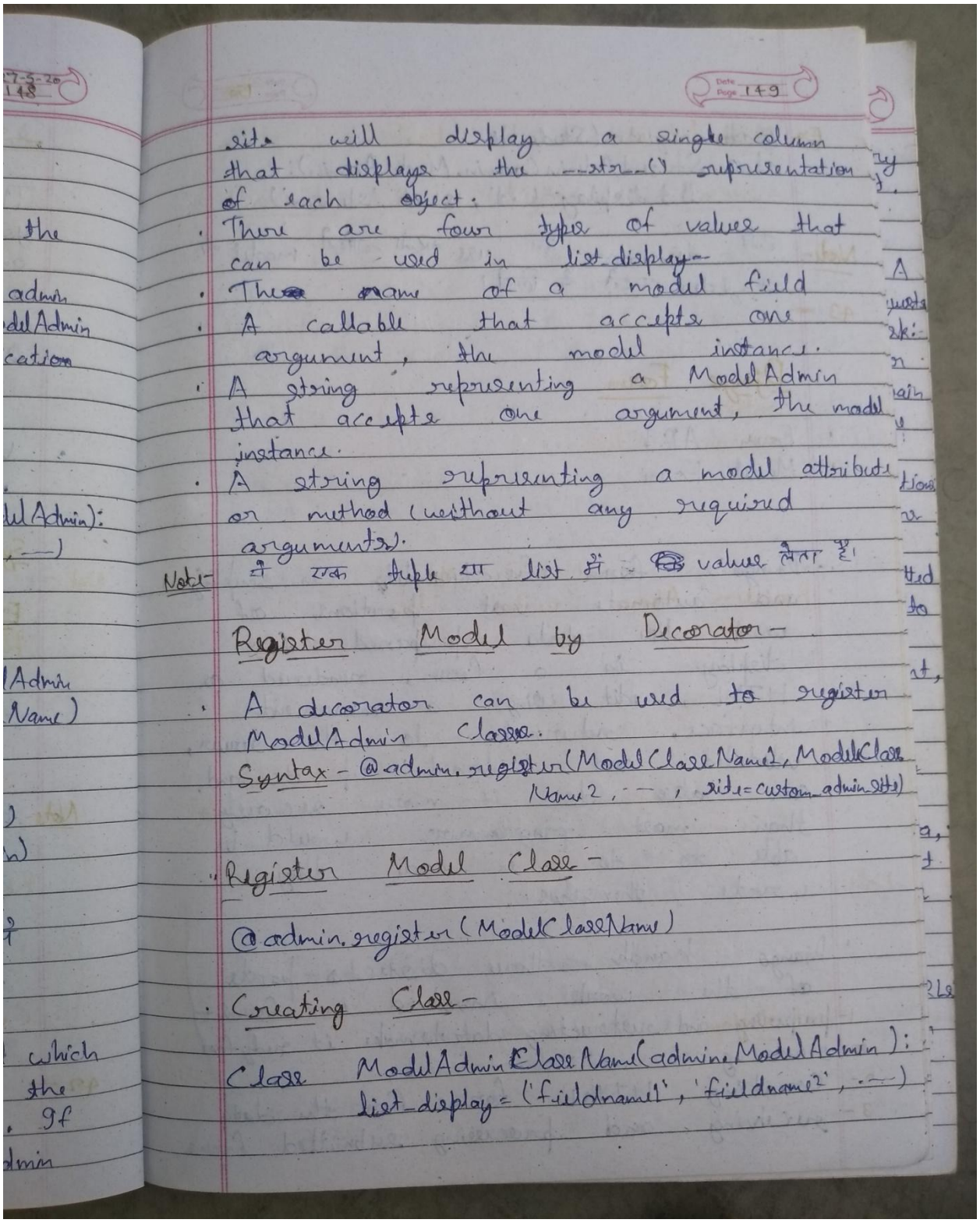
- Register Above Created Class -

```
admin.site.register(ModelClassName, ModelAdmin
                                    ClassName)
```

Ex-
```
class StudentAdmin(admin.ModelAdmin):
    list_display = ('id', 'stuid', 'stuname')
admin.site.register(Student, StudentAdmin)
```

Note- हमें ModelAmin class का नाम अपने modul का नाम + Admin रखना चाहिये जैसे कि - StudentAdmin.

list_display - Set list_display to control which fields are displayed on the change list page of the admin. If you don't set list_display, the 'admin



site will display a single column that displays the __str__() representation of each object.

- There are four type of values that can be used in list_display-
- The name of a model field
- A callable that accepts one argument, the model instance.
- A string representing a ModelAdmin that accepts one argument, the model instance.
- A string representing a model attribute or method (without any required arguments).

Note- हम एक tuple या list में value देता है।

Register Model by Decorator-

- A decorator can be used to register ModelAdmin classes.

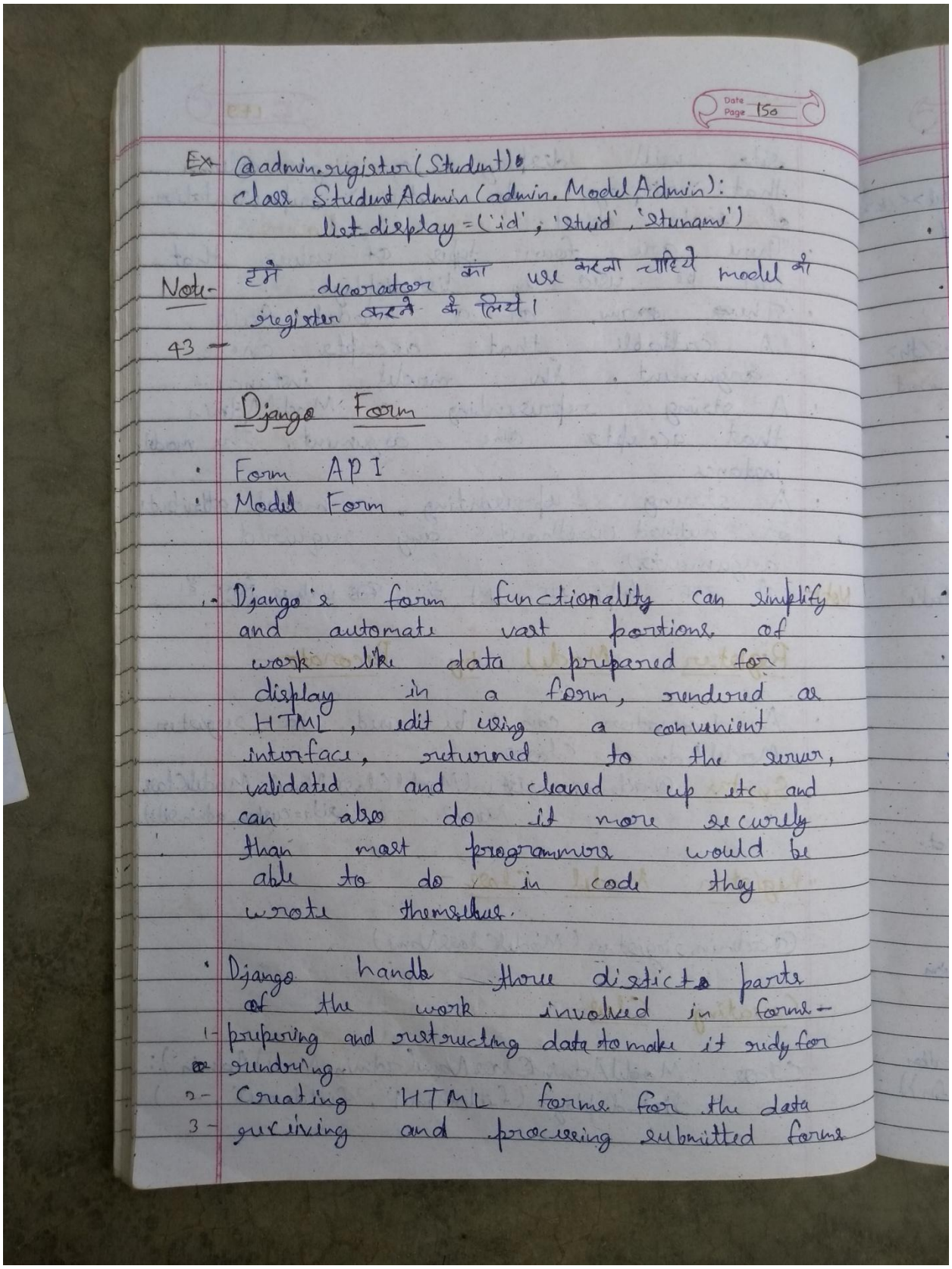
Syntax - @admin.register(ModelClassName1, ModelClassName2, --, site=custom_admin_site)

- Register Model Class-

@admin.register(ModelClassName)

- Creating Class-

Class ModelAdminClassName(admin.ModelAdmin):
    list_display = ('fieldname1', 'fieldname2', ---)



Date Page 150

Ex- @admin.register(Student)
class StudentAdmin(admin.ModelAdmin):
list_display = ('id', 'stuid', 'stuname')

Note- हमें decorator का use करना चाहिये model को register करने के लिये।

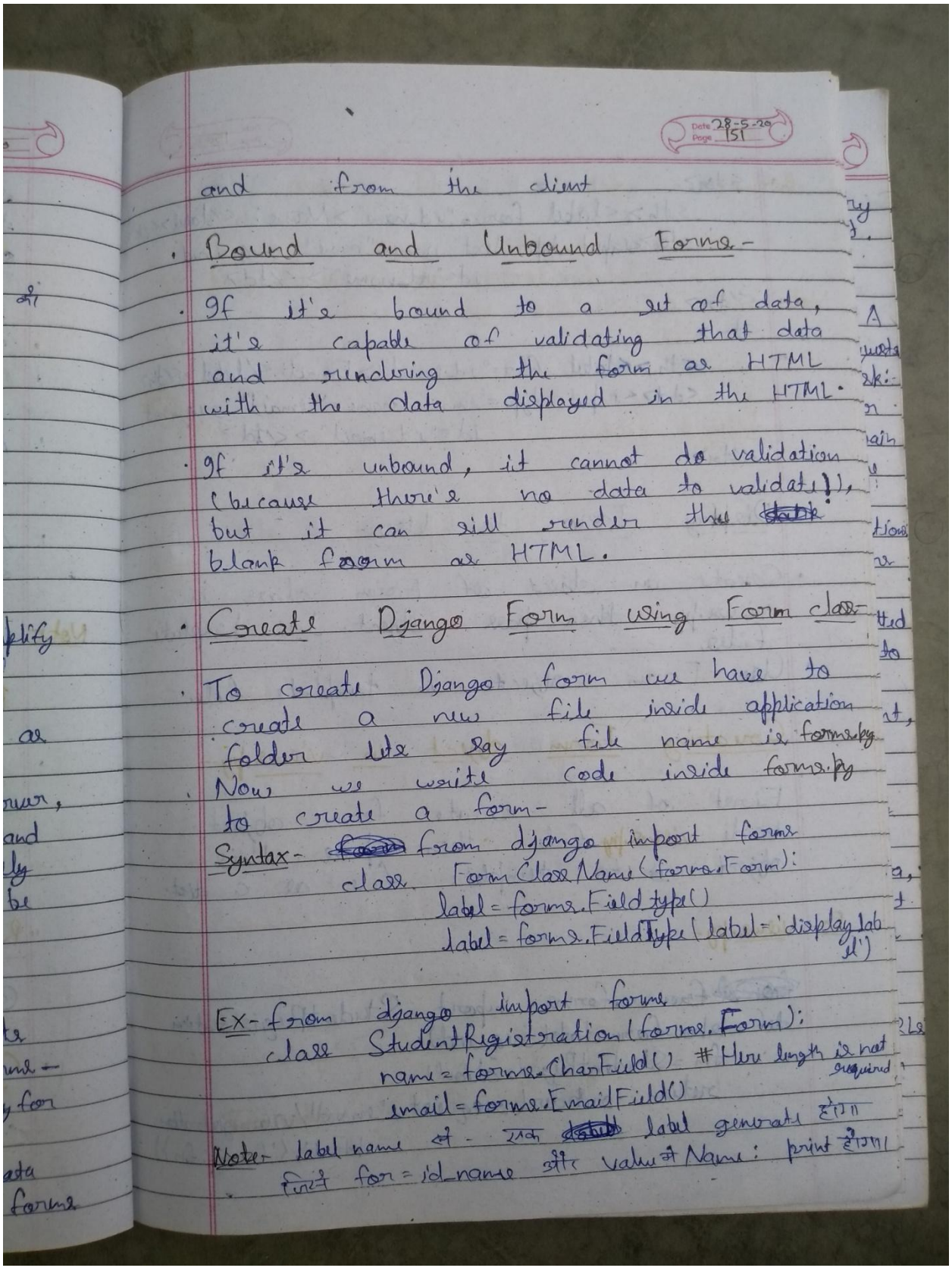
43 -

Django Form

- Form API
- Model Form

- Django's form functionality can simplify and automate vast portions of work like data prepared for display in a form, rendered as HTML, edit using a convenient interface, returned to the server, validated and cleaned up etc and can also do it more securely than most programmers would be able to do in code they wrote themselves.

- Django handle three distinct parts of the work involved in forms -
1- preparing and restructing data to make it ready for rendering.
2- Creating HTML forms for the data
3- receiving and processing submitted forms



and from the client

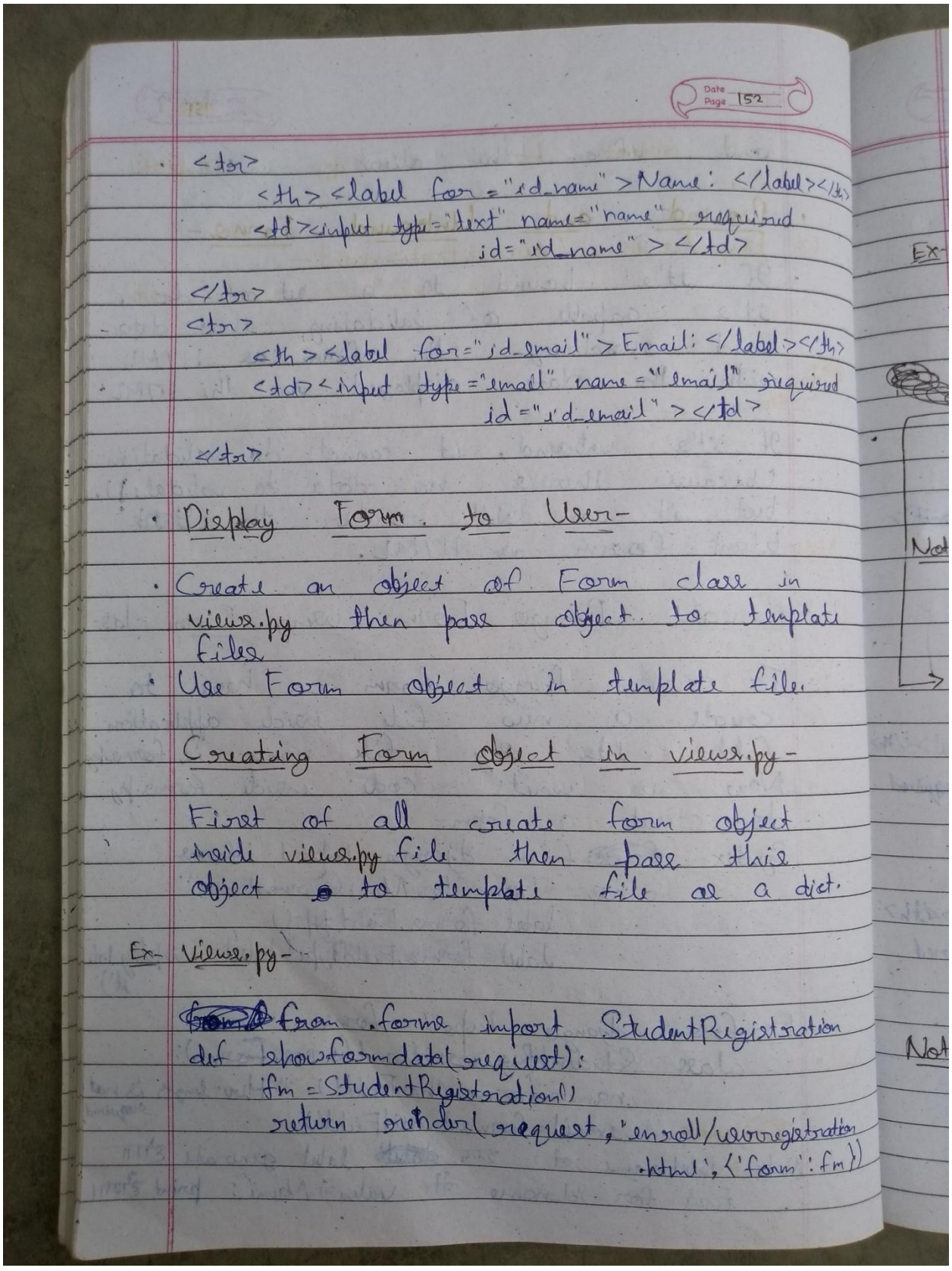
- **Bound and Unbound Forms-**
- If it's bound to a set of data, it's capable of validating that data and rendering the form as HTML with the data displayed in the HTML.
- If it's unbound, it cannot do validation (because there's no data to validate!!), but it can still render the blank form as HTML.

- **Create Django Form using Form class**
- To create Django form we have to create a new file inside application folder like say file name is forms.py
- Now we write code inside forms.py to create a form-

Syntax- from django import forms
class FormClassName(forms.Form):
label = forms.FieldType()
label = forms.FieldType(label='display label')

Ex- from django import forms
class StudentRegistration(forms.Form):
name = forms.CharField() # Here length is not required
email = forms.EmailField()

Note- label name से - स्पेस label generate होगा
first for = id_name और value = Name: print होगा।



```
<tr>
    <th><label for="id_name">Name:</label></th>
    <td><input type="text" name="name" required
              id="id_name"></td>
</tr>
<tr>
    <th><label for="id_email">Email:</label></th>
    <td><input type="email" name="email" required
              id="id_email"></td>
</tr>
```

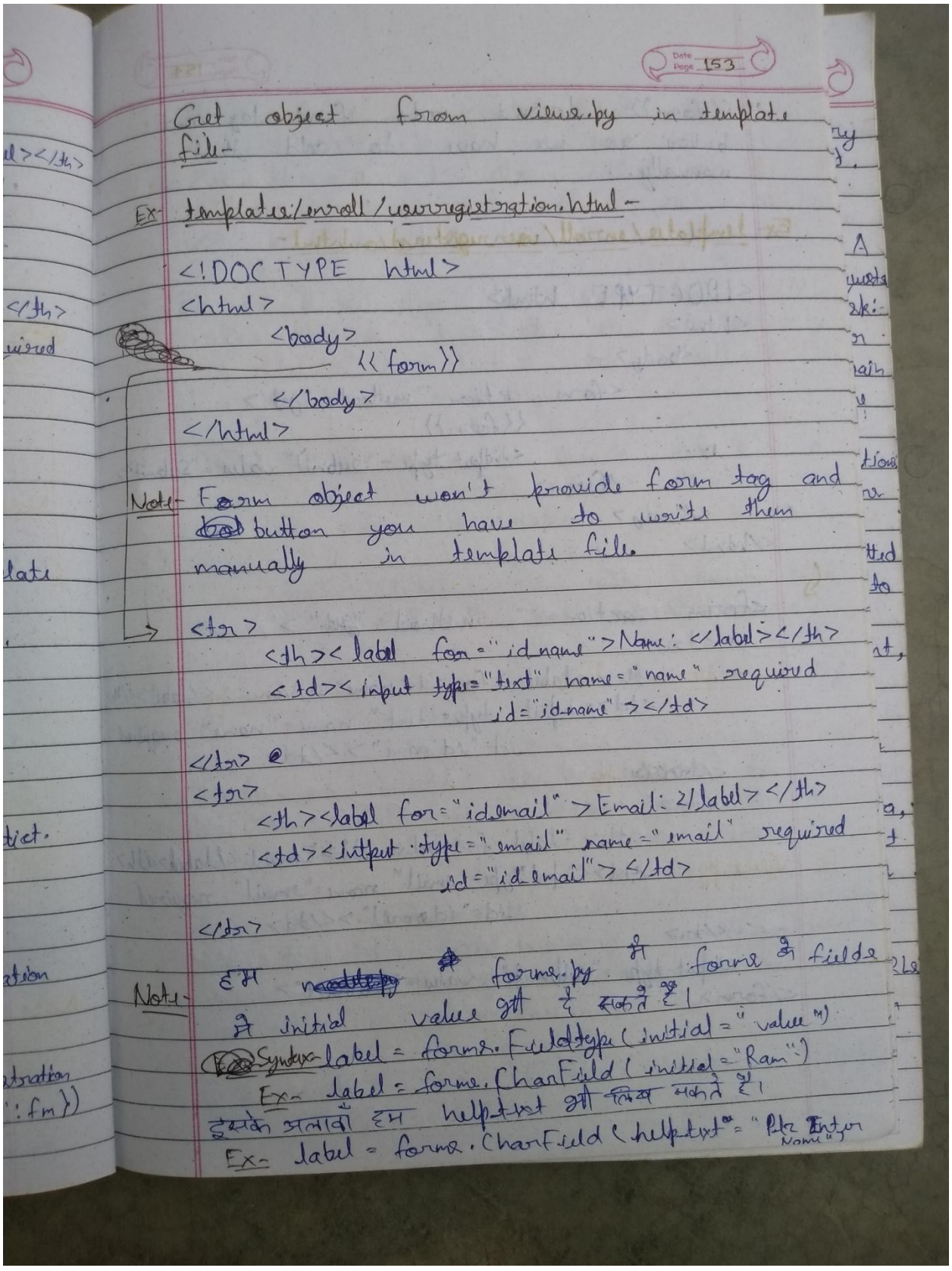
- Display Form to User-
- Create an object of Form class in views.py then pass object to template files
- Use Form object in template file.

Creating Form object in views.py -

First of all create form object inside views.py file then pass this object to template file as a dict.

Ex- views.py -

```
from .forms import StudentRegistration
def showformdata(request):
    fm = StudentRegistration()
    return render(request, 'enroll/userregistration
                  .html', {'form': fm})
```



Get object from views.py in template file-

Ex- templates/enroll/userregistration.html -

```
<!DOCTYPE html>
<html>
    <body>
        {{ form }}
    </body>
</html>
```

Note- Form object won't provide form tag and button you have to write them manually in template file.

```
<tr>
    <th><label for="id_name">Name:</label></th>
    <td><input type="text" name="name" required
        id="id_name"></td>
</tr>
<tr>
    <th><label for="id_email">Email:</label></th>
    <td><input type="email" name="email" required
        id="id_email"></td>
</tr>
```

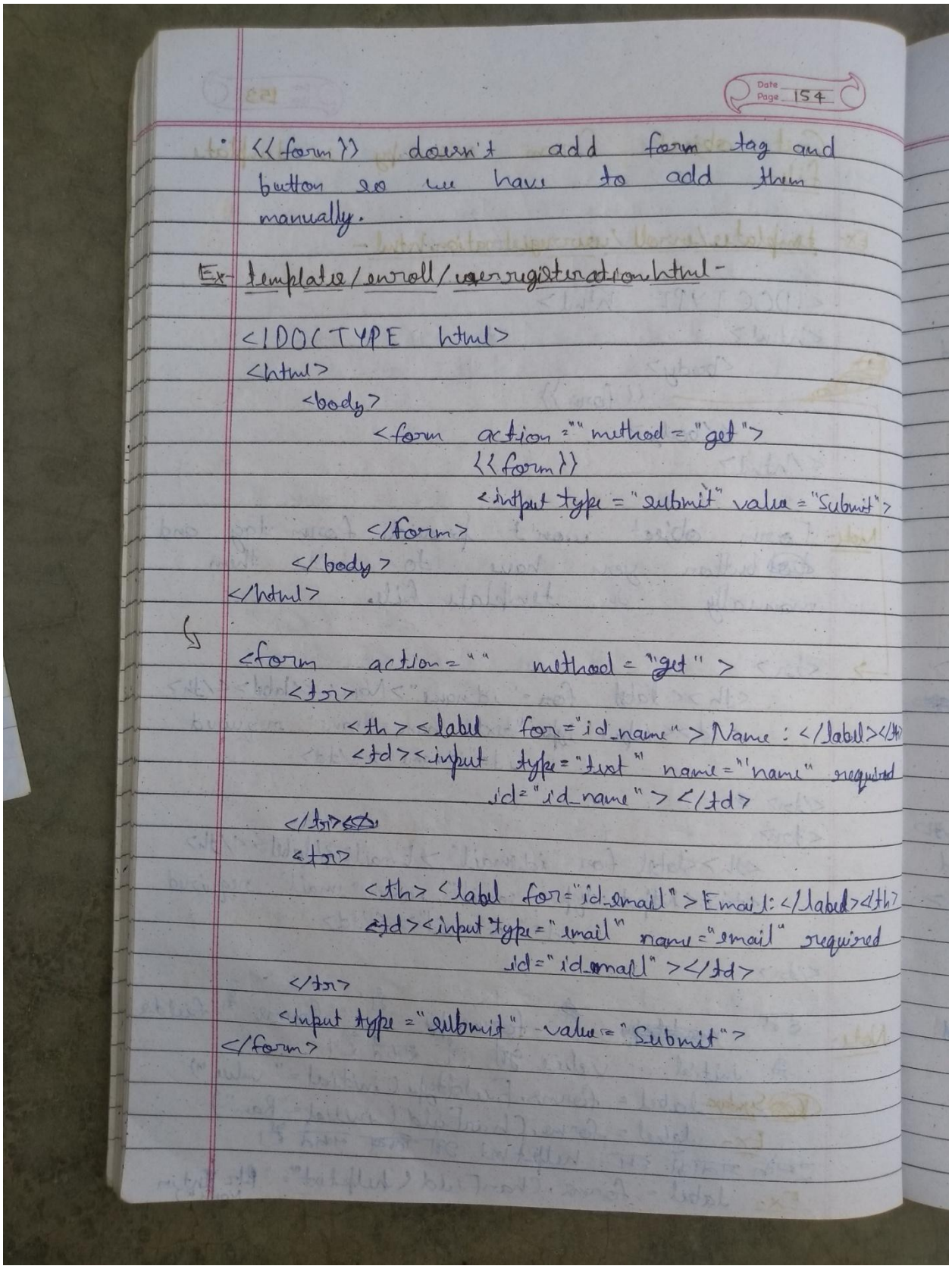
Note- हम forms.py में forms के fields में initial value भी दे सकते हैं।

Syntax- label = forms.Fieldtype(initial="value")

Ex- label = forms.CharField(initial="Ram")

इसके अलावा हम help_text भी लिख सकते हैं।

Ex- label = forms.CharField(help_text="Plz Enter Name")



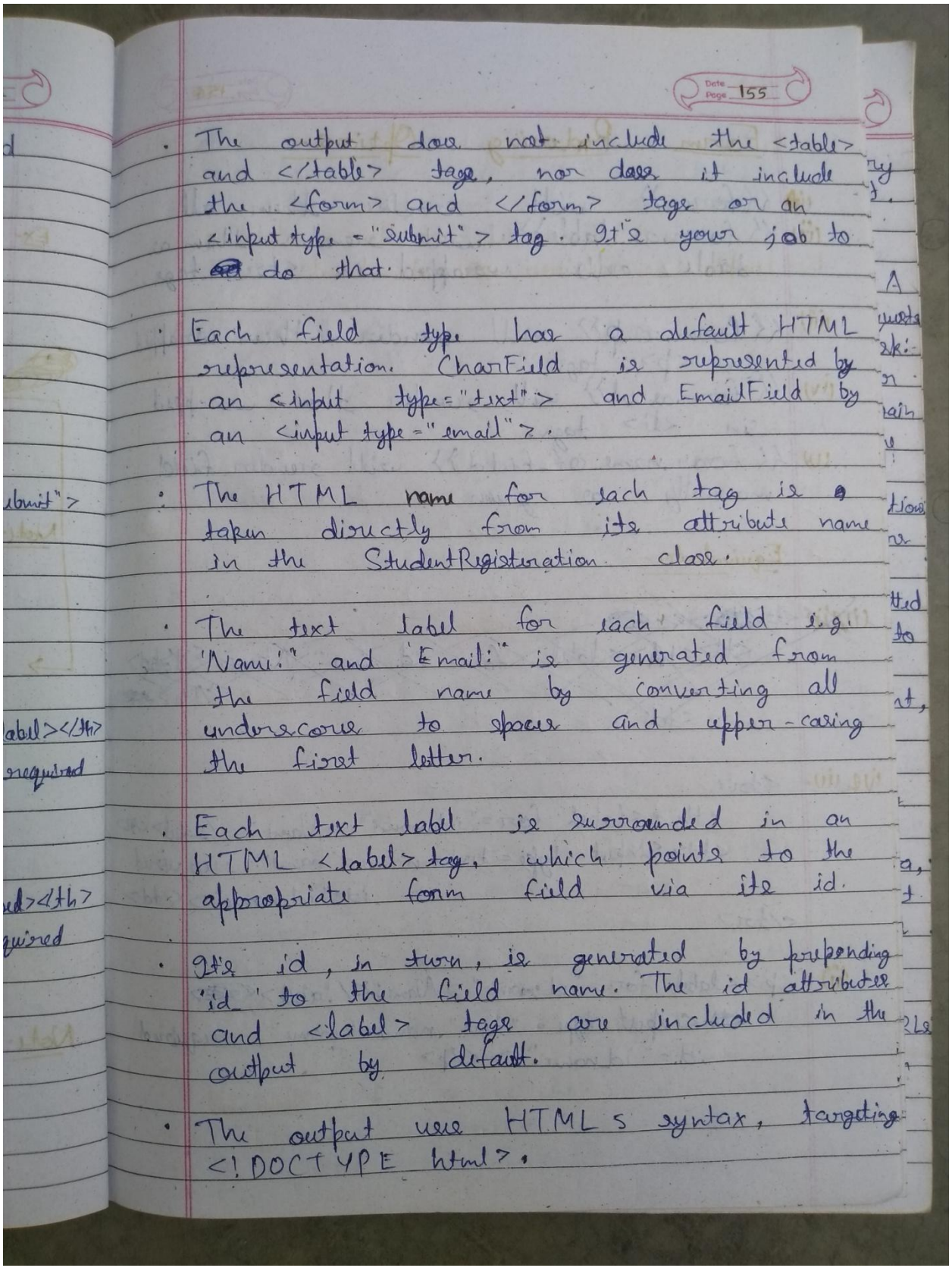
Date
Page 154

- {{form}} doesn't add form tag and button so we have to add them manually.

Ex- templates/enroll/userregistration.html -

```
<!DOCTYPE html>
<html>
    <body>
        <form action="" method="get">
            {{form}}
            <input type="submit" value="Submit">
        </form>
    </body>
</html>
```

```
<form action="" method="get">
    <tr>
        <th><label for="id_name">Name :</label></th>
        <td><input type="text" name="name" required
            id="id_name"></td>
    </tr>
    <tr>
        <th><label for="id_email">Email :</label></th>
        <td><input type="email" name="email" required
            id="id_email"></td>
    </tr>
    <input type="submit" value="Submit">
</form>
```



- The output does not include the <table> and </table> tags, nor does it include the <form> and </form> tags or an <input type="submit"> tag. It's your job to do that.
- Each field type has a default HTML representation. CharField is represented by an <input type="text"> and EmailField by an <input type="email">.
- The HTML name for each tag is taken directly from its attribute name in the StudentRegistration class.
- The text label for each field i.e. 'Name:' and 'Email:' is generated from the field name by converting all underscore to spaces and upper-casing the first letter.
- Each text label is surrounded in an HTML <label> tag, which points to the appropriate form field via its id.
- It's id, in turn, is generated by prepending 'id_' to the field name. The id attributes and <label> tags are included in the output by default.
- The output uses HTML's syntax, targeting <!DOCTYPE html>.

Date Page 156

Form Rendering Options -

(i) {{form}} will render them all.
(ii) {{ form.as_table }} will render them as table cells wrapped in <tr> tags.
(iii) {{ form.as_p }} will render them wrapped in <p> tags.
(iv) {{ form.as_ul }} will render them wrapped in <li> tags.
(v) {{ form.name_of_field }} will render field manually as given.

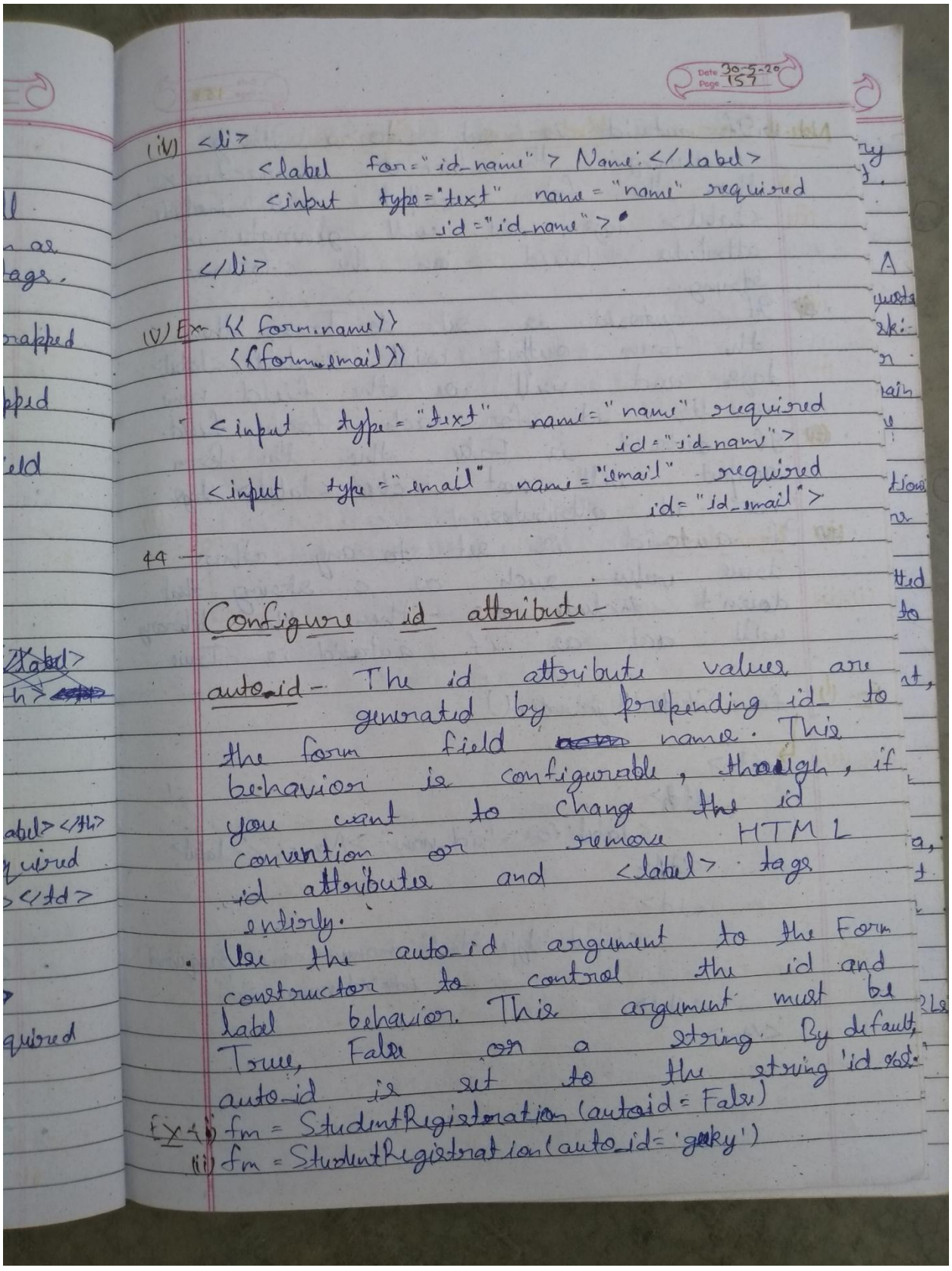
Equivalent to -

(i) & (ii) -

```
<tr>
    <th> <label for="id_name"> Name: </label> </th>
    <td> <input type="text" name="name" required
         id="id_name"> </td>
</tr>
```

(iii) -

```
<p> <label for="id_name"> Name: </label>
    <input type="text" name="name" required
    id="id_name" > </p>
```



(iv)
<label for="id_name">Name:</label>
<input type="text" name="name" required
id="id_name">

(v) Ex {{ form.name }}
{{ form.email }}

<input type="text" name="name" required
id="id_name">
<input type="email" name="email" required
id="id_email">

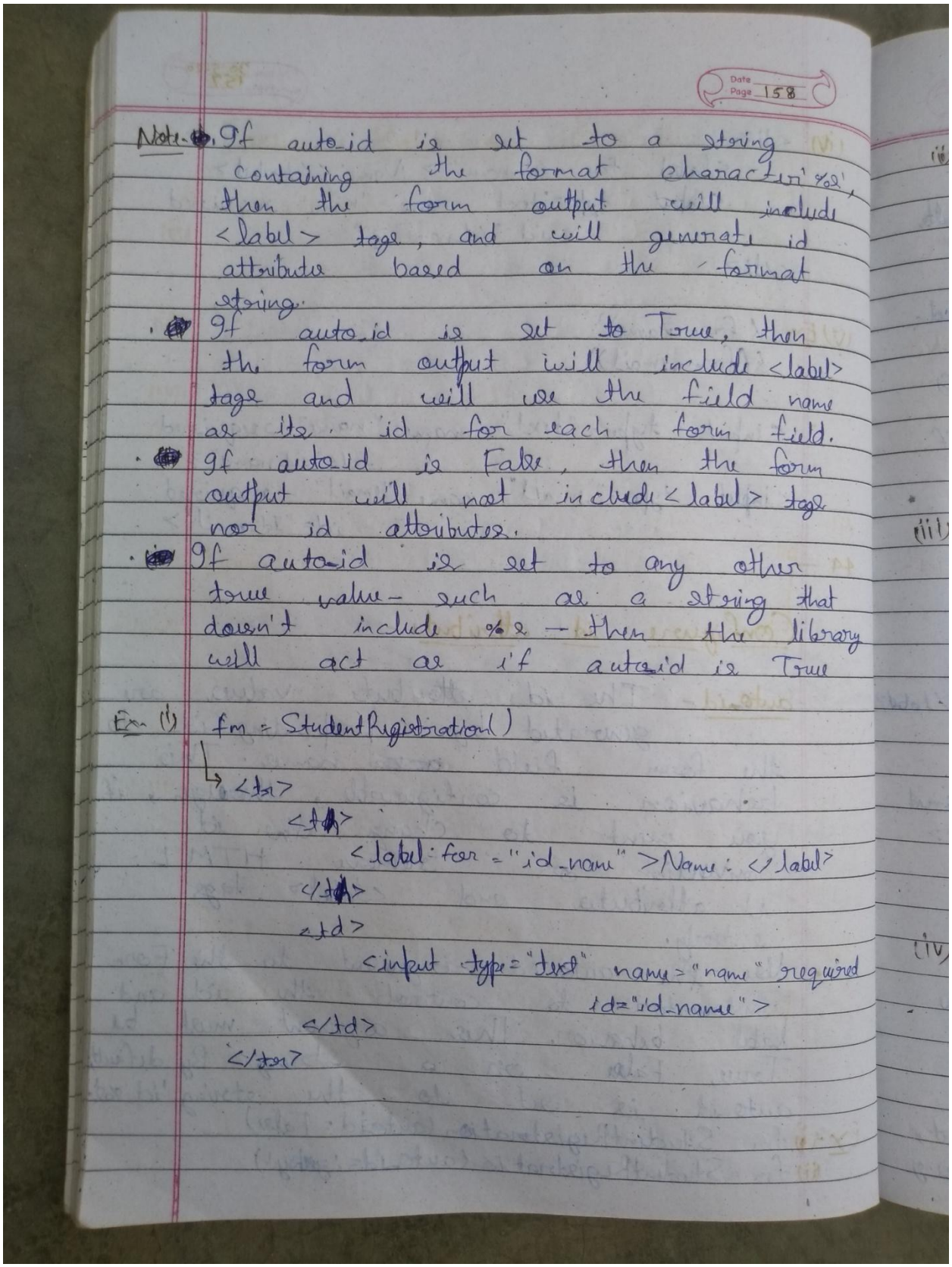
44 —

Configure id attribute -

auto_id - The id attribute values are generated by prepending id_ to the form field name. This behavior is configurable, though, if you want to change the id convention or remove HTML id attributes and <label> tags entirely.

- Use the auto_id argument to the Form constructor to control the id and label behavior. This argument must be True, False or a string. By default, auto_id is set to the string 'id_%s'.

Ex - (i) fm = StudentRegistration(autoid=False)
(ii) fm = StudentRegistration(auto_id='geeky')



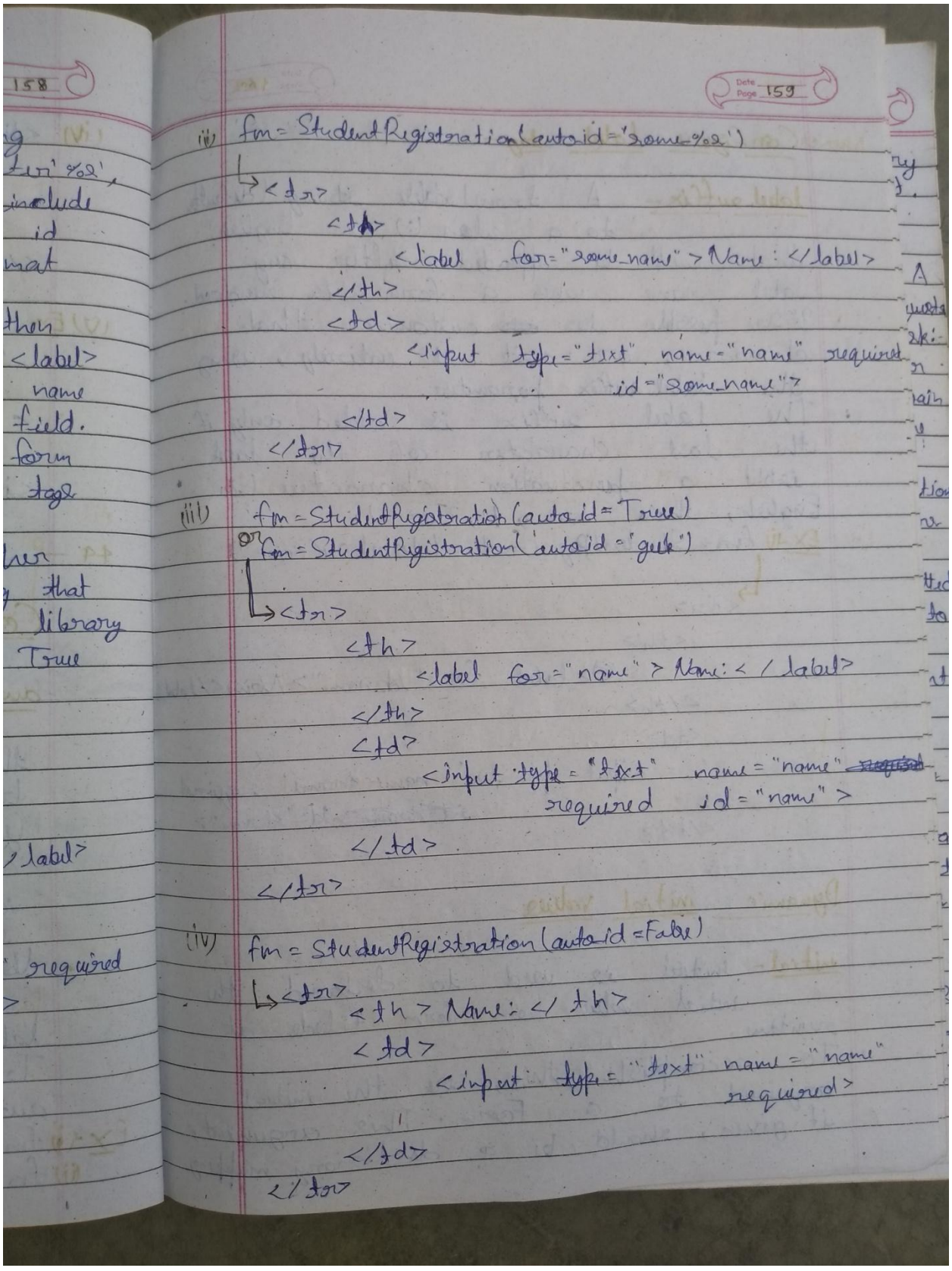
Date
Page 158

Note:- If auto_id is set to a string containing the format character '%s', then the form output will include <label> tags, and will generate id attributes based on the format string.

- If auto_id is set to True, then the form output will include <label> tags and will use the field name as its id for each form field.
- If auto_id is False, then the form output will not include <label> tags nor id attributes.
- If auto_id is set to any other true value - such as a string that doesn't include %s - then the library will act as if auto_id is True

Ex (i) fm = StudentRegistration()

↳ <tr>
<td>
<label for="id_name">Name:</label>
</td>
<td>
<input type="text" name="name" required id="id_name">
</td>
</tr>



(ii) fm = StudentRegistration(auto_id='some_%02')

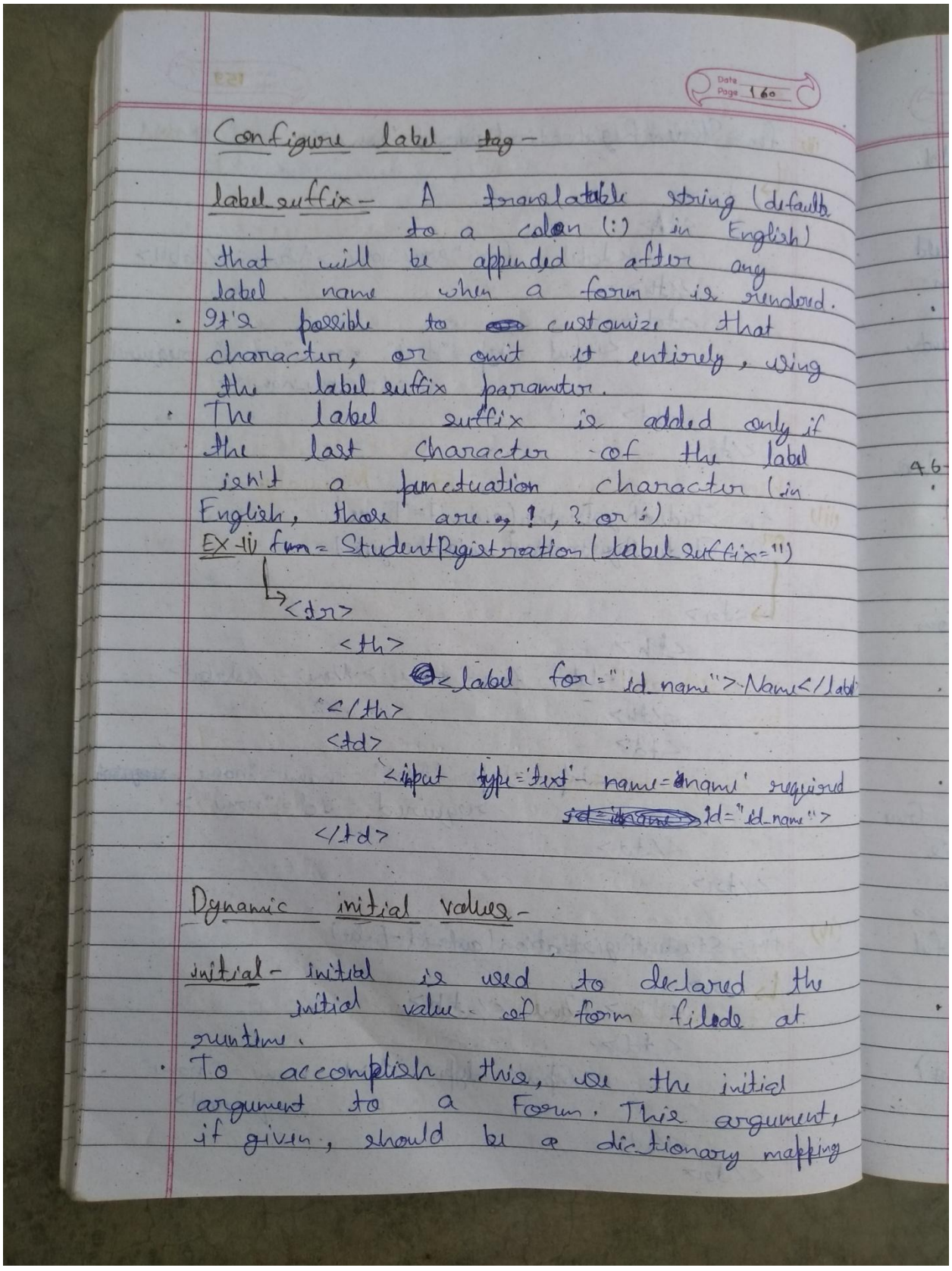
```
<tr>
    <th>
        <label for="some_name">Name :</label>
    </th>
    <td>
        <input type="text" name="name" required
               id="some_name">
    </td>
</tr>
```

(iii) fm = StudentRegistration(auto_id=True)
or fm = StudentRegistration(auto_id='geek')

```
<tr>
    <th>
        <label for="name">Name:</label>
    </th>
    <td>
        <input type="text" name="name"
               required id="name">
    </td>
</tr>
```

(iv) fm = StudentRegistration(auto_id=False)

```
<tr>
    <th>Name:</th>
    <td>
        <input type="text" name="name"
               required>
    </td>
</tr>
```



Configure label tag -

label_suffix - A translatable string (default to a colon (:) in English) that will be appended after any label name when a form is rendered.

- It's possible to customize that character, or omit it entirely, using the label_suffix parameter.
- The label_suffix is added only if the last character of the label isn't a punctuation character (in English, those are ., !, ? or :)

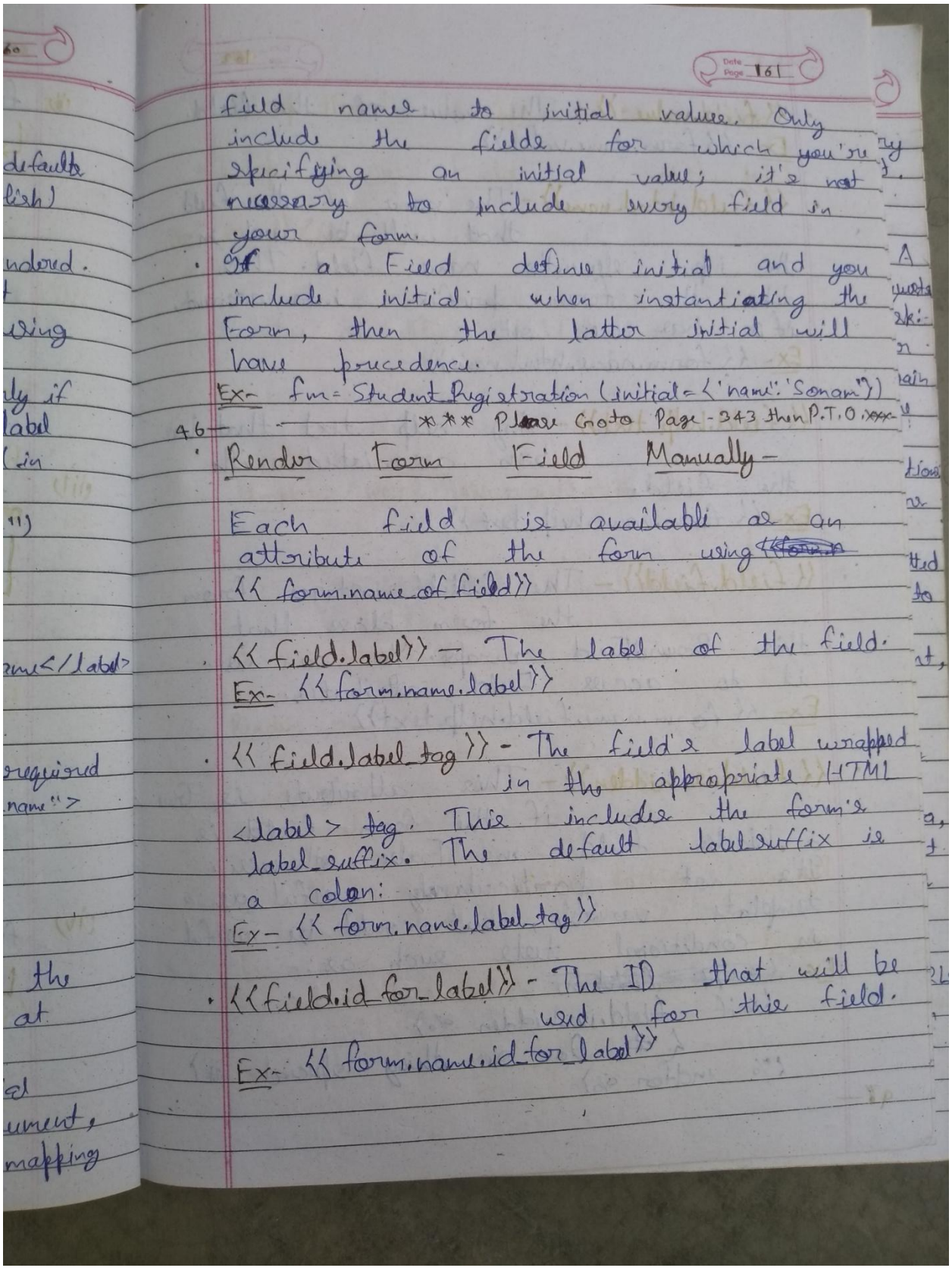
Ex - ii) fm = StudentRegistration(label_suffix="")

```
<tr>
    <th>
        <label for="id_name">Name</label>
    </th>
    <td>
        <input type='text' name='name' required id="id_name">
    </td>
```

Dynamic initial values -

initial - initial is used to declared the initial value of form fields at runtime.

- To accomplish this, use the initial argument to a Form. This argument, if given, should be a dictionary mapping



field names to initial values. Only include the fields for which you're specifying an initial value; it's not necessary to include every field in your form.

- If a Field define initial and you include initial when instantiating the Form, then the latter initial will have precedence.

Ex- fm = StudentRegistration(initial={'name':'Sonam'})

46 ——— *** Please Goto Page-343 then P.T.O. ***

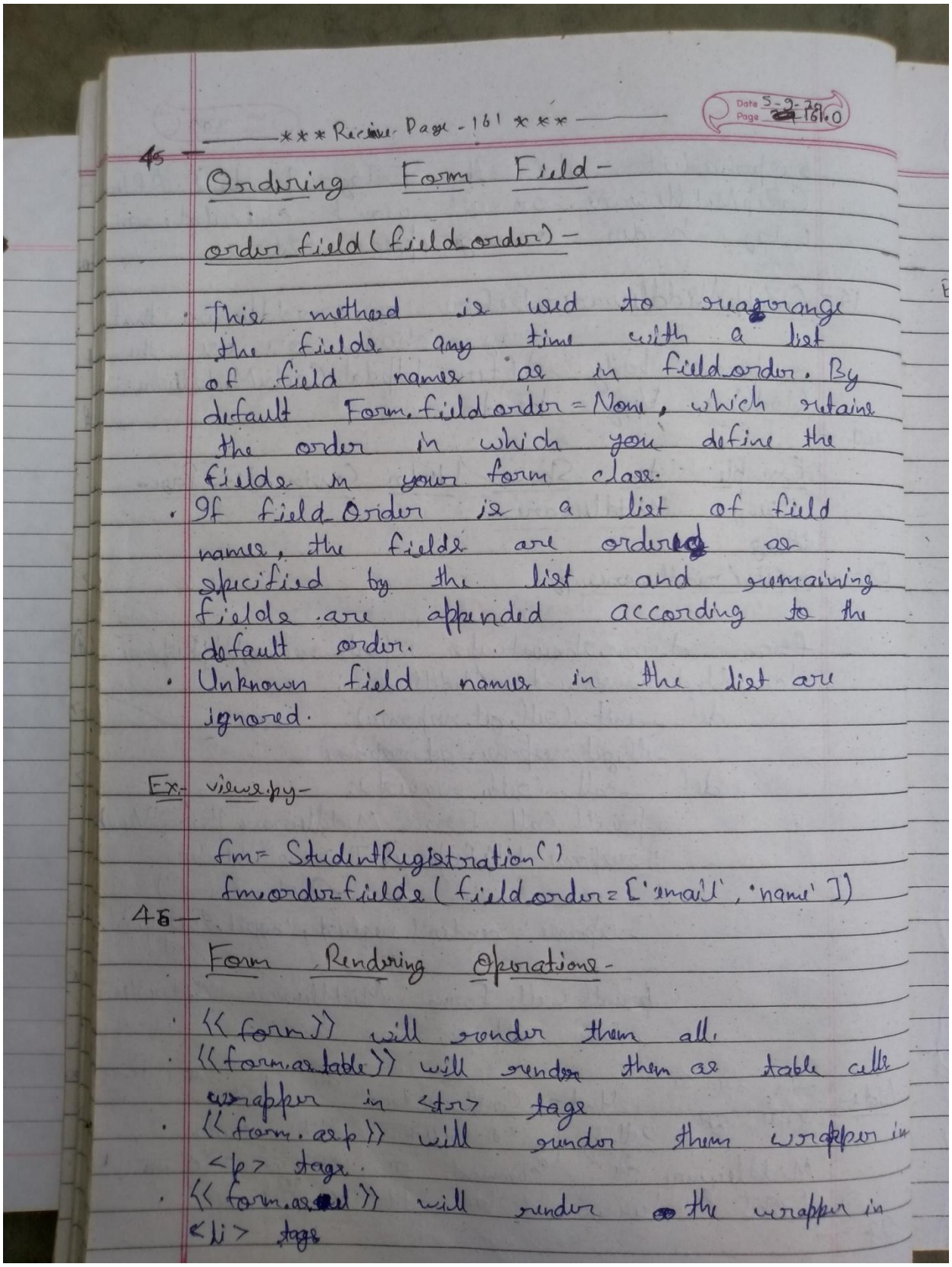
- Render Form Field Manually -

Each field is available as an attribute of the form using {{ form.name_of_field }}

- {{ field.label }} - The label of the field.
  Ex- {{ form.name.label }}

- {{ field.label_tag }} - The field's label wrapped in the appropriate HTML <label> tag. This includes the form's label_suffix. The default label_suffix is a colon:
  Ex- {{ form.name.label_tag }}

- {{ field.id_for_label }} - The ID that will be used for this field.
  Ex- {{ form.name.id_for_label }}



*** Revise Page - 161 ***

Date 5-9-21
Page 161.0

45 — Ordering Form Field -

order_field(field_order) -

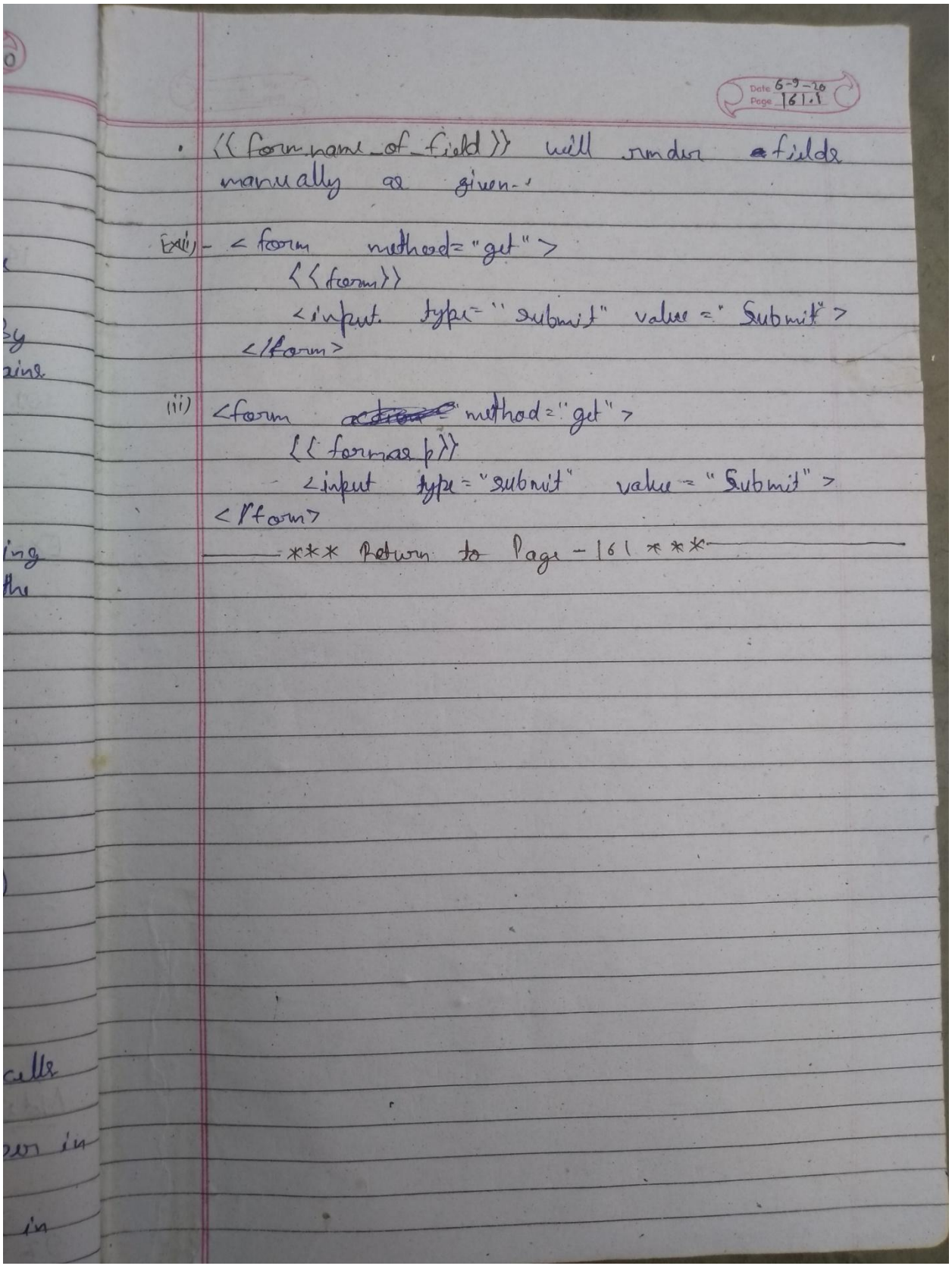
- This method is used to rearrange the fields any time with a list of field names as in field_order. By default Form.field_order = None, which retains the order in which you define the fields in your form class.
- If field_Order is a list of field names, the fields are ordered as specified by the list and remaining fields are appended according to the default order.
- Unknown field names in the list are ignored.

Ex- views.py -

```
fm = StudentRegistration()
fm.order_fields(field_order=['email', 'name'])
```

46 — Form Rendering Operations -

- {{ form }} will render them all.
- {{ form.as_table }} will render them as table cells wrapper in <tr> tags
- {{ form.as_p }} will render them wrapper in <p> tags.
- {{ form.as_ul }} will render the wrapper in <li> tags



- {{ form.name_of_field }} will render a field manually as given -

(ii) - <form method="get">
{{ form }}
<input type="submit" value="Submit">
</form>

(iii) <form method="get">
{{ form.as_p }}
<input type="submit" value="Submit">
</form>

*** Return to Page - 161 ***

field names to initial values. Only include the fields for which you're specifying an initial value; it's not necessary to include every field in your form.

- If a Field define initial and you include initial when instantiating the Form, then the latter initial will have precedence.

Ex- fm = StudentRegistration(initial={'name':'Sonam'})

*** Please Goto Page - 343 then P.T.O.

46

Render Form Field Manually -

Each field is available as an attribute of the form using {{ form.name_of_field }}

- {{ field.label }} - The label of the field.
  Ex- {{ form.name.label }}

- {{ field.label_tag }} - The field's label wrapped in the appropriate HTML <label> tag. This includes the form's label_suffix. The default label_suffix is a colon:
  Ex- {{ form.name.label_tag }}

- {{ field.id_for_label }} - The ID that will be used for this field.
  Ex- {{ form.name.id_for_label }}

Date Page 162

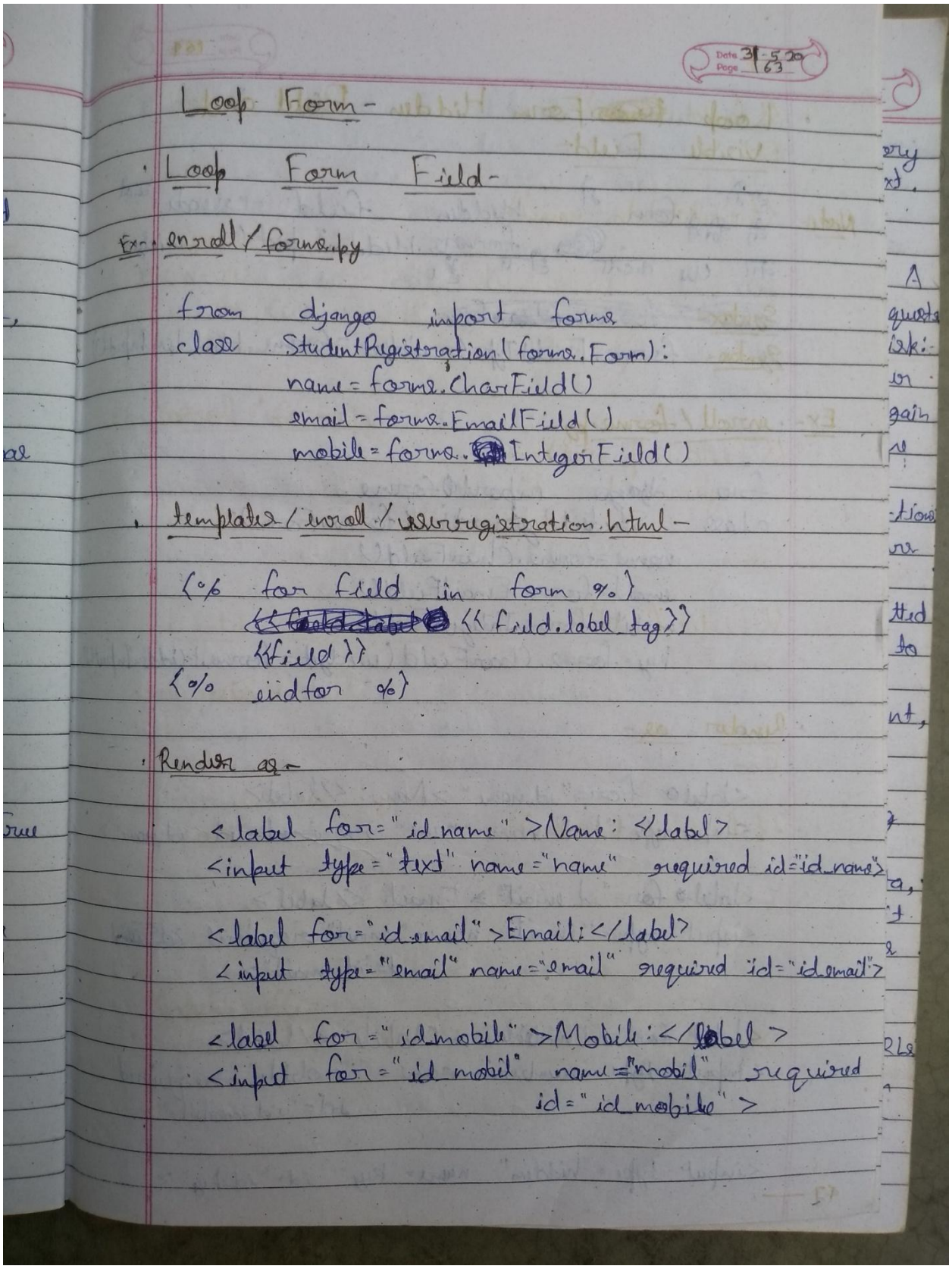
- {{ field.value }} - The value of the field.
  Ex- {{ form.name.value }}

- {{ field.html_name }} - The name of the field that will be used in the input element's name field. This takes the form prefix into account, if it has been set.
  Ex- {{ form.name.html_name }}

- {{ field.help_text }} - Any help text that has been associated with the field.
  Ex- {{ form.name.help_text }}

- {{ field.field }} - The field instance from the form class that this BoundField wraps. You can use it to access Field attributes.
  Ex- {{ form.name.field.help_text }}

- {{ field.is_hidden }} - This attribute is True if the form field is a hidden field and False otherwise. It's not particularly useful as a template variable, but could be useful in conditional tests such as:-
  Ex- {% if field.is_hidden %}
  {# Do something special #}
  {% endfor %}



Loop Form -

- Loop Form Field -

Ex - enroll / forms.py

```
from django import forms
class StudentRegistration(forms.Form):
    name = forms.CharField()
    email = forms.EmailField()
    mobile = forms.IntegerField()
```

- templates / enroll / userregistration.html -

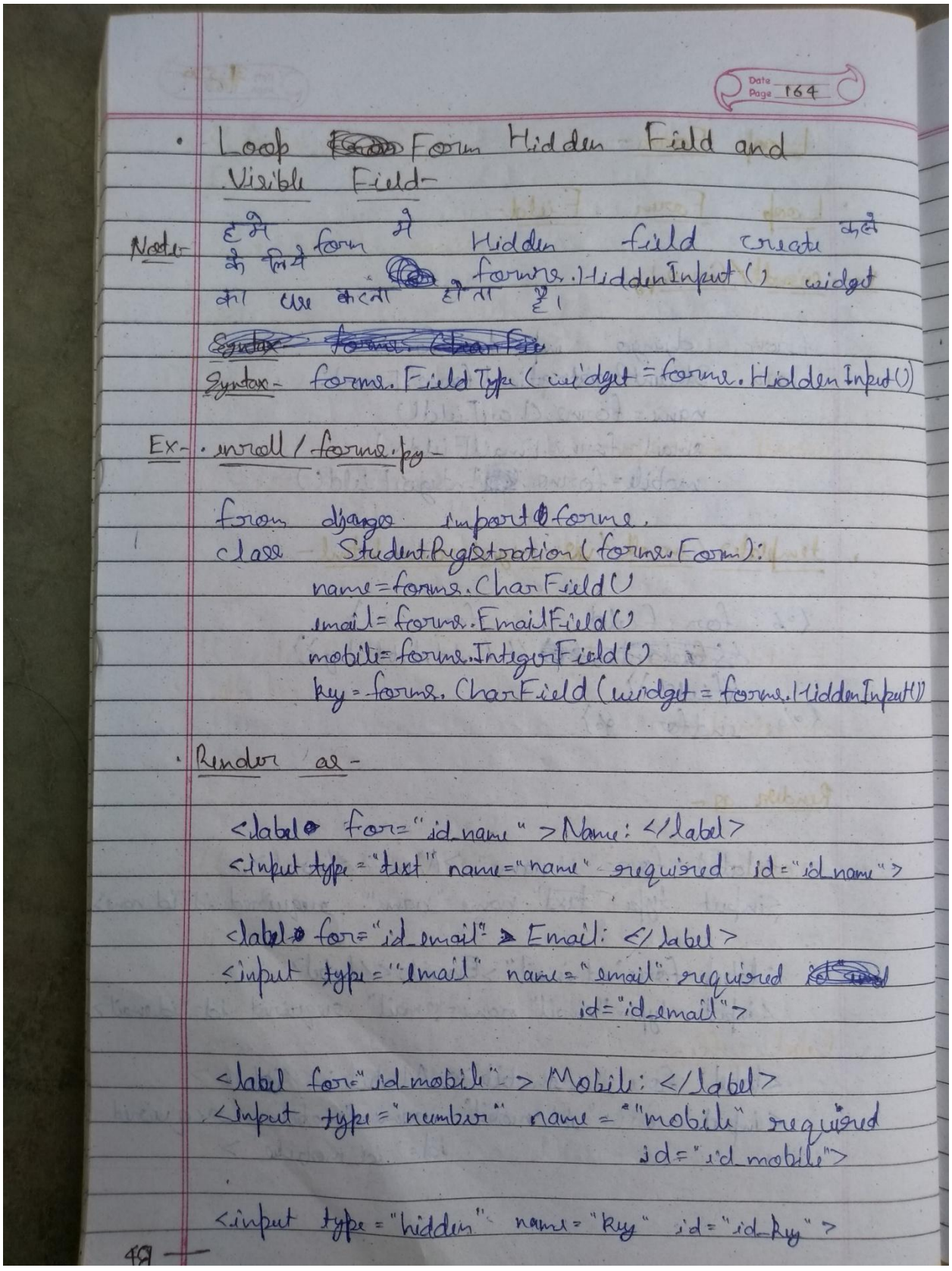
```
{% for field in form %}
    {{ field.label_tag }}
    {{ field }}
{% endfor %}
```

- Render as -

```
<label for="id_name">Name:</label>
<input type="text" name="name" required id="id_name">

<label for="id_email">Email:</label>
<input type="email" name="email" required id="id_email">

<label for="id_mobile">Mobile:</label>
<input for="id_mobil" name="mobil" required
    id="id_mobile">
```



Date Page 164

- Loop Form Hidden Field and Visible Field-

Note- हमे किसी form में Hidden field create करने के लिये forms.HiddenInput() widget का use करना होता है।

Syntax- forms.FieldType(widget=forms.HiddenInput())

Ex- · myall/forms.py-

```
from django import forms
class StudentRegistration(forms.Form):
    name=forms.CharField()
    email=forms.EmailField()
    mobile=forms.IntegerField()
    key=forms.CharField(widget=forms.HiddenInput())
```

· Render as-

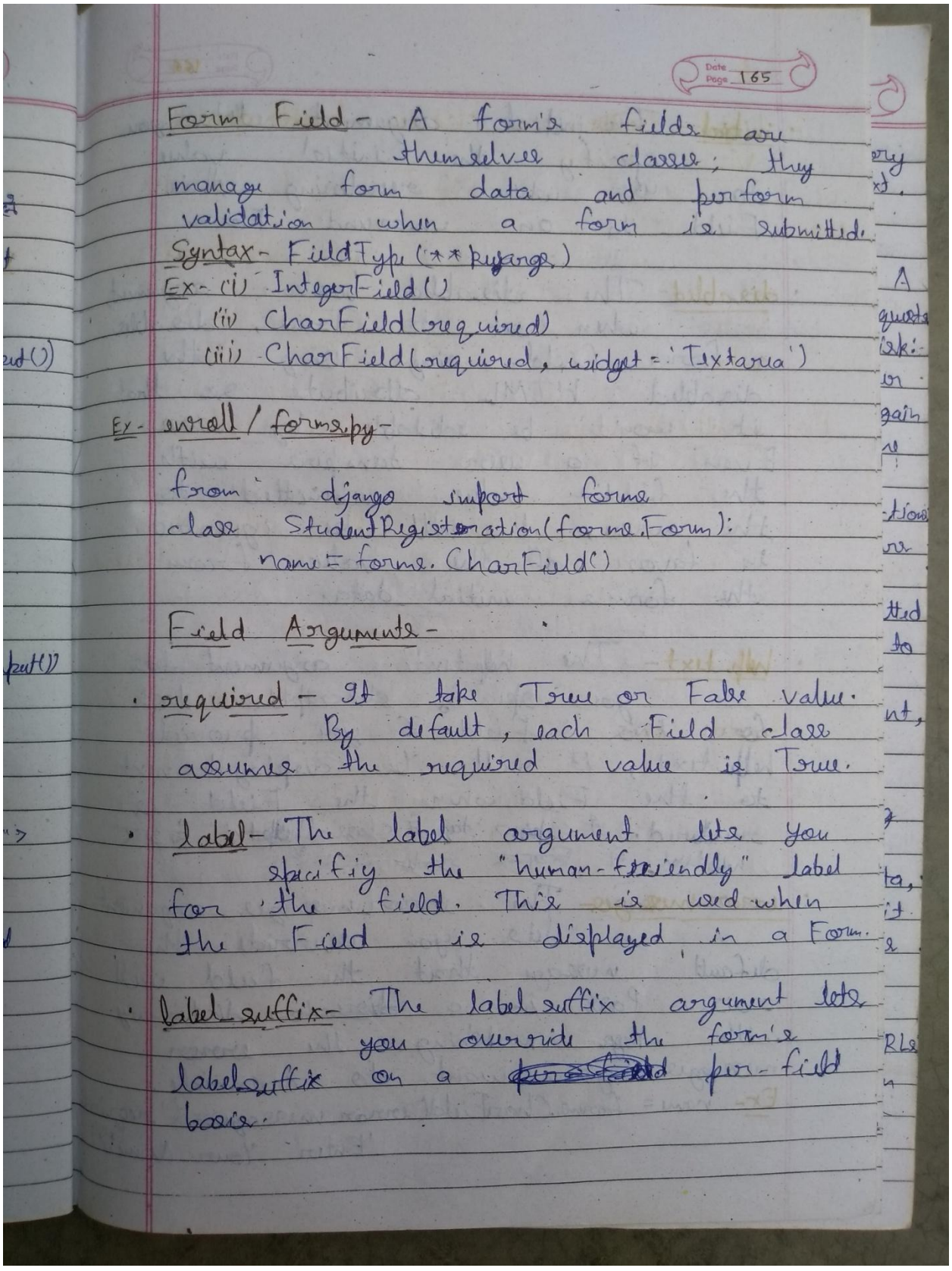
```
<label for="id_name">Name:</label>
<input type="text" name="name" required id="id_name">

<label for="id_email">Email:</label>
<input type="email" name="email" required
    id="id_email">

<label for="id_mobile">Mobile:</label>
<input type="number" name="mobile" required
    id="id_mobile">

<input type="hidden" name="key" id="id_key">
```

49



Date Page 165

Form Field - A form's fields are themselves classes; they manage form data and perform validation when a form is submitted.

Syntax - FieldType (**kwargs)

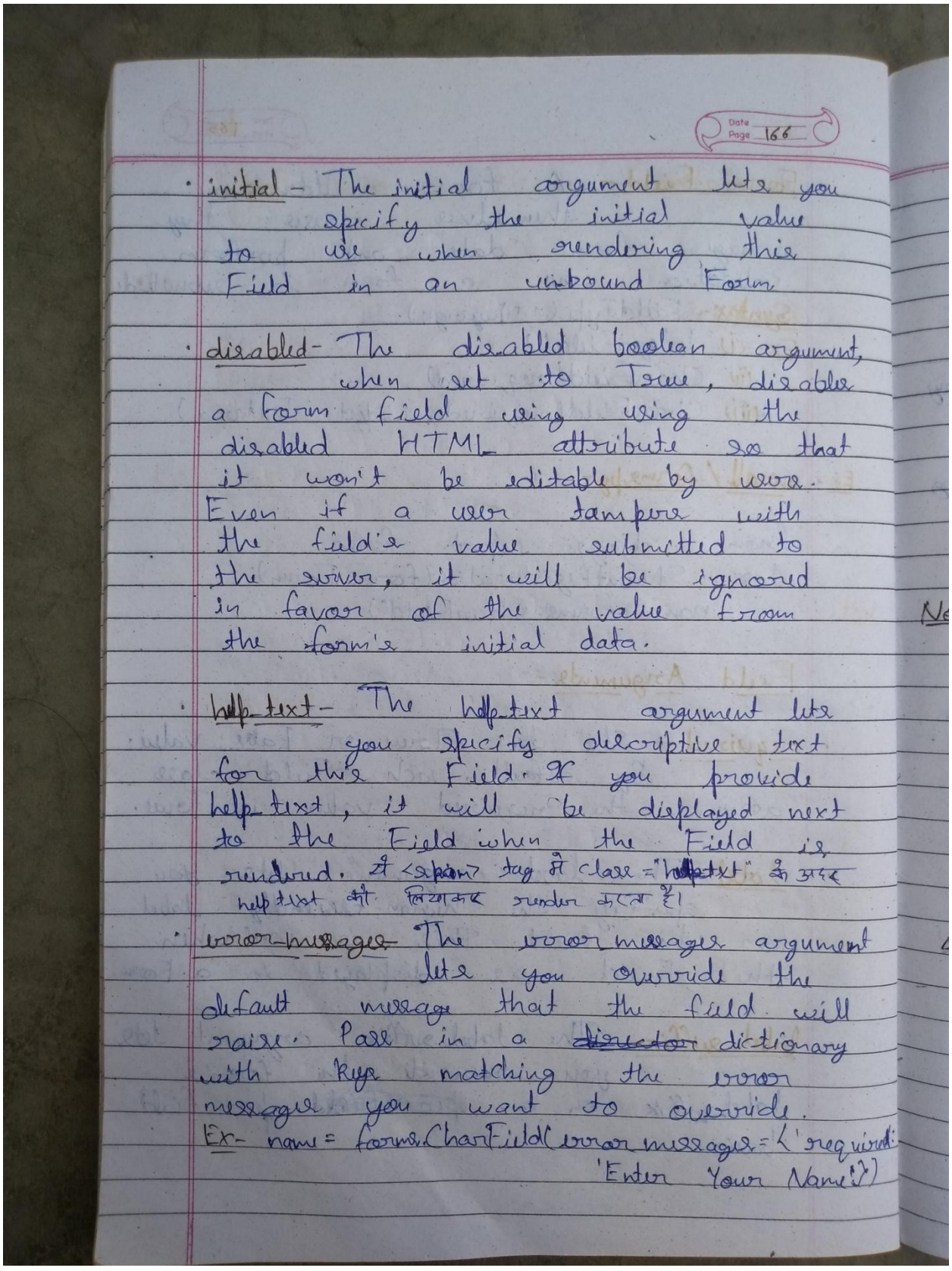
Ex - (i) IntegerField()
(ii) CharField(required)
(iii) CharField(required, widget = 'Textarea')

Ex - enroll / forms.py -

```
from django import forms
class StudentRegistration(forms.Form):
    name = forms.CharField()
```

Field Arguments -

- required - It take True or False value. By default, each Field class assumes the required value is True.
- label - The label argument lets you specify the "human-friendly" label for the field. This is used when the Field is displayed in a Form.
- label_suffix - The label_suffix argument lets you override the form's labelsuffix on a ~~per-field~~ per-field basis.



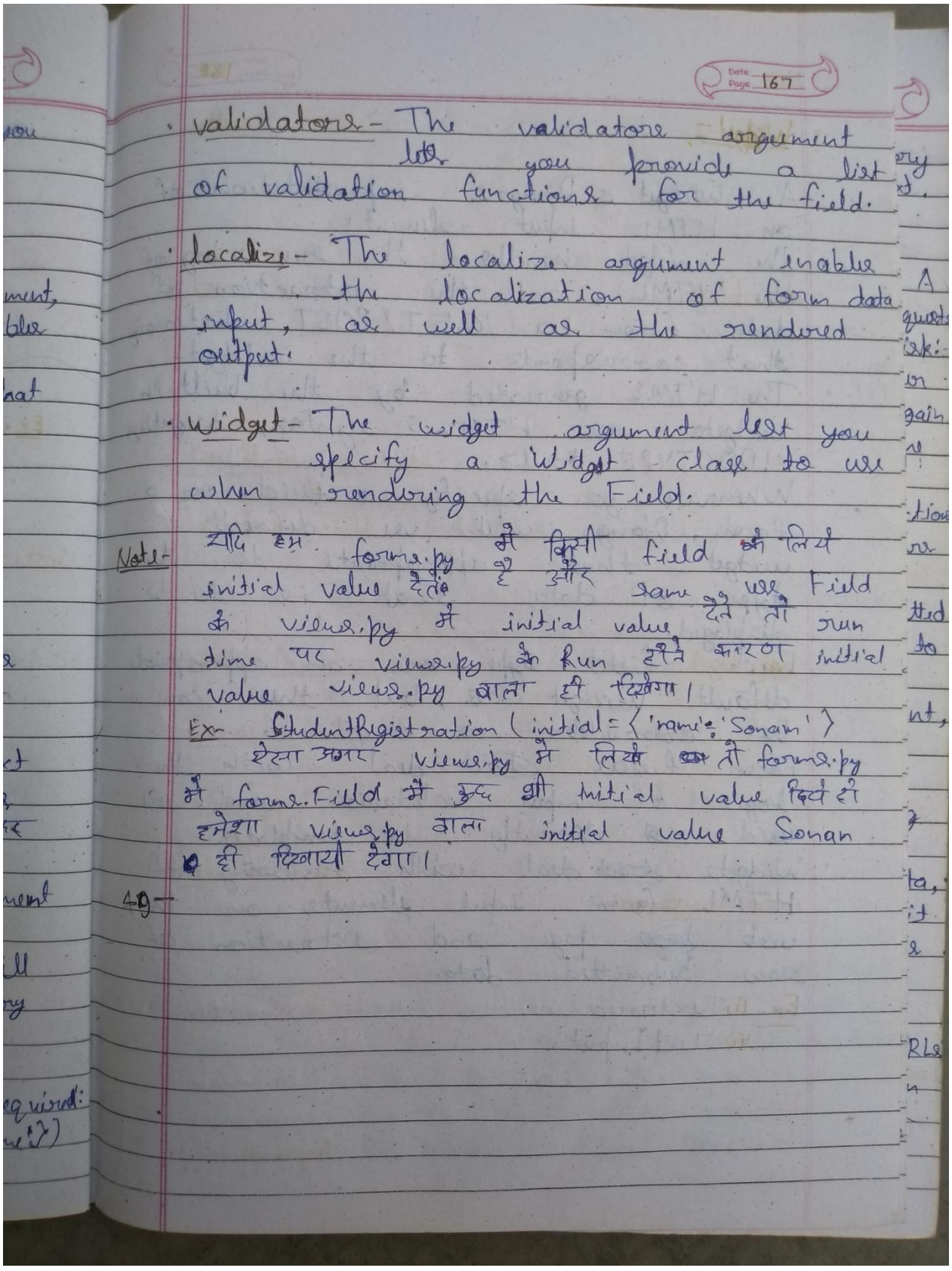
Date Page 166

- initial - The initial argument lets you specify the initial value to use when rendering this Field in an unbound Form.

- disabled - The disabled boolean argument, when set to True, disables a form field using using the disabled HTML attribute so that it won't be editable by users. Even if a user tampers with the field's value submitted to the server, it will be ignored in favor of the value from the form's initial data.

- help_text - The help_text argument lets you specify descriptive text for this Field. If you provide help_text, it will be displayed next to the Field when the Field is rendered. ये tag में class = "helptext" के अंदर help text को डिफाल्ट render करता है।

- error_messages - The error_messages argument lets you override the default message that the field will raise. Pass in a ~~dictionary~~ dictionary with keys matching the error messages you want to override.
Ex- name = forms.CharField(error_messages={'required': 'Enter Your Name'})

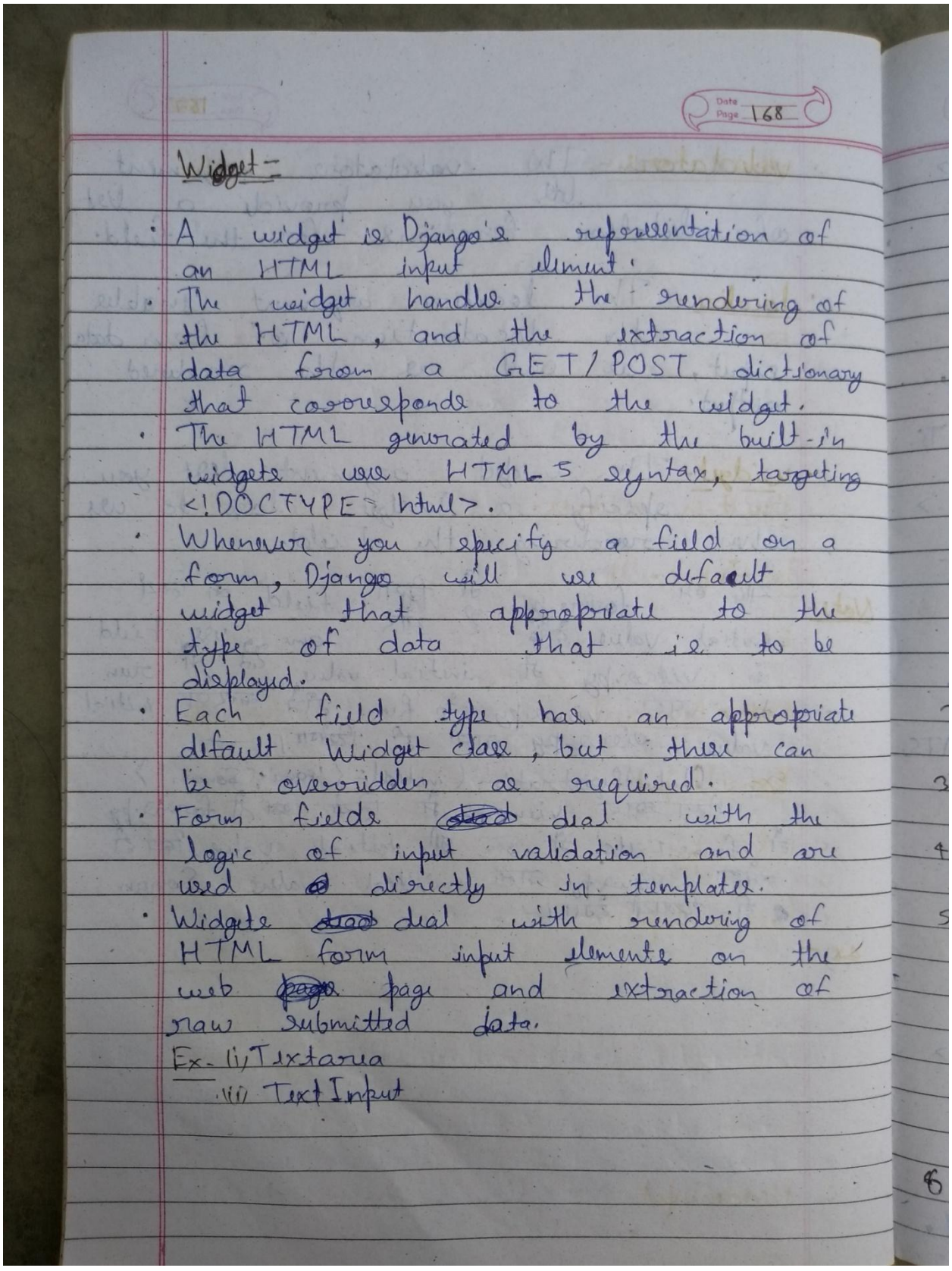


- validators - The validators argument lets you provide a list of validation functions for the field.
- localize - The localize argument enables the localization of form data input, as well as the rendered output.
- widget - The widget argument lets you specify a Widget class to use when rendering the Field.

Note - यदि हम forms.py में किसी field के लिये initial value देते हैं और same use Field के views.py में initial value देते तो run time पर views.py के Run होने कारण initial value views.py वाला ही दिखेगा।

Ex- StudentRegistration(initial={'name':'Sonam'})
ऐसा अगर views.py में लिखे तो forms.py में forms.Field में कुछ भी initial value दिये हो हमेशा views.py वाला initial value Sonan ही दिखायी देगा।

49-

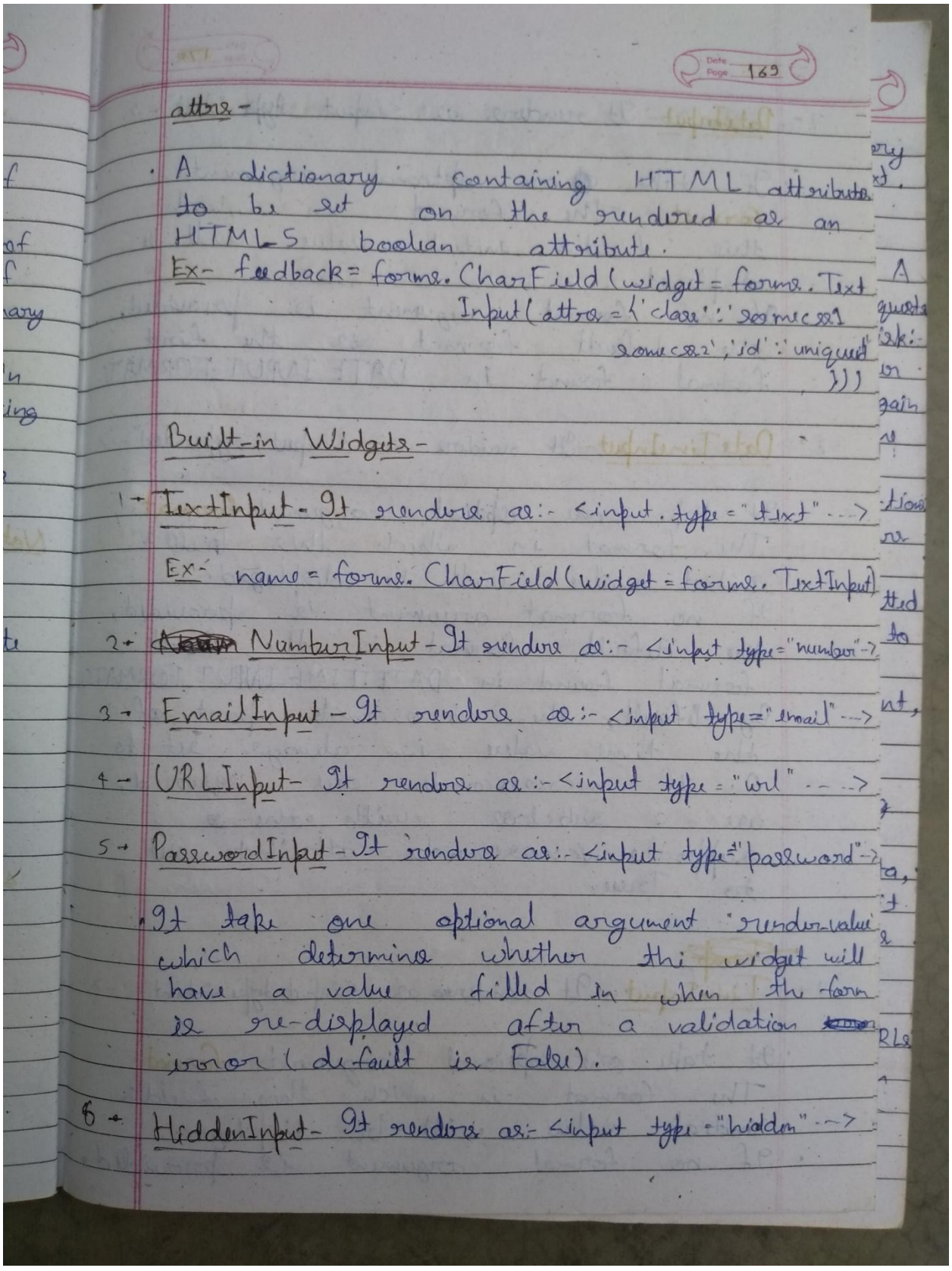


Date Page 168

Widget :-

- A widget is Django's representation of an HTML input element.
- The widget handles the rendering of the HTML, and the extraction of data from a GET/POST dictionary that corresponds to the widget.
- The HTML generated by the built-in widgets use HTML 5 syntax, targeting <!DOCTYPE html>.
- Whenever you specify a field on a form, Django will use default widget that appropriate to the type of data that is to be displayed.
- Each field type has an appropriate default Widget class, but these can be overridden as required.
- Form fields deal with the logic of input validation and are used directly in templates.
- Widgets deal with rendering of HTML form input elements on the web page and extraction of raw submitted data.

Ex- i) Textarea
ii) TextInput



attrs -

- A dictionary containing HTML attributes to be set on the rendered as an HTML-5 boolean attribute.

Ex- feedback = forms.CharField(widget = forms.TextInput(attrs = {'class': 'some css1 some css2', 'id': 'unique'}))

Built-in Widgets -

1- TextInput - It renders as:- <input type = "text" ...>

Ex- name = forms.CharField(widget = forms.TextInput)

2- NumberInput - It renders as:- <input type = "number">

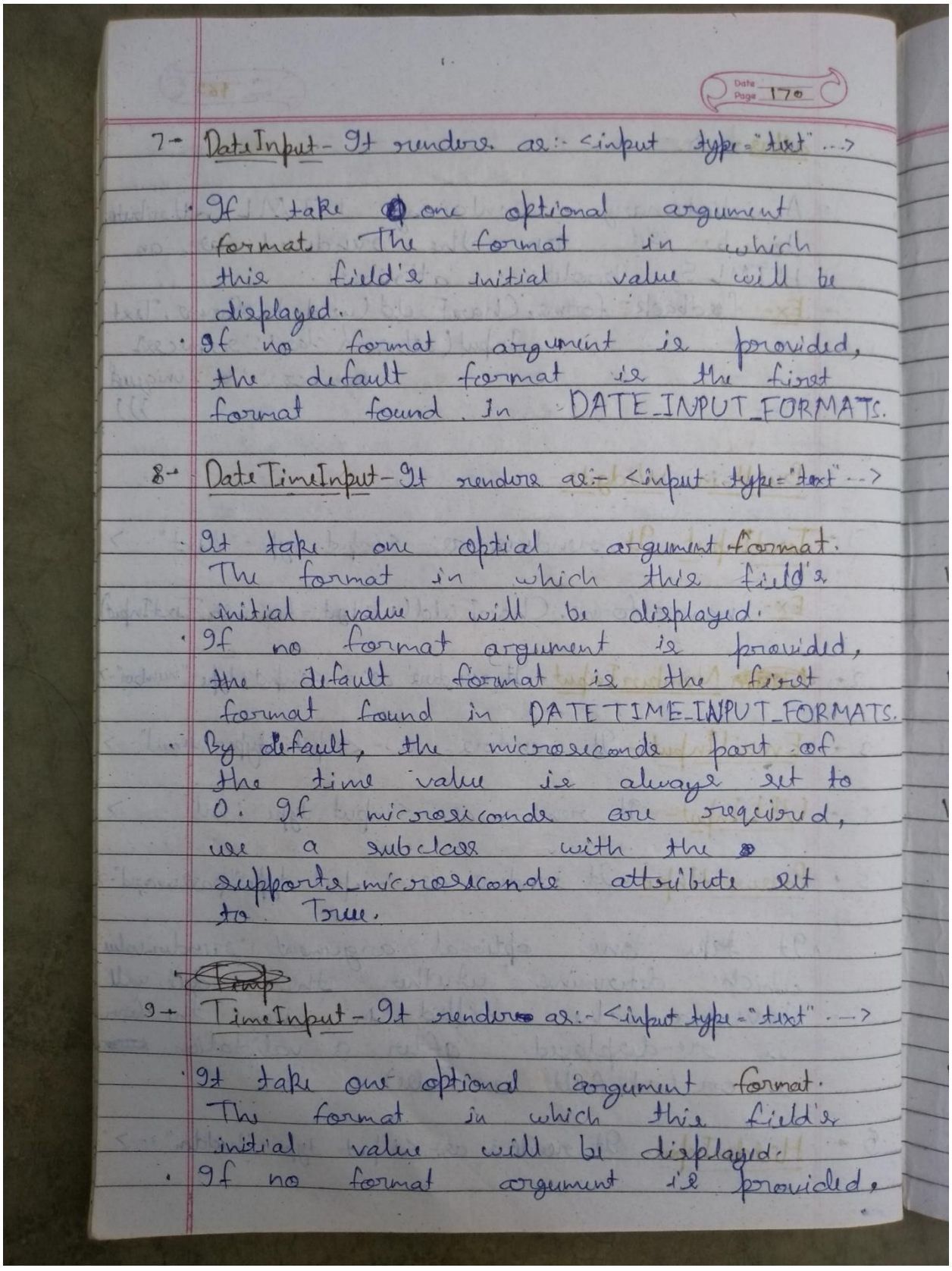
3- EmailInput - It renders as:- <input type = "email" ...>

4- URLInput - It renders as:- <input type = "url" ...>

5- PasswordInput - It renders as:- <input type = "password">

It take one optional argument 'render_value' which determine whether thi widget will have a value filled in when the form is re-displayed after a validation error (default is False).

6- HiddenInput - It renders as:- <input type = "hidden" ...>



Date
Page 170

7→ DateInput - It renders as:- <input type="text" ...>

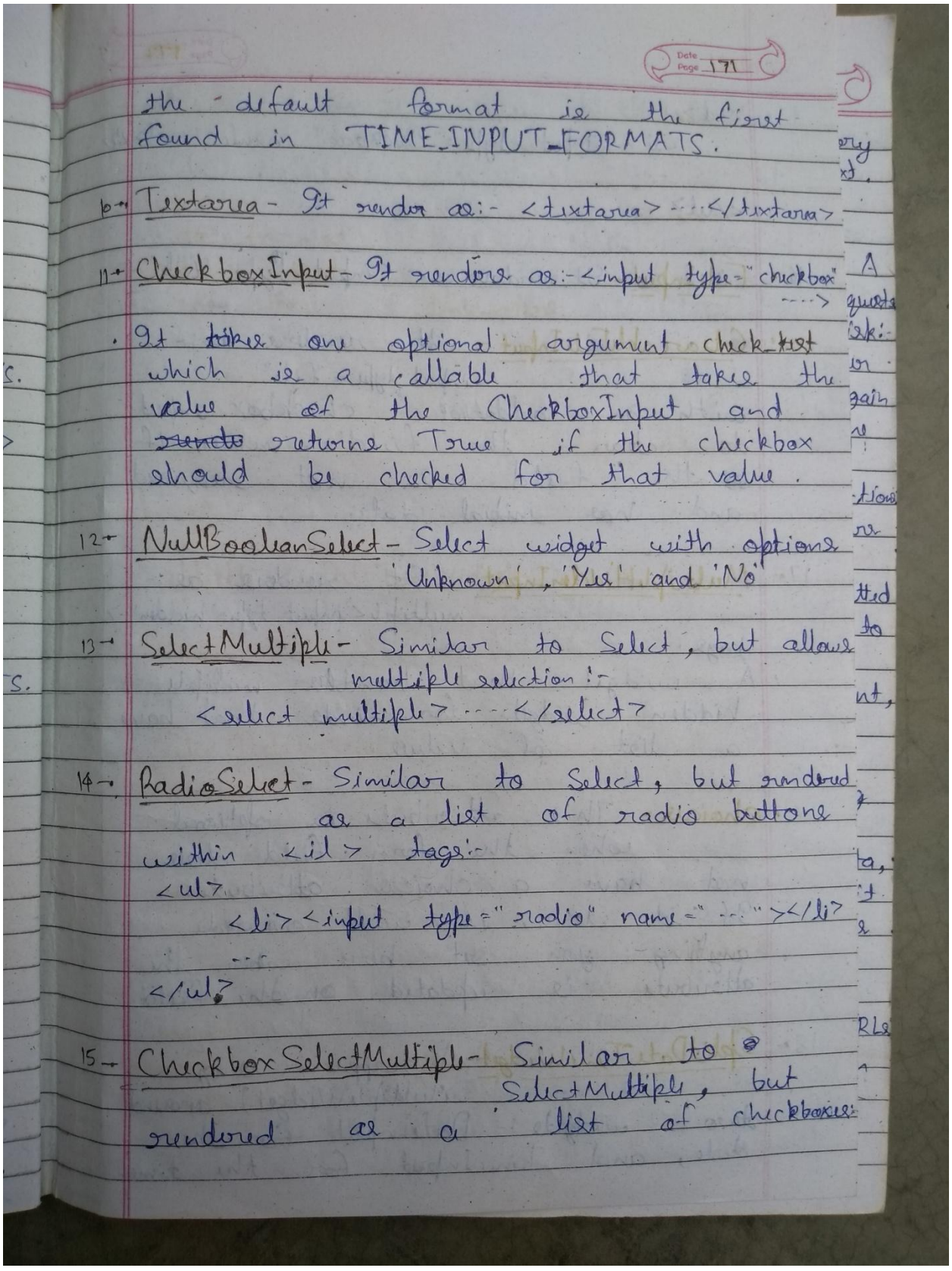
- It take one optional argument format. The format in which this field's initial value will be displayed.
- If no format argument is provided, the default format is the first format found in DATE_INPUT_FORMATS.

8→ DateTimeInput - It renders as:- <input type="text" ...>

- It take one optial argument format. The format in which this field's initial value will be displayed.
- If no format argument is provided, the default format is the first format found in DATETIME_INPUT_FORMATS.
- By default, the microseconds part of the time value is always set to 0. If microseconds are required, use a subclass with the supports_microseconds attribute set to True.

9→ TimeInput - It renders as:- <input type="text" ...>

- It take one optional argument format. The format in which this field's initial value will be displayed.
- If no format argument is provided,



the default format is the first found in TIME_INPUT_FORMATS.

10- Textarea - It renders as:- <textarea>...</textarea>

11- CheckboxInput - It renders as:- <input type="checkbox" ...>

. It takes one optional argument check_test which is a callable that takes the value of the CheckboxInput and returns True if the checkbox should be checked for that value.

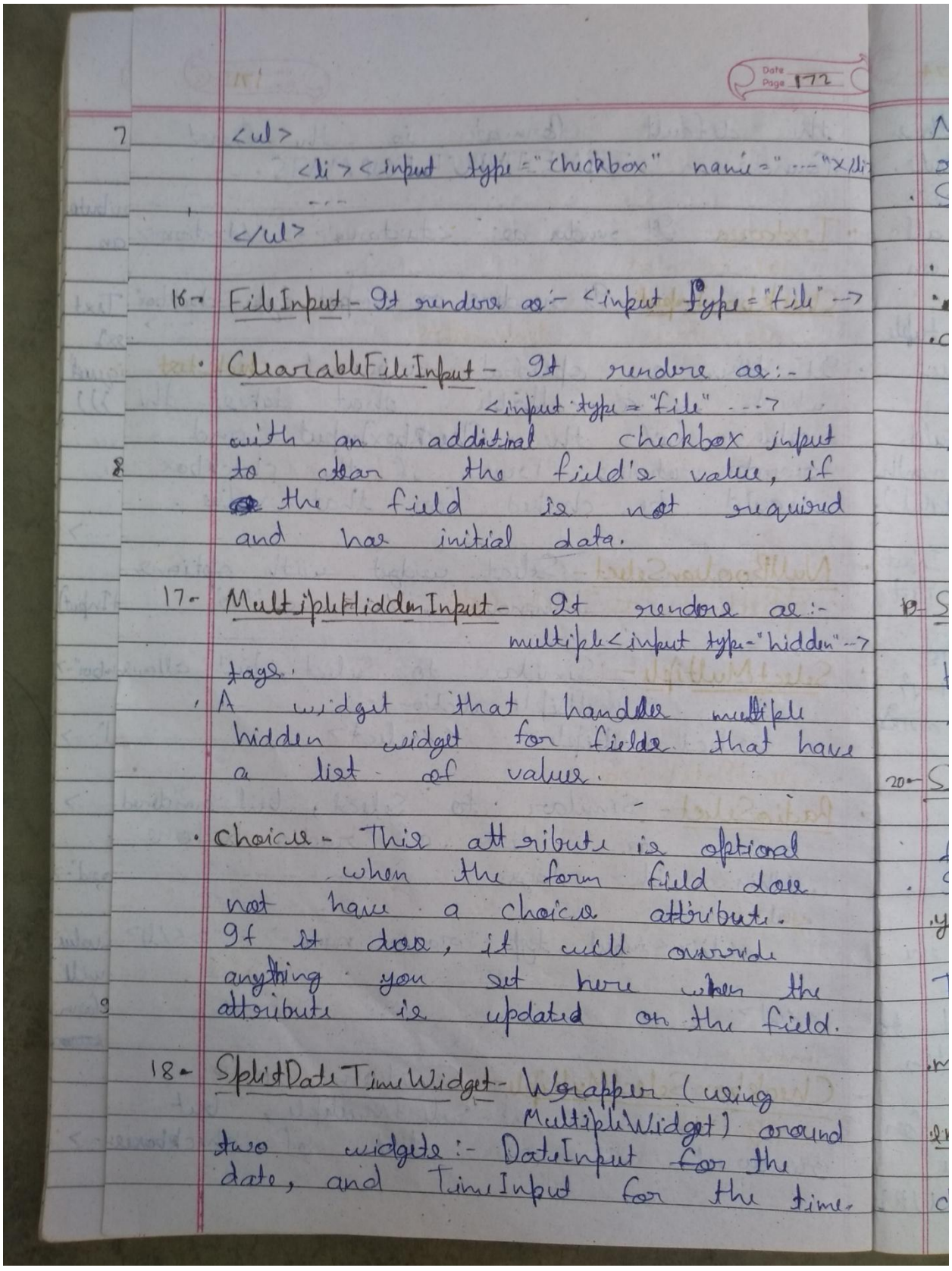
12- NullBooleanSelect - Select widget with options 'Unknown', 'Yes' and 'No'

13- SelectMultiple - Similar to Select, but allows multiple selection :-
<select multiple>...</select>

14- RadioSelect - Similar to Select, but rendered as a list of radio buttons within <li> tags:-

<input type="radio" name="...">
...

15- CheckboxSelectMultiple - Similar to SelectMultiple, but rendered as a list of checkboxes:



<input type="checkbox" name="...">
...

16- FileInput - It renders as:- <input type="file" ...>

- ClearableFileInput - It renders as:-
<input type="file" ...>
with an additional checkbox input to clear the field's value, if the field is not required and has initial data.

17- MultipleHiddenInput - It renders as:-
multiple <input type="hidden" ...>
tags.
A widget that handles multiple hidden widget for fields that have a list of values.

- choices - This attribute is optional when the form field does not have a choices attribute. If it does, it will override anything you set here when the attribute is updated on the field.

18- SplitDateTimeWidget - Wrapper (using MultipleWidget) around two widgets:- DateInput for the date, and TimeInput for the time.

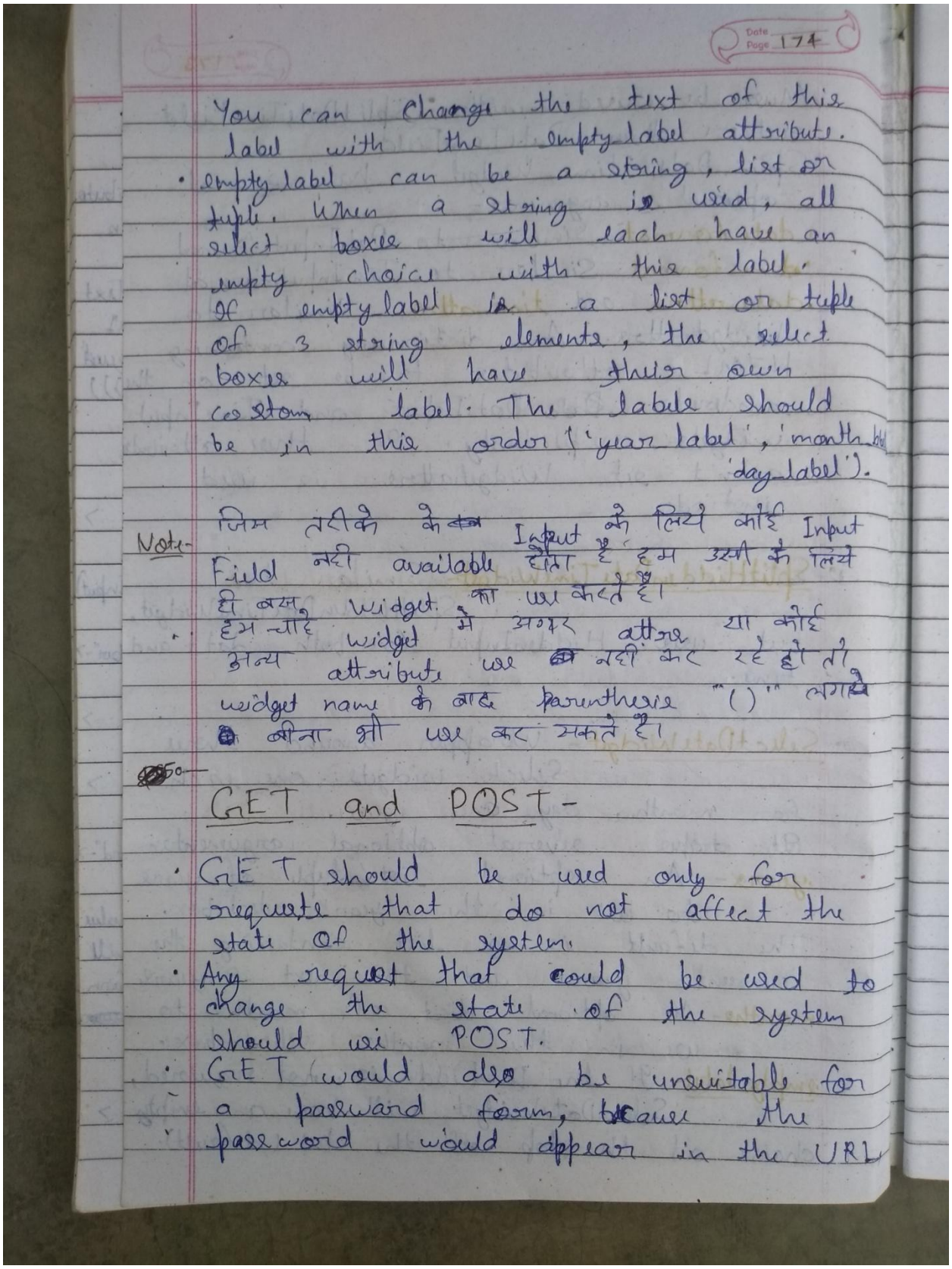
Must be used with SplitDateTimeField rather than DateTimeField.

- SplitDateTimeWidget has several optional arguments-
- date_format Similar to DateInput.format
- time_format Similar to TimeInput.format
- date_attrs and time_attrs Similar to Widget.attrs. A dictionary containing HTML attributes to be set on the rendered DateInput and TimeInput widgets, respectively. If these attributes aren't set Widget.attrs is used instead.

19- SplitHiddenDateTimeWidget- Similar to SplitHiddenDateTimeWidget, but uses HiddenInput for both date and time.

20- SelectDateWidget- Wrapper around three Select widgets:- one each for month, day and year.

- It takes several optional arguments:-
- years- An optional list/tuple of years to use in the "year" select box. The default is a list containing the current year and the next 9 years.
- months- An optional dict of months to use in the "months" select box.
- empty_label- If the DateField is not required, SelectDateWidget will have an empty choice at the top of the list (default).



Date Page 174

You can change the text of this label with the empty_label attribute.

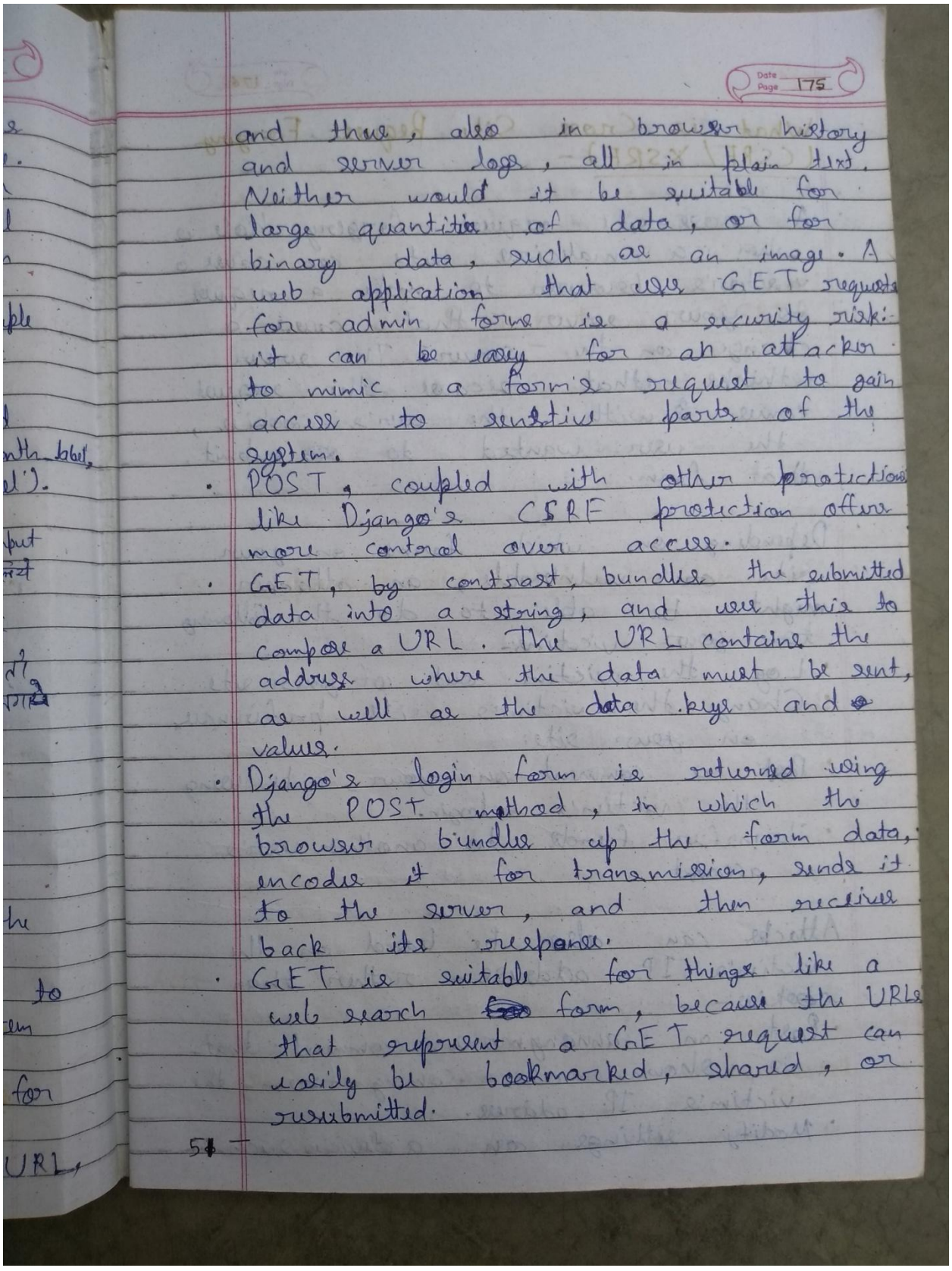
- empty_label can be a string, list or tuple. When a string is used, all select boxes will each have an empty choice with this label. If empty_label is a list or tuple of 3 string elements, the select boxes will have their own custom label. The labels should be in this order ('year_label', 'month_label', 'day_label').

Note- जिस तरीके के Input Field नहीं available होता है हम उसी के लिये ही बस widget का use करते है।

- हम चाहे widget में अगर attrs या कोई अन्य attribute use नहीं कर रहे हो तो, widget name के बाद parenthesis "()" लगाये बिना भी use कर सकते है।

GET and POST-

- GET should be used only for requests that do not affect the state of the system.
- Any request that could be used to change the state of the system should use POST.
- GET would also be unsuitable for a password form, because the password would appear in the URL

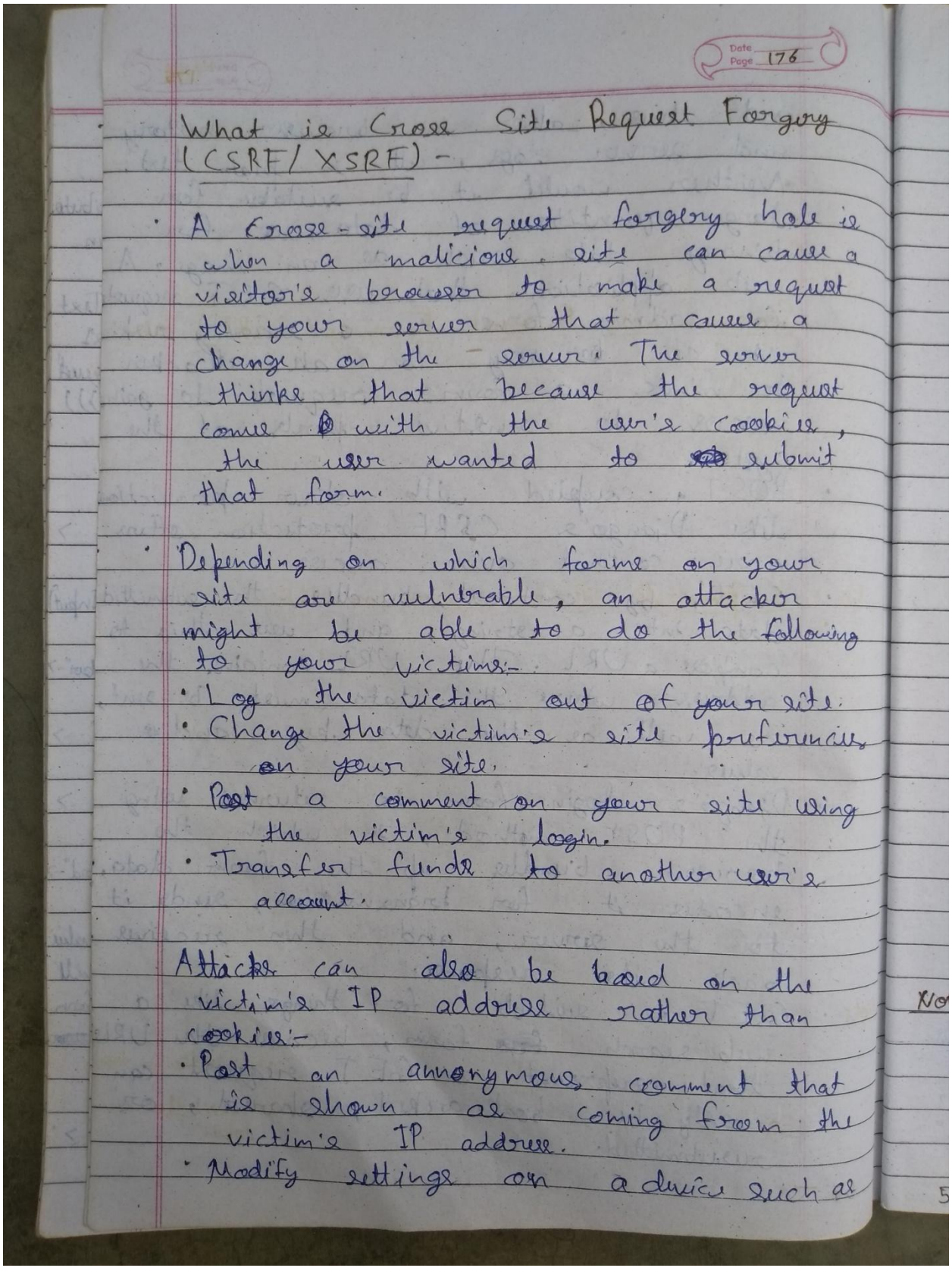


Date Page 175

and thus, also in browser history and server logs, all in plain text. Neither would it be suitable for large quantities of data, or for binary data, such as an image. A web application that use GET requests for admin forms is a security risk: it can be easy for an attacker to mimic a form's request to gain access to sensitive parts of the system.

- POST, coupled with other protections like Django's CSRF protection offers more control over access.
- GET, by contrast, bundles the submitted data into a string, and uses this to compose a URL. The URL contains the address where the data must be sent, as well as the data keys and values.
- Django's login form is returned using the POST method, in which the browser bundles up the form data, encodes it for transmission, sends it to the server, and then receives back its response.
- GET is suitable for things like a web search form, because the URLs that represent a GET request can easily be bookmarked, shared, or resubmitted.

54



Date Page 176

What is Cross Site Request Forgery (CSRF/XSRF) -

- A Cross-site request forgery hole is when a malicious site can cause a visitor's browser to make a request to your server that causes a change on the server. The server thinks that because the request comes with the user's cookies, the user wanted to submit that form.

- Depending on which forms on your site are vulnerable, an attacker might be able to do the following to your victims:-
  - Log the victim out of your site.
  - Change the victim's site preferences on your site.
  - Post a comment on your site using the victim's login.
  - Transfer funds to another user's account.

Attacks can also be based on the victim's IP address rather than cookies:-
- Post an anonymous comment that is shown as coming from the victim's IP address.
- Modify settings on a device such as

Date 15-9-20

a wireless router or cable modem.
- Modify an internet wiki page.
- Perform a distributed password-guessing attack without a botnet.

51 —

Cross Site Request Forgery (CSRF) Token-

Django provides CSRF Protection with csrf_token which we need to add inside form tag. This token will add a hidden input field with random value in form tag.

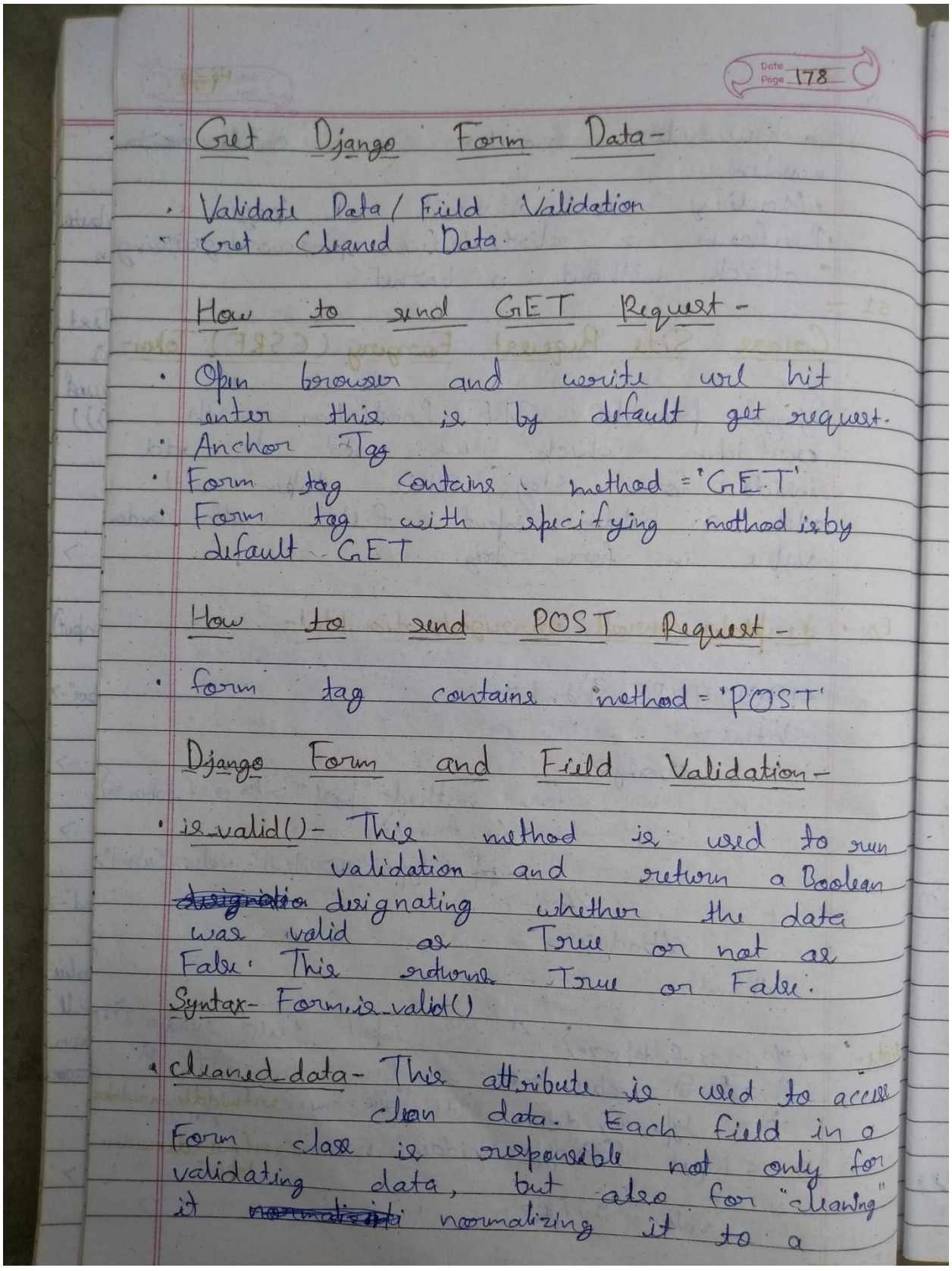
Ex- templates/enroll/userregistration.html-

```
<!DOCTYPE html>
<html>
    <body>
        <form method="post"> {% csrf_token %}
            {{ form }}
            <input type="submit" value="Submit">
        </form>
    </body>
</html>
```

Note- {% csrf_token %} से एक input field render होता है जिसकी value एक random value होती है और type = hidden होता है और name=csrfmiddlewaretoken होता है।

Ex- <input type="hidden" name="csrfmiddlewaretoken" value="1vf42 - - ———">

53 —



Date
Page 178

Get Django Form Data-

- Validate Data / Field Validation
- Get Cleaned Data.

How to send GET Request -

- Open browser and write url hit enter this is by default get request.
- Anchor Tag
- Form tag contains : method = 'GET'
- Form tag with specifying method is by default GET

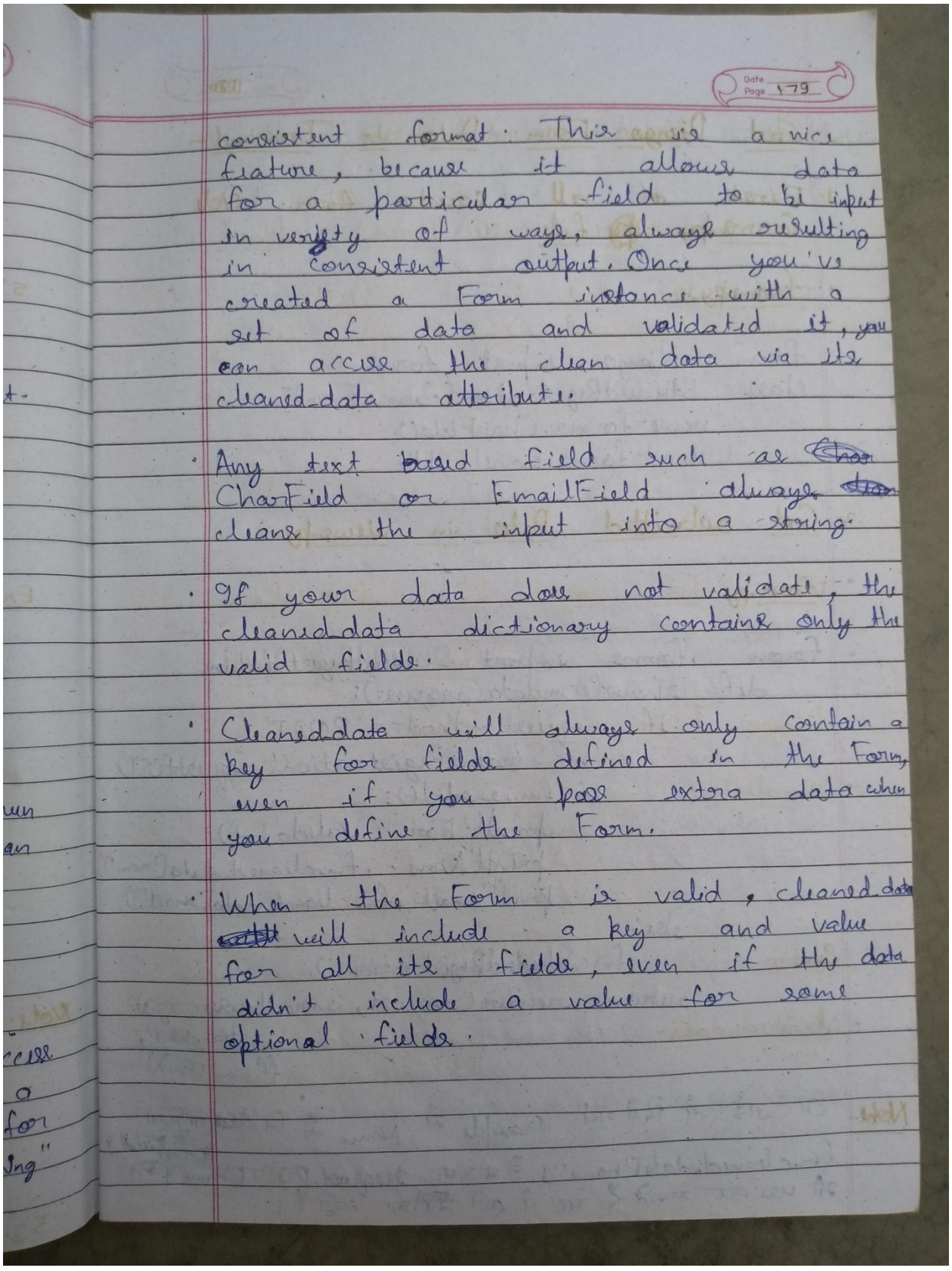
How to send POST Request -

- form tag contains method = 'POST'

Django Form and Field Validation-

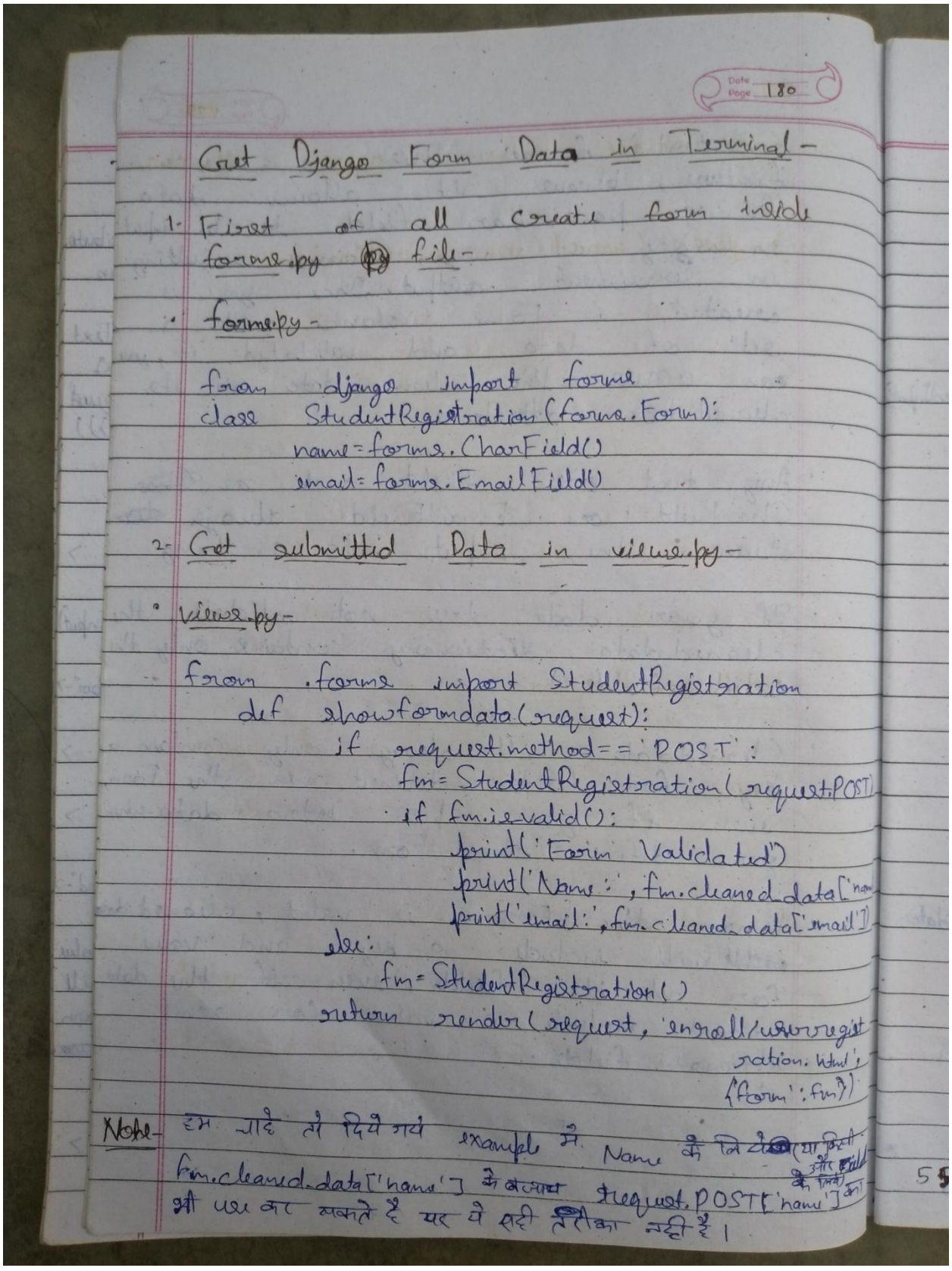
- is_valid() - This method is used to run validation and return a Boolean designating whether the data was valid as True or not as False. This returns True or False.
  Syntax- Form.is_valid()

- cleaned_data - This attribute is used to access clean data. Each field in a Form class is responsible not only for validating data, but also for "cleaning" it normalizing it to a



consistent format. This is a nice feature, because it allows data for a particular field to be input in variety of ways, always resulting in consistent output. Once you've created a Form instance with a set of data and validated it, you can access the clean data via its cleaned_data attribute.

- Any text based field such as CharField or EmailField always cleans the input into a string.
- If your data does not validate, the cleaned_data dictionary contains only the valid fields.
- Cleaned_data will always only contain a key for fields defined in the Form, even if you pass extra data when you define the Form.
- When the Form is valid, cleaned_data will include a key and value for all its fields, even if the data didn't include a value for some optional fields.



Date Page 180

Get Django Form Data in Terminal -

1. First of all create form inside forms.py file -

- forms.py -

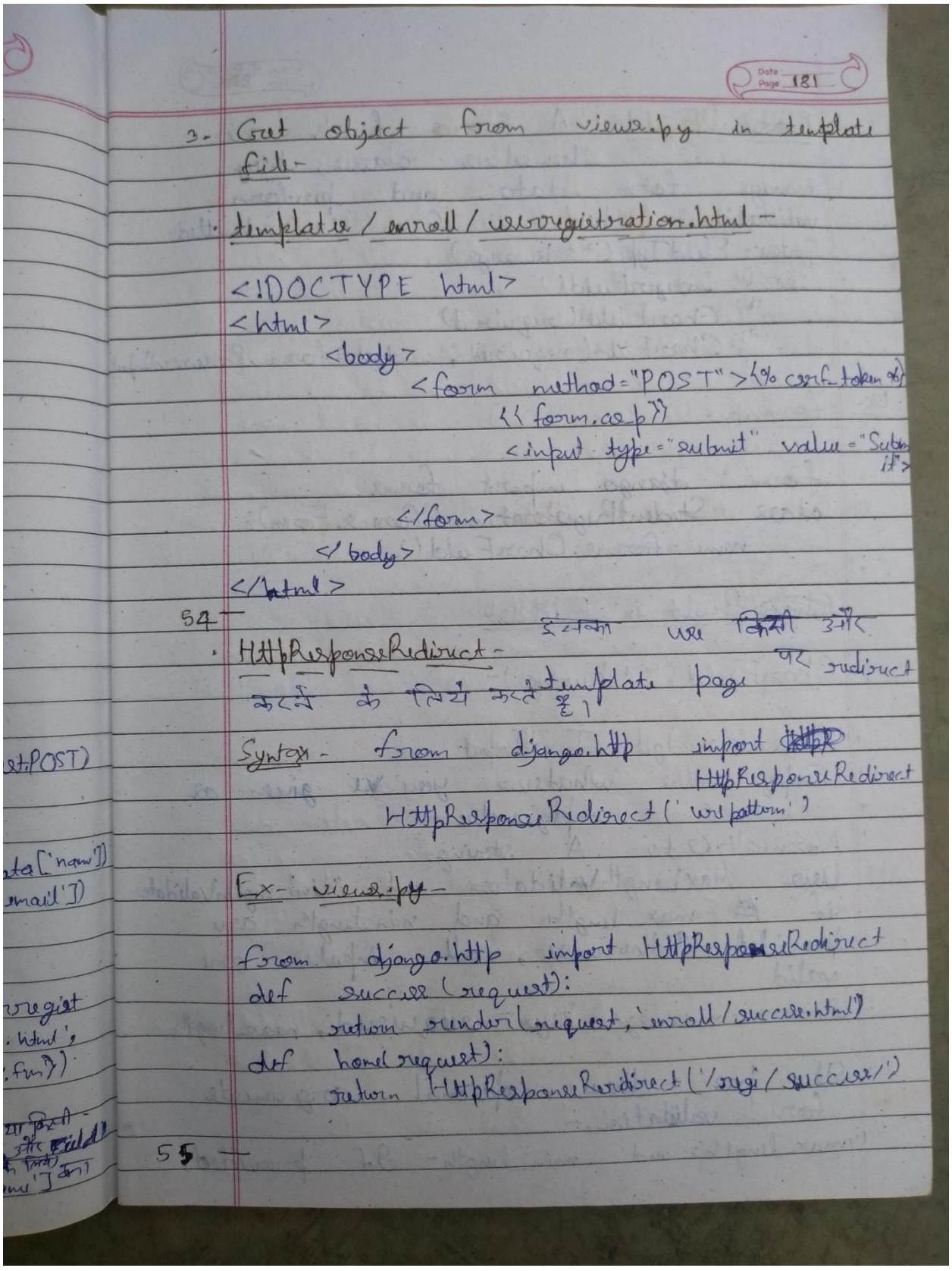
```
from django import forms
class StudentRegistration(forms.Form):
    name = forms.CharField()
    email = forms.EmailField()
```

2. Get submitted Data in views.py -

- views.py -

```
from .forms import StudentRegistration
def showformdata(request):
    if request.method == 'POST':
        fm = StudentRegistration(request.POST)
        if fm.is_valid():
            print('Form Validated')
            print('Name:', fm.cleaned_data['name'])
            print('email:', fm.cleaned_data['email'])
    else:
        fm = StudentRegistration()
    return render(request, 'enroll/userregistration.html', {'form': fm})
```

Note - हम ऊपर दे दिये गये example में Name के लिये (या किसी और Field के लिये) fm.cleaned_data['name'] के बजाय request.POST['name'] का भी use कर सकते है पर वे सही तरीका नहीं है।



Date Page 181

3- Get object from views.py in template file-

· templates / enroll / userregistration.html -

```
<!DOCTYPE html>
<html>
    <body>
        <form method="POST">{% csrf_token %}
            {{ form.as_p }}
            <input type="submit" value="Submit">
        </form>
    </body>
</html>
```

54 —

· HttpResponseRedirect - इसका use किसी और page पर redirect करने के लिये करते है।

Syntax - from django.http import HttpResponseRedirect
HttpResponseRedirect('url pattern')

Ex- views.py -

```
from django.http import HttpResponseRedirect
def success(request):
    return render(request, 'enroll/success.html')
def home(request):
    return HttpResponseRedirect('/regi/success/')
```

55 —