PART-3

django Web Framework

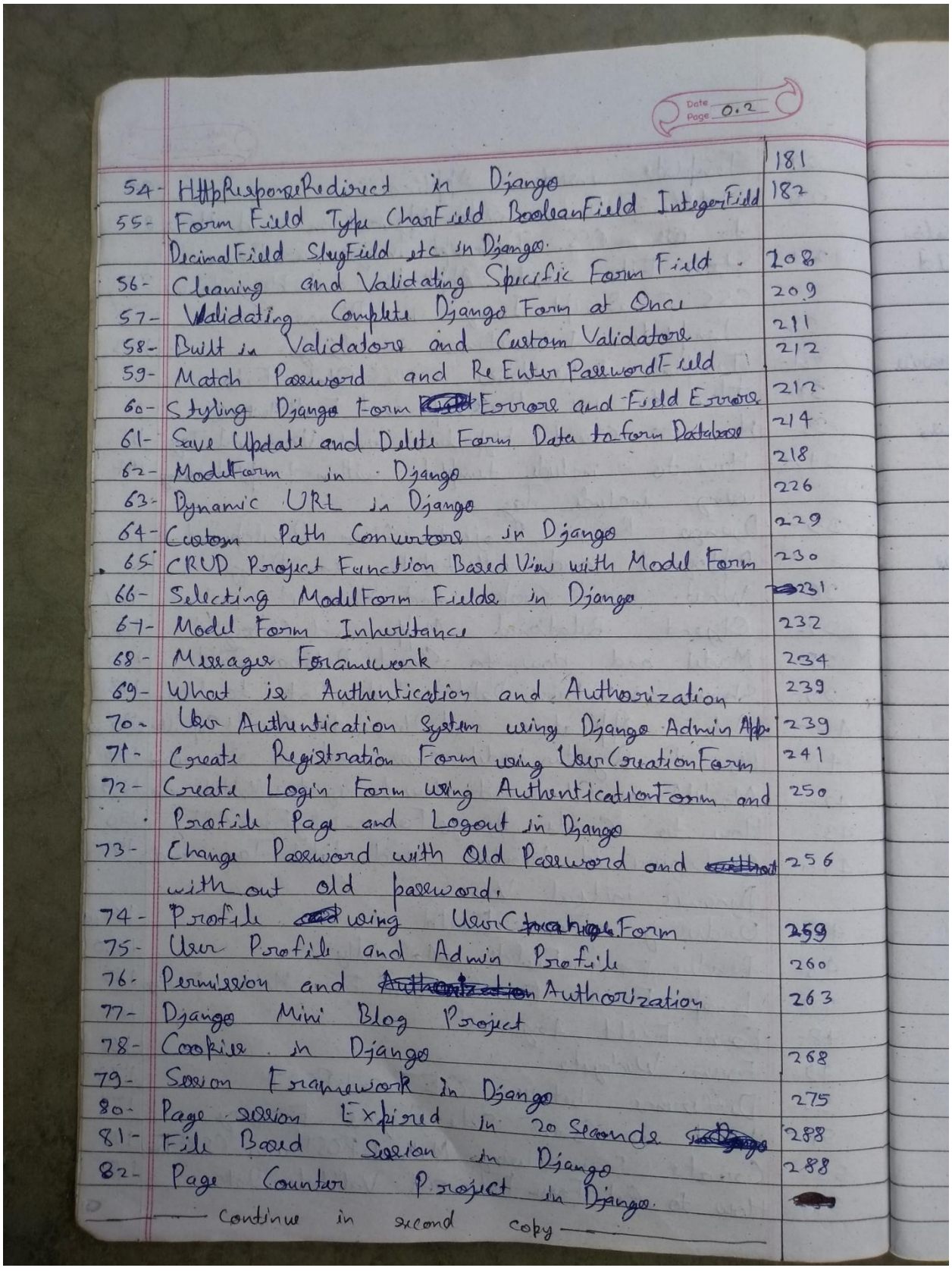
Geeky Shows YouTube Channel Learning Notes

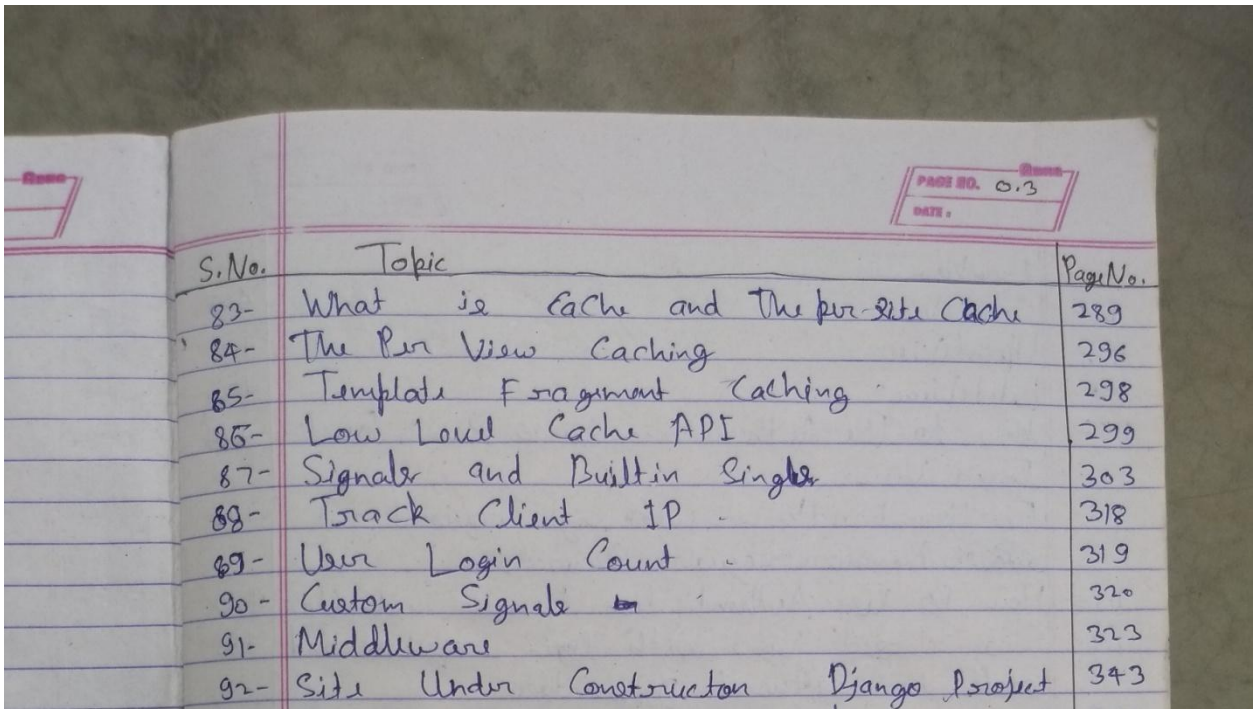
SATYAM SETH

14/09/2020

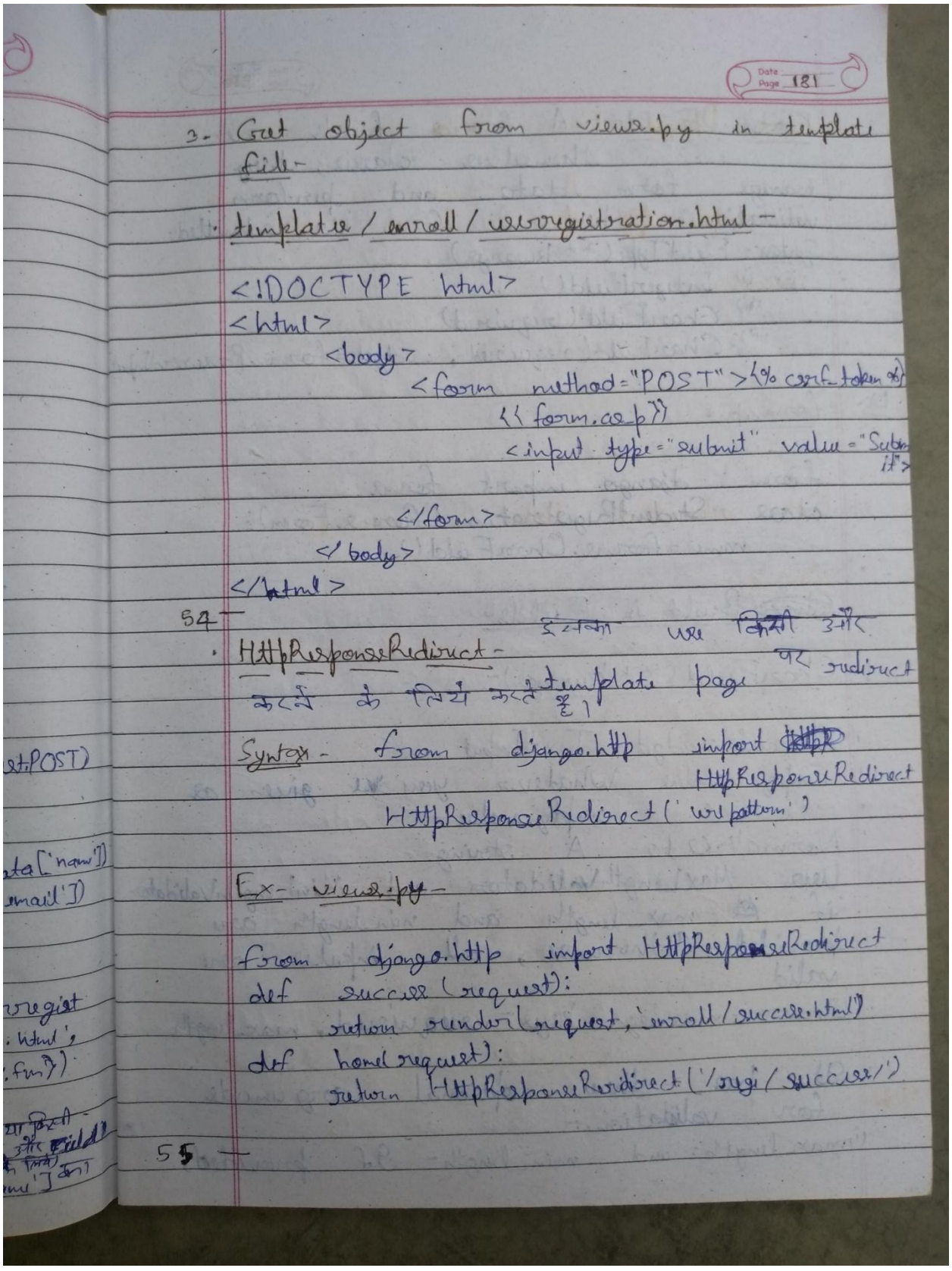
Source Code – https://github.com/satyam-seth/django_learning

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





PAGE NO. 0.3
DATE :



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 —

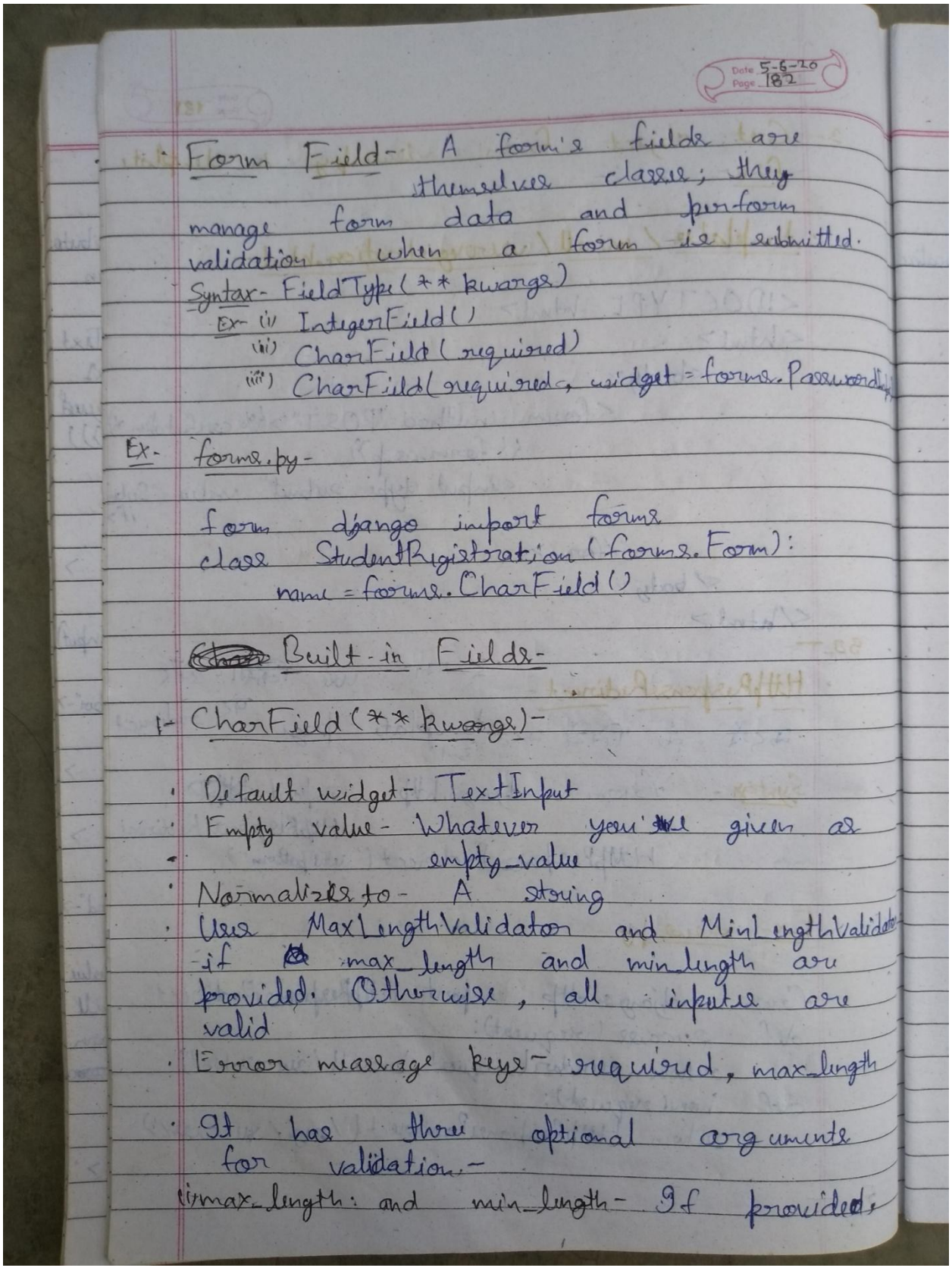
· HttpResponseRedirect - इसका use किसी और page पर redirect करने के लिये करते है।

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 —



Date 5-6-20
Page 182

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=forms.PasswordInput)

Ex- forms.py-

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

Built-in Fields-

1- CharField(**kwargs)-

- Default widget- TextInput
- Empty value- Whatever you've given as empty_value
- Normalizes to- A string
- Uses MaxLengthValidator and MinLengthValidator if max_length and min_length are provided. Otherwise, all inputs are valid
- Error message keys- required, max_length
- It has three optional arguments for validation.-

i) max_length: and min_length- If provided,

Date Page 183

these arguments ensure that the string is at most or at least the given length.

(ii) strip - If True (default), the value will be stripped of leading and trailing whitespace.

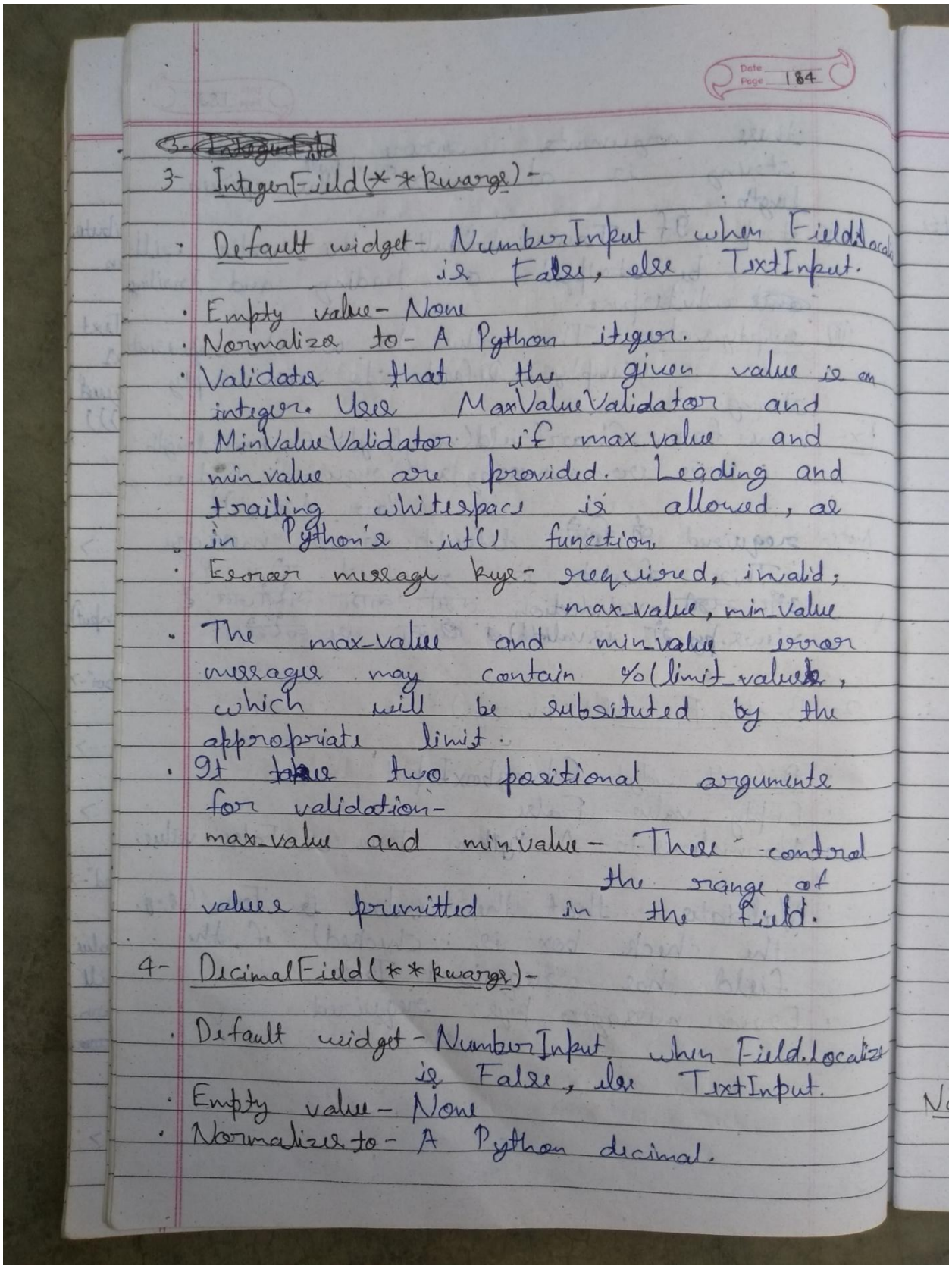
(iii) empty_value - The value to use to represent "empty". Defaults to an empty string.

Ex- name = forms.CharField(min_length=5, max_length=10, error_messages={'required': 'Enter Your Name'})

Note- required के लिये default error message 'This field is required' होता है। और सभी validation तभी काम करेंगे जब है views.py में is_valid() का use करेंगे।

2- BooleanField(**kwargs)-

- Default widget - CheckboxInput
- Empty value - False
- Normalize to - A Python True or False value.
- Validates that the value is True (i.e. the check box is checked) if the field has required = True.
- Error message key :- required



3- IntegerField(**kwargs)-

- Default widget- NumberInput when Field.localize is False, else TextInput.
- Empty value- None
- Normalize to- A Python integer.
- Validates that the given value is an integer. Uses MaxValueValidator and MinValueValidator if max_value and min_value are provided. Leading and trailing whitespace is allowed, as in Python's int() function.
- Error message keys- required, invalid, max_value, min_value
- The max_value and min_value error message may contain %(limit_value)s, which will be substituted by the appropriate limit.
- It takes two positional arguments for validation-
  max_value and min_value- These control the range of values permitted in the field.

4- DecimalField(**kwargs)-

- Default widget- NumberInput when Field.localize is False, else TextInput.
- Empty value- None
- Normalizes to- A Python decimal.

Date Page 185

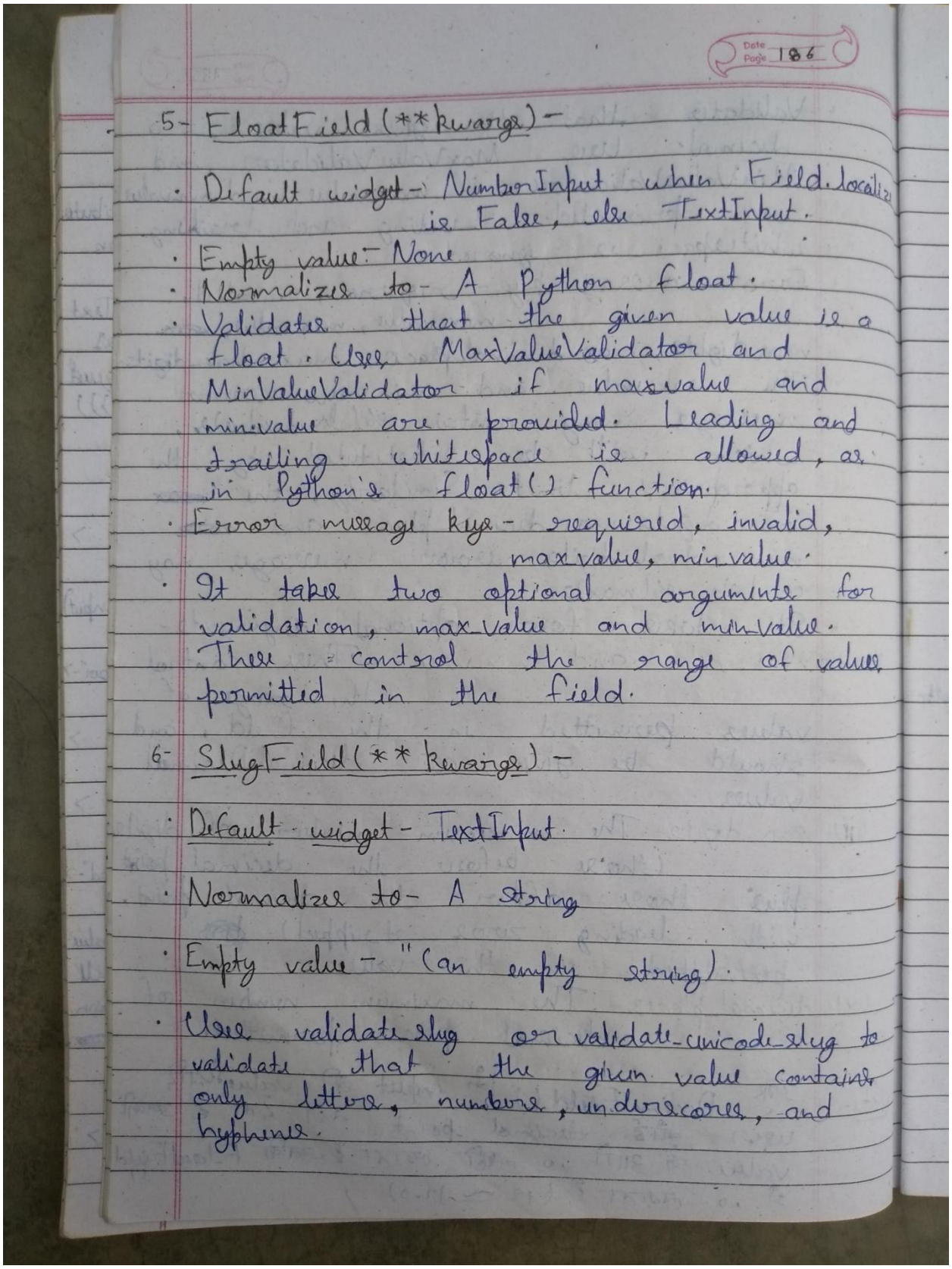
- Validates that the given value is a decimal. Uses MaxValueValidator and MinValueValidator if max_value and min_value are provided. Leading and trailing whitespace is ignored.
- Error message keys - required, invalid, max_value, min_value, max_digits, max_decimal_places, max_whole_digits.
- The max_value and min_value error message may contain %(limit_value)s, which will be substituted by the appropriate limit. Similarly, the max_digits, max_decimal_places and max_whole_digits error messages may contain %(max)s.
- It takes four optional arguments-

(i) and (ii) max_value and min_value - These control the range of values permitted in the field, and should be given as decimal.Decimal values.

(iii) max_digits - The maximum number of digits (those before the decimal point plus those after the decimal point, with leading zeros stripped) permitted in the value.

(iv) decimal_places - The maximum number of decimal places permitted.

Note - अगर DecimalField() के input की value वैसादि user कोई decimal point नहीं देता है उसकी value के आगे .0 नहीं लगाता है जबकि FloatField में .0 लगाता है (12 ≠ 12.0) /



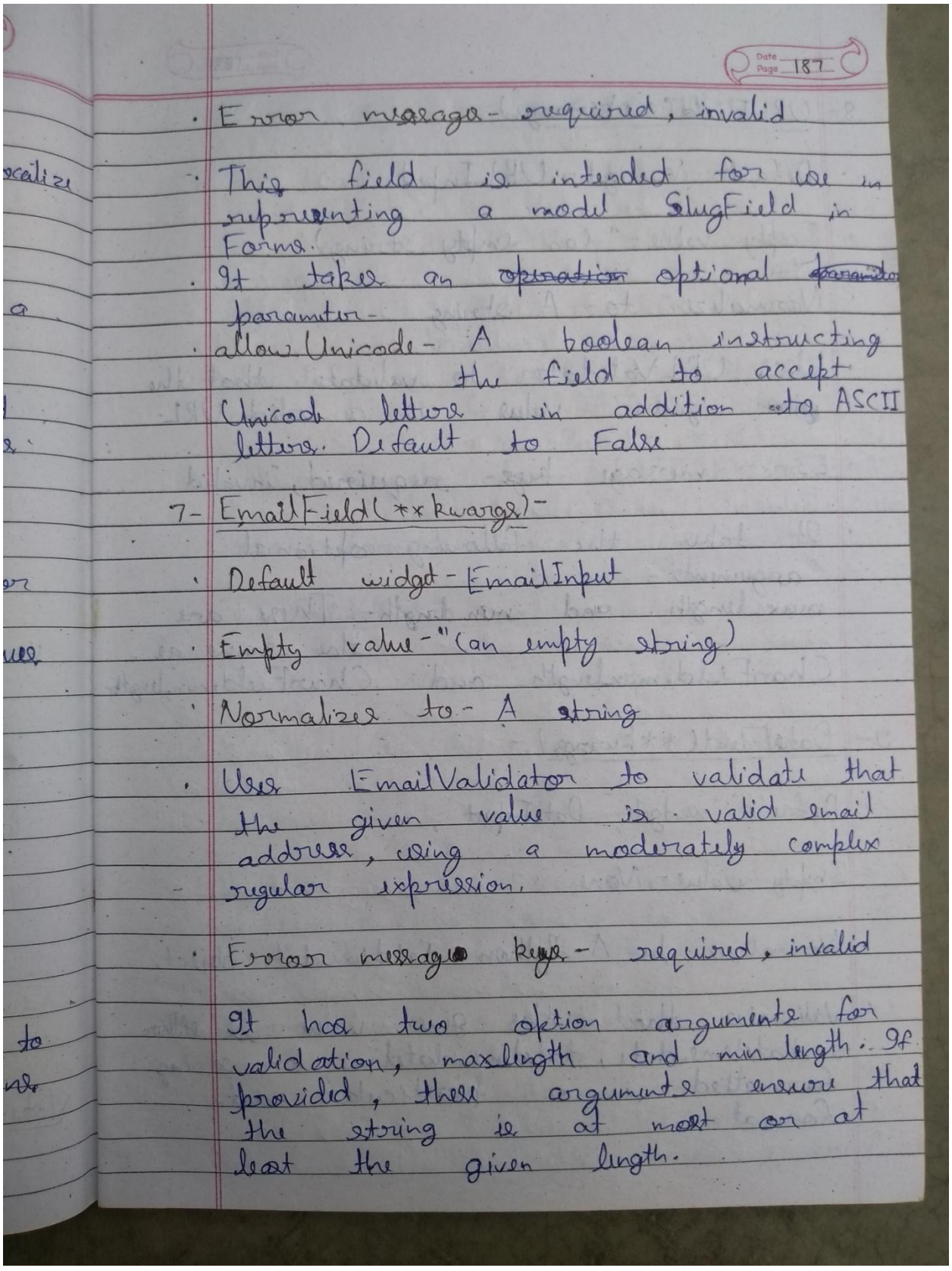
Date Page 186

5- FloatField(**kwargs) -

- Default widget - NumberInput when Field.localize is False, else TextInput.
- Empty value - None
- Normalizes to - A Python float.
- Validates that the given value is a float. Uses MaxValueValidator and MinValueValidator if max_value and min_value are provided. Leading and trailing whitespace is allowed, as in Python's float() function.
- Error message keys - required, invalid, max_value, min_value.
- It takes two optional arguments for validation, max_value and min_value. These control the range of values permitted in the field.

6- SlugField(**kwargs) -

- Default widget - TextInput.
- Normalizes to - A string
- Empty value - '' (an empty string).
- Uses validate_slug or validate_unicode_slug to validate that the given value contains only letters, numbers, underscores, and hyphens.



- Error message - required, invalid
- This field is intended for use in representing a model SlugField in Forms.
- It takes an optional parameter -
- allow_unicode - A boolean instructing the field to accept Unicode letters in addition to ASCII letters. Default to False

7- EmailField(**kwargs)-

- Default widget - EmailInput
- Empty value - '' (an empty string)
- Normalizes to - A string
- Uses EmailValidator to validate that the given value is valid email address, using a moderately complex regular expression.
- Error message keys - required, invalid

It has two option arguments for validation, max_length and min_length. If provided, these arguments ensure that the string is at most or at least the given length.

Date Page 188

8- URLField(**kwargs)-

- Default widget - URLInput
- Empty value - '' (an empty string)
- Normalizes to - A string
- Uses URLValidator to validate that the given value is a valid URL.
- Error message keys - required, invalid
- It takes the following optional arguments -

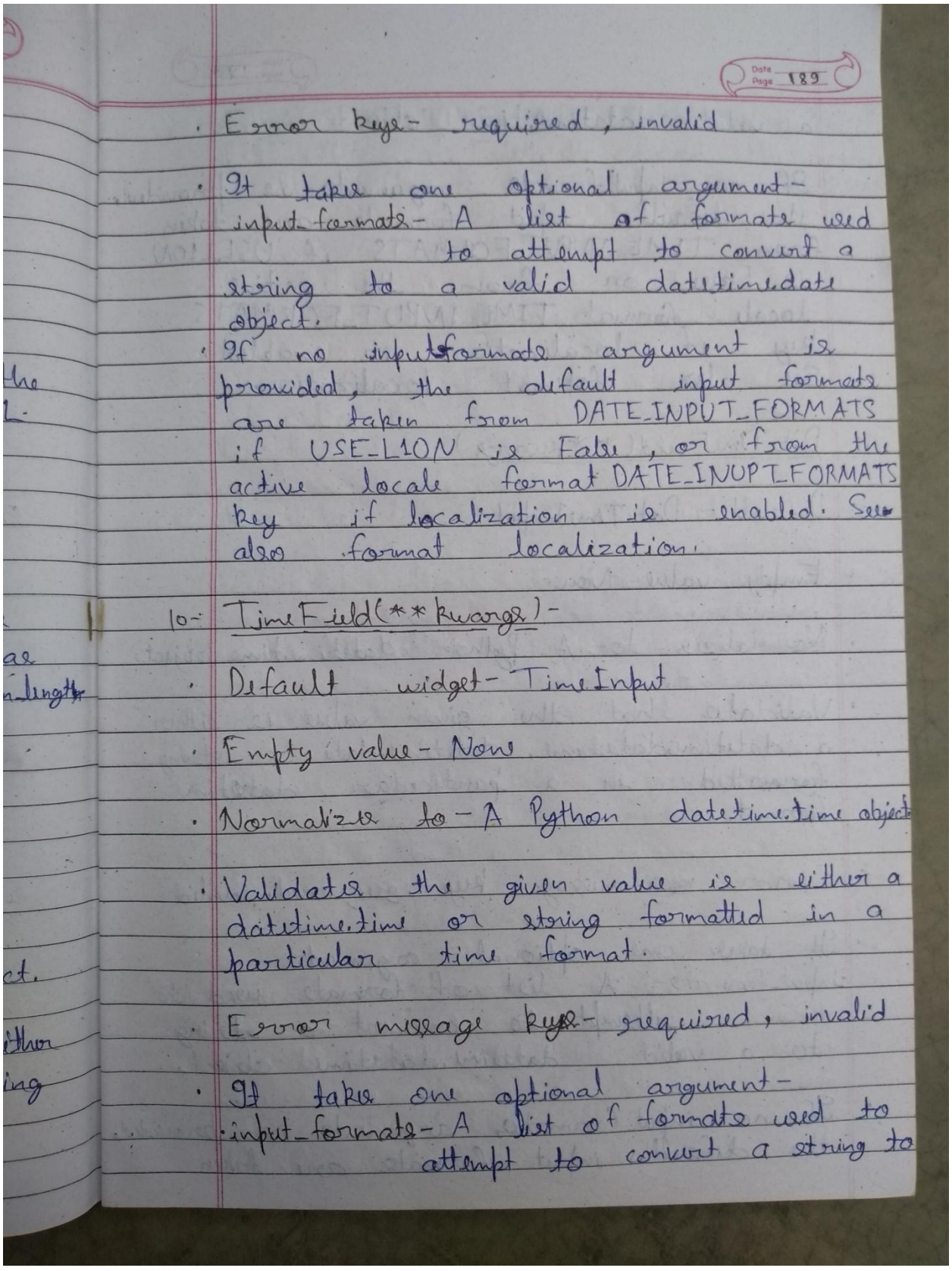
max_length and min_length - These are the same as CharField.max_length and CharField.min_length

9- DateField(**kwargs)-

- Default widget - DateInput
- Empty value - None

Normalizes to - A Python datetime.date object.

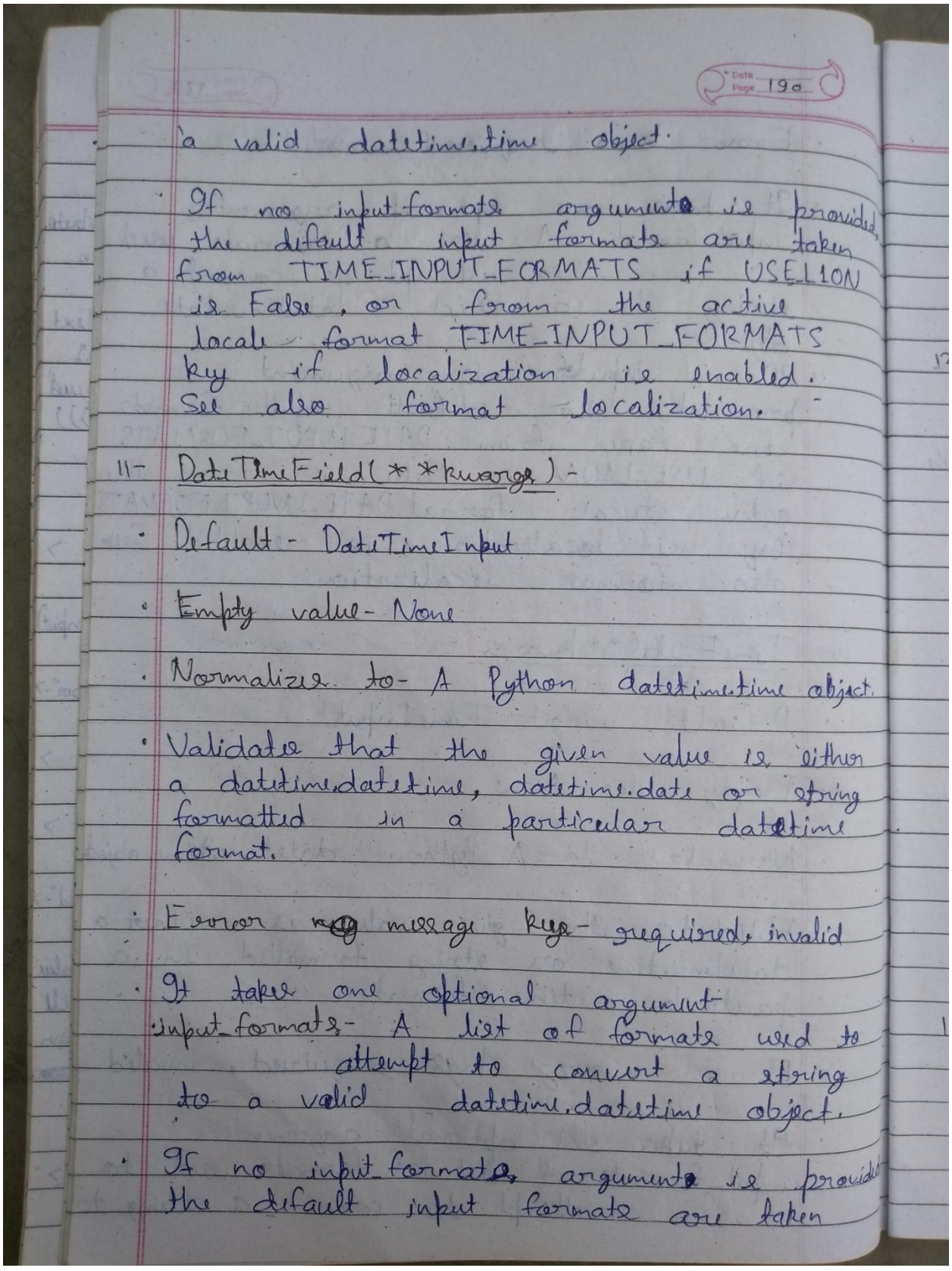
- Validates that the given value is either a datetime.date, datetime.datetime or string formatted in a particular date format.



- Error keys - required, invalid
- It take one optional argument - input_formats - A list of formats used to attempt to convert a string to a valid datetime.date object.
- If no input_formats argument is provided, the default input formats are taken from DATE_INPUT_FORMATS if USE_L10N is False, or from the active locale format DATE_INUPT_FORMATS key if localization is enabled. See also format localization.

10- TimeField(**kwargs) -

- Default widget - TimeInput
- Empty value - None
- Normalizes to - A Python datetime.time object
- Validates the given value is either a datetime.time or string formatted in a particular time format.
- Error message keys - required, invalid
- It take one optional argument - input_formats - A list of formats used to attempt to convert a string to

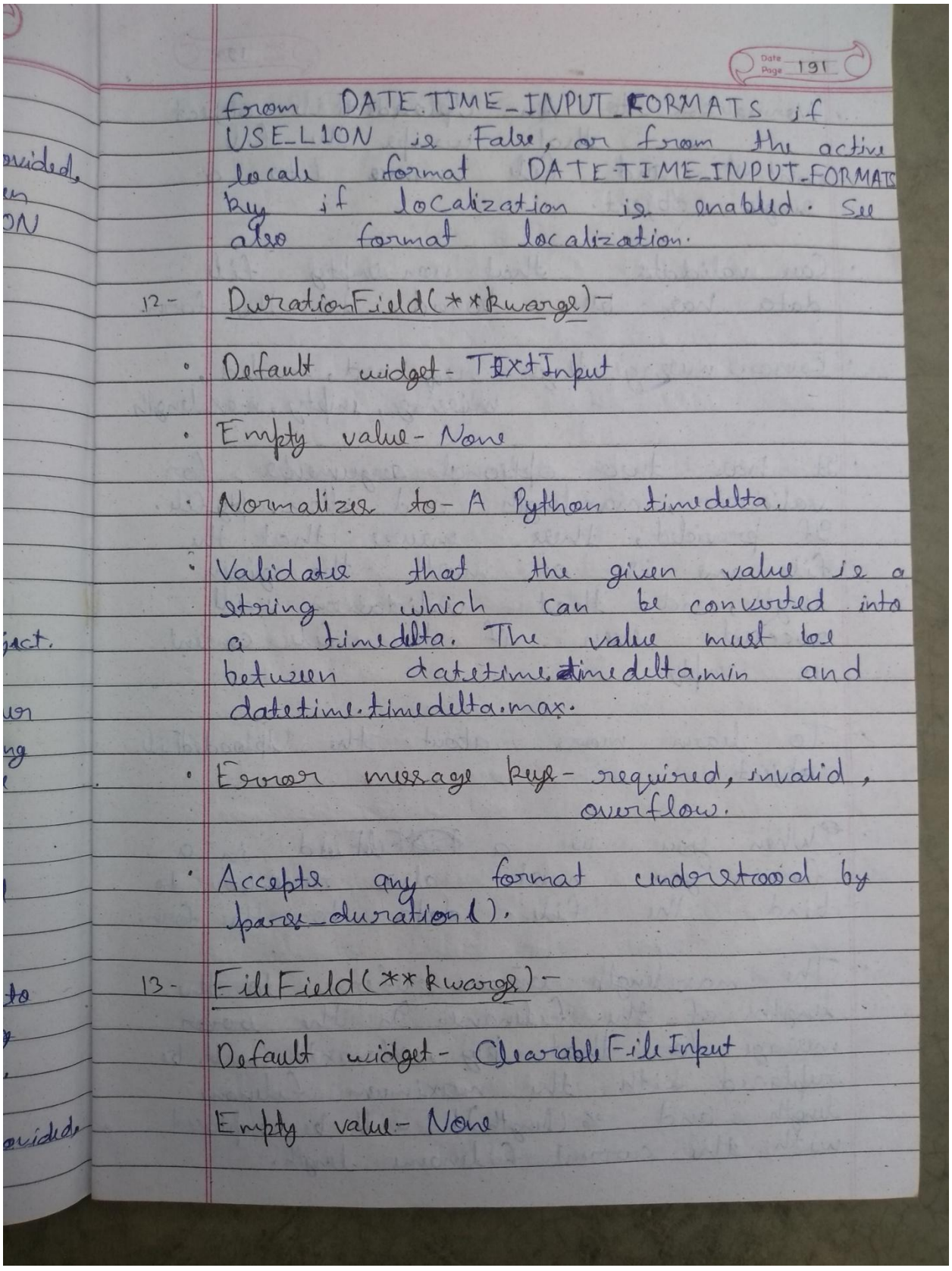


a valid datetime.time object.

- If no input_formats argument is provided, the default input formats are taken from TIME_INPUT_FORMATS if USE_L10N is False, or from the active locale format TIME_INPUT_FORMATS key if localization is enabled. See also format localization.

11- DateTimeField(**kwargs) :-

- Default - DateTimeInput
- Empty value - None
- Normalizes to - A Python datetime.time object.
- Validates that the given value is either a datetime.datetime, datetime.date or string formatted in a particular datetime format.
- Error message keys - required, invalid
- It takes one optional argument-
  input_formats :- A list of formats used to attempt to convert a string to a valid datetime.datetime object.
- If no input_formats argument is provided the default input formats are taken



from DATE_TIME_INPUT_FORMATS if USE_L10N is False, or from the active locale format DATETIME_INPUT_FORMATS key if localization is enabled. See also format localization.

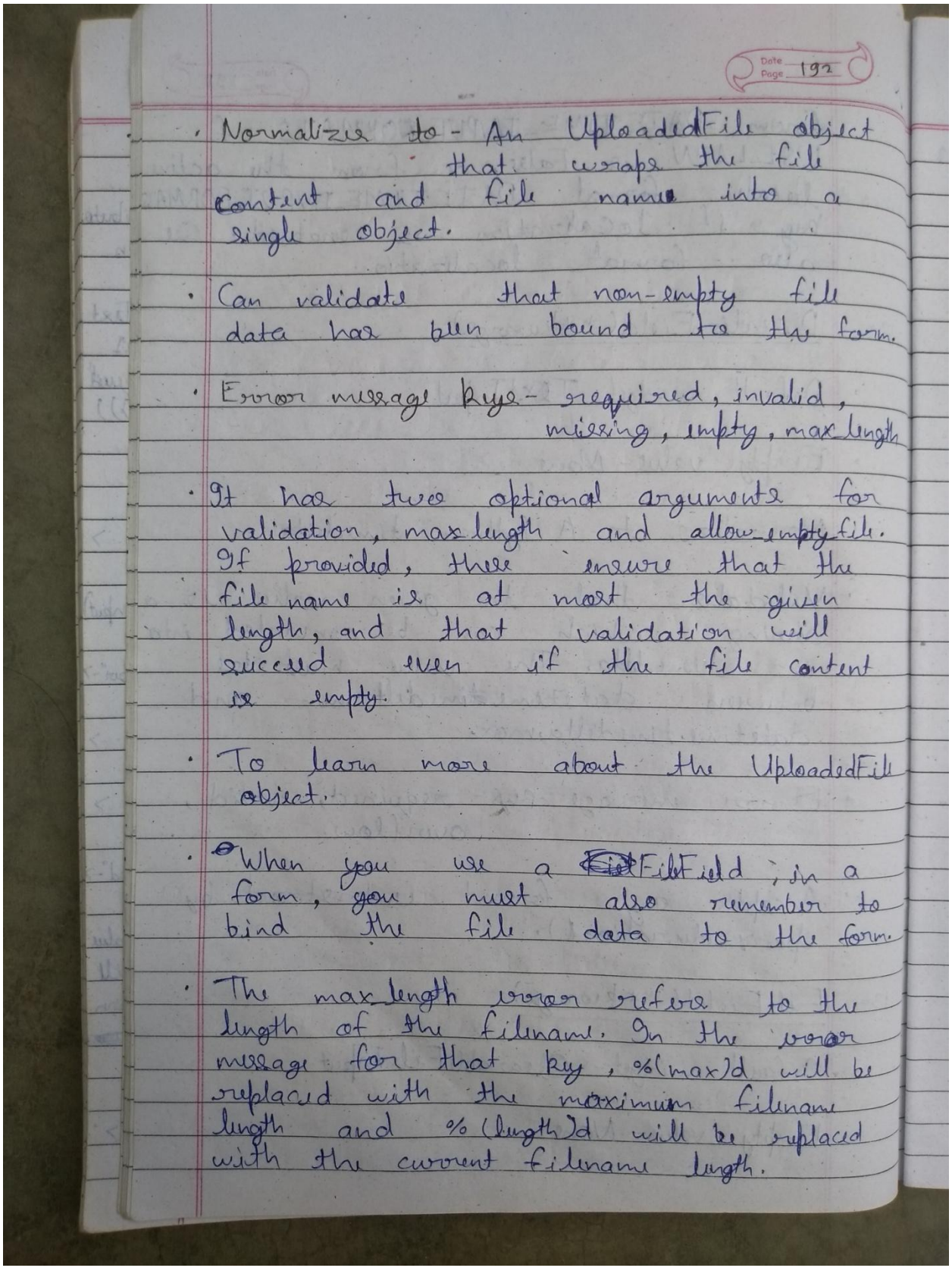
12- DurationField(**kwargs)-

- Default widget - TextInput
- Empty value - None
- Normalizes to - A Python timedelta.
- Validates that the given value is a string which can be converted into a timedelta. The value must be between datetime.timedelta.min and datetime.timedelta.max.
- Error message keys - required, invalid, overflow.
- Accepts any format understood by parse_duration().

13- FileField(**kwargs)-

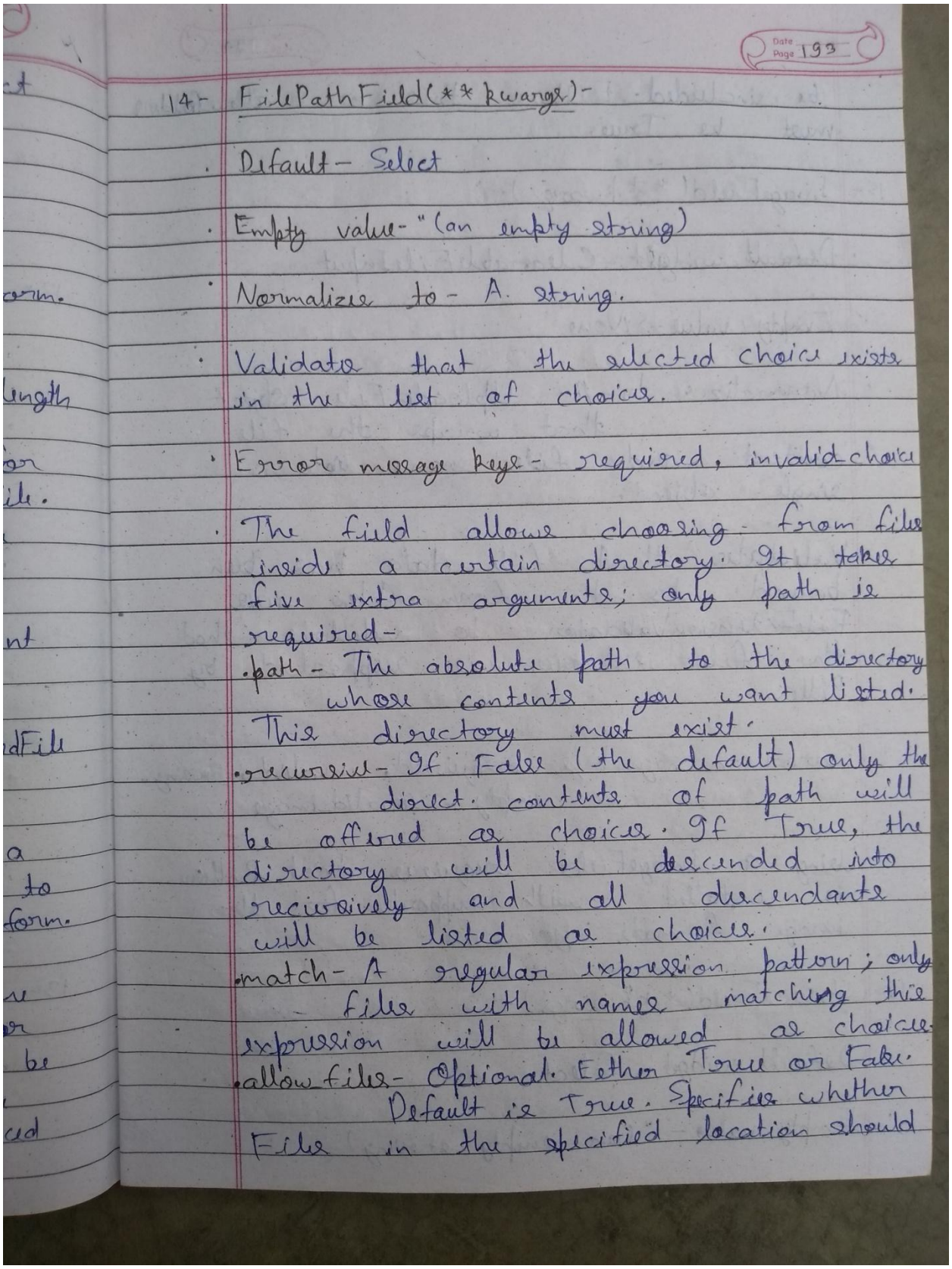
Default widget - ClearableFileInput

Empty value - None



Date Page 192

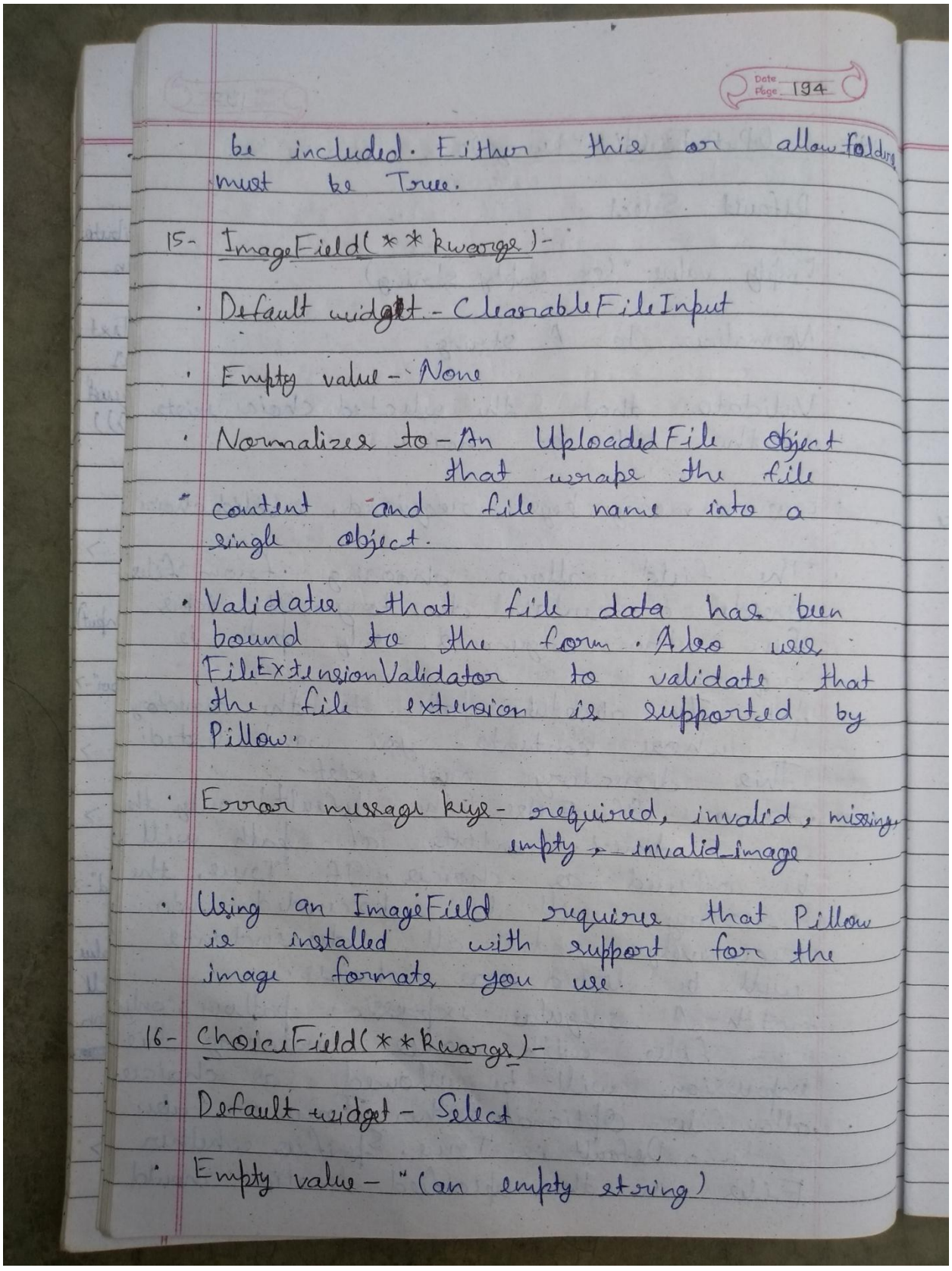
- Normalizes to - An UploadedFile object that wraps the file content and file name into a single object.
- Can validate that non-empty file data has been bound to the form.
- Error message keys - required, invalid, missing, empty, max_length
- It has two optional arguments for validation, max_length and allow_empty_file. If provided, these ensure that the file name is at most the given length, and that validation will succeed even if the file content is empty.
- To learn more about the UploadedFile object.
- When you use a FileField in a form, you must also remember to bind the file data to the form.
- The max_length error refers to the length of the filename. In the error message for that key, %(max)d will be replaced with the maximum filename length and %(length)d will be replaced with the current filename length.



Date Page 193

14- FilePathField(**kwargs)-

- Default - Select
- Empty value - "" (an empty string)
- Normalize to - A string.
- Validates that the selected choice exists in the list of choices.
- Error message keys - required, invalid_choice
- The field allows choosing from files inside a certain directory. It takes five extra arguments; only path is required -
  - path - The absolute path to the directory whose contents you want listed. This directory must exist.
  - recursive - If False (the default) only the direct contents of path will be offered as choices. If True, the directory will be descended into recursively and all descendants will be listed as choices.
  - match - A regular expression pattern; only files with names matching this expression will be allowed as choices.
  - allow_files - Optional. Either True or False. Default is True. Specifies whether files in the specified location should



Date Page 194

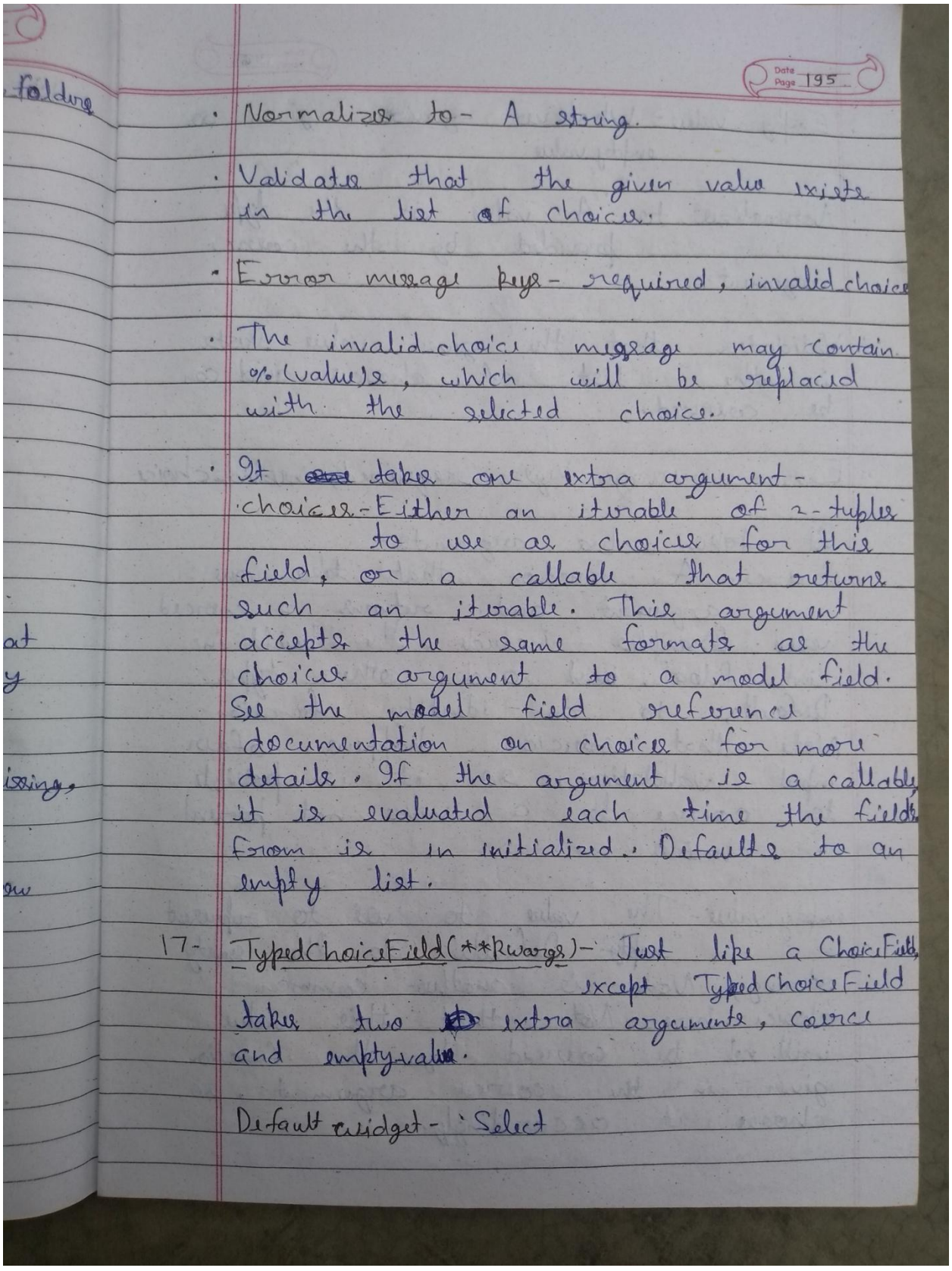
be included. Either this or allow folding must be True.

15- ImageField(**kwargs)-

- Default widget - ClearableFileInput
- Empty value - None
- Normalizes to - An UploadedFile object that wraps the file content and file name into a single object.
- Validates that file data has been bound to the form. Also uses FileExtensionValidator to validate that the file extension is supported by Pillow.
- Error message keys - required, invalid, missing, empty, invalid_image
- Using an ImageField requires that Pillow is installed with support for the image formats you use.

16- ChoiceField(**kwargs)-

- Default widget - Select
- Empty value - "" (an empty string)



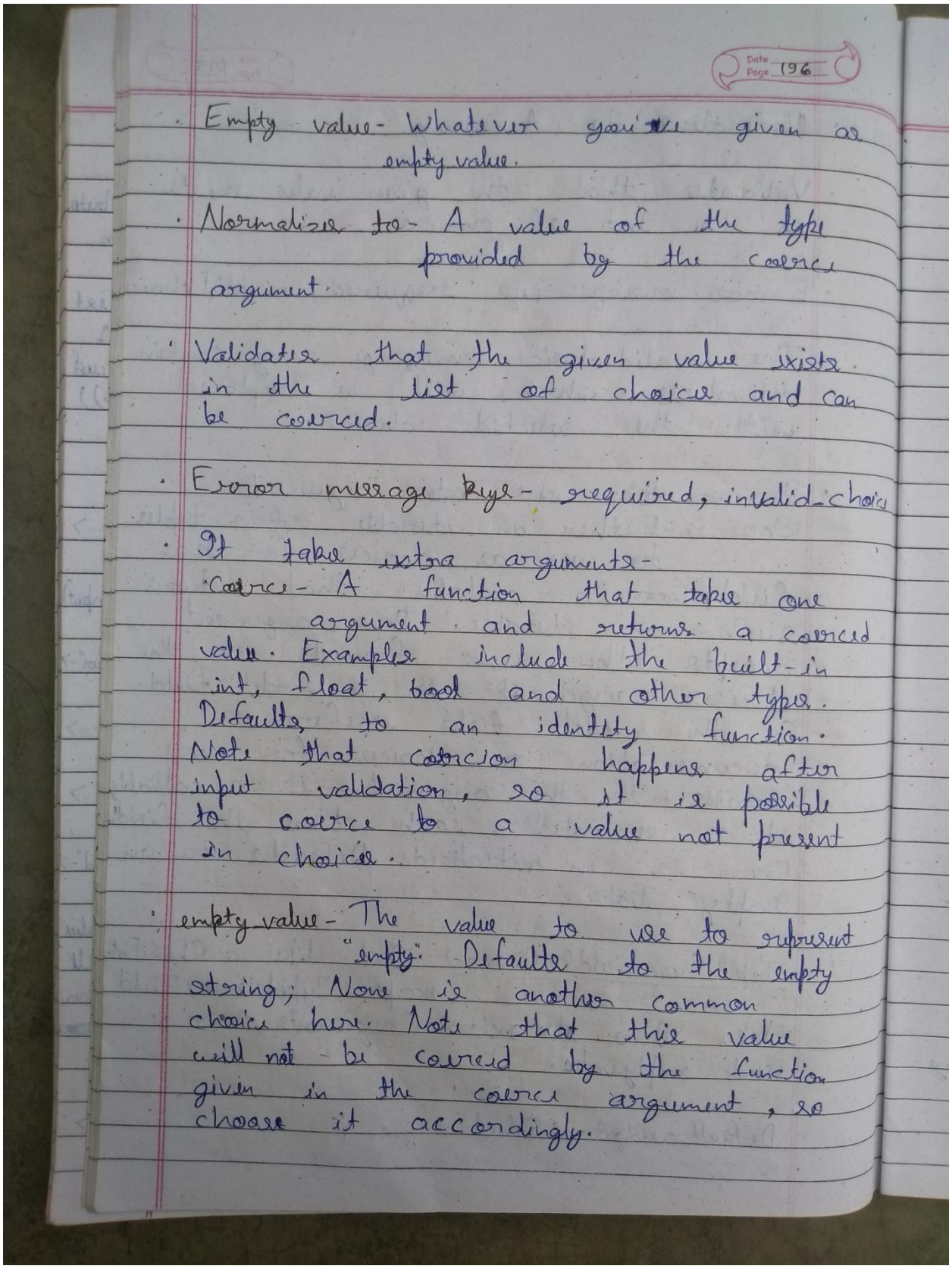
folding

Date Page 195

- Normalizes to - A string.
- Validates that the given value exists in the list of choices.
- Error message keys - required, invalid_choice
- The invalid_choice message may contain %(value)s, which will be replaced with the selected choice.
- It takes one extra argument -
  choices - Either an iterable of 2-tuples to use as choices for this field, or a callable that returns such an iterable. This argument accepts the same formats as the choices argument to a model field. See the model field reference documentation on choices for more details. If the argument is a callable, it is evaluated each time the field's form is initialized. Defaults to an empty list.

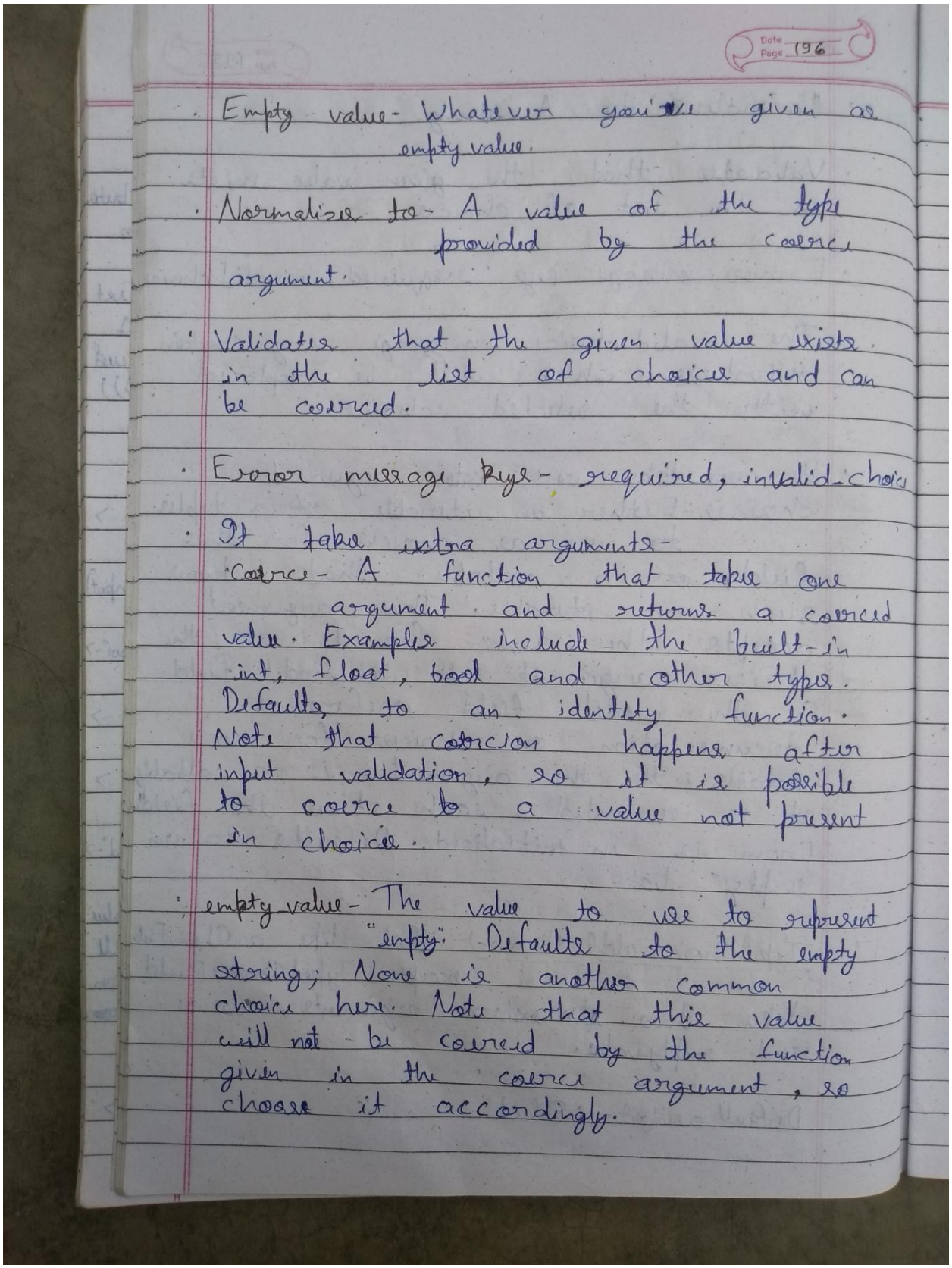
17- TypedChoiceField(**kwargs) - Just like a ChoiceField except TypedChoiceField takes two extra arguments, coerce and empty_value.

Default widget - Select



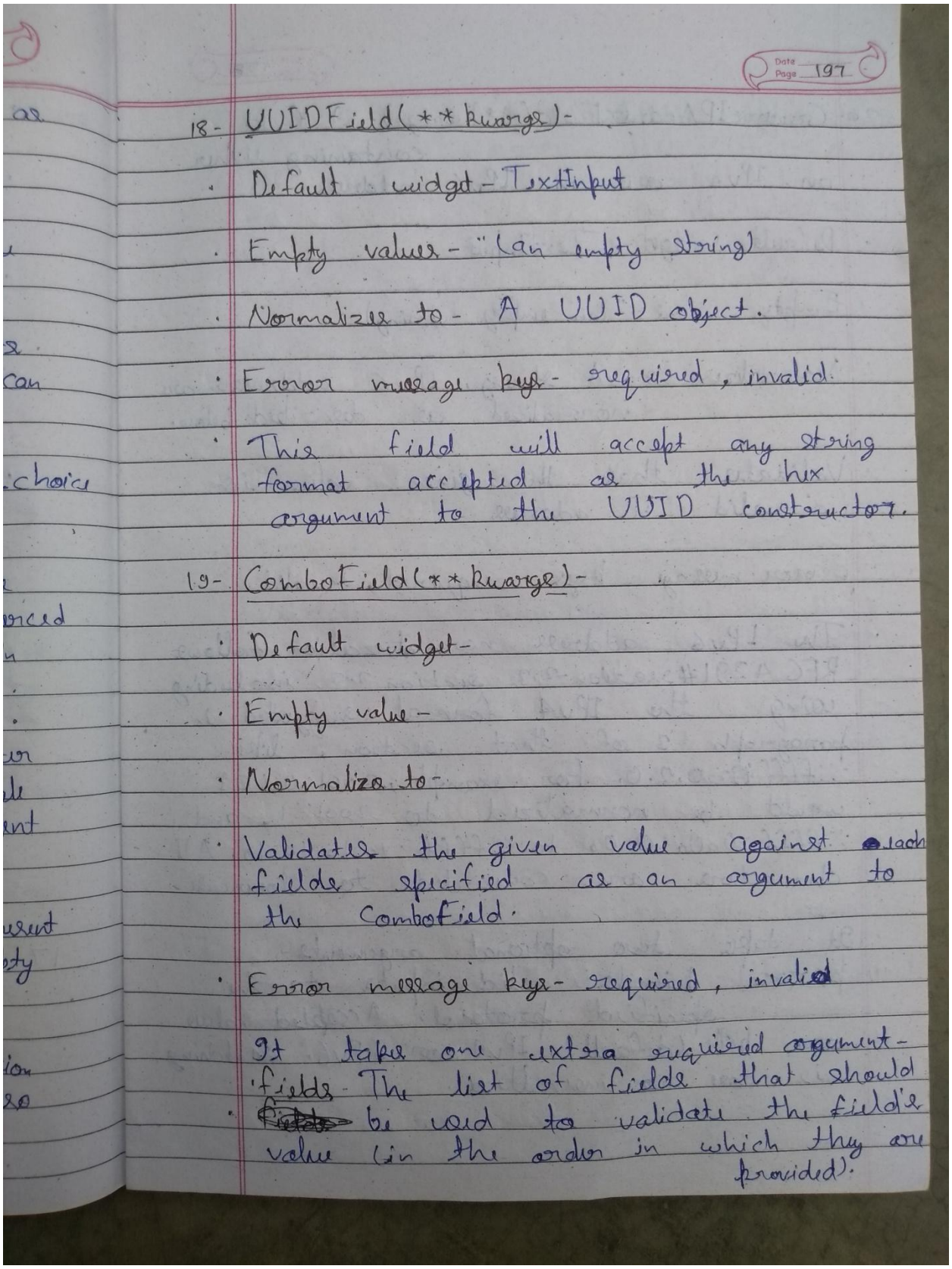
Date Page 196

- Empty value - Whatever you've given as empty value.
- Normalize to - A value of the type provided by the coerce argument.
- Validates that the given value exists in the list of choices and can be coerced.
- Error message keys - required, invalid_choice
- It takes extra arguments -
  - coerce - A function that takes one argument and returns a coerced value. Examples include the built-in int, float, bool and other types. Defaults to an identity function. Note that coercion happens after input validation, so it is possible to coerce to a value not present in choices.
  - empty_value - The value to use to represent "empty". Defaults to the empty string; None is another common choice here. Note that this value will not be coerced by the function given in the coerce argument, so choose it accordingly.



Date Page 196

- Empty value - Whatever you're given as empty value.
- Normalize to - A value of the type provided by the coerce argument.
- Validates that the given value exists in the list of choices and can be coerced.
- Error message keys - required, invalid_choice
- It take extra arguments -
  - coerce - A function that take one argument and returns a coerced value. Examples include the built-in int, float, bool and other types. Defaults to an identity function. Note that coercion happens after input validation, so it is possible to coerce to a value not present in choices.
  - empty_value - The value to use to represent "empty". Defaults to the empty string; None is another common choice here. Note that this value will not be coerced by the function given in the coerce argument, so choose it accordingly.



18- UUIDField(**kwargs)-

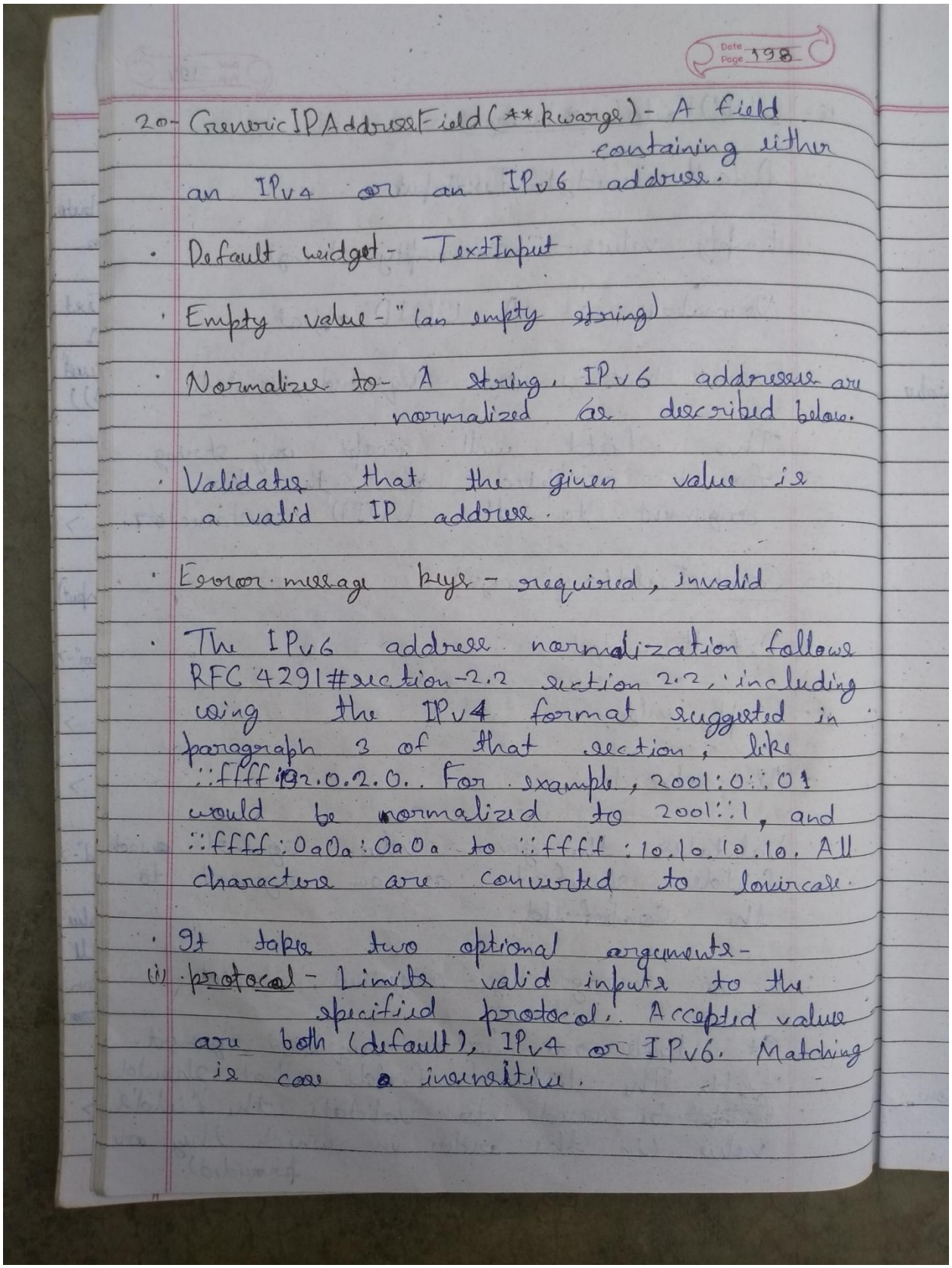
- Default widget - TextInput
- Empty values - '' (an empty string)
- Normalizes to - A UUID object.
- Error message keys - required, invalid.
- This field will accept any string format accepted as the hex argument to the UUID constructor.

19- ComboField(**kwargs)-

- Default widget -
- Empty value -
- Normalizes to -
- Validates the given value against each of the fields specified as an argument to the ComboField.
- Error message keys - required, invalid

It takes one extra required argument -

- fields - The list of fields that should be used to validate the field's value (in the order in which they are provided).

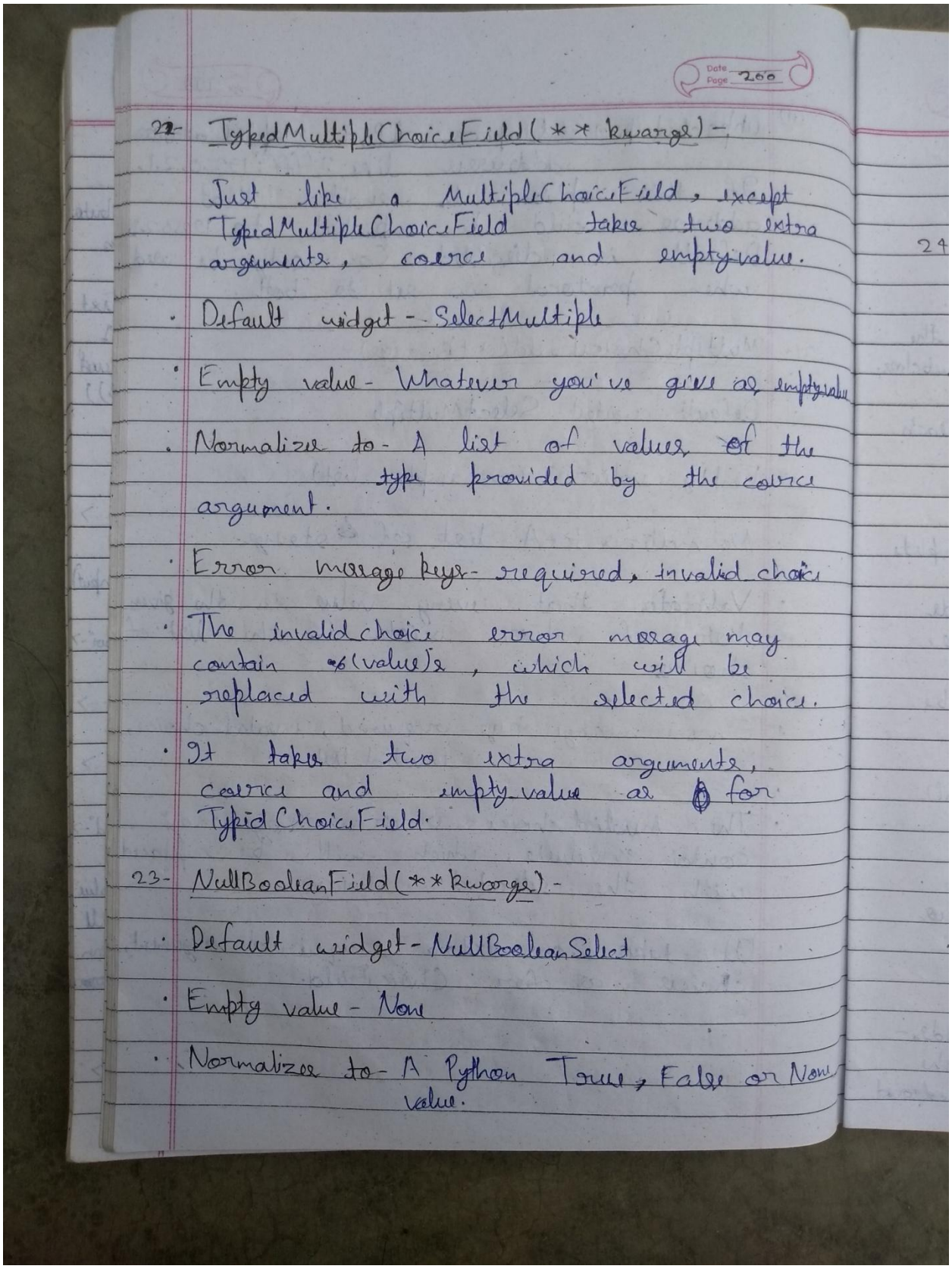


Date Page 198

20- GenericIPAddressField(**kwargs) - A field containing either an IPv4 or an IPv6 address.

- Default widget - TextInput
- Empty value - "" (an empty string)
- Normalize to - A string. IPv6 addresses are normalized as described below.
- Validates that the given value is a valid IP address.
- Error message keys - required, invalid
- The IPv6 address normalization follows RFC 4291#section-2.2 section 2.2, including using the IPv4 format suggested in paragraph 3 of that section, like ::ffff:192.0.2.0. For example, 2001:0::01 would be normalized to 2001::1, and ::ffff:0a0a:0a0a to ::ffff:10.10.10.10. All characters are converted to lowercase.
- It takes two optional arguments -

(i) protocol - Limits valid inputs to the specified protocol. Accepted values are both (default), IPv4 or IPv6. Matching is case insensitive.



Date Page 200

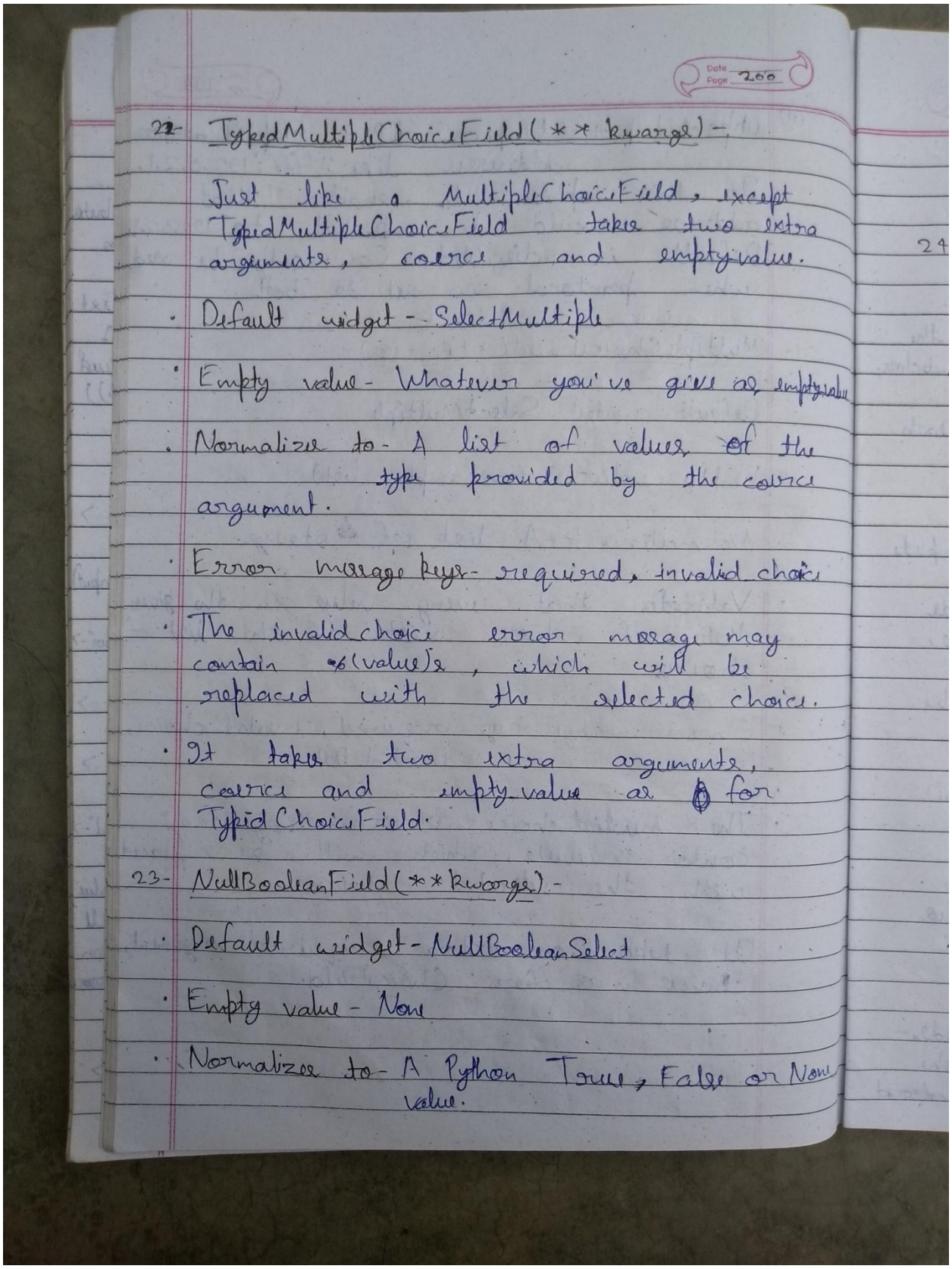
22- TypedMultipleChoiceField(**kwargs) -

Just like a MultipleChoiceField, except TypedMultipleChoiceField takes two extra arguments, coerce and empty_value.

- Default widget - SelectMultiple
- Empty value - Whatever you've given as empty_value
- Normalizes to - A list of values of the type provided by the coerce argument.
- Error message keys - required, invalid_choice
- The invalid_choice error message may contain %(value)s, which will be replaced with the selected choice.
- It takes two extra arguments, coerce and empty_value as '' for TypedChoiceField.

23- NullBooleanField(**kwargs) -

- Default widget - NullBooleanSelect
- Empty value - None
- Normalizes to - A Python True, False or None value.



Date Page 200

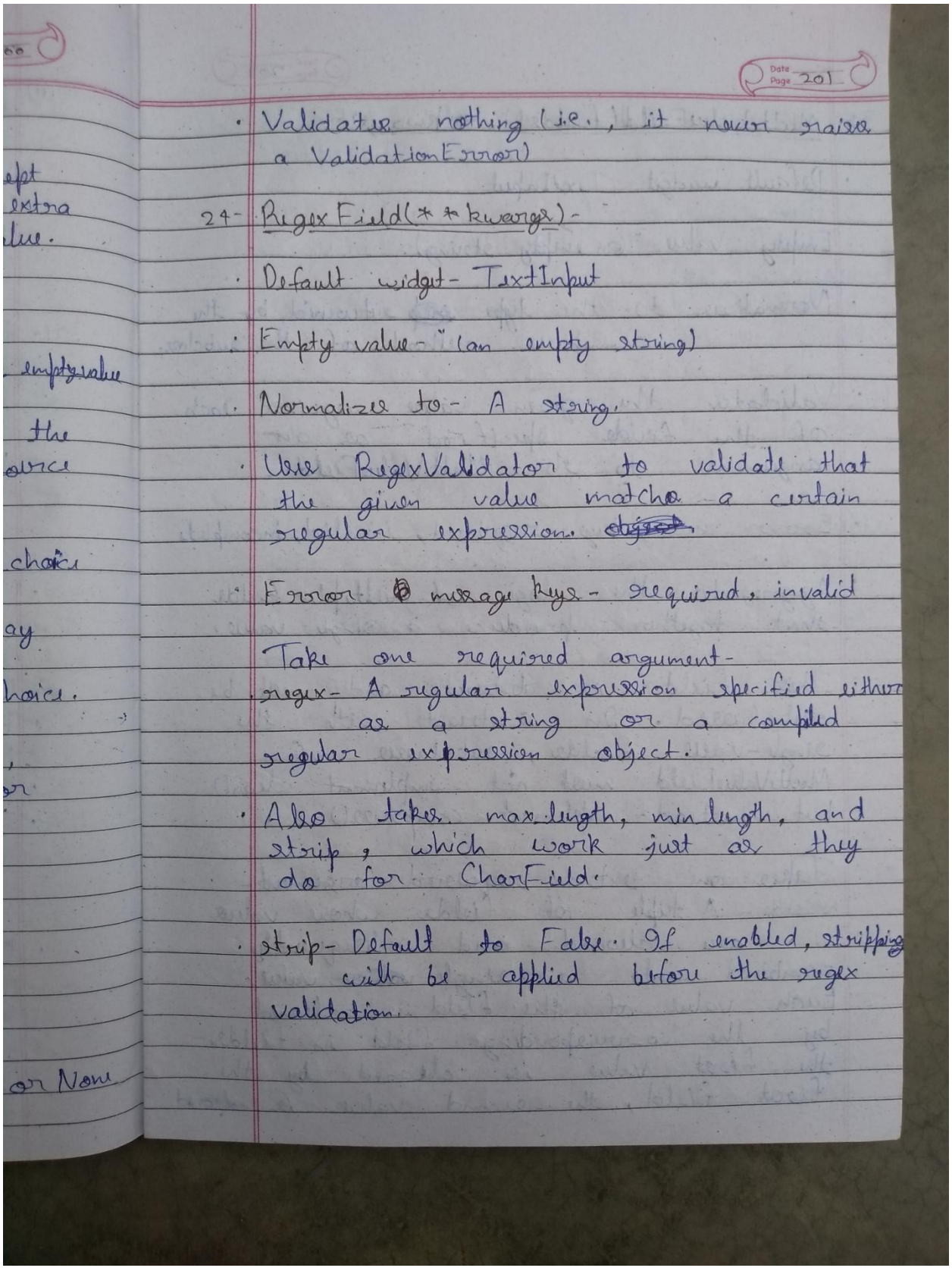
22- TypedMultipleChoiceField(**kwargs) -

Just like a MultipleChoiceField, except TypedMultipleChoiceField takes two extra arguments, coerce and empty_value.

- Default widget - SelectMultiple
- Empty value - Whatever you've given as empty_value
- Normalizes to - A list of values of the type provided by the coerce argument.
- Error message keys - required, invalid_choice
- The invalid_choice error message may contain %(value)s, which will be replaced with the selected choice.
- It takes two extra arguments, coerce and empty_value as '' for TypedChoiceField.

23- NullBooleanField(**kwargs) -

- Default widget - NullBooleanSelect
- Empty value - None
- Normalizes to - A Python True, False or None value.



- Validates nothing (i.e., it never raises a ValidationError)

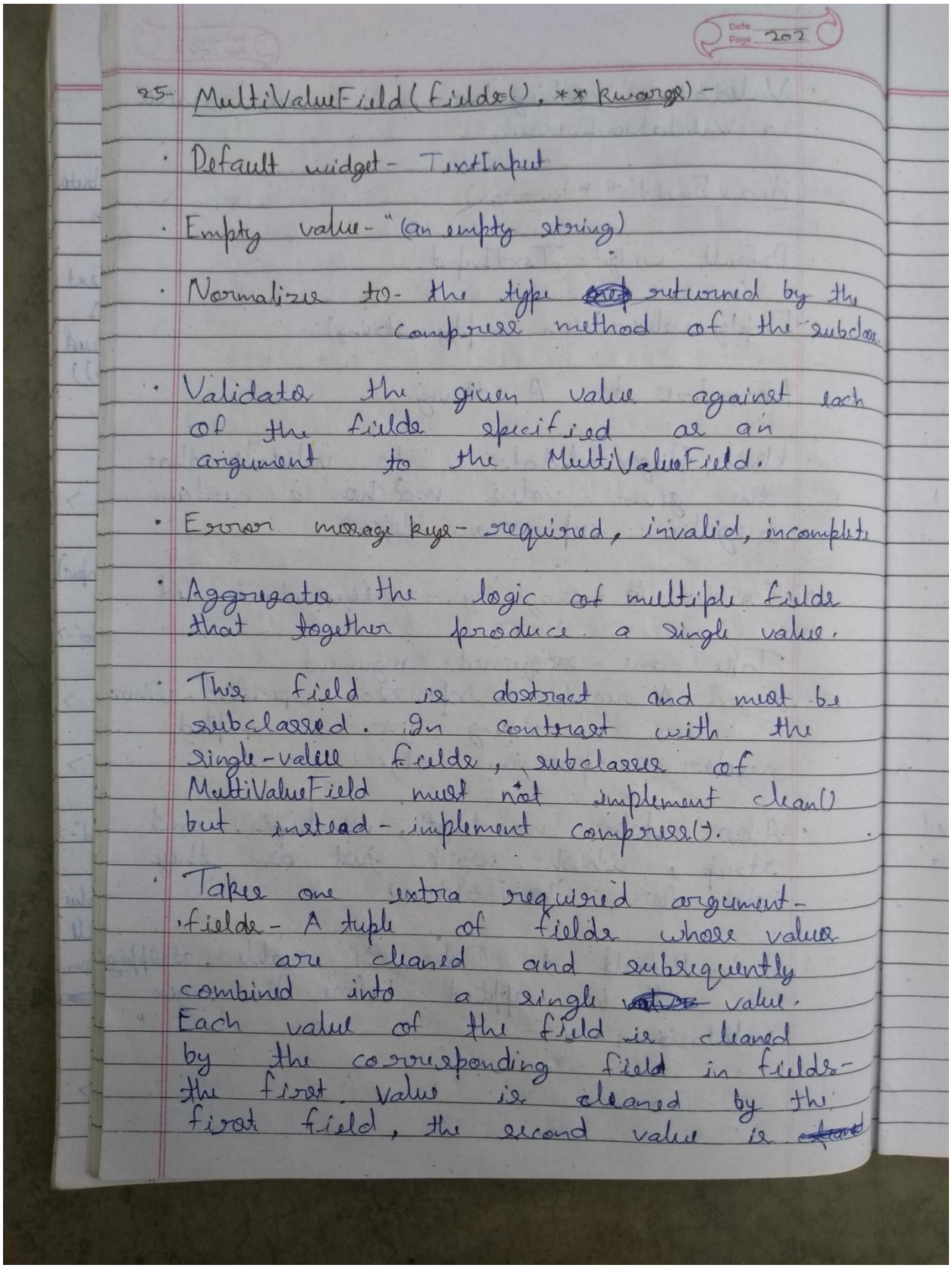
24- RegexField(**kwargs)-

- Default widget- TextInput
- Empty value- '' (an empty string)
- Normalizes to- A string.
- Uses RegexValidator to validate that the given value matches a certain regular expression.
- Error message keys- required, invalid

Takes one required argument-

regex- A regular expression specified either as a string or a compiled regular expression object.

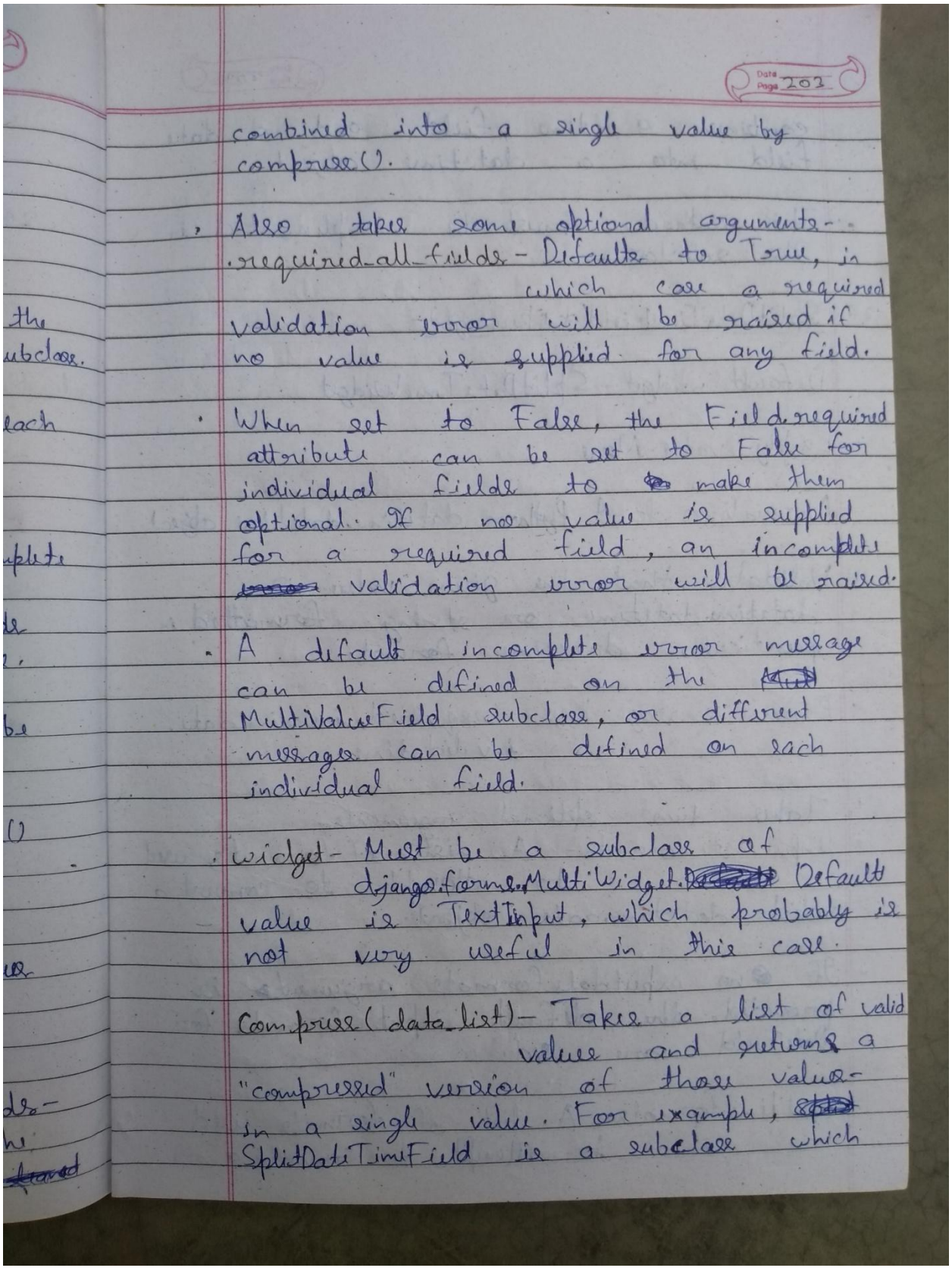
- Also takes max_length, min_length, and strip, which work just as they do for CharField.
- strip- Default to False. If enabled, stripping will be applied before the regex validation.



Date Page 202

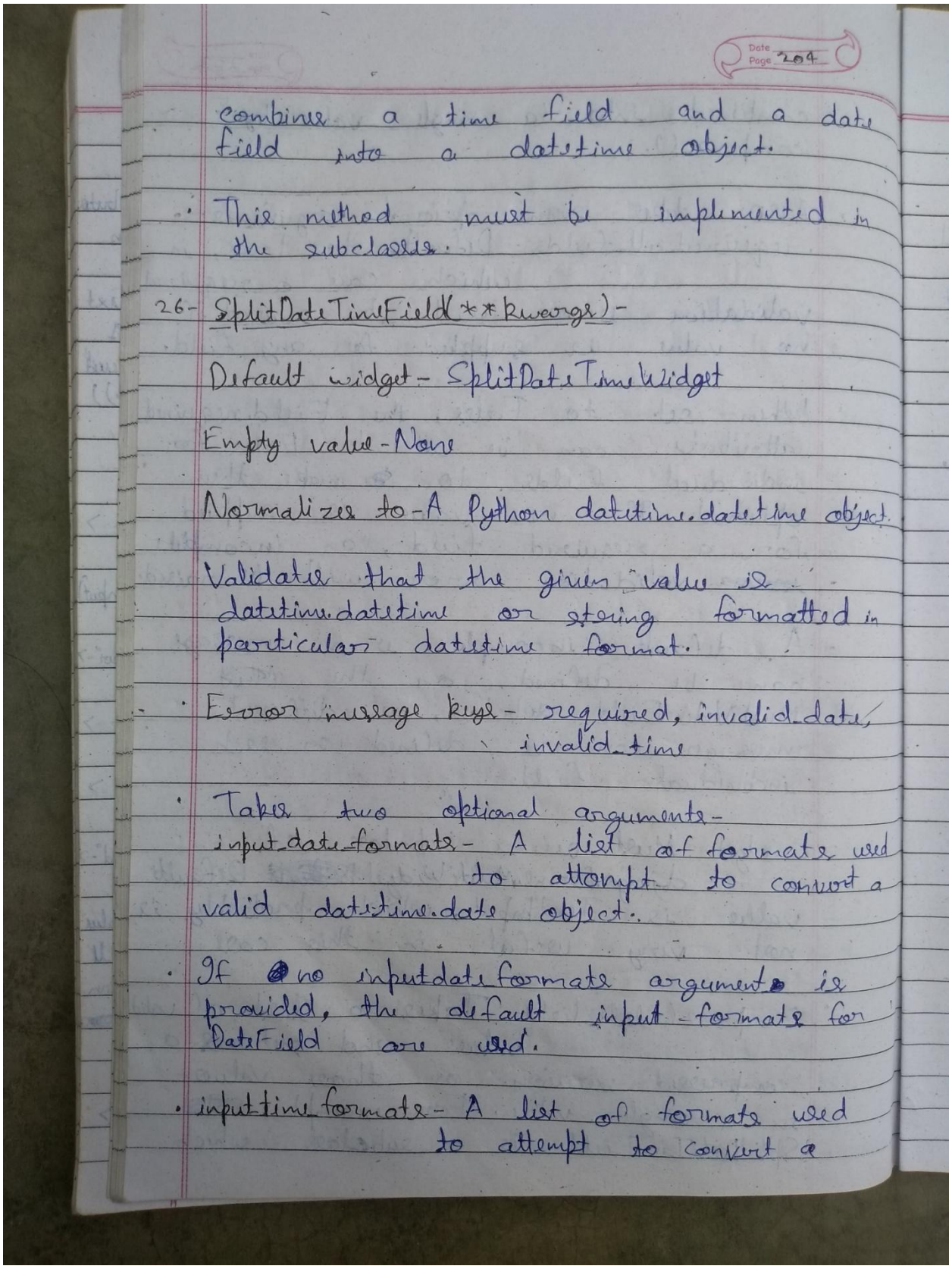
25- MultiValueField(fields(), **kwargs) -

- Default widget - TextInput
- Empty value - "" (an empty string)
- Normalize to - the type returned by the compress method of the subclass.
- Validates the given value against each of the fields specified as an argument to the MultiValueField.
- Error message keys - required, invalid, incomplete
- Aggregates the logic of multiple fields that together produce a single value.
- This field is abstract and must be subclassed. In contrast with the single-value fields, subclasses of MultiValueField must not implement clean() but instead - implement compress().
- Takes one extra required argument -
  fields - A tuple of fields whose values are cleaned and subsequently combined into a single value. Each value of the field is cleaned by the corresponding field in fields - the first value is cleaned by the first field, the second value is



combined into a single value by compress().

- Also takes some optional arguments-
  - required_all_fields - Defaults to True, in which case a required validation error will be raised if no value is supplied for any field.
  - When set to False, the Field.required attribute can be set to False for individual fields to make them optional. If no value is supplied for a required field, an incomplete validation error will be raised.
  - A default incomplete error message can be difined on the MultiValueField subclass, or different messages can be difined on each individual field.
  - widget - Must be a subclass of django.forms.MultiWidget. Default value is TextInput, which probably is not very useful in this case.
  - Compress(data_list) - Takes a list of valid values and returns a "compressed" version of those values- in a single value. For example, SplitDateTimeField is a subclass which



Date Page 204

combines a time field and a date field into a datetime object.

- This method must be implemented in the subclasses.

26- SplitDateTimeField(**kwargs)-

Default widget - SplitDateTimeWidget

Empty value - None

Normalizes to - A Python datetime.datetime object.

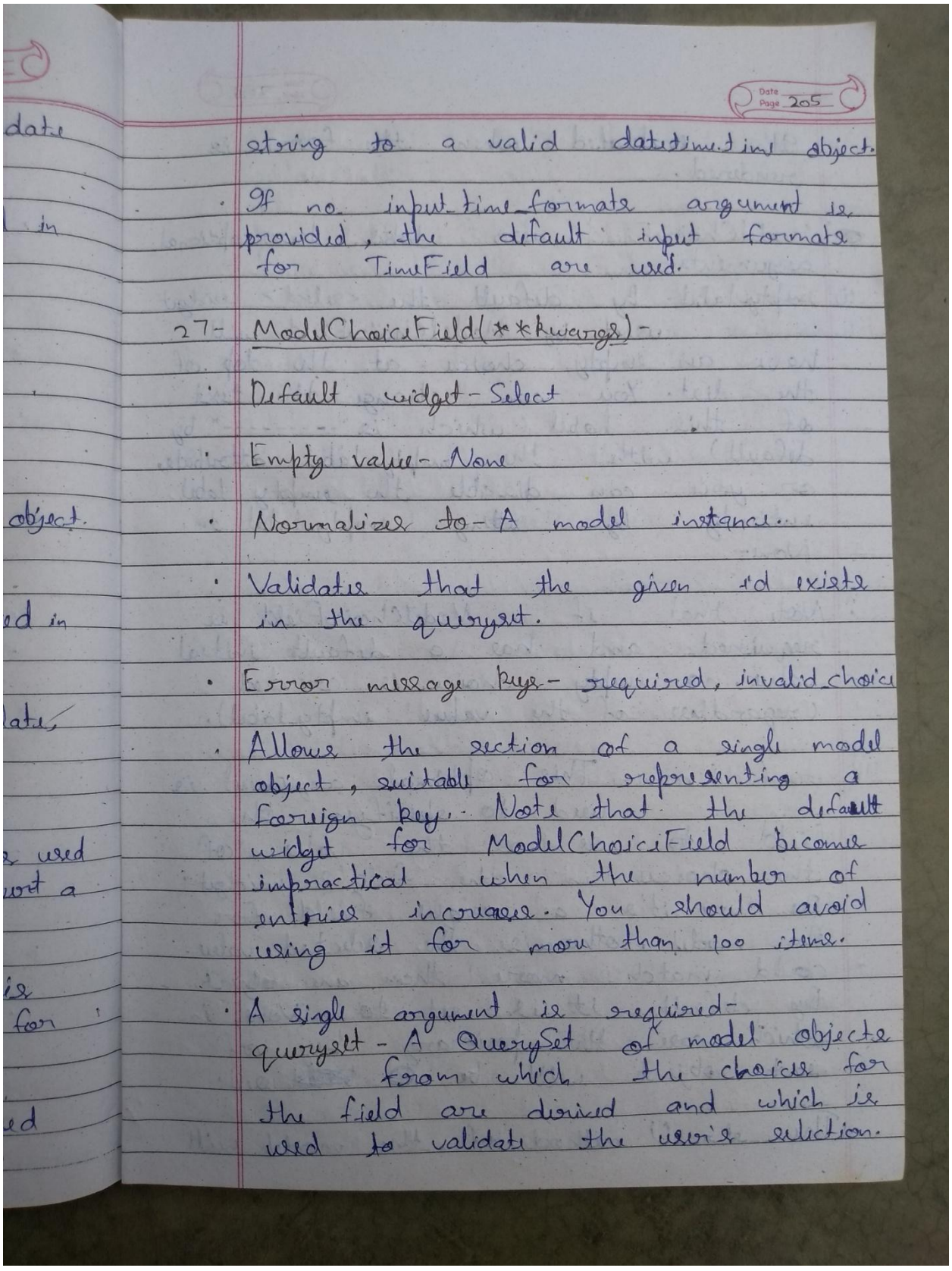
Validates that the given value is a datetime.datetime or string formatted in particular datetime format.

- Error message keys - required, invalid_date, invalid_time

- Takes two optional arguments -
  input_date_formats - A list of formats used to attempt to convert a valid datetime.date object.

- If no input_date_formats argument is provided, the default input formats for DateField are used.

- input_time_formats - A list of formats used to attempt to convert a

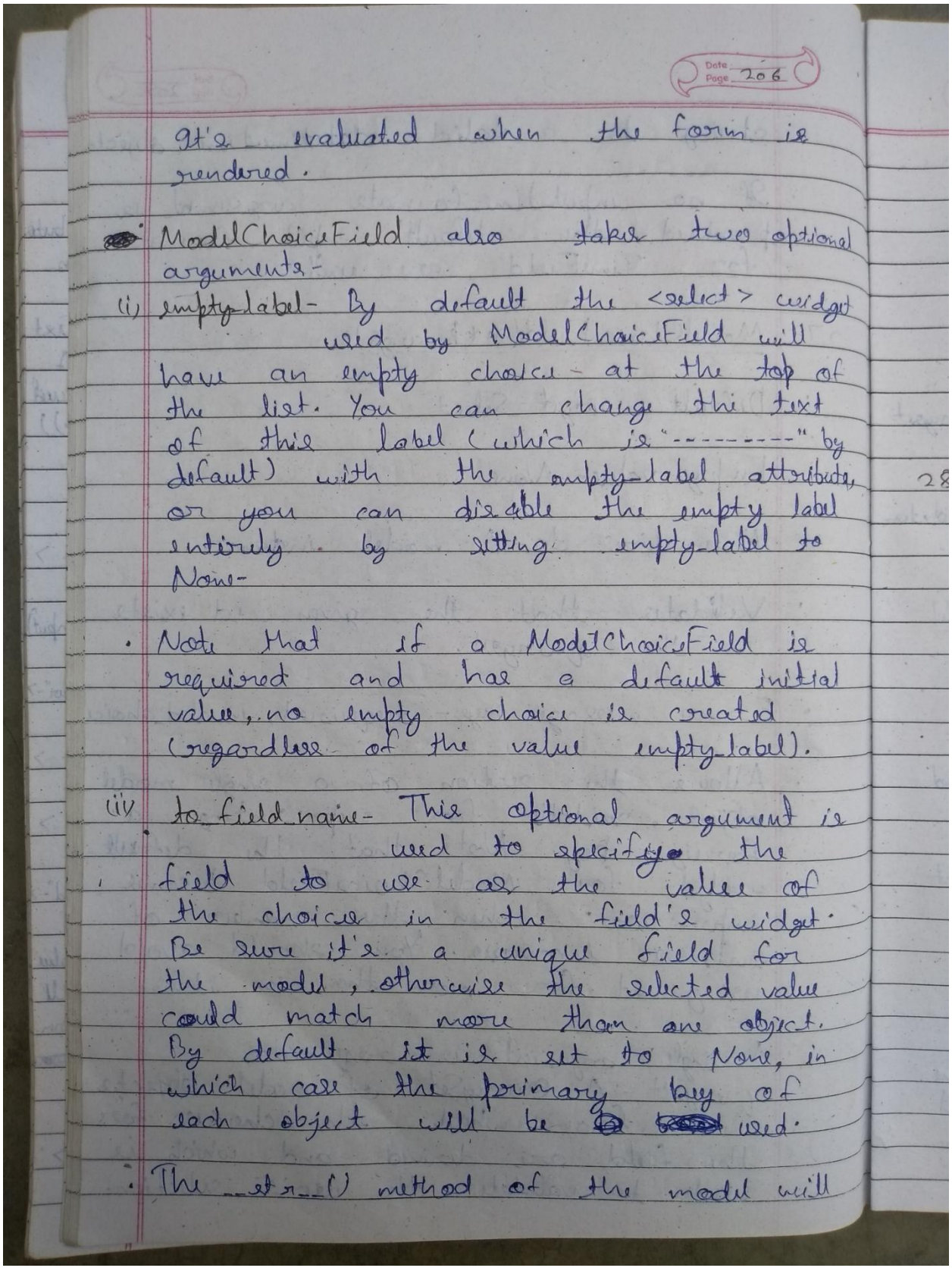


string to a valid datetime.time object.

- If no input_time_formats argument is provided, the default input formats for TimeField are used.

27- ModelChoiceField(**kwargs) -

- Default widget - Select
- Empty value - None
- Normalizes to - A model instance.
- Validates that the given id exists in the queryset.
- Error message keys - required, invalid_choice
- Allows the selection of a single model object, suitable for representing a foreign key. Note that the default widget for ModelChoiceField becomes impractical when the number of entries increases. You should avoid using it for more than 100 items.
- A single argument is required -
  queryset - A QuerySet of model objects from which the choices for the field are derived and which is used to validate the user's selection.



It's evaluated when the form is rendered.

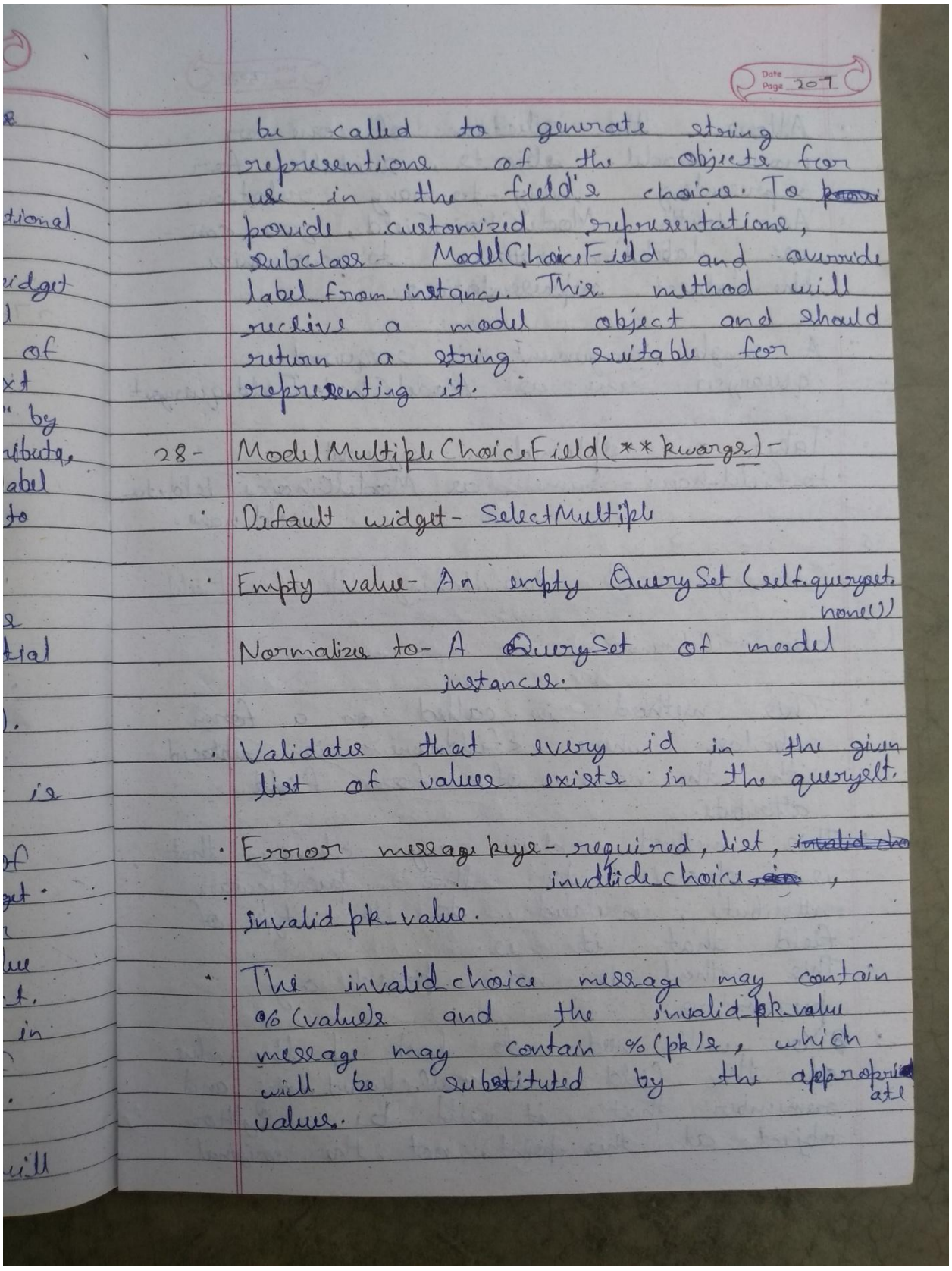
- ModelChoiceField also takes two optional arguments-

(i) empty_label- By default the <select> widget used by ModelChoiceField will have an empty choice at the top of the list. You can change the text of this label (which is "---------" by default) with the empty_label attribute, or you can disable the empty label entirely by setting empty_label to None-

- Note that if a ModelChoiceField is required and has a default initial value, no empty choice is created (regardless of the value empty_label).

(ii) to_field_name- This optional argument is used to specify the field to use as the value of the choices in the field's widget. Be sure it's a unique field for the model, otherwise the selected value could match more than one object. By default it is set to None, in which case the primary key of each object will be used.

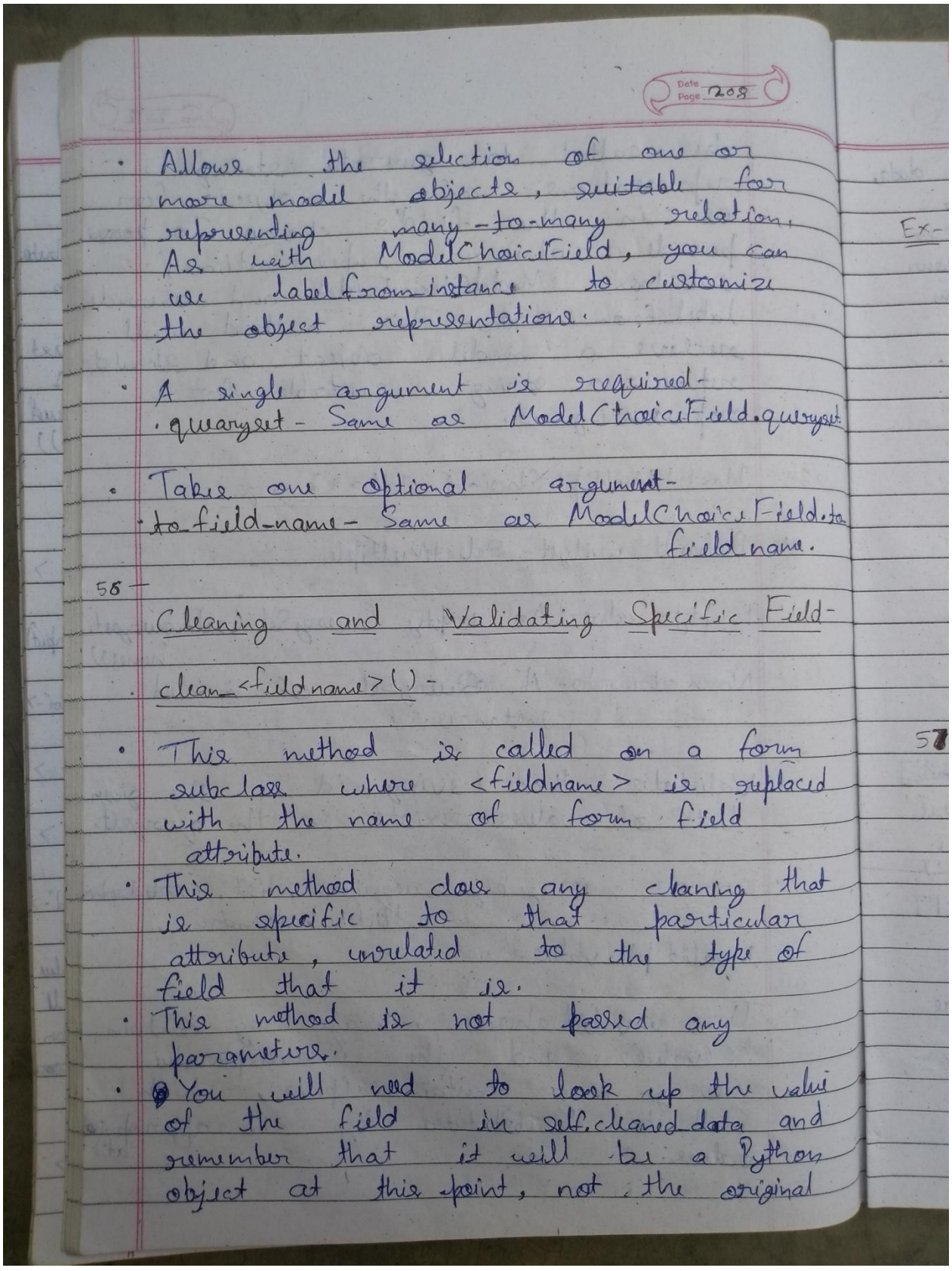
- The __str__() method of the model will



be called to generate string representations of the objects for use in the field's choices. To provide customized representations, subclass ModelChoiceField and override label_from_instance. This method will receive a model object and should return a string suitable for representing it.

28- ModelMultipleChoiceField(**kwargs)-

- Default widget- SelectMultiple
- Empty value- An empty QuerySet (self.queryset.none())
- Normalize to- A QuerySet of model instances.
- Validates that every id in the given list of values exists in the queryset.
- Error message keys- required, list, invalid_choice, invalid_pk_value.
- The invalid_choice message may contain %(value)s and the invalid_pk_value message may contain %(pk)s, which will be substituted by the appropriate values.



Date Page 208

- Allows the selection of one or more model objects, suitable for representing many-to-many relation. As with ModelChoiceField, you can use label_from_instance to customize the object representations.
- A single argument is required-
  - queryset - Same as ModelChoiceField.queryset
- Takes one optional argument-
  - to_field_name - Same as ModelChoiceField.to_field_name.

Ex-

56 -

Cleaning and Validating Specific Field-

clean_<fieldname>()-

- This method is called on a form subclass where <fieldname> is replaced with the name of form field attribute.
- This method does any cleaning that is specific to that particular attribute, unrelated to the type of field that it is.
- This method is not passed any parameters.
- You will need to look up the value of the field in self.cleaned_data and remember that it will be a Python object at this point, not the original

57

Date 8-6-20
Page 209

string submitted in the form.

Ex- forms.py -

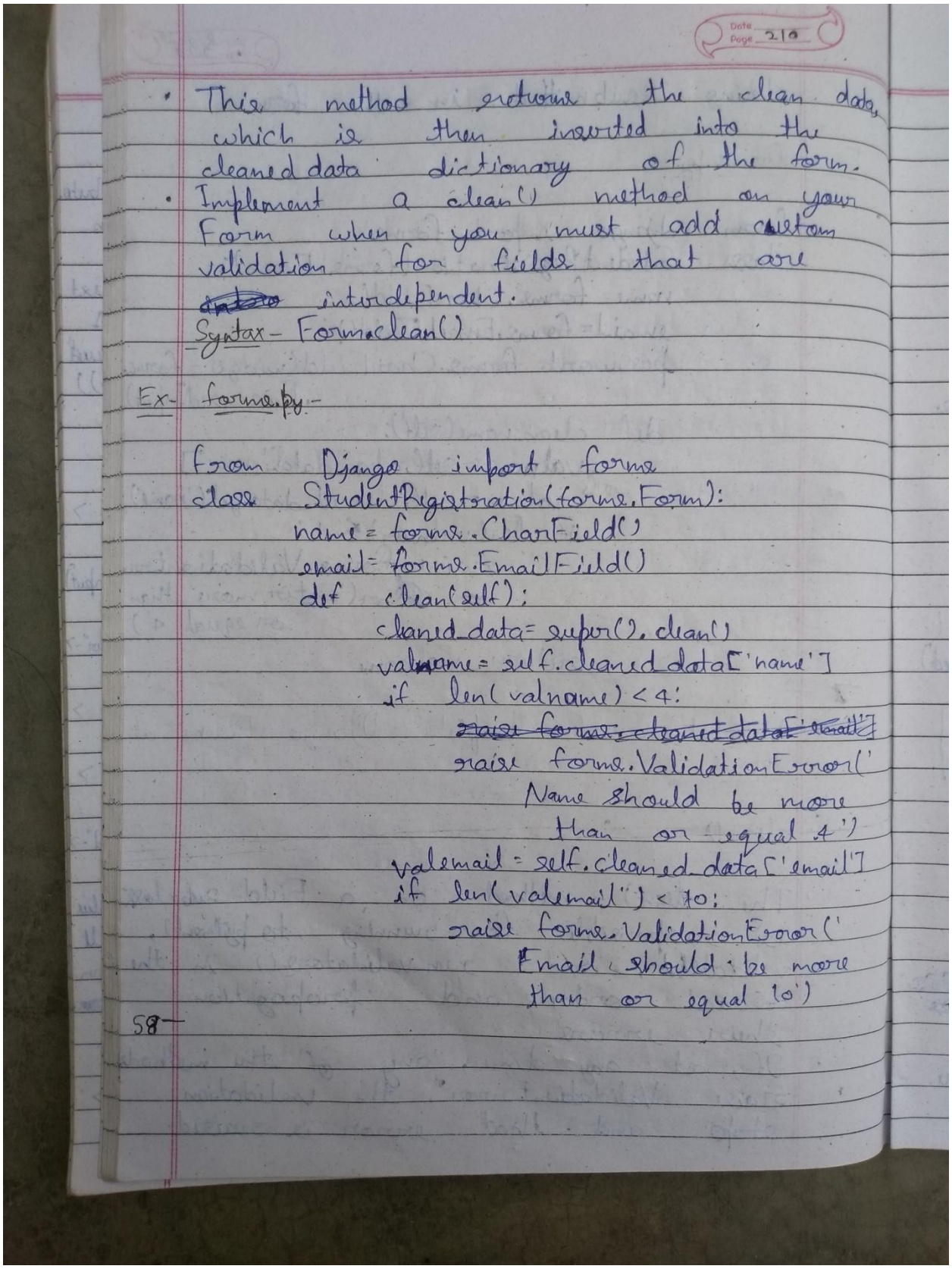
```
from django import forms
class StudentRegistration(forms.Form):
    name = forms.CharField()
    email = forms.EmailField()
    password = forms.CharField(widget = forms
                .PasswordInput)
    def clean_name(self):
        valname = self.clean_data['name']
    or  valnum = self.cleaned_data.get('name')
        if len(valnum) < 4:
            raise forms.ValidationError
            or('Enter more than
                or equal 4')
        return valname
```

5 -

Validation of Complete Django Form at once -

- clean() -

- The clean() method on a Field subclass is responsible for running to_python(), validate(), and run_validators() in the correct order and propagating their errors.
- If, at any time, any of the methods raise ValidationError, the validation stops and that error is raised.



- This method returns the clean data, which is then inserted into the cleaned data dictionary of the form.
- Implement a clean() method on your Form when you must add custom validation for fields that are ~~dato~~ interdependent.

Syntax - Form.clean()

Ex- forms.py -

```
From Django import forms
class StudentRegistration(forms.Form):
    name = forms.CharField()
    email = forms.EmailField()
    def clean(self):
        cleaned_data = super().clean()
        valname = self.cleaned_data['name']
        if len(valname) < 4:
            ~~raise forms.cleaned_data['email']~~
            raise forms.ValidationError('
                Name should be more
                than or equal 4')
        valemail = self.cleaned_data['email']
        if len(valemail') < 10:
            raise forms.ValidationError('
                Email should be more
                than or equal 10')
```

58

Using Built-in Validators – We can use Built-in Validators, available in django.core module

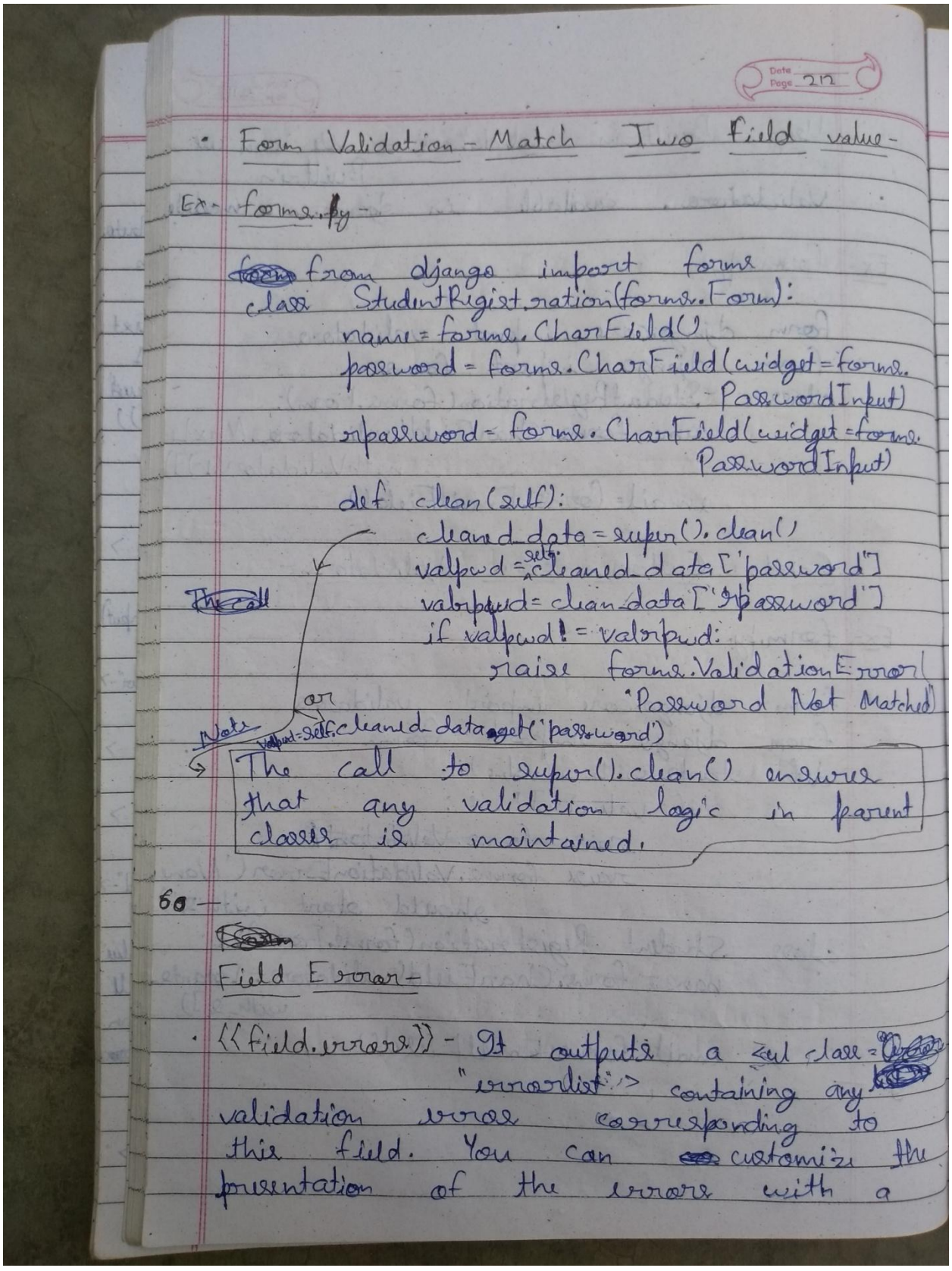
Ex- forms.py -

```
form django.core import validators
form django import forms
class StudentRegistration(forms.Form):
    name = forms.CharField(validators:MaxLength Validator(10)])
    email = forms.EmailField()
```

Create Custom Form Validators -

Ex- form.py

```
form django.core import validators
from django import forms
def starts_with_s(value):
    if value[0] != 's':
        ~~raise forms.ValidationError~~
        raise forms.ValidationError('Name should start with s')
class Student Registration(forms.Form):
    name = forms.CharField(validators = [starts_with_s])
    email = forms.EmailField()
```



Date Page 217

• Form Validation - Match Two field value -

Ex - forms.py -

```
from django import forms
class StudentRigistration(forms.Form):
    name = forms.CharField()
    password = forms.CharField(widget=forms.
                               PasswordInput)
    rpassword = forms.CharField(widget=forms.
                                PasswordInput)

    def clean(self):
        cleaned_data = super().clean()
        valpwd = self.cleaned_data['password']
        valrpwd = clean_data['rpassword']
        if valpwd != valrpwd:
            raise forms.ValidationError(
                'Password Not Matched')
```

The call

or

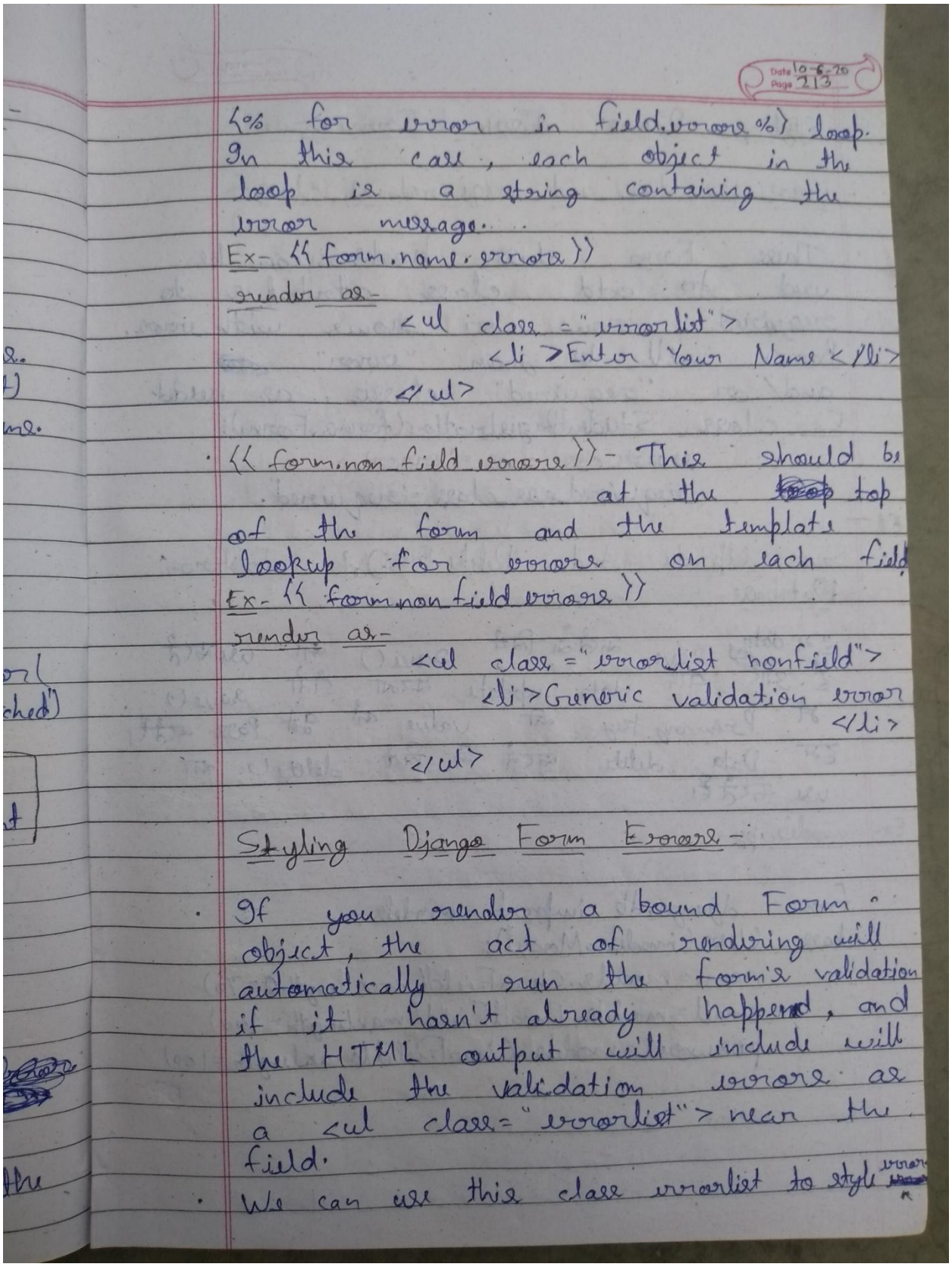
Note: valpwd = self.cleaned_data.get('password')

The call to super().clean() ensures that any validation logic in parent classes is maintained.

60 -

Field Errors -

• {{field.errors}} - It outputs a <ul class="errorlist"> containing any validation errors corresponding to this field. You can customize the presentation of the errors with a



{% for error in field.errors %} loop. In this case, each object in the loop is a string containing the error message.

Ex- {{ form.name.errors }}

render as-

```
<ul class="errorlist">
    <li>Enter Your Name</li>
</ul>
```

- {{ form.non_field_errors }} - This should be at the top of the form and the template lookup for errors on each field.

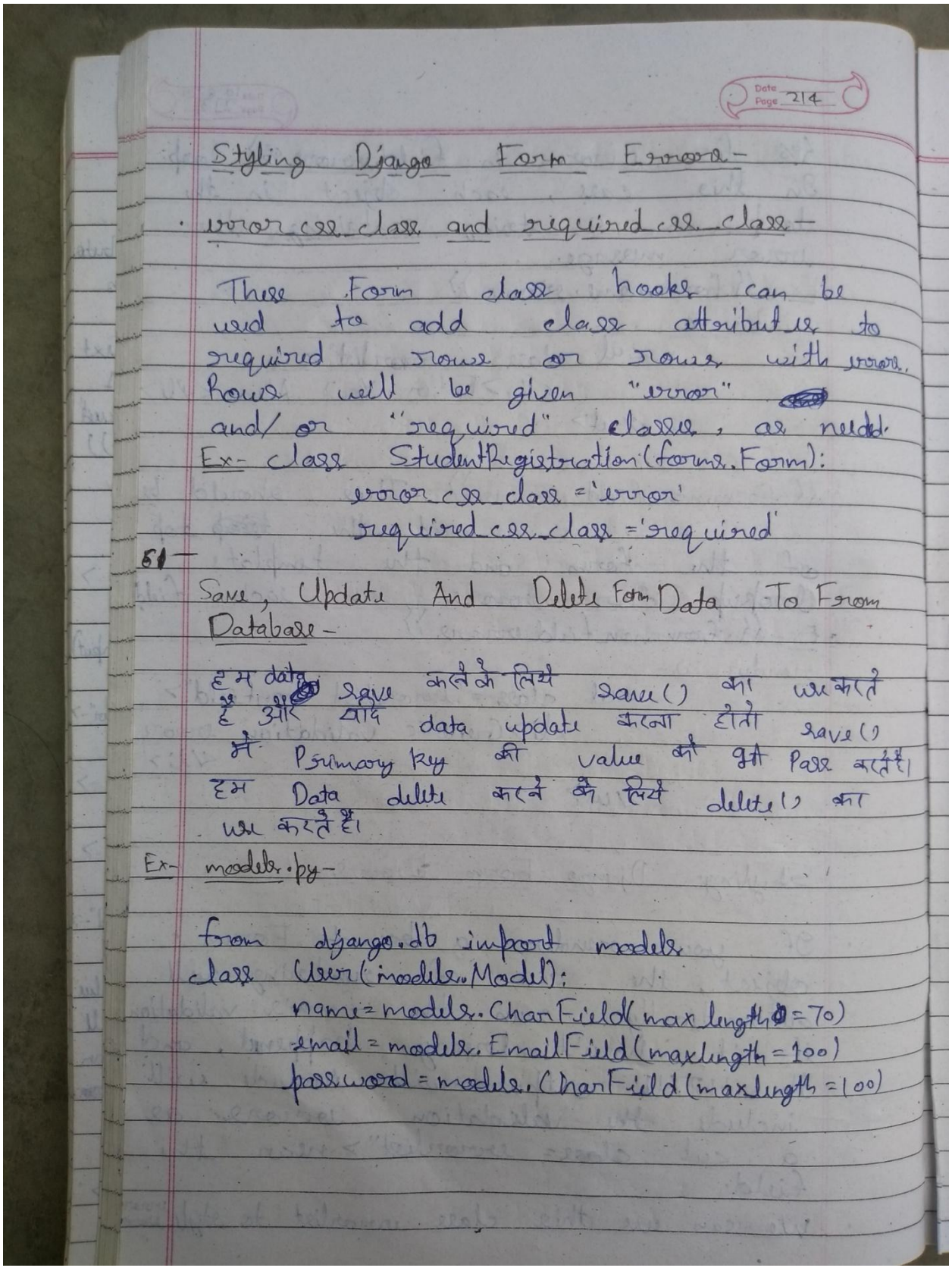
Ex- {{ form.non_field_errors }}

render as-

```
<ul class="errorlist nonfield">
    <li>Generic validation error</li>
</ul>
```

Styling Django Form Errors -

- If you render a bound Form object, the act of rendering will automatically run the form's validation if it hasn't already happened, and the HTML output will include will include the validation errors as a <ul class="errorlist"> near the field.
- We can use this class errorlist to style



Date Page 214

Styling Django Form Errors-

· error_css_class and required_css_class-

These Form class hooks can be used to add class attributes to required rows or rows with errors. Rows will be given "error" and/or "required" class, as needed.

Ex- class StudentRegistration(forms.Form):
    error_css_class = 'error'
    required_css_class = 'required'

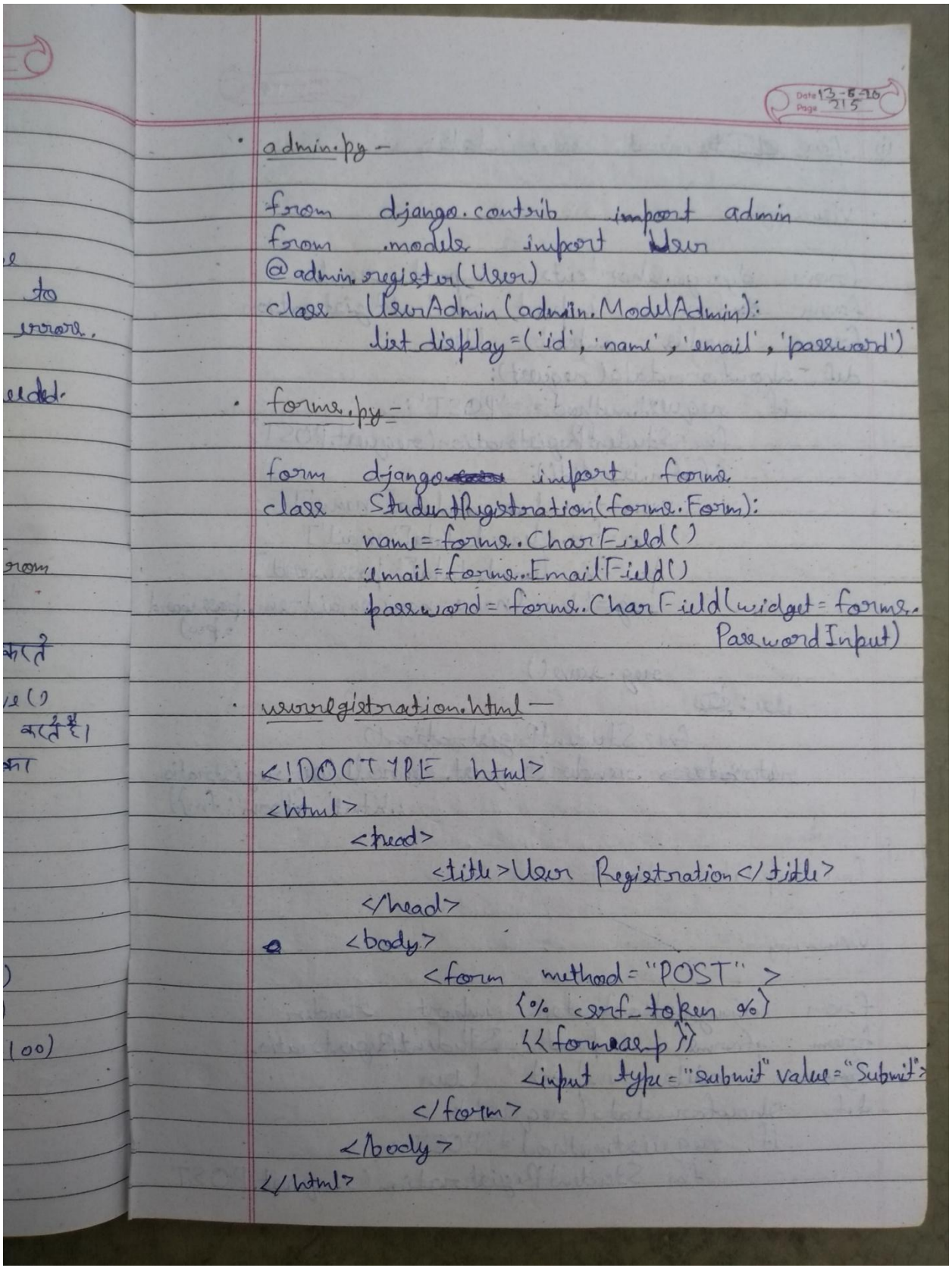
51-

Save, Update And Delete Form Data To From Database-

हम data save करने के लिये save() का use करते है और यदि data update करना होतो save() में Primary key की value को भी Pass करते है। हम Data delete करने के लिये delete() का use करते है।

Ex- models.py-

from django.db import models
class User(models.Model):
    name = models.CharField(max_length=70)
    email = models.EmailField(max_length=100)
    password = models.CharField(max_length=100)



admin.py –

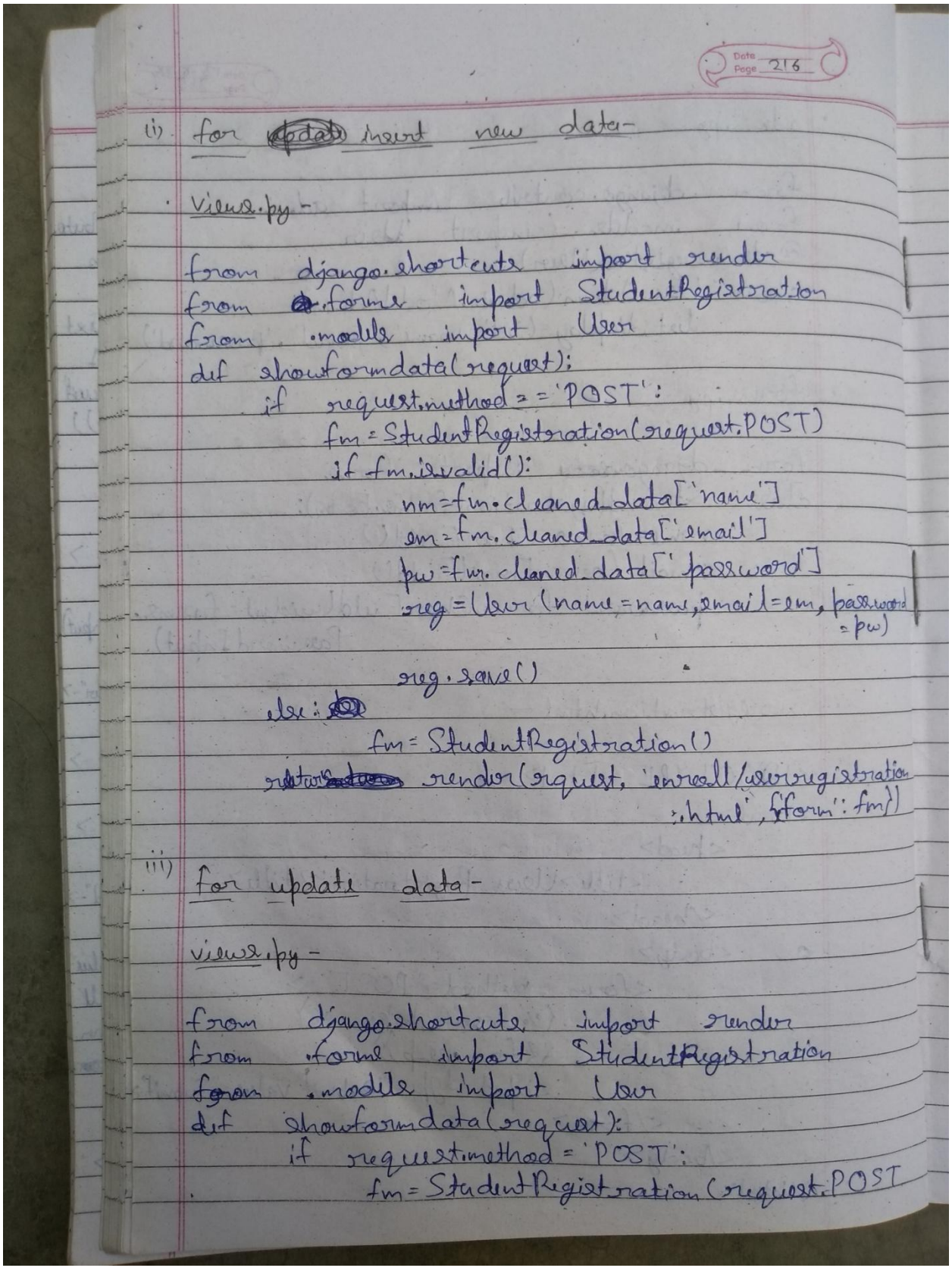
```
from django.contrib import admin
from .models import User
@admin.register(User)
class UserAdmin(admin.ModelAdmin):
    list_display=('id','name','email','password')
```

forms.py –

```
form django import forms
class StudentRegistration(forms.Form):
    name=forms.CharField()
    email=forms.EmailField()
    password=forms.CharField(widget=forms.
                             PasswordInput)
```

userregistration.html –

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



Date Page 216

(i) for insert new data-

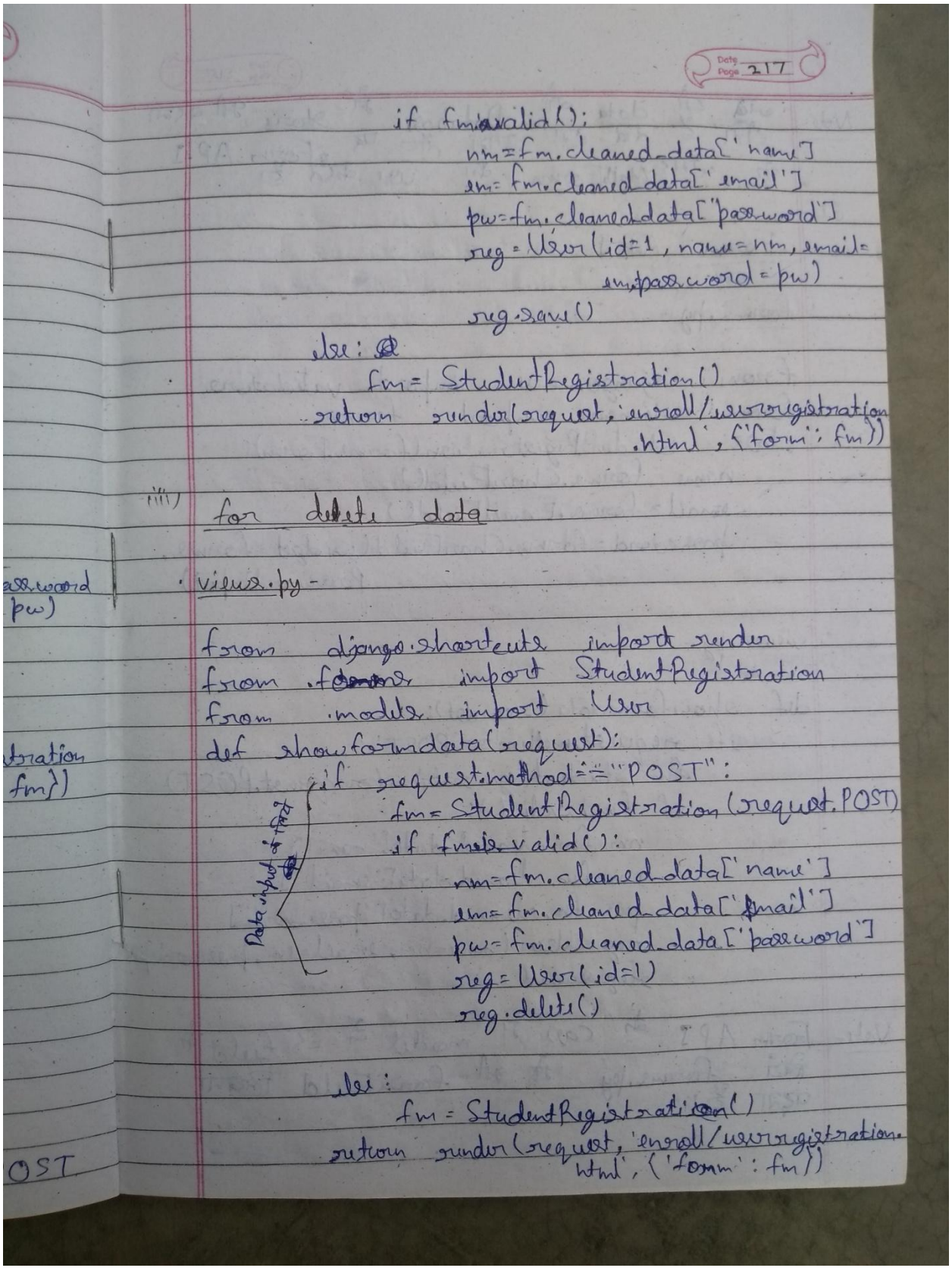
views.py -

```
from django.shortcuts import render
from .forms import StudentRegistration
from .models import User
def showformdata(request):
    if request.method == 'POST':
        fm = StudentRegistration(request.POST)
        if fm.is_valid():
            nm = fm.cleaned_data['name']
            em = fm.cleaned_data['email']
            pw = fm.cleaned_data['password']
            reg = User(name=name, email=em, password=pw)
            reg.save()
    else:
        fm = StudentRegistration()
    return render(request, 'enroll/userregistration.html', {'form': fm})
```

(ii) for update data-

views.py -

```
from django.shortcuts import render
from .forms import StudentRegistration
from .models import User
def showformdata(request):
    if request.method == 'POST':
        fm = StudentRegistration(request.POST
```



```
if fm.is_valid():
    nm=fm.cleaned_data['name']
    em=fm.cleaned_data['email']
    pw=fm.cleaned_data['password']
    reg=User(id=1, name=nm, email=
        em, password=pw)
    reg.save()
else:
    fm=StudentRegistration()
return render(request, 'enroll/userregistration
    .html', {'form': fm})
```

iii) for delete data-

- views.py -

```
from django.shortcuts import render
from .forms import StudentRegistration
from .models import User
def showformdata(request):
    if request.method=="POST":
        fm=StudentRegistration(request.POST)
        if fm.is_valid():
            nm=fm.cleaned_data['name']
            em=fm.cleaned_data['email']
            pw=fm.cleaned_data['password']
            reg=User(id=1)
            reg.delete()
    else:
        fm=StudentRegistration()
    return render(request, 'enroll/userregistration.
        html', {'form': fm})
```

Data input & save

Date Page 218

Note- जब हमें data को Database में store भी करना होता है तब हम आज तक पर Form API के जगह ModelForm का use करते है।

62 -

Form API -

- forms.py -

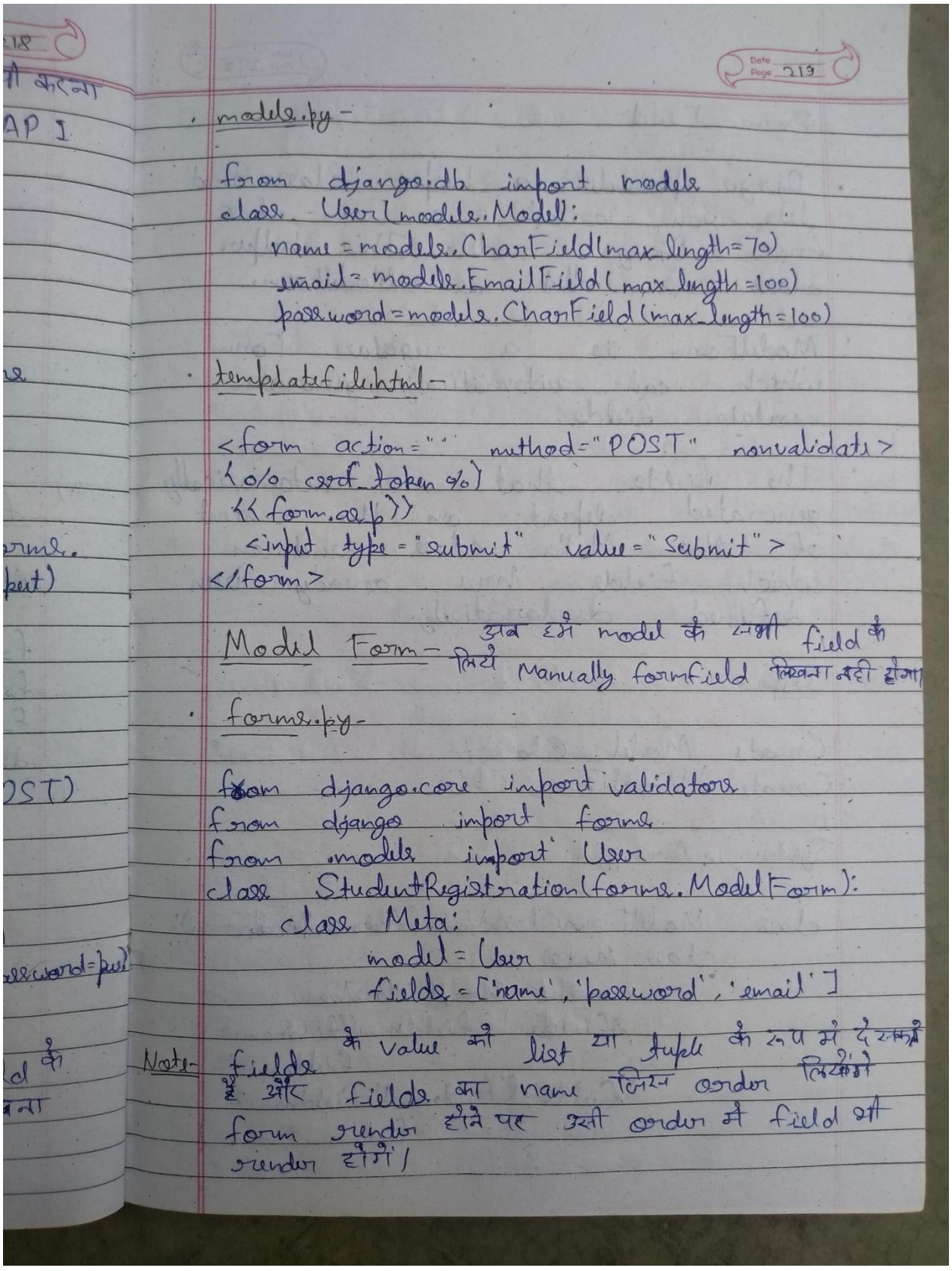
```
from django.core import validators
from django import forms
class StudentRegistration(forms.Form):
    name = forms.CharField()
    email = forms.EmailField()
    password = forms.CharField(widget = forms.
                               PasswordInput)
```

- views.py -

```
def showformdata(request):
    if request.method == 'POST':
        fm = StudentRegistration(request.POST)
        if fm.isvalid():
            nm = fm.cleaned_data['name']
            em = fm.cleaned_data['email']
            pw = fm.cleaned_data['password']
            reg = User(name=nm, email=em, password=pw)
            reg.save()
```

Note- Form API के case में model के कई field के लिये forms.py में भी formField लिखना पड़ता है।

Note-



- models.py -

```
from django.db import models
class User(models.Model):
    name = models.CharField(max_length=70)
    email = models.EmailField(max_length=100)
    password = models.CharField(max_length=100)
```

- templatefile.html -

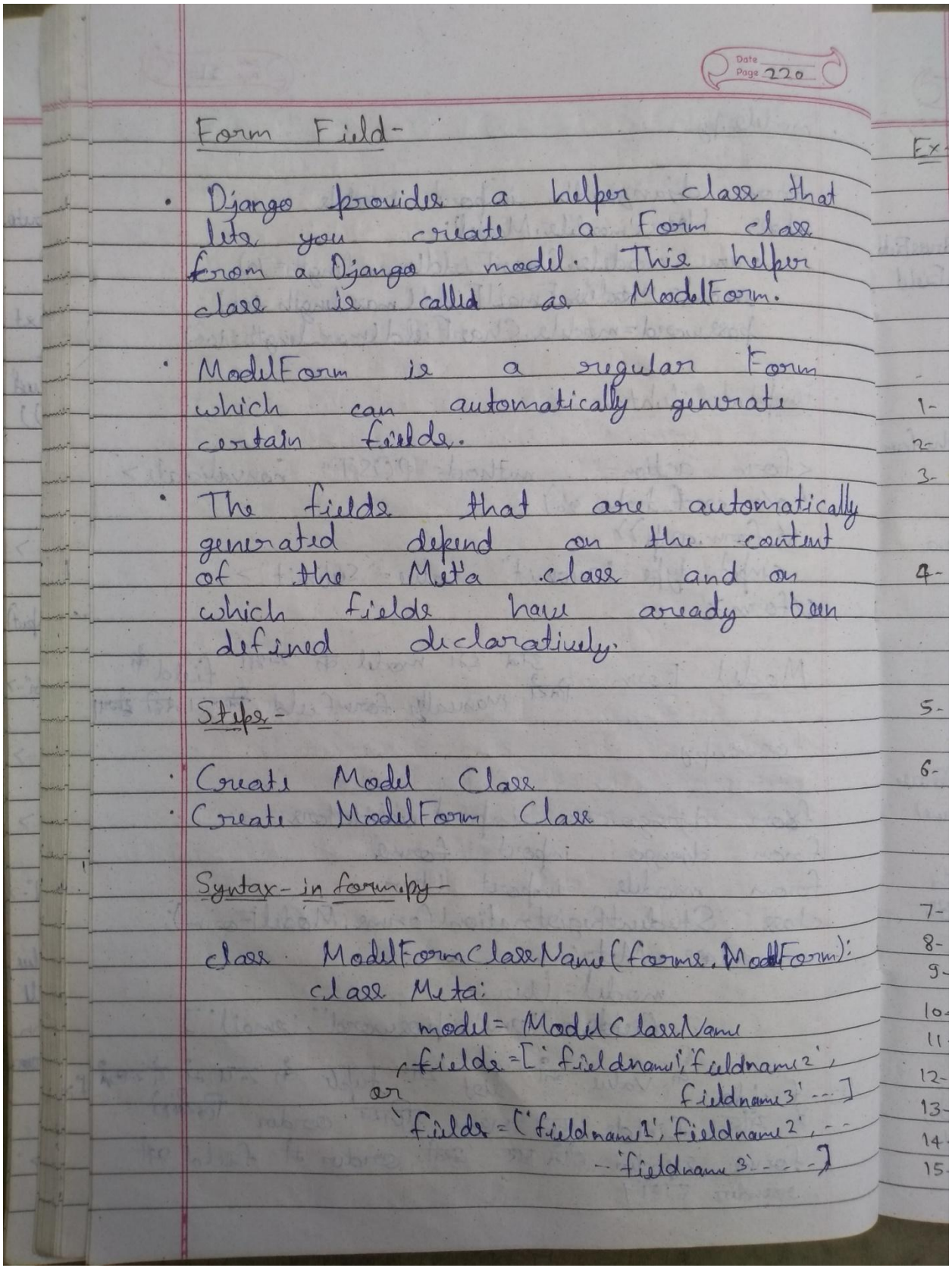
```
<form action="" method="POST" novalidate>
{% csrf_token %}
{{form.as_p}}
<input type="submit" value="Submit">
</form>
```

Model Form - इसमें model के सभी field के लिये Manually formfield लिखना नहीं होगा

- forms.py -

```
from django.core import validators
from django import forms
from module import User
class StudentRegistration(forms.ModelForm):
    class Meta:
        model = User
        fields = ['name', 'password', 'email']
```

Note- fields के value को list या tuple के रूप में दे सकते हैं और fields का name जिस order लिखेंगे form render होने पर उसी order में field भी render होंगे।



Date
Page 220

Form Field-

- Django provides a helper class that lets you create a Form class from a Django model. This helper class is called as ModelForm.
- ModelForm is a regular Form which can automatically generate certain fields.
- The fields that are automatically generated depend on the content of the Meta class and on which fields have already been defined declaratively.

Steps-

- Create Model Class
- Create ModelForm Class

Syntax- in form.py-

```
class ModelFormClassName(forms.ModelForm):
    class Meta:
        model = ModelClassName
        fields = ['fieldname1', 'fieldname2',
                  'fieldname3' ...]
    or
        fields = ('fieldname1', 'fieldname2', ...
                  'fieldname3' ...)
```

Date Page 222

16-	ImageField	ImageField
17-	IntegerField	IntegerField
18-	IPAddressField	IPAddressField
19-	GenericIPAddressField	GenericIPAddressField
20-	ManyToManyField	ModelMultipleChoiceField
21-	NullBooleanField	NullBooleanField
22-	PositiveIntegerField	IntegerField
23-	PositiveSmallIntegerField	IntegerField
24-	SlugField	SlugField
25-	SmallAutoField	Not represented in the form
26-	SmallIntegerField	IntegerField
27-	TextField	CharField with widget = forms.Textarea
28-	TimeField	TimeField
29-	URLField	URLField
30-	UUIDField	UUIDField

- If the model field has blank=True, the required is set to False on the form field. Otherwise, required=True.
  Ex- name = models.CharField(max_length=70, blank=True)

- The form field's label is set to the verbose_name of the model field, with the first character capitalized.

- The form field's help_text is set to the help_text of the model field.

- If the model field has choices set, then the form field's widget will be set to Select, with choices

Date Page 222

16-	ImageField	ImageField
17-	IntegerField	IntegerField
18-	IPAddressField	IPAddressField
19-	GenericIPAddressField	GenericIPAddressField
20-	ManyToManyField	ModelMultipleChoiceField
21-	NullBooleanField	NullBooleanField
22-	PositiveIntegerField	IntegerField
23-	PositiveSmallIntegerField	IntegerField
24-	SlugField	SlugField
25-	SmallAutoField	Not represented in the form
26-	SmallIntegerField	IntegerField
27-	TextField	CharField with widget = forms.Textarea
28-	TimeField	TimeField
29-	URLField	URLField
30-	UUIDField	UUIDField

- If the model field has blank=True, the required is set to False on the form field. Otherwise, required=True.
  Ex- name = models.CharField(max_length=70, blank=True)

- The form field's label is set to the verbose_name of the model field, with the first character capitalized.

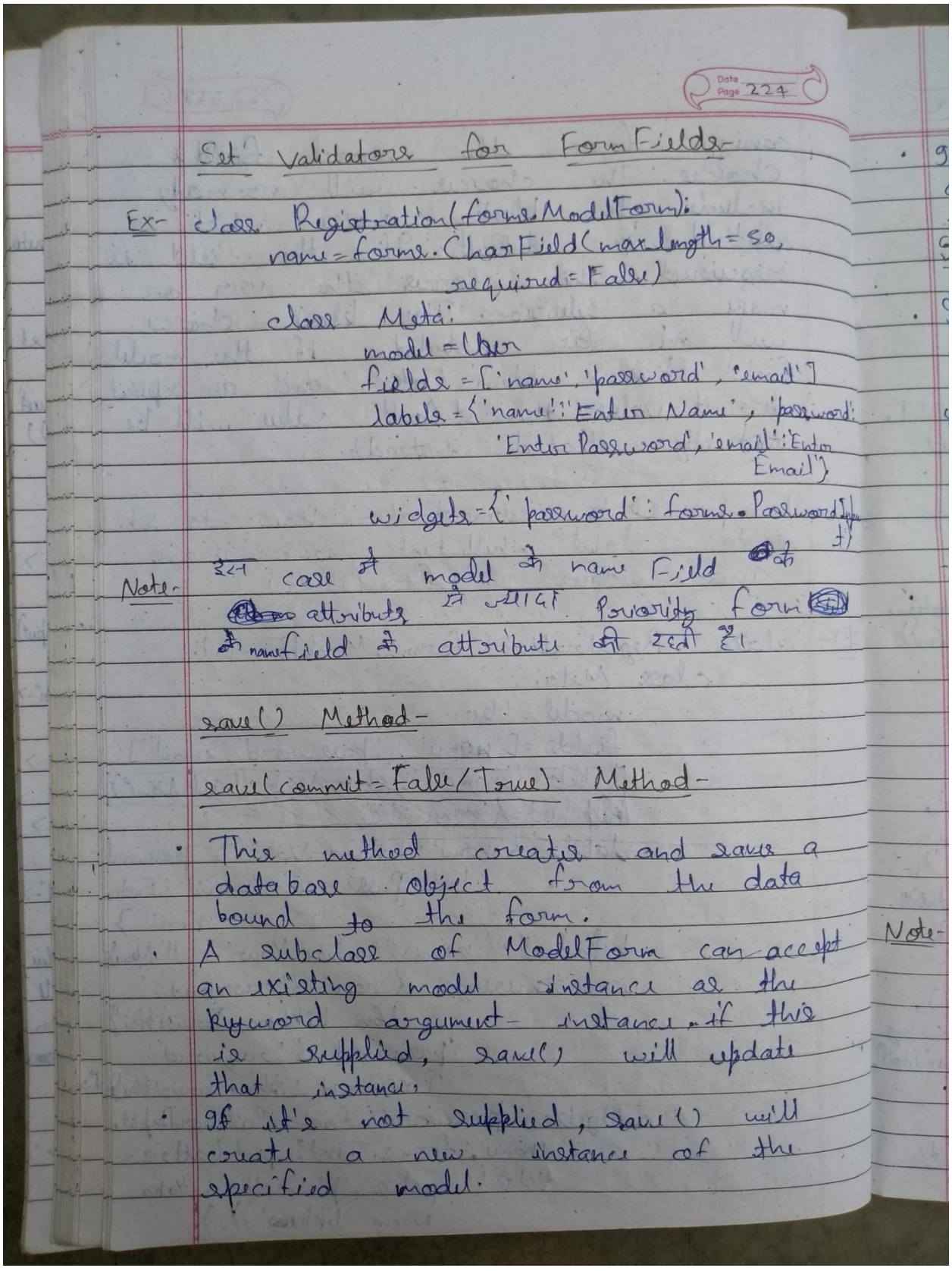
- The form field's help_text is set to the help_text of the model field.

- If the model field has choices set, then the form field's widget will be set to Select, with choices

coming from the model field's choices. The choices will normally include the blank choice which is selected by default. If the field is required, this forces the user or make a selection. The blank choice will not be included if the model field has blank=False and an explicit default value (the default value will be initially selected instead).

Example to demostrate how to set custom label, help_text errormessages and widgets in formfield-

Ex-
```
class Registration(forms.ModelForm):
    class Meta:
        model = User
        fields = ['name', 'password', 'email']
        labels = {'name': 'Enter Name', 'password'
                  : 'EnterPassword', 'email': 'Enter
                                              Email'}
        help_text = {'name': 'Enter Your Full Name'}
        errormessages = {'name': {'required':
                        'Name Likhna Jaruri Hai'}
                        'password': {'required':
                        'Password Likhna Jaruri Hai'}
        widgets = {'password': forms.PasswordInput,
                   'name': forms.TextInput(attrs=
                   {'class': 'myclass', 'placeholder': 'Yaha
                                        Naam likhna'})}
```



Date Page 224

Set Validators for Form Fields-

Ex- class Registration(forms.ModelForm):
    name = forms.CharField(max_length=50, required=False)
    class Meta:
        model = User
        fields = ['name', 'password', 'email']
        labels = {'name': 'Enter Name', 'password': 'Enter Password', 'email': 'Enter Email'}
        widgets = {'password': forms.PasswordInput}

Note- इस case में model के name Field के attributes से अलग Priority form field के attributes की रहती है।

save() Method-

save(commit=False/True) Method-

- This method creates and saves a database object from the data bound to the form.
- A subclass of ModelForm can accept an existing model instance as the keyword argument instance. if this is supplied, save() will update that instance.
- If it's not supplied, save() will create a new instance of the specified model.

Note-

- If the form hasn't been validated, calling save() will do so by checking form.errors.

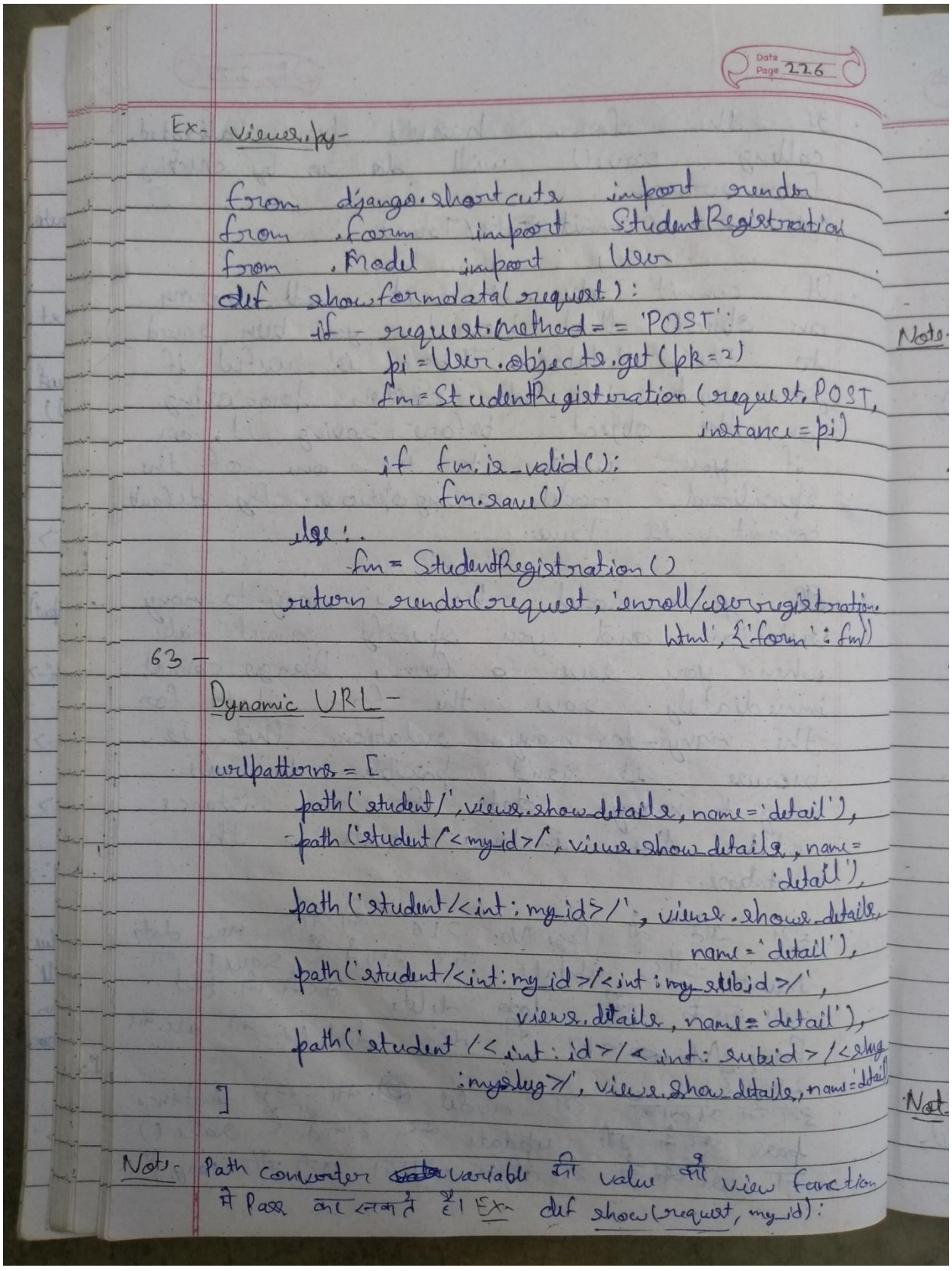
Syntax- save(commit=False/True)

- If commit=False, then it will return an object that hasn't yet been saved to the database. This is useful if you want to do custom processing on the object before saving it, or if you want to use one of the specialized model saving options. By default commit is True.

- If your model has a many-to-many relation and you specify commit=False when you save a form, Django cannot immediately save the form data for the many-to-many relation. This is because it isn't possible to save many-to-many data for an instance until the instance exists in the database.

Note- हम चाहे तो Page No. 216 की तरह new data insert और update कर सकते है save() का use करके और data delete करने के लिये Page No. 217 की तरह delete() का use कर सकते है।

- इस के अलावा हम model का एक instance pass करके भी update कर सकते है save() का use करके ये तरीका ज्यादा सही और easy है।



Date Page 226

Ex- views.py-

```
from django.shortcuts import render
from .form import StudentRegistration
from .model import User
def showformdata(request):
    if request.method == 'POST':
        pi = User.objects.get(pk=2)
        fm = StudentRegistration(request.POST,
                                 instance=pi)
        if fm.is_valid():
            fm.save()
    else:
        fm = StudentRegistration()
    return render(request, 'enroll/userregistration.
                  html', {'form': fm})
```

Note

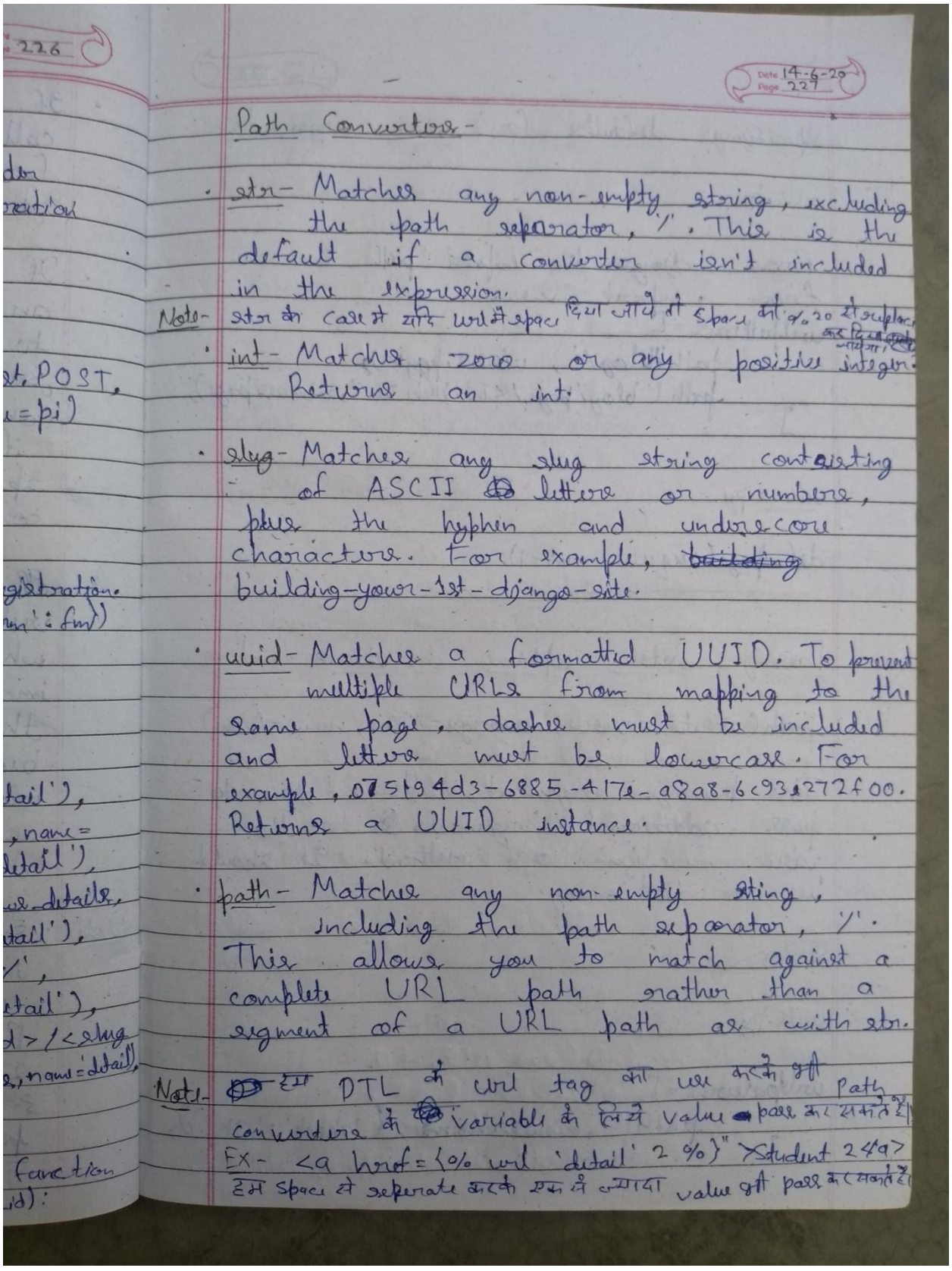
63 -

Dynamic URL -

```
urlpatterns = [
    path('student/', views.show_details, name='detail'),
    path('student/<my_id>/', views.show_details, name=
                                'detail'),
    path('student/<int:my_id>/', views.show_details,
                                name='detail'),
    path('student/<int:my_id>/<int:my_subid>/',
                views.details, name='detail'),
    path('student/<int:id>/<int:subid>/<slug
                :myslug>/', views.show_details, name='detail'),
]
```

Note

Note- Path converter variable की value को view function में Pass करने के लिए Ex- def show(request, my_id):



Path Converters -

- str - Matches any non-empty string, excluding the path separator, '/'. This is the default if a converter isn't included in the expression.

Note- str को case में यदि url में space दिया जाये तो space को %20 से replace कर दिया जायेगा।

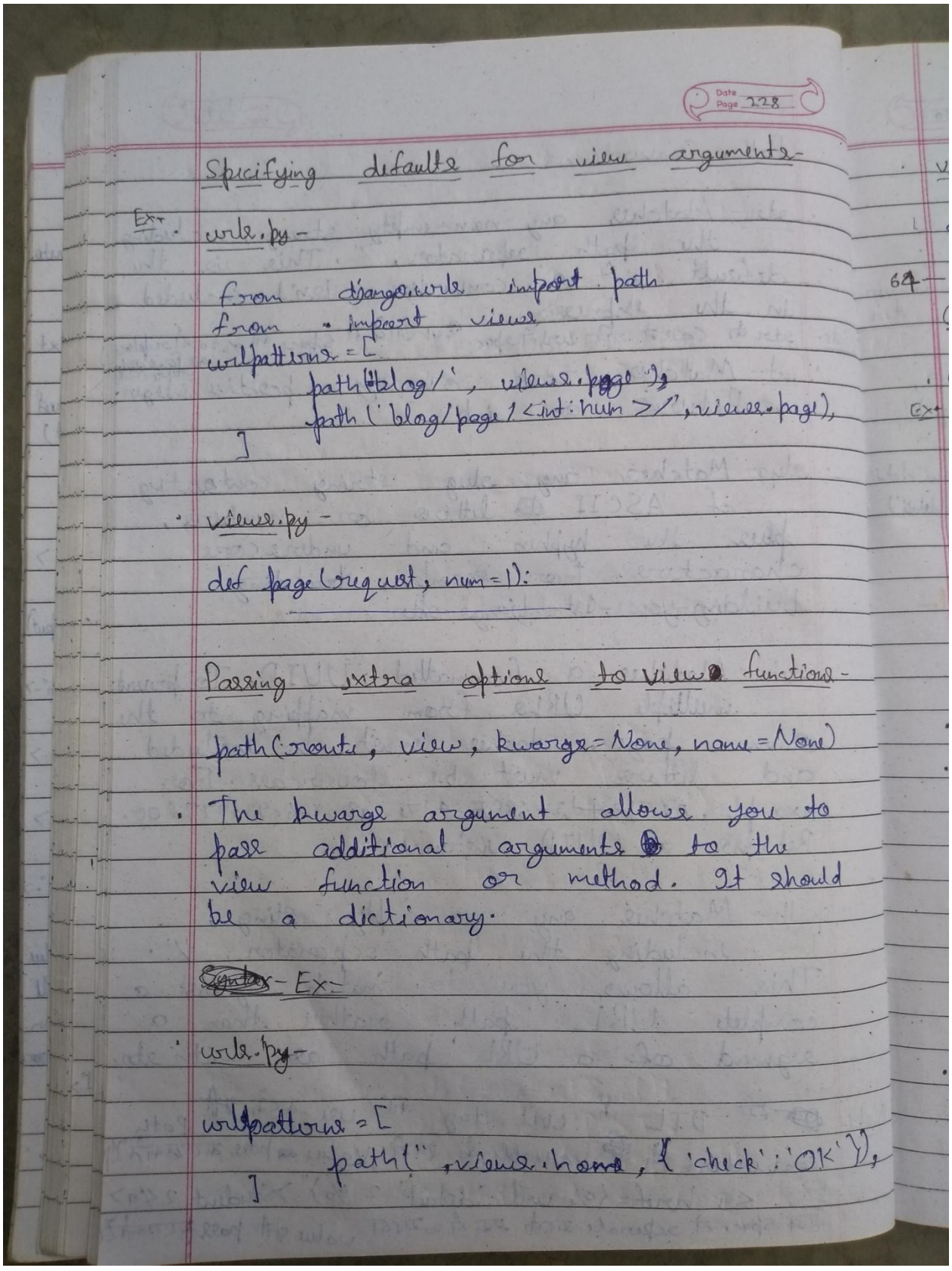
- int - Matches zero or any positive integer. Returns an int.

- slug - Matches any slug string consisting of ASCII letters or numbers, plus the hyphen and underscore characters. For example, ~~building~~ building-your-1st-django-site.

- uuid - Matches a formatted UUID. To prevent multiple URLs from mapping to the same page, dashes must be included and letters must be lowercase. For example, 075194d3-6885-417e-a8a8-6c931272f00. Returns a UUID instance.

- path - Matches any non-empty string, including the path separator, '/'. This allows you to match against a complete URL path rather than a segment of a URL path as with str.

Note:- हम DTL में url tag का use करके भी Path converters में variable के लिये value pass कर सकते है।
Ex- Student 2
हम space से seperate करके एक से ज्यादा value भी pass कर सकते है।



Specifying defaults for view arguments-

Ex- urls.py -

```
from django.urls import path
from . import views
urlpatterns = [
    path('blog/', views.page),
    path('blog/page/<int:num>/', views.page),
]
```

- views.py -

```
def page(request, num=1):
```

Passing extra options to view functions -

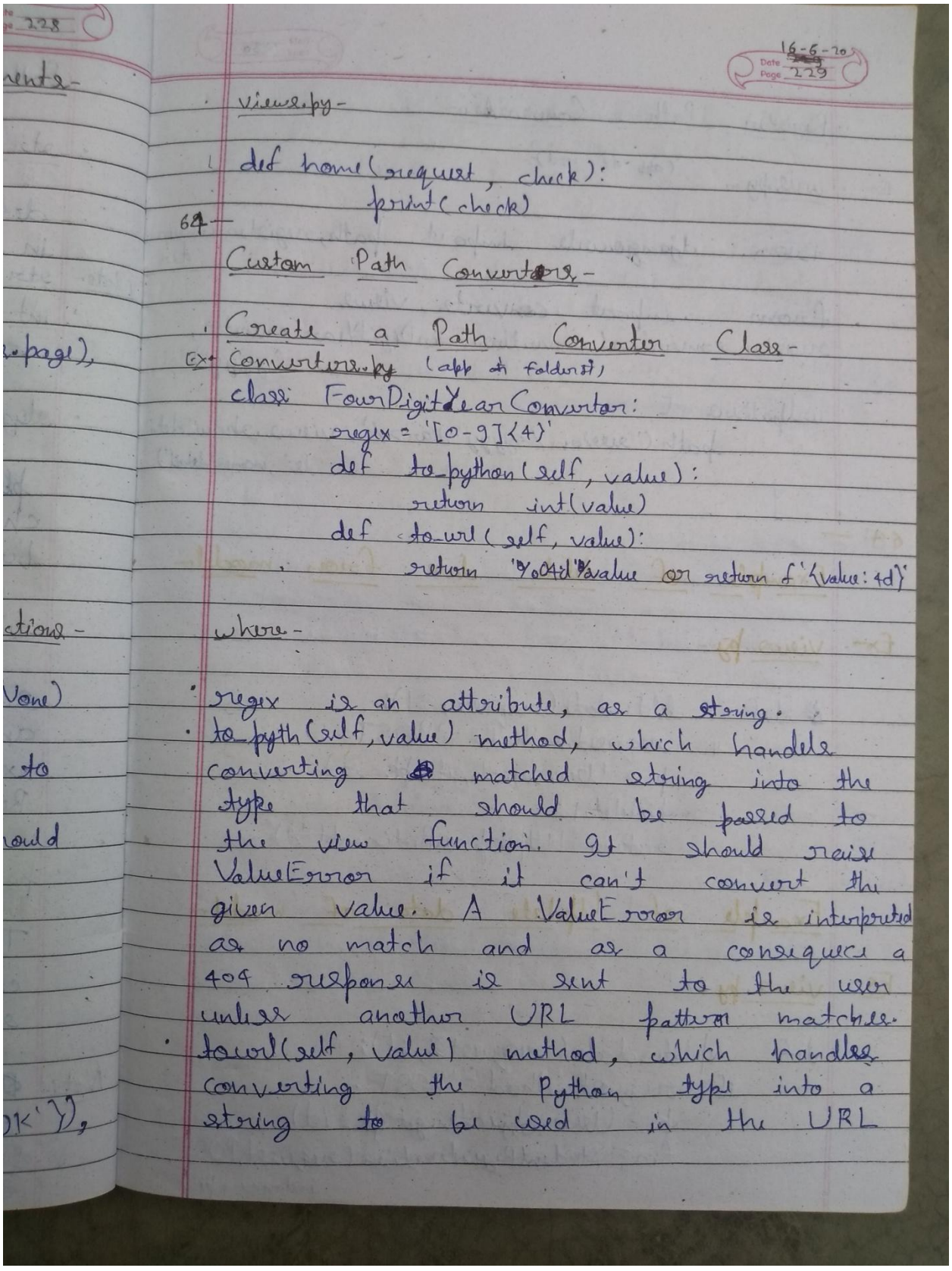
path(route, view, kwargs=None, name=None)

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

Syntax - Ex=

- urls.py -

```
urlpatterns = [
    path('', views.home, {'check': 'OK'}),
]
```



views.py -

```
def home(request, check):
    print(check)
```

64 -

Custom Path Converters -

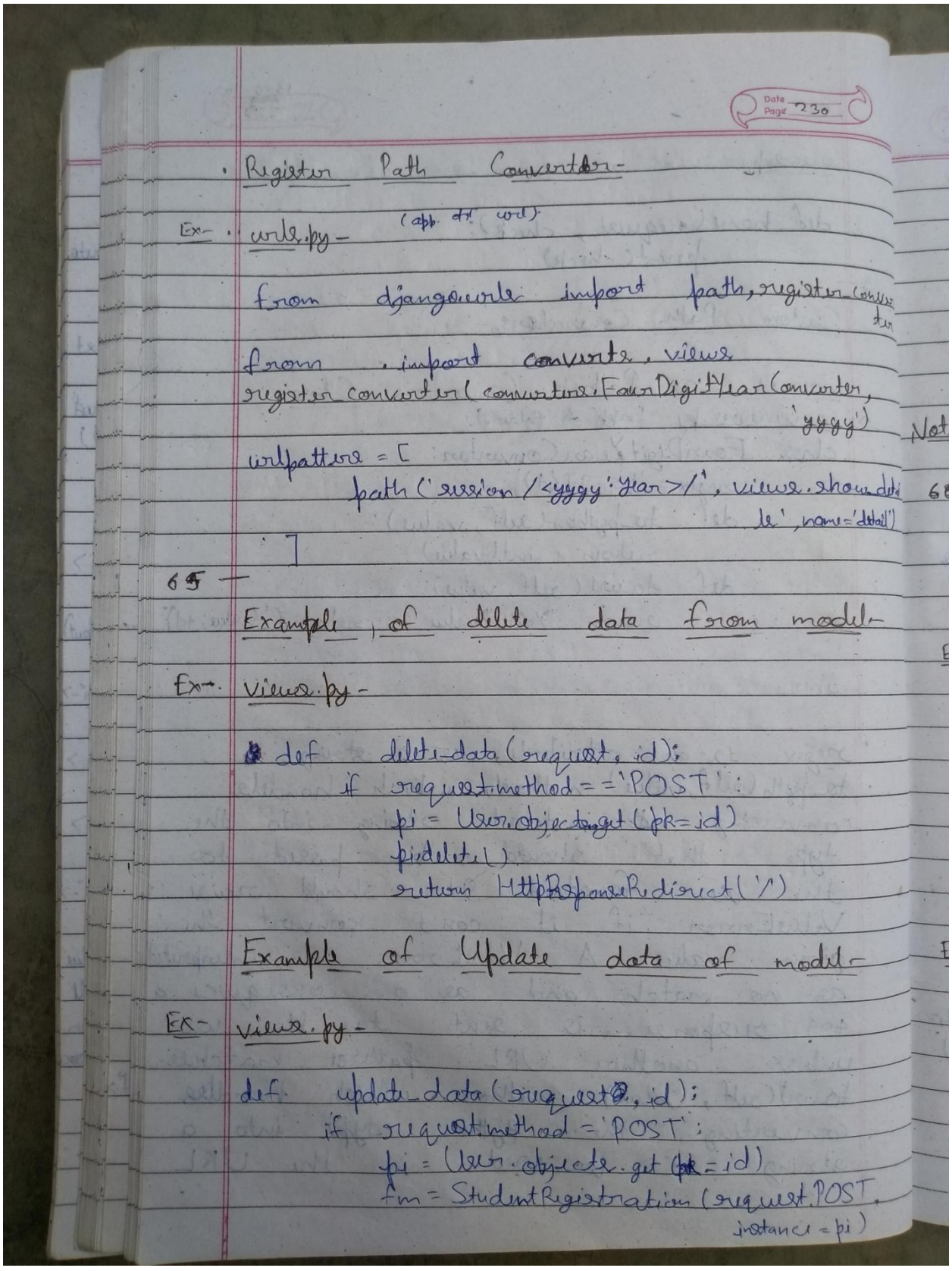
Create a Path Converter Class -

Ex- converters.py (app ki folder me)

```
class FourDigitYearConvertor:
    regex = '[0-9]{4}'
    def to_python(self, value):
        return int(value)
    def to_url(self, value):
        return '%04d'%value or return f'{value:4d}'
```

where -

- regex is an attribute, as a string.
- to_pyth(self, value) method, which handles converting matched string into the type that should be passed to the view function. It should raise ValueError if it can't convert the given value. A ValueError is interpreted as no match and as a consequence a 404 response is sent to the user unless another URL pattern matches.
- to_url(self, value) method, which handles converting the Python type into a string to be used in the URL



Date Page 230

• Register Path Converter -

Ex- urls.py - (app. dt url).

from django.urls import path, register_converter

from . import converter, views

register_converter(converters.FourDigitYearConverter, 'yyyy')

urlpatterns = [

path('session/<yyyy:year>/', views.showdetail, name='detail')

]

65 —

Example of delete data from model -

Ex- views.py -

def delete_data(request, id):

if request.method == 'POST':

pi = User.objects.get(pk=id)

pi.delete()

return HttpResponseRedirect('/')

Example of Update data of model -

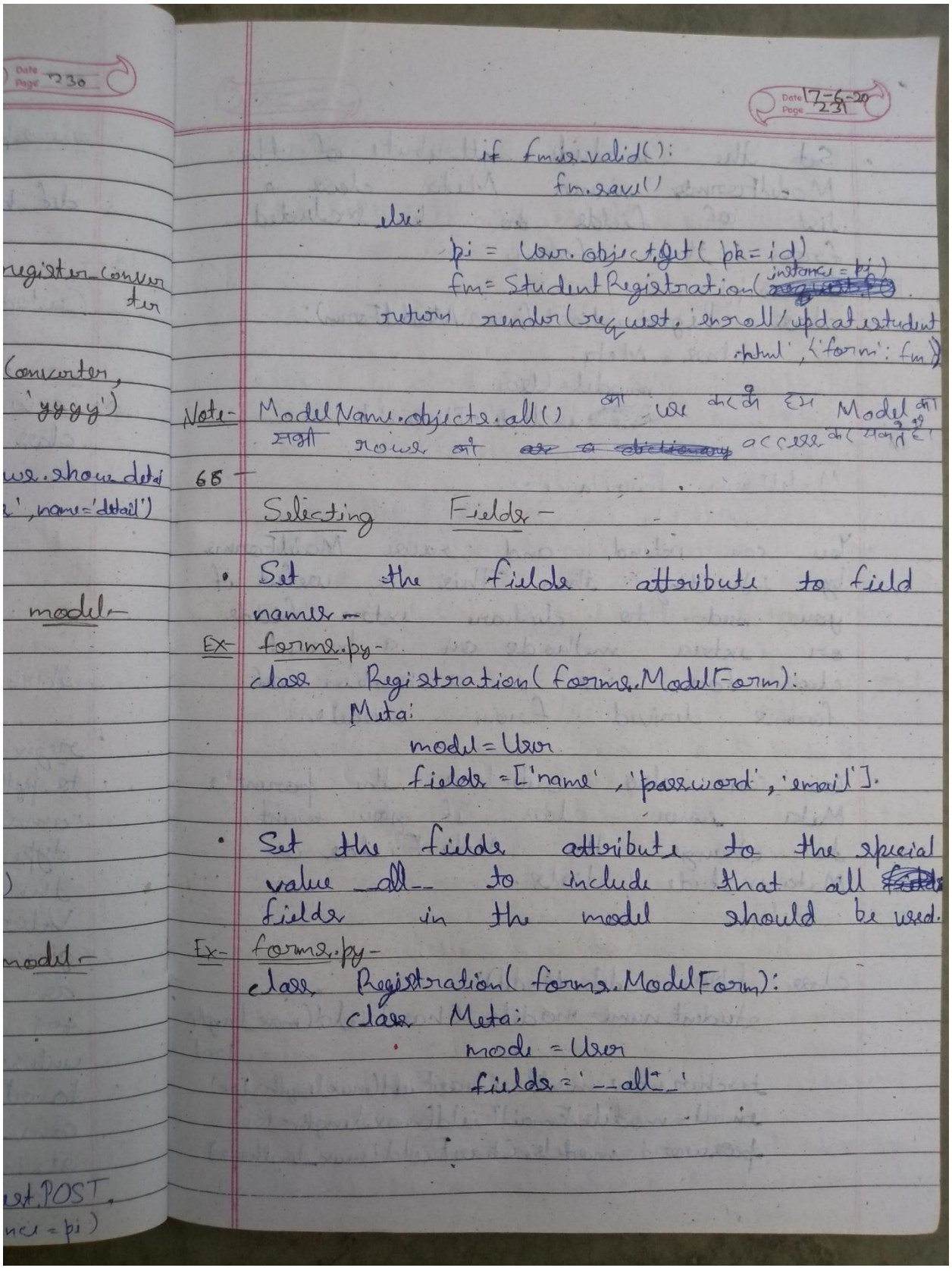
Ex- views.py -

def update_data(request, id):

if request.method = 'POST':

pi = User.objects.get(pk=id)

fm = StudentRegistration(request.POST, instance=pi)



```
                if fm.is_valid():
                    fm.save()
            else:
                pi = User.objects.get(pk=id)
                fm = StudentRegistration(instance=pi)
            return render(request, 'enroll/updatestudent.html', {'form': fm})
```

Note- ModelName.objects.all() का use करके हम Model का सारा record का access कर सकते है।

65 -

Selecting Fields -

- Set the fields attribute to field names -

Ex- forms.py -

```
class Registration(forms.ModelForm):
    Meta:
        model = User
        fields = ['name', 'password', 'email'].
```

- Set the fields attribute to the special value __all__ to include that all fields in the model should be used.

Ex- forms.py -

```
class Registration(forms.ModelForm):
    class Meta:
        model = User
        fields = '__all__'
```

Date Page 232

- Set the exclude attribute of the ModelForm's inner Meta class a list of fields to be excluded from the form.

Ex- forms.py -

```
class Registration(forms.ModelForm):
    class Meta:
        model = User
        exclude = ['name']
```

67 —

ModelForm Inheritance -

Note

- You can extend and reuse ModelForms by inheriting them. This is useful if you need to declare extra fields or extra methods on a parent class for use in a number of forms derived from models.

- You can also subclass the parent's Meta inner class if you want to change the Meta.fields or Meta.exclude lists.

Ex- models.py -

```
class User(models.Model):
    student_name = models.CharField(max_length=100)
    teacher_name = models.CharField(max_length=100)
    email = models.EmailField(max_length=100)
    password = models.CharField(max_length=100)
```

68

Date Page 232

- Set the exclude attribute of the ModelForm's inner Meta class a list of fields to be excluded from the form.

Ex- forms.py -

```
class Registration(forms.ModelForm):
    class Meta:
        model = User
        exclude = ['name']
```

67 -

ModelForm Inheritance -

Note

- You can extend and reuse ModelForms by inheriting them. This is useful if you need to declare extra fields or extra methods on a parent class for use in a number of forms derived from models.

- You can also subclass the parent's Meta inner class if you want to change the Meta.Fields or Meta.exclude lists.

Ex- models.py -

```
class User(models.Model):
    student_name = models.CharField(max_length=100)
    teacher_name = models.CharField(max_length=100)
    email = models.EmailField(max_length=100)
    password = models.CharField(max_length=100)
```

68

forms.py -

```
class StudentRegistration(forms.ModelForm):
    class Meta:
        model = User
        fields = ['student_name', 'email', 'password']

class TeacherRegistration(StudentRegistration):
    class Meta(StudentRegistration.Meta):
        fields = ['teacher_name', 'email', 'password']
```

Note -
- Normal Python name resolution rules apply. If you have multiple base classes that declare a Meta inner class, only the first one will be used. This means the child's Meta, if it exists, otherwise the Meta of the first parent, etc.
- It's possible to inherit from both Form and ModelForm simultaneously, however, you must ensure that ModelForm appears first in the MRO. This is because these classes rely on different metaclasses and a class can only have one metaclass.
- It's possible to declaratively remove a Field inherited from a parent class by setting the name to be None on the subcla

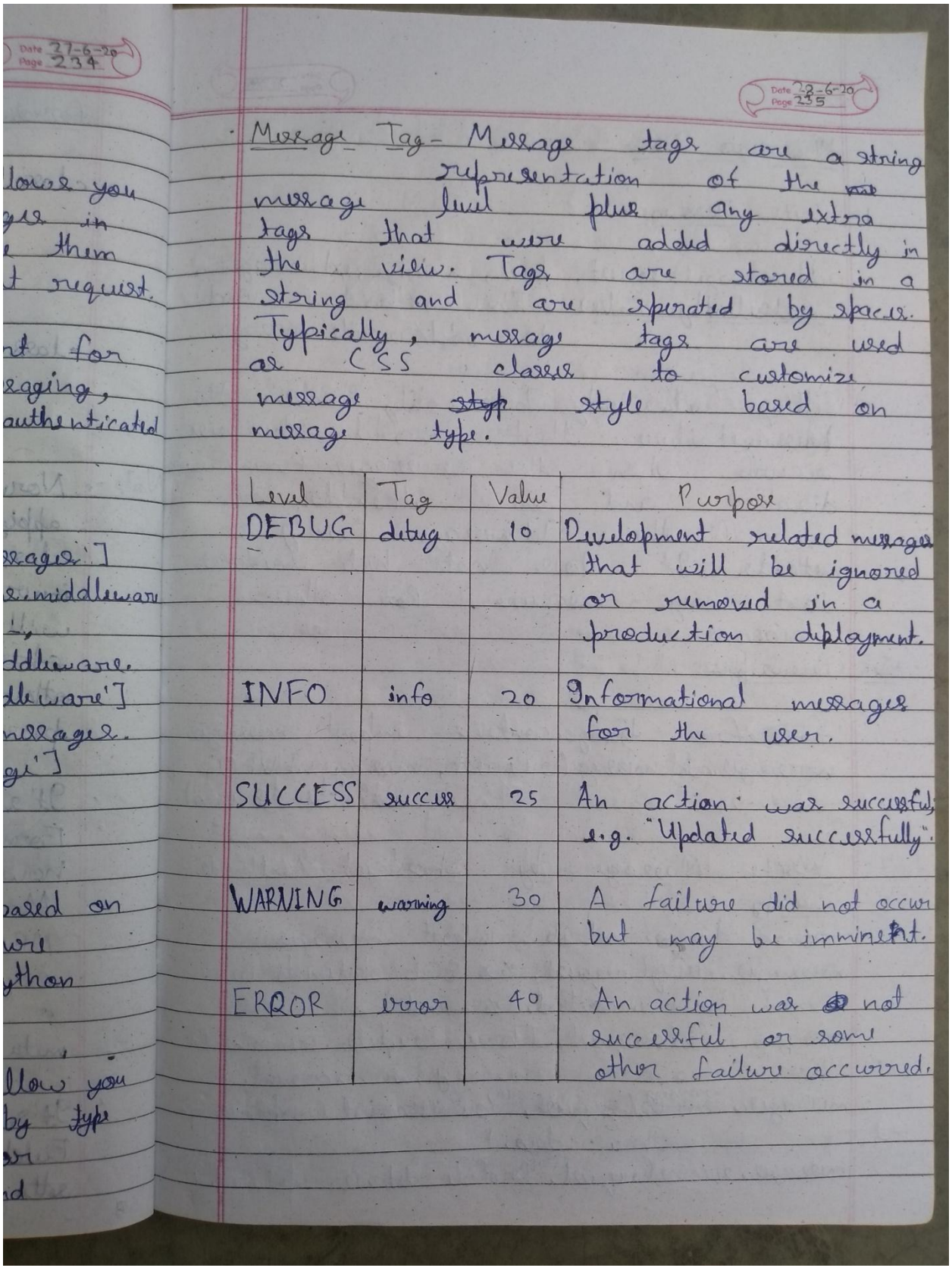
Date 27-6-20
Page 234

Messages Framework-

- The messages framework allows you to temporily store messages in one request and retrieve them for display in a subsequent request.
- Django provides full support for cookie- and session-based messaging, for both anonymous and authenticated users.
- In settings.py-
- INSTALLED_APPS=['django.contrib.messages']
  MIDDLEWARE=['django.contrib.sessions.middleware.SessionMiddleware',
  'django.contrib.messages.middleware.MessageMiddleware']
  'context_processors':['django.contrib.messages.context_processors.messages']

Message Levels and Tags-

- The messages framework is based on a configurable level architecture similar to that of the Python logging module.
- Message Level- Message levels allow you to group messages by type so they can be filtered or displayed differently in views and templates.

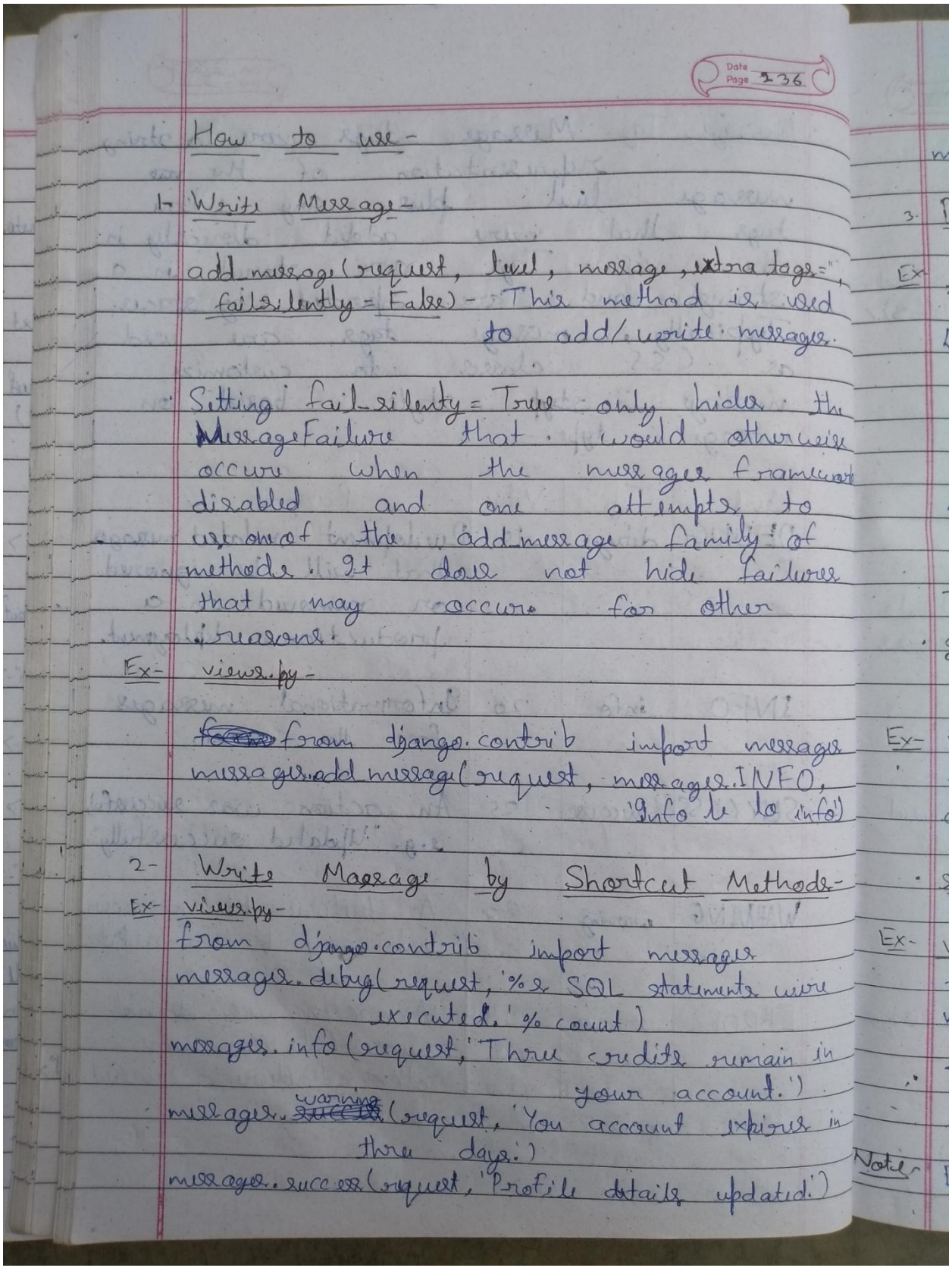


Date 27-6-20 Page 234

Date 28-6-20 Page 235

Message Tag - Message tags are a string representation of the message level plus any extra tags that were added directly in the view. Tags are stored in a string and are separated by spaces. Typically, message tags are used as CSS classes to customize message style based on message type.

Level	Tag	Value	Purpose
DEBUG	debug	10	Development related messages that will be ignored or removed in a production deployment.
INFO	info	20	Informational messages for the user.
SUCCESS	success	25	An action was successful, e.g. "Updated successfully".
WARNING	warning	30	A failure did not occur but may be imminent.
ERROR	error	40	An action was not successful or some other failure occurred.



Date Page 236

How to use -

1- Write Message -

add_message(request, level, message, extra_tags='', fail_silently=False) - This method is used to add/write messages.

Setting fail_silenty = True only hides the MessageFailure that would otherwise occure when the messages framework disabled and one attempts to use one of the add_message family of methods. It does not hide failures that may occure for other reasons.

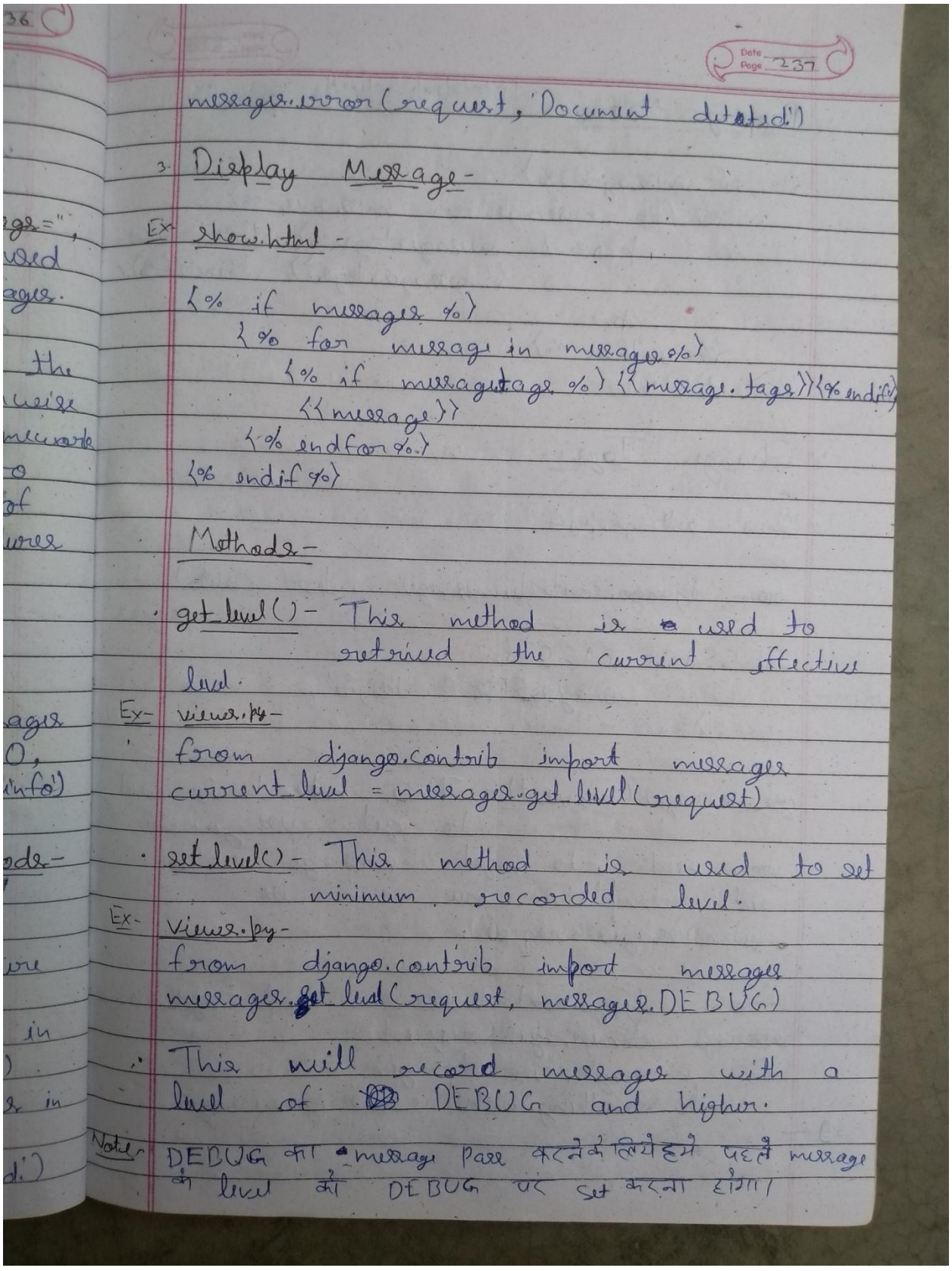
Ex- views.py -

from django.contrib import messages
messages.add_message(request, messages.INFO, 'Info to do info')

2- Write Message by Shortcut Methods -

Ex- views.py -

from django.contrib import messages
messages.debug(request, '%s SQL statements were executed.' % count)
messages.info(request, 'Three credits remain in your account.')
messages.warning(request, 'You account expires in three days.')
messages.success(request, 'Profile details updated.')



messages.error(request, 'Document deleted')

3. Display Message-

Ex- show.html -

```
{% if messages %}
    {% for message in messages %}
        {% if message.tags %} {{ message.tags }} {% endif %}
        {{ message }}
    {% endfor %}
{% endif %}
```

Methods-

- get_level() - This method is used to retrieved the current effective level.

Ex- views.py -

```
from django.contrib import messages
current_level = messages.get_level(request)
```

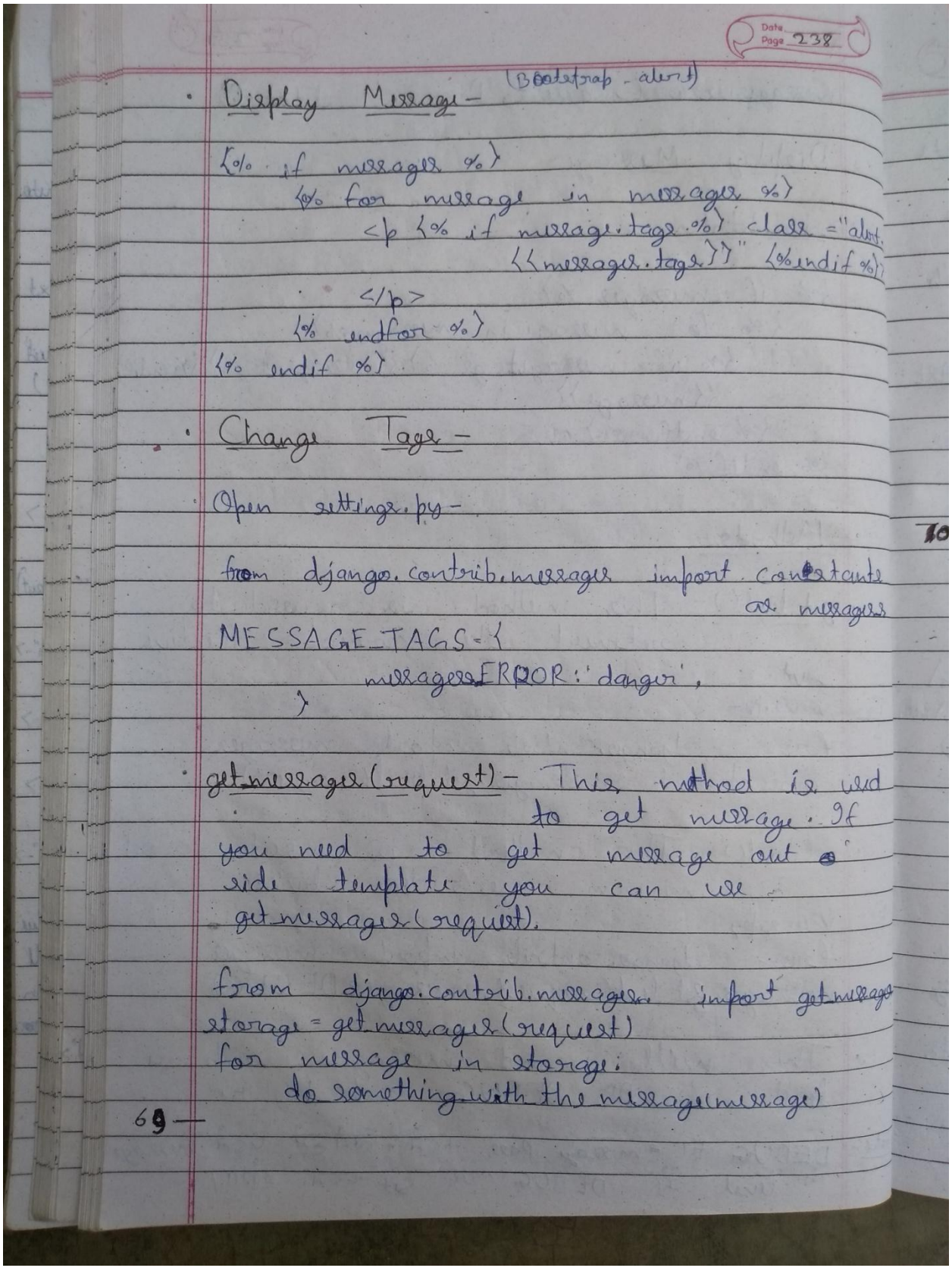
- set_level() - This method is used to set minimum recorded level.

Ex- views.py -

```
from django.contrib import messages
messages.set_level(request, messages.DEBUG)
```

∴ This will record messages with a level of DEBUG and higher.

Note- DEBUG का message pass करने के लिये हमें पहले message के level को DEBUG पर set करना होगा।



Date Page 238

- Display Message - (Bootstrap - alert)

```
{% if messages %}
    {% for message in messages %}
        <p {% if message.tags %} class = "alert
            {{message.tags}}" {% endif %}>
        </p>
    {% endfor %}
{% endif %}
```

- Change Tags -

Open settings.py -

```
from django.contrib.messages import constants as messages
MESSAGE_TAGS = {
        messages.ERROR : 'danger',
    }
```

- get_messages(request) - This method is used to get message. If you need to get message out side template you can use get_messages(request).

```
from django.contrib.messages import get_messages
storage = get_messages(request)
for message in storage:
    do_something_with_the_message(message)
```

69 —

Date 29-6-20
Page 239

Authentication and Authorization-

- **Authentication-** Authentication is about validating your credentials like Username and password to verify your identity.

- **Authorization-** Authorization is the process to determine whether the authenticated user has access to the particular resources. It checks your rights to grant you access to resources such as information, databases, files, etc.

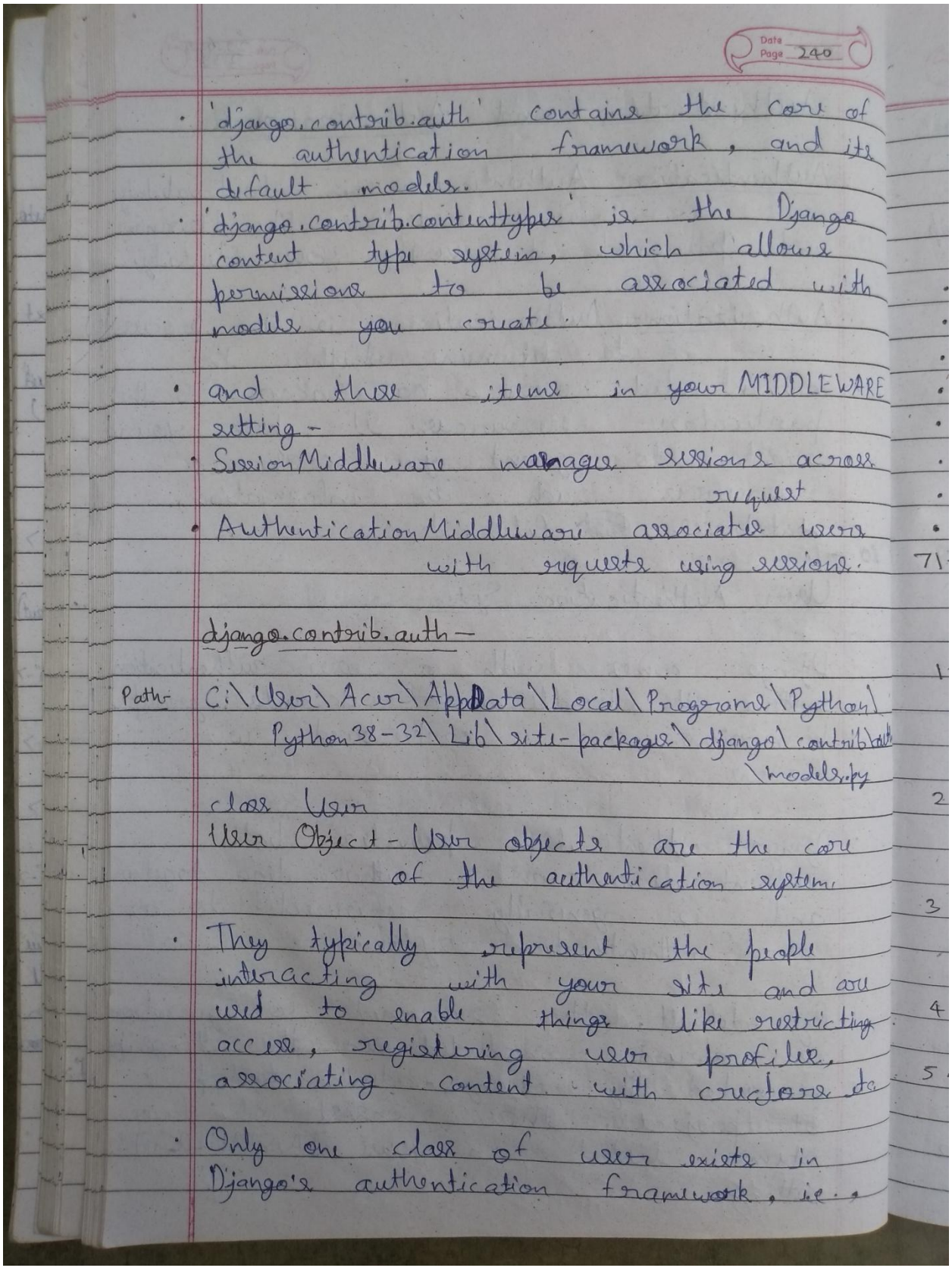
10-

User Authentication System-

- Django comes with a user authentication system. It handles user accounts, groups, permissions and cookie-based user sessions.

- Django authentication provides both authentication and authorization together and is generally referred to as the authentication system.

- By default, the required configuration is already included in the settings.py generated by django-admin startproject, these consist of two items listed in your INSTALLED_APPS setting-



Date
Page 240

- 'django.contrib.auth' contains the core of the authentication framework, and its default models.
- 'django.contrib.contenttypes' is the Django content type system, which allows permissions to be associated with models you create.

- and these items in your MIDDLEWARE setting -
- SessionMiddleware manages sessions across request
- AuthenticationMiddleware associates users with requests using sessions.

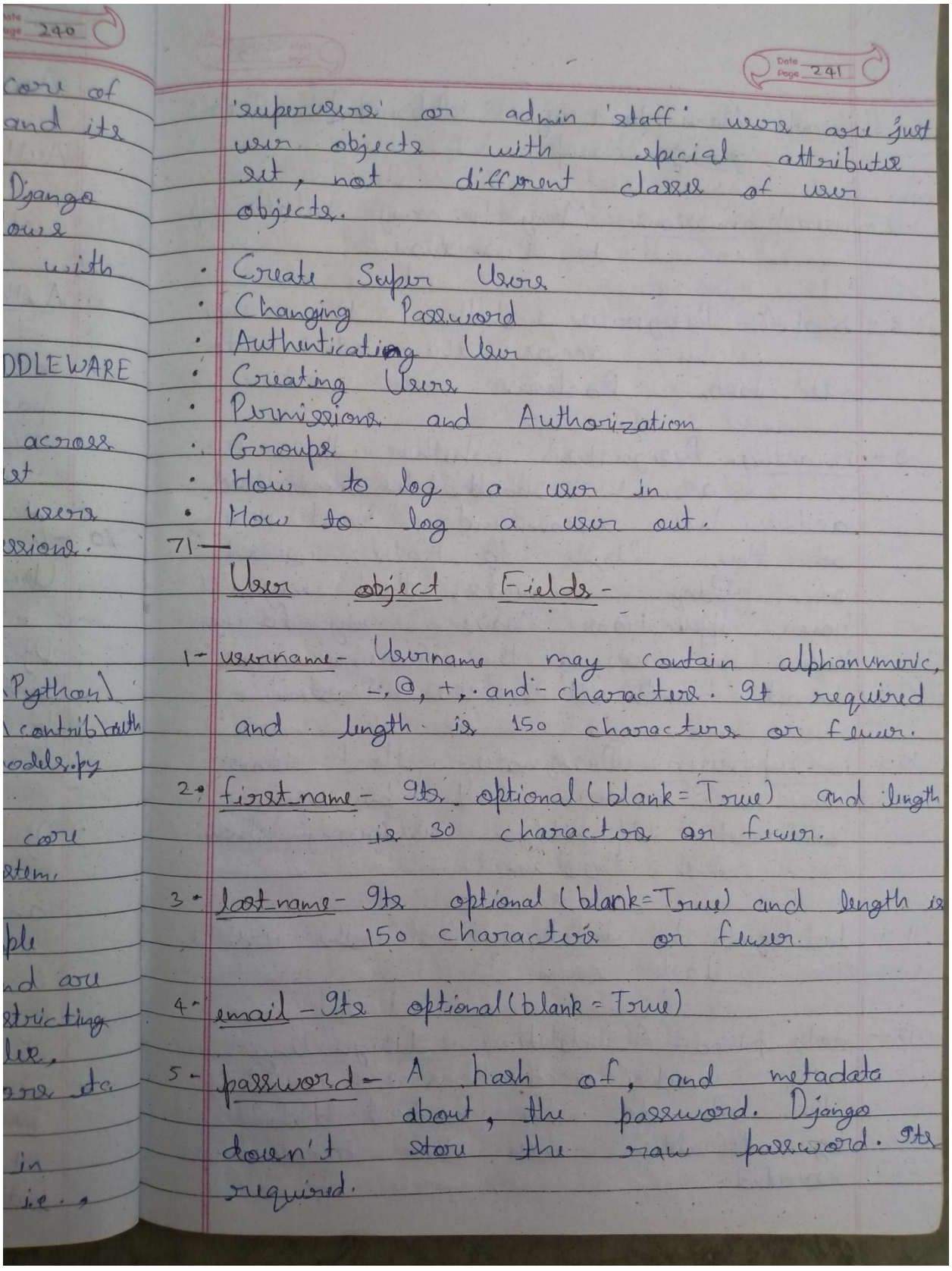
django.contrib.auth -

Path- C:\Users\Acer\AppData\Local\Programs\Python\Python38-32\Lib\site-packages\django\contrib\auth\models.py

class User

User Object - User objects are the core of the authentication system.

- They typically represent the people interacting with your site and are used to enable things like restricting access, registering user profiles, associating content with creators etc.
- Only one class of user exists in Django's authentication framework, i.e.,



'superusers' or admin 'staff' users are just user objects with special attribute set, not different classes of user objects.

- Create Super Users
- Changing Password
- Authenticating User
- Creating Users
- Permissions and Authorization
- Groups
- How to log a user in
- How to log a user out.

71—

User object Fields -

1- username - Username may contain alphanumeric, _, @, +, . and - character. It required and length is 150 characters or fewer.

2. first_name - Its optional (blank=True) and length is 30 characters or fewer.

3. last_name - Its optional (blank=True) and length is 150 characters or fewer.

4. email - Its optional (blank=True)

5 - password - A hash of, and metadata about, the password. Django doesn't store the raw password. Its required.

Date 30-6-20
Page 242

6- groups - Many-to-many relationship to Group.

7- user_permissions - Many-to-many relationship to Permission.

8- is_staff - Designates whether this user can access the admin site. It takes Boolean.

9- is_active - Designates whether this user account should be considered active. We recommend that you set this flag to False instead of deleting accounts; that way, if your applications have any foreign keys to users, the foreign keys won't break. It takes Boolean.

10- is_superuser - Designates that this has all permissions without explicitly assigning them. It takes Boolean.

11- last_login - A datetime of the user's last login.

12- date_joined - A datetime designating when the account was created. It set to the current date/time by default when the account is created.

13- is_authenticated - This is a way to tell if the user has been ~~auth~~ authenticated. This does not imply any permissions and doesn't check if the user is active or has a valid session. It's a ready-only attribute which is always True.

14- is_anonymous - This is a way of differentiating User and AnonymousUser objects. It's a read-only attribute which is always False.

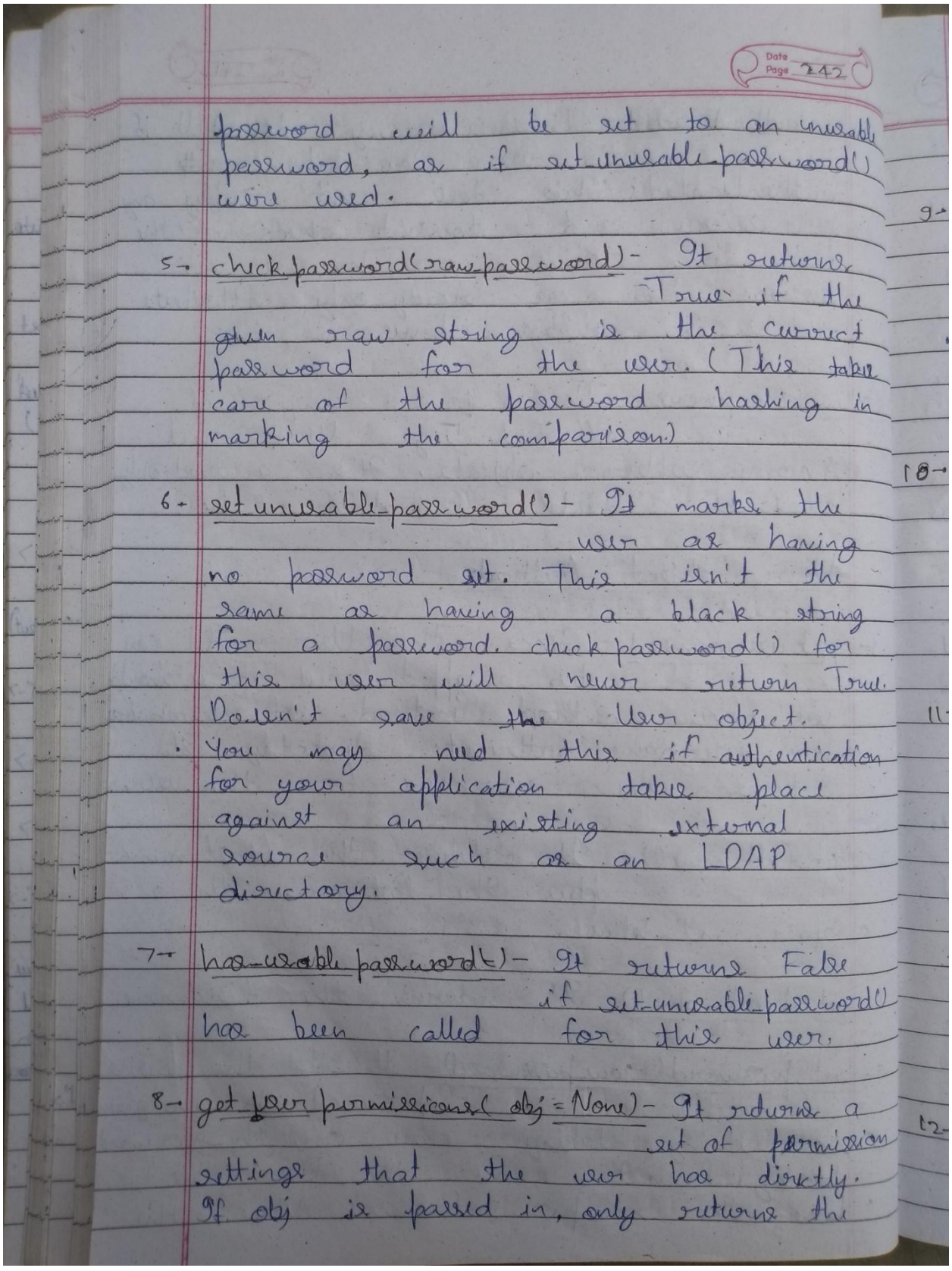
User object Methods -

1- get_username() - Since the User model can be swapped out, you should use this method instead of referencing the username attribute directly. It returns the username for the user.

2- get_full_name() - It returns the first_name plus the last_name, with a space in between.

3- get_short_name() - It returns the first_name.

4- set_password(raw_password) - It sets the user's password to the given raw string, taking care of the password hashing. Doesn't save the User object. When the raw_password is None, the



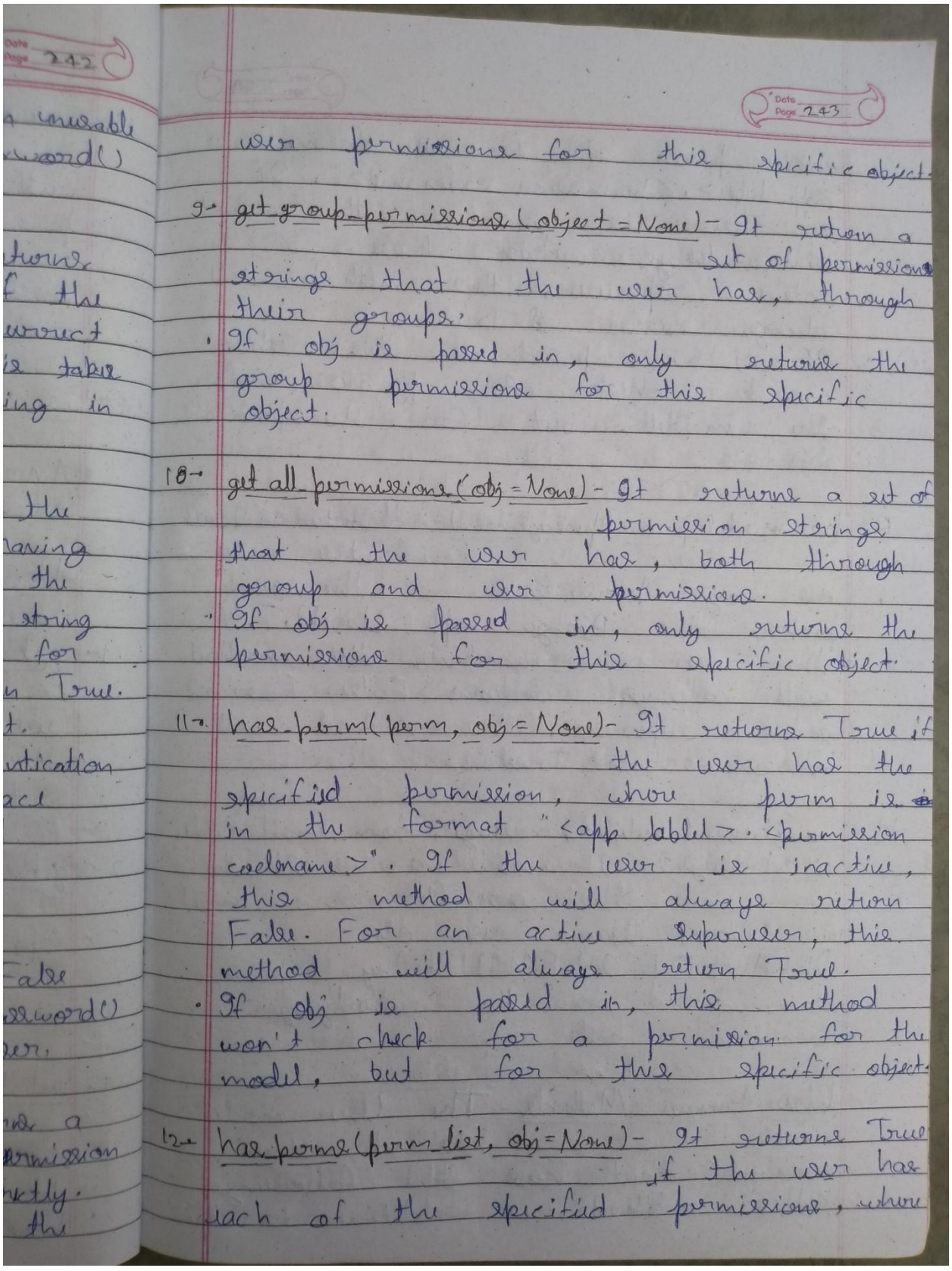
password will be set to an unusable password, as if set_unusable_password() were used.

5- check_password(raw_password) - It returns True if the given raw string is the correct password for the user. (This takes care of the password hashing in making the comparison.)

6- set_unusable_password() - It marks the user as having no password set. This isn't the same as having a blank string for a password. check_password() for this user will never return True. Doesn't save the User object.
You may need this if authentication for your application takes place against an existing external source such as an LDAP directory.

7- has_usable_password() - It returns False if set_unusable_password() has been called for this user.

8- get_user_permissions(obj=None) - It returns a set of permission strings that the user has directly. If obj is passed in, only returns the



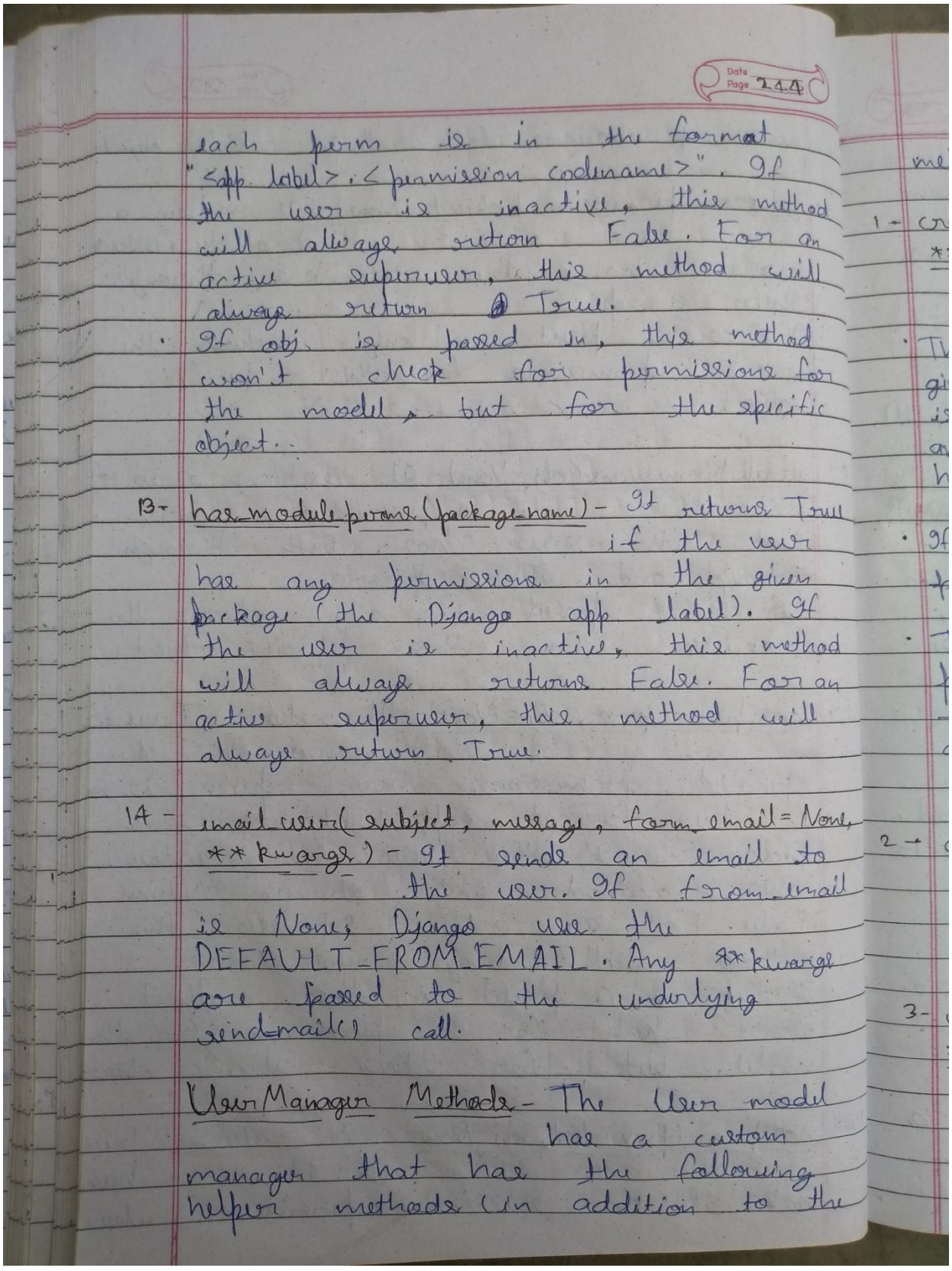
user permissions for this specific object.

9- get_group_permissions(object=None) - It return a set of permissions strings that the user has, through their groups.
- If obj is passed in, only returns the group permissions for this specific object.

10- get_all_permissions(obj=None) - It returns a set of permission strings that the user has, both through group and user permissions.
- If obj is passed in, only returns the permissions for this specific object.

11- has_perm(perm, obj=None) - It returns True if the user has the specified permission, where perm is in the format "<app label>.<permission codename>". If the user is inactive, this method will always return False. For an active superuser, this method will always return True.
- If obj is passed in, this method won't check for a permission for the model, but for this specific object.

12- has_perms(perm_list, obj=None) - It returns True if the user has each of the specified permissions, where



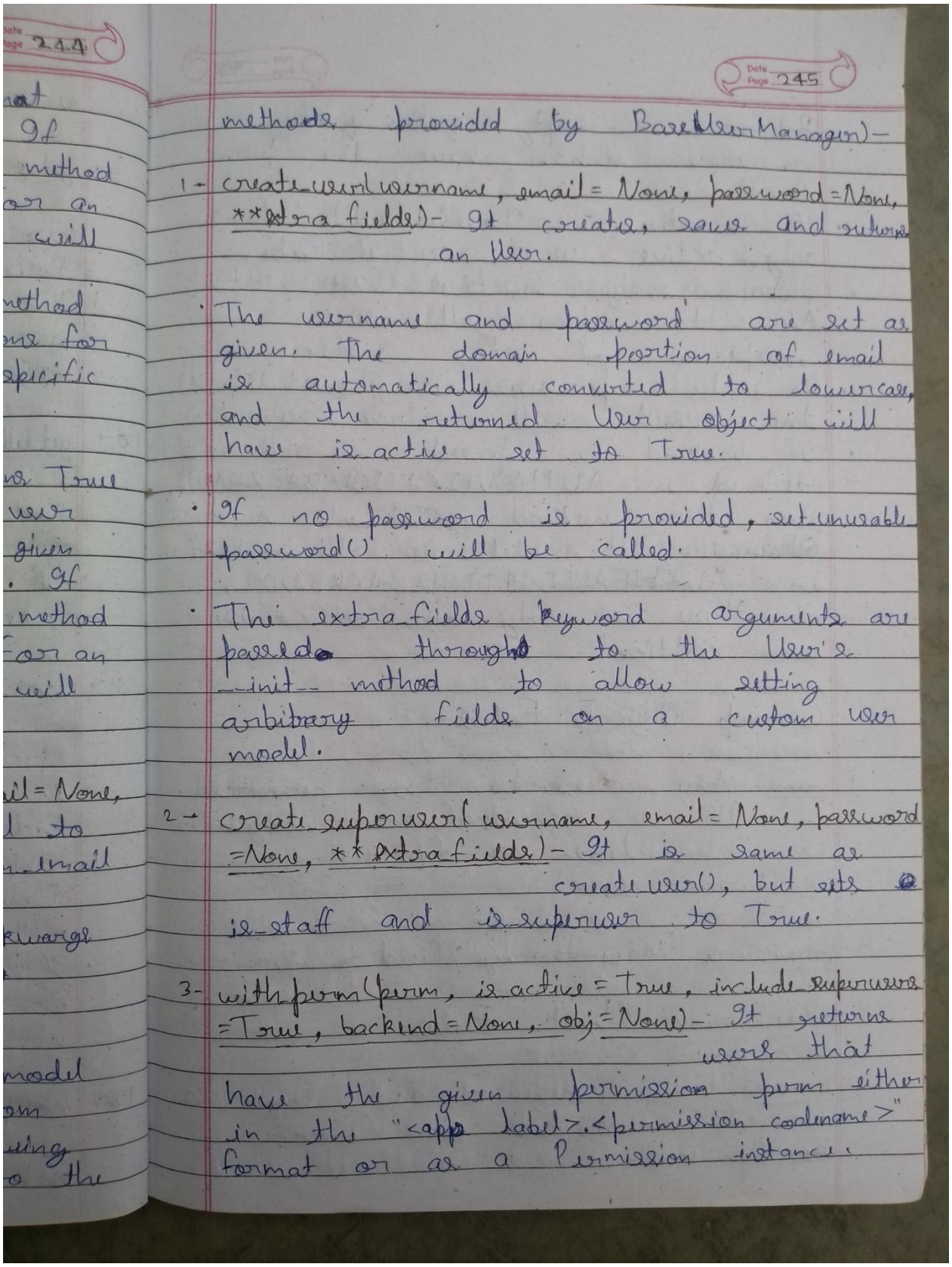
each perm is in the format "<app label>.<permission codename>". If the user is inactive, this method will always return False. For an active superuser, this method will always return True.

If obj is passed in, this method won't check for permissions for the model, but for the specific object.

13- has_module_perms(package_name) - It returns True if the user has any permissions in the given package (the Django app label). If the user is inactive, this method will always returns False. For an active superuser, this method will always return True.

14 - email_user(subject, message, from_email=None, **kwargs) - It sends an email to the user. If from_email is None, Django uses the DEFAULT_FROM_EMAIL. Any **kwargs are passed to the underlying send_mail() call.

User Manager Methods - The User model has a custom manager that has the following helper methods (in addition to the



methods provided by BaseUserManager)-

1- create_user(username, email=None, password=None, **extra_fields)- It creates, saves and returns an User.

- The username and password are set as given. The domain portion of email is automatically converted to lowercase, and the returned User object will have is_active set to True.
- If no password is provided, set_unusable_password() will be called.
- The extra_fields keyword arguments are passed through to the User's __init__ method to allow setting arbitrary fields on a custom user model.

2- create_superuser(username, email=None, password=None, **extra_fields)- It is same as create_user(), but sets is_staff and is_superuser to True.

3- with_perm(perm, is_active=True, include_superusers=True, backend=None, obj=None)- It returns users that have the given permission perm either in the "<app label>.<permission codename>" format or as a Permission instance.

Date Page 246

Returns an empty ~~query~~ queryset if no users who have the perm found.

- If is_active is True (default), returns only active users, or if False, returns only inactive users. Use None to return all users irrespective of active state.
- If include_superusers is True (default), the result will include superusers.
- If backend is passed in and it's defined in AUTHENTICATION_BACKENDS, then this method will use it. Otherwise, it will use the backend in AUTHENTICATION_BACKENDS, if there is only one, or raise an exception.

Group Object Fields-

1- name- Any characters are permitted. It is required and length is 150 characters or fewer.
Ex- 'Awsome Users'.

2- permissions- Many-to-many field to Permission

Permission Object Fields-

1- name- It is required and length is 255 characters or fewer.
Ex- 'Can vote'.

2- content_type - A reference to the ~~django~~ django_content_type database table, which contains a record for each installed model. It is required.

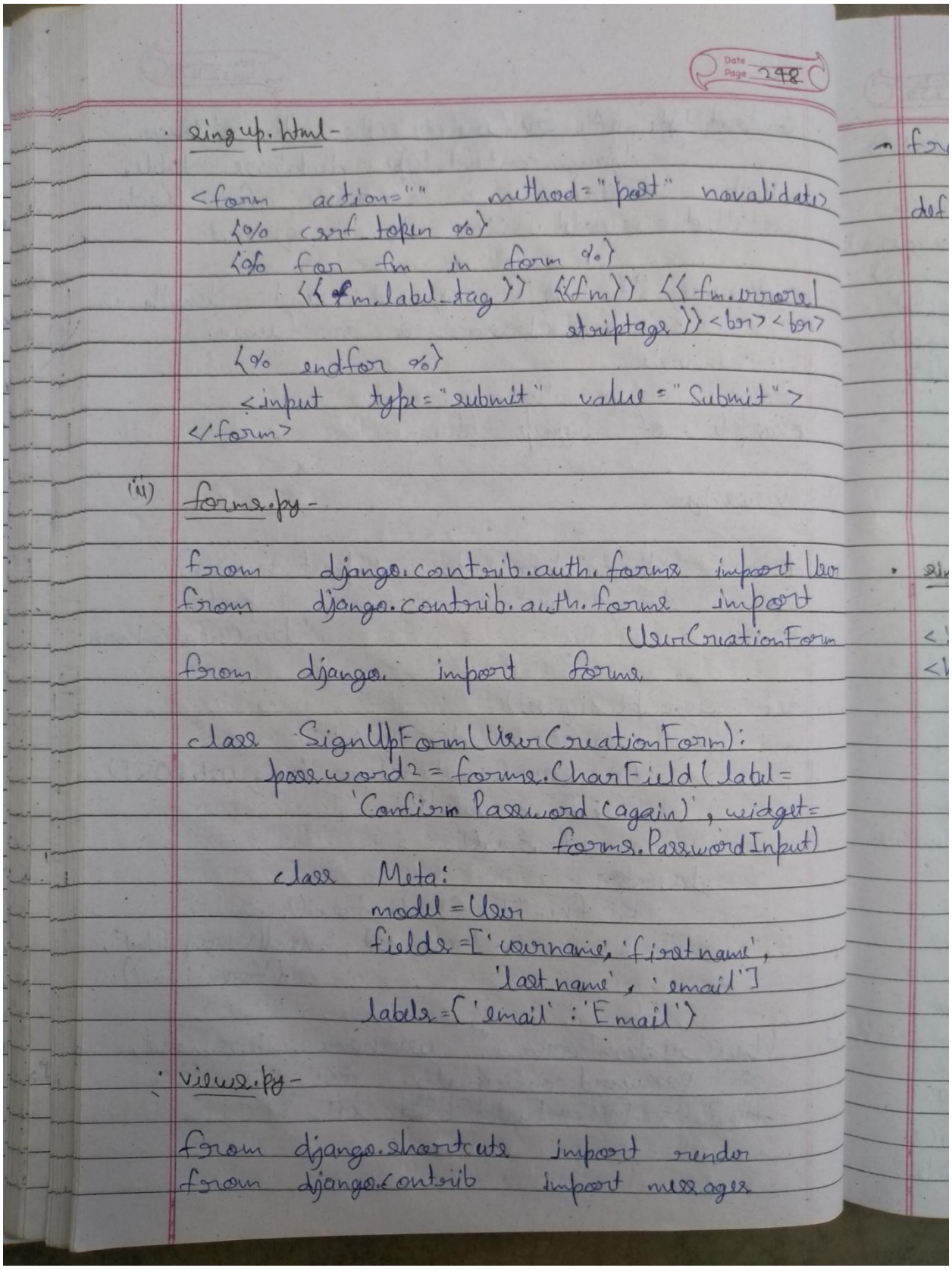
3- codename - It is required and length is 100 characters or fewer.
Ex- 'can_vote'.

Example of Signup form-

i) views.py

```
from django.shortcuts import render
from django.contrib.auth.forms import
                                UserCreationForm

def sign_up(request):
    if request.method == 'POST':
        fm = UserCreationForm(request.POST)
        if fm.is_valid():
            fm.save()
    else:
        fm = UserCreationForm()
    return render(request, 'enroll/signup.html',
                  {'form': fm})
```

Note- UserCreationForm में username, password, और password confirmation जैसा इसी तीन चीजों के लिए FormField होता है।



Date
Page 248

· signup.html -

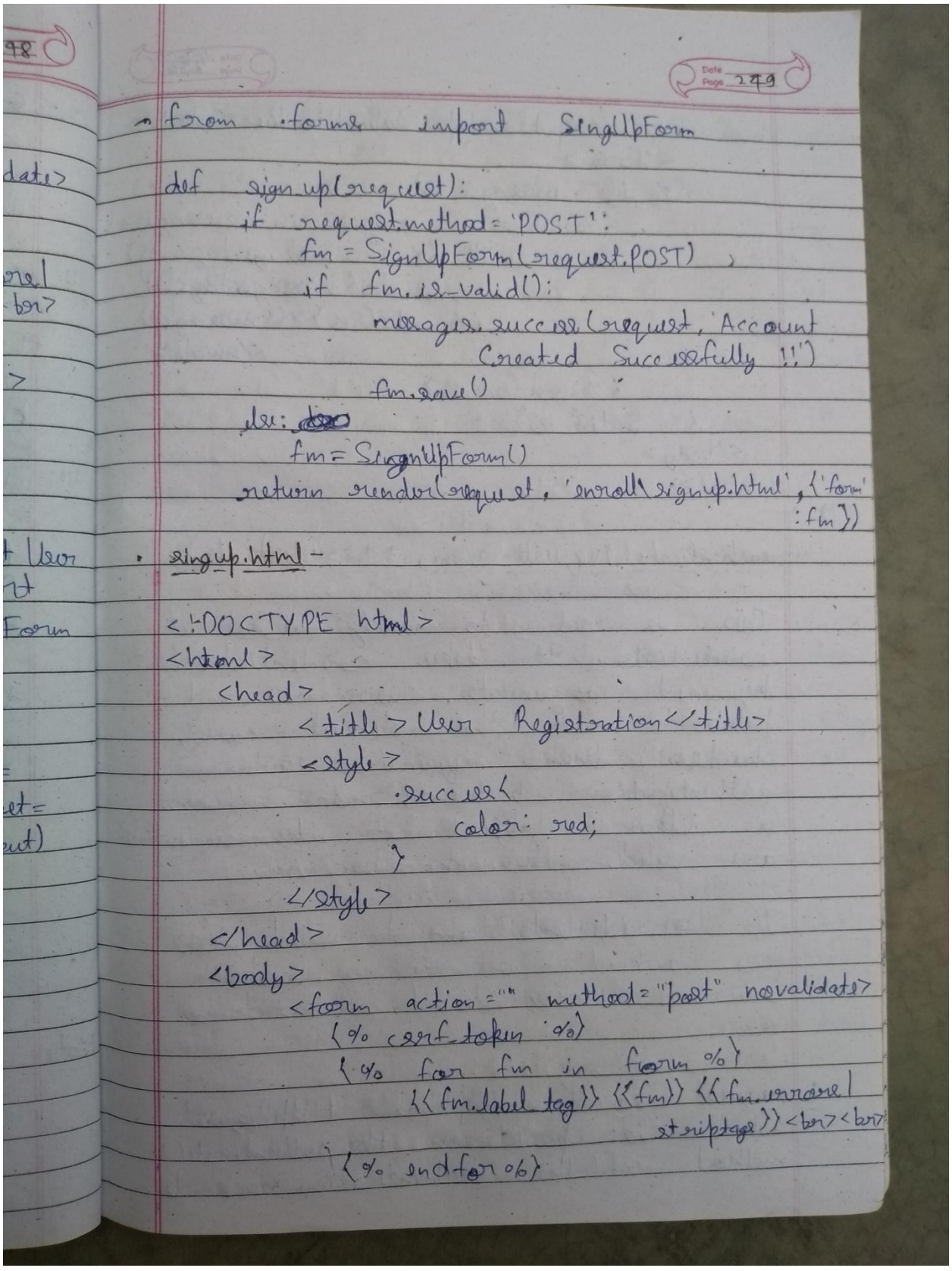
```
<form action="" method="post" novalidate>
  {% csrf token %}
  {% for fm in form %}
      {{ fm.label_tag }} {{ fm }} {{ fm.errors|
                                striptags }} <br> <br>
  {% endfor %}
  <input type="submit" value="Submit">
</form>
```

(ii) forms.py -

```
from django.contrib.auth.forms import User
from django.contrib.auth.forms import
                                UserCreationForm
from django import forms

class SignUpForm(UserCreationForm):
    password2 = forms.CharField(label=
        'Confirm Password (again)', widget=
                                forms.PasswordInput)
    class Meta:
        model = User
        fields = ['username', 'firstname',
                  'last name', 'email']
        labels = {'email': 'Email'}
```

· views.py -

```
from django.shortcuts import render
from django.contrib import messages
```

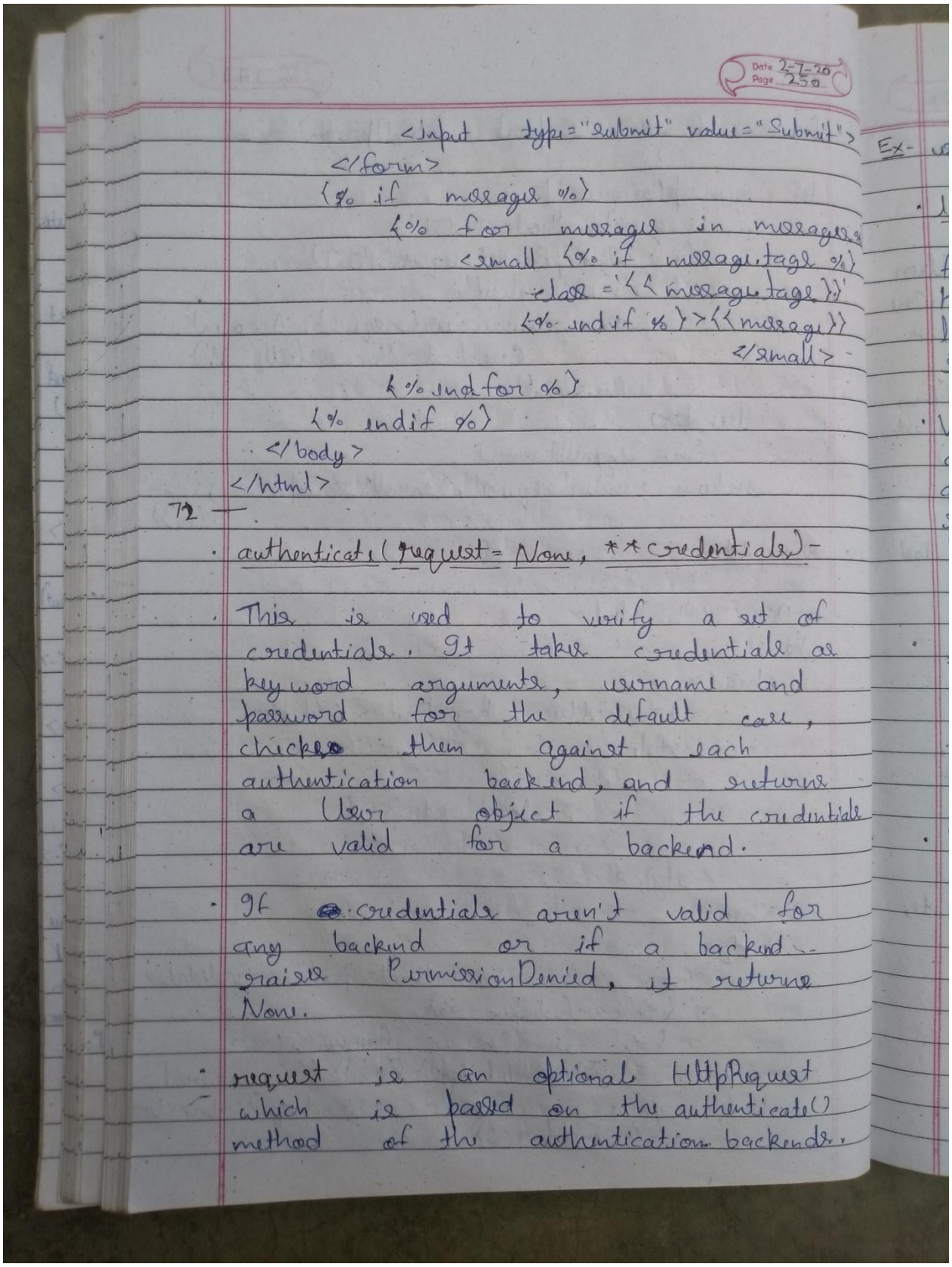


- from .forms import SignUpForm

```
def sign_up(request):
    if request.method == 'POST':
        fm = SignUpForm(request.POST)
        if fm.is_valid():
            messages.success(request, 'Account
                Created Successfully !!')
            fm.save()
    else:
        fm = SignUpForm()
    return render(request, 'enroll/signup.html', {'form'
                                                  :fm})
```

- signup.html -

```
<!DOCTYPE html>
<html>
  <head>
    <title> User Registration</title>
    <style>
      .success{
        color: red;
      }
    </style>
  </head>
  <body>
    <form action="" method="post" novalidate>
      {% csrf_token %}
      {% for fm in form %}
        {{ fm.label_tag }} {{ fm }} {{ fm.errors|
                                    striptags }} <br> <br>
      {% endfor %}
```



```
        <input type="submit" value="Submit">
    </form>
    {% if message %}
        {% for message in message %}
            <small {% if message.tags %}
                class = '{{ message.tags }}'
            {% endif %}>{{ message }}
                </small>
        {% endfor %}
    {% endif %}
</body>
</html>
```

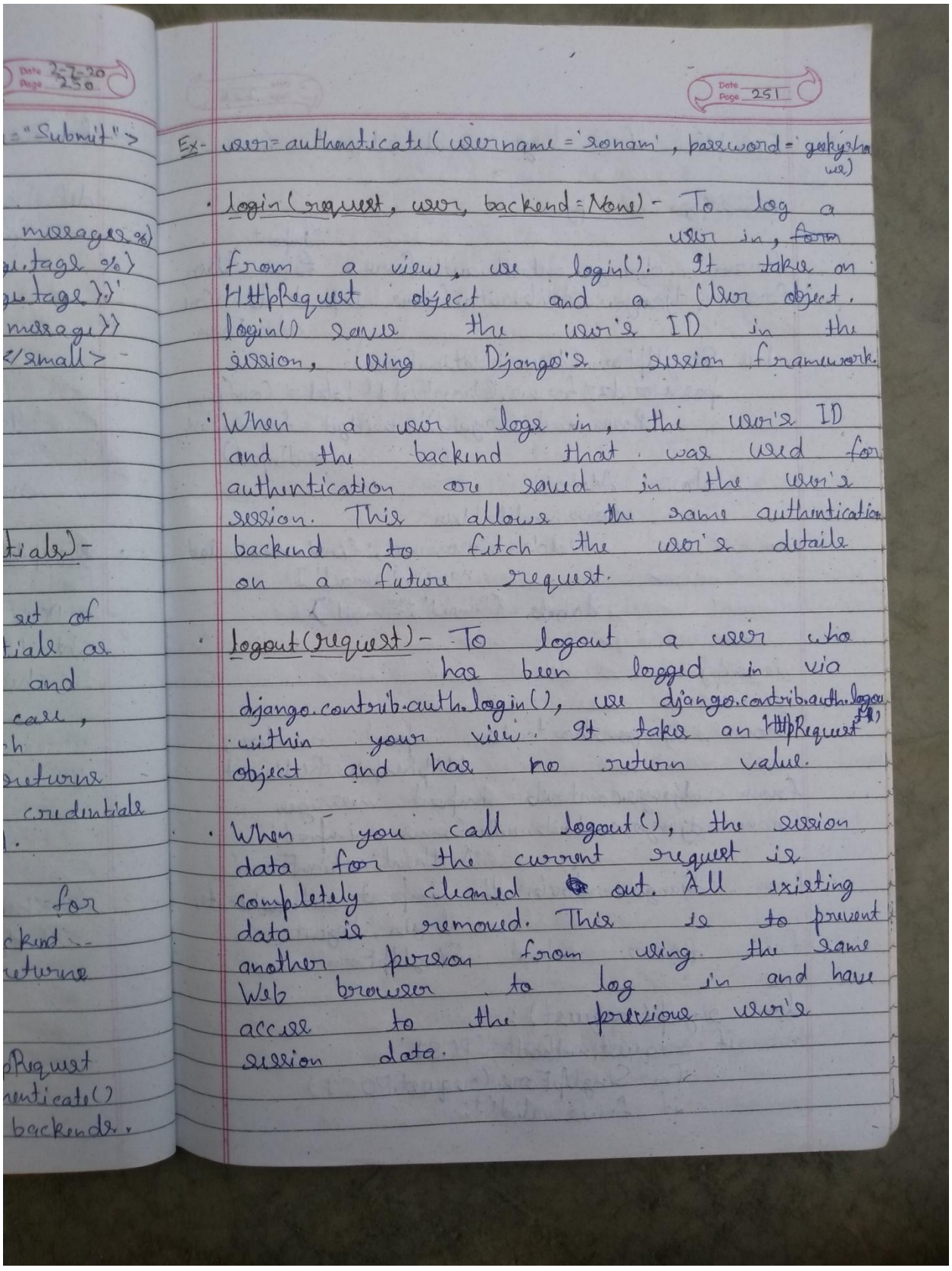
72 —

- authenticate(request=None, **credentials)-

- This is used to verify a set of credentials. It takes credentials as keyword arguments, username and password for the default case, checks them against each authentication backend, and returns a User object if the credentials are valid for a backend.

- If credentials aren't valid for any backend or if a backend raises PermissionDenied, it returns None.

- request is an optional HttpRequest which is passed on the authenticate() method of the authentication backends.



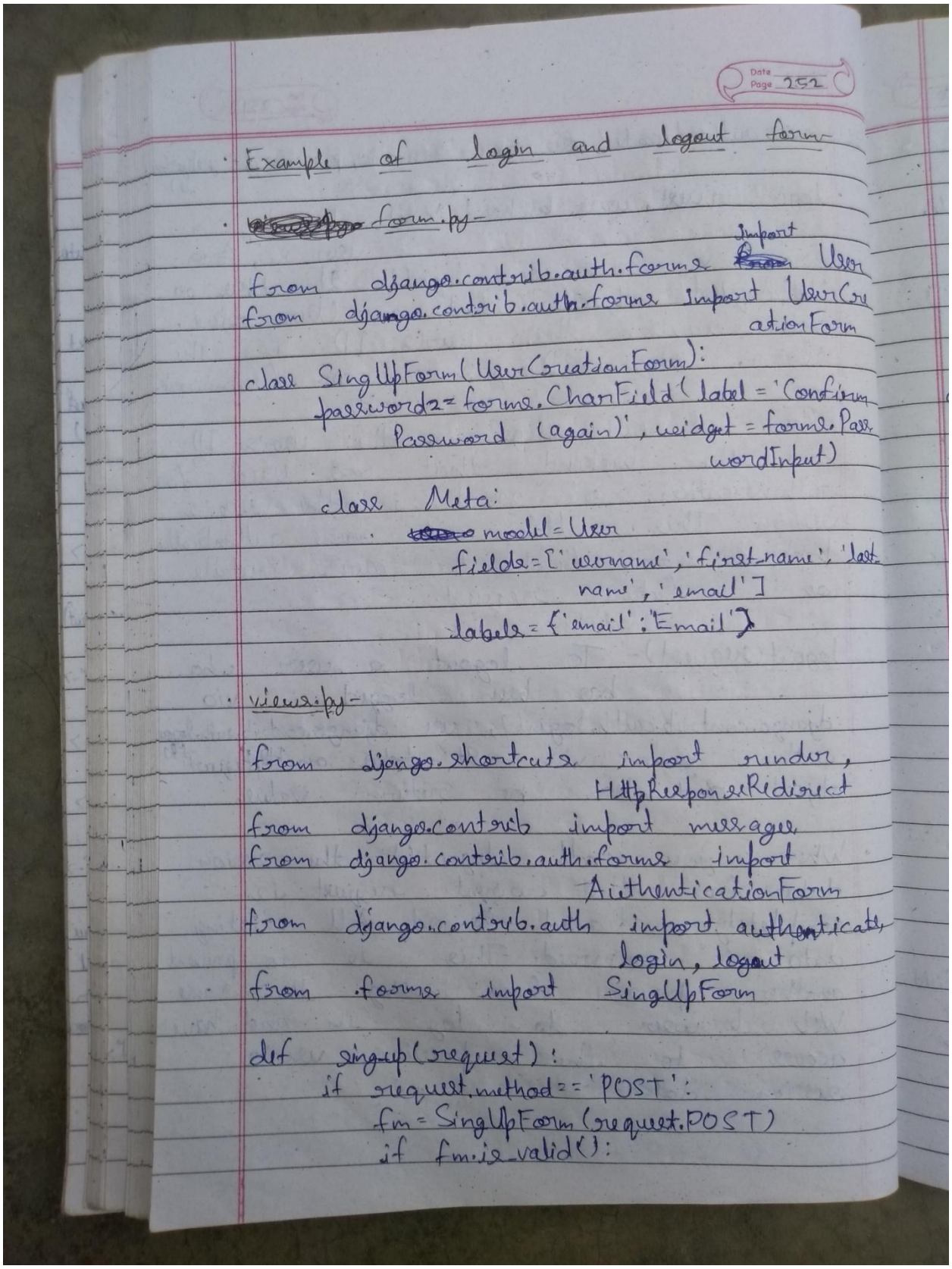
Ex- user = authenticate(username='sonam', password='geekyshows')

- login(request, user, backend=None) - To log a user in, from a view, use login(). It takes an HttpRequest object and a User object. login() saves the user's ID in the session, using Django's session framework.

- When a user logs in, the user's ID and the backend that was used for authentication are saved in the user's session. This allows the same authentication backend to fetch the user's details on a future request.

- logout(request) - To logout a user who has been logged in via django.contrib.auth.login(), use django.contrib.auth.logout() within your view. It takes an HttpRequest object and has no return value.

- When you call logout(), the session data for the current request is completely cleaned out. All existing data is removed. This is to prevent another person from using the same Web browser to log in and have access to the previous user's session data.



Date Page 252

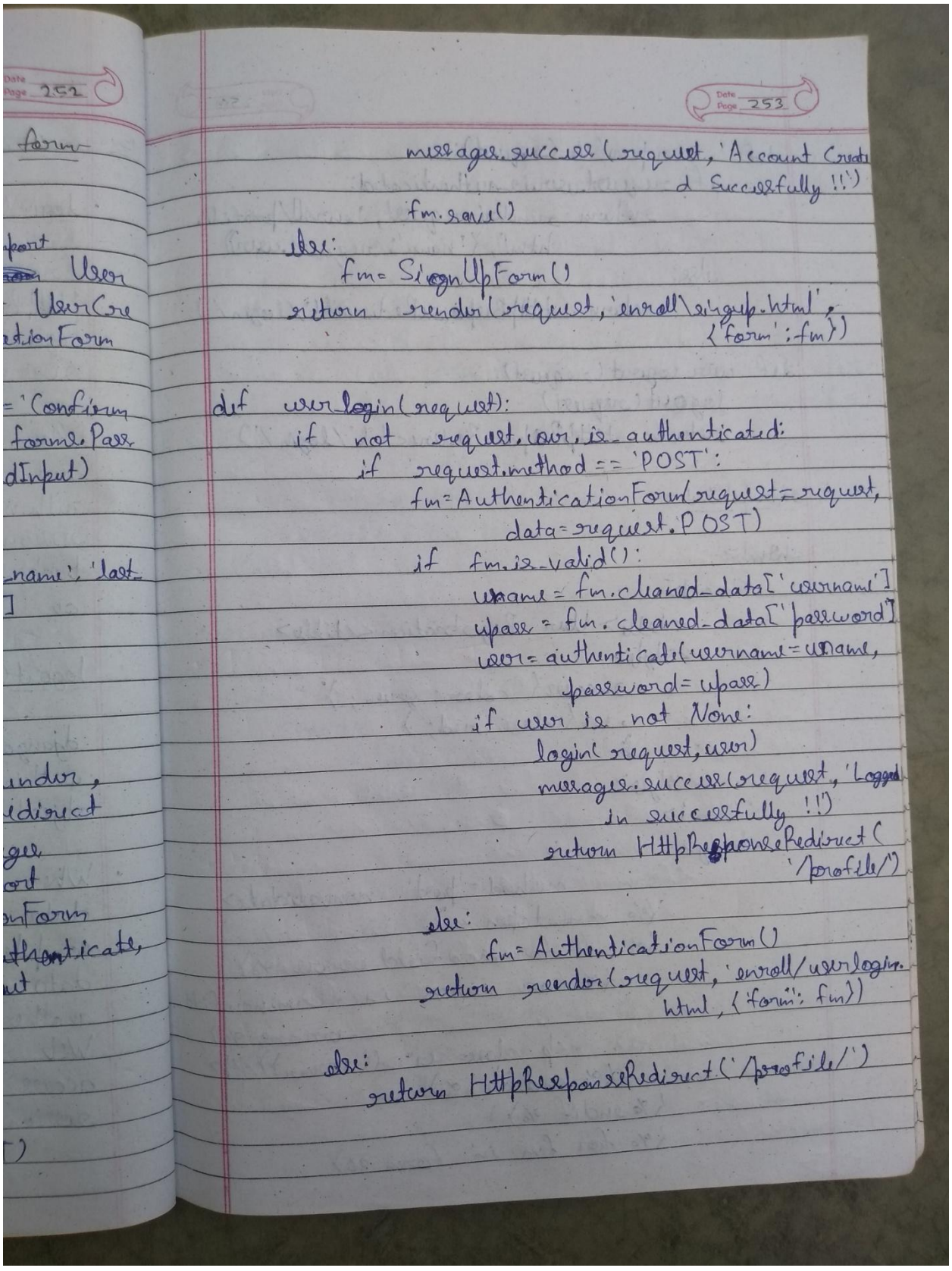
Example of login and logout form

- form.py -

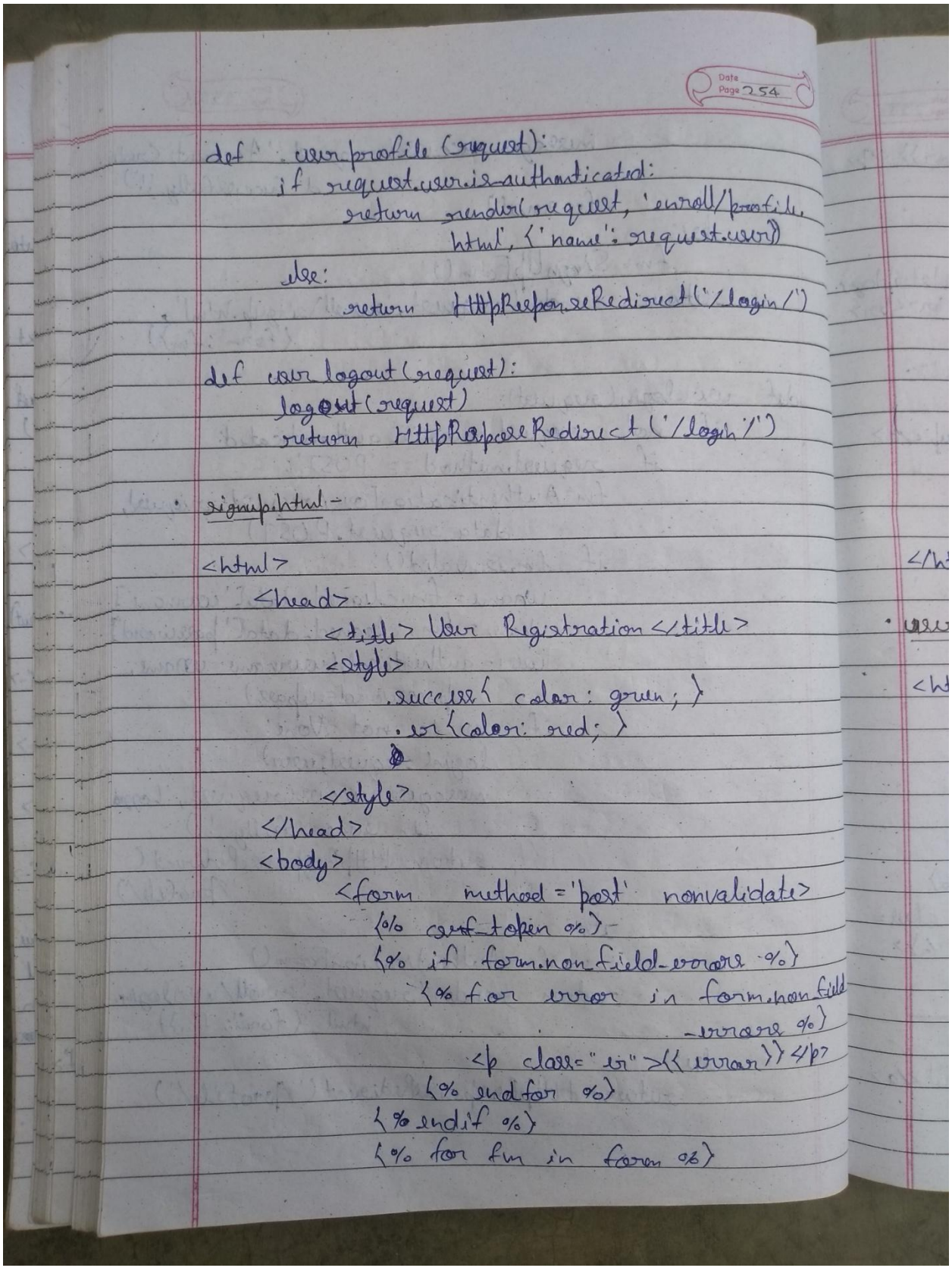
```
from django.contrib.auth.forms import UserCreationForm

class SingUpForm(UserCreationForm):
    password2 = forms.CharField(label = 'Confirm Password (again)', widget = forms.PasswordInput)

    class Meta:
        model = User
        fields = ['username', 'first_name', 'last_name', 'email']
        labels = {'email': 'Email'}
```

- views.py -

```
from django.shortcuts import render, HttpResponseRedirect
from django.contrib import messages
from django.contrib.auth.forms import AuthenticationForm
from django.contrib.auth import authenticate, login, logout
from .forms import SingUpForm

def singup(request):
    if request.method == 'POST':
        fm = SingUpForm(request.POST)
        if fm.is_valid():
```



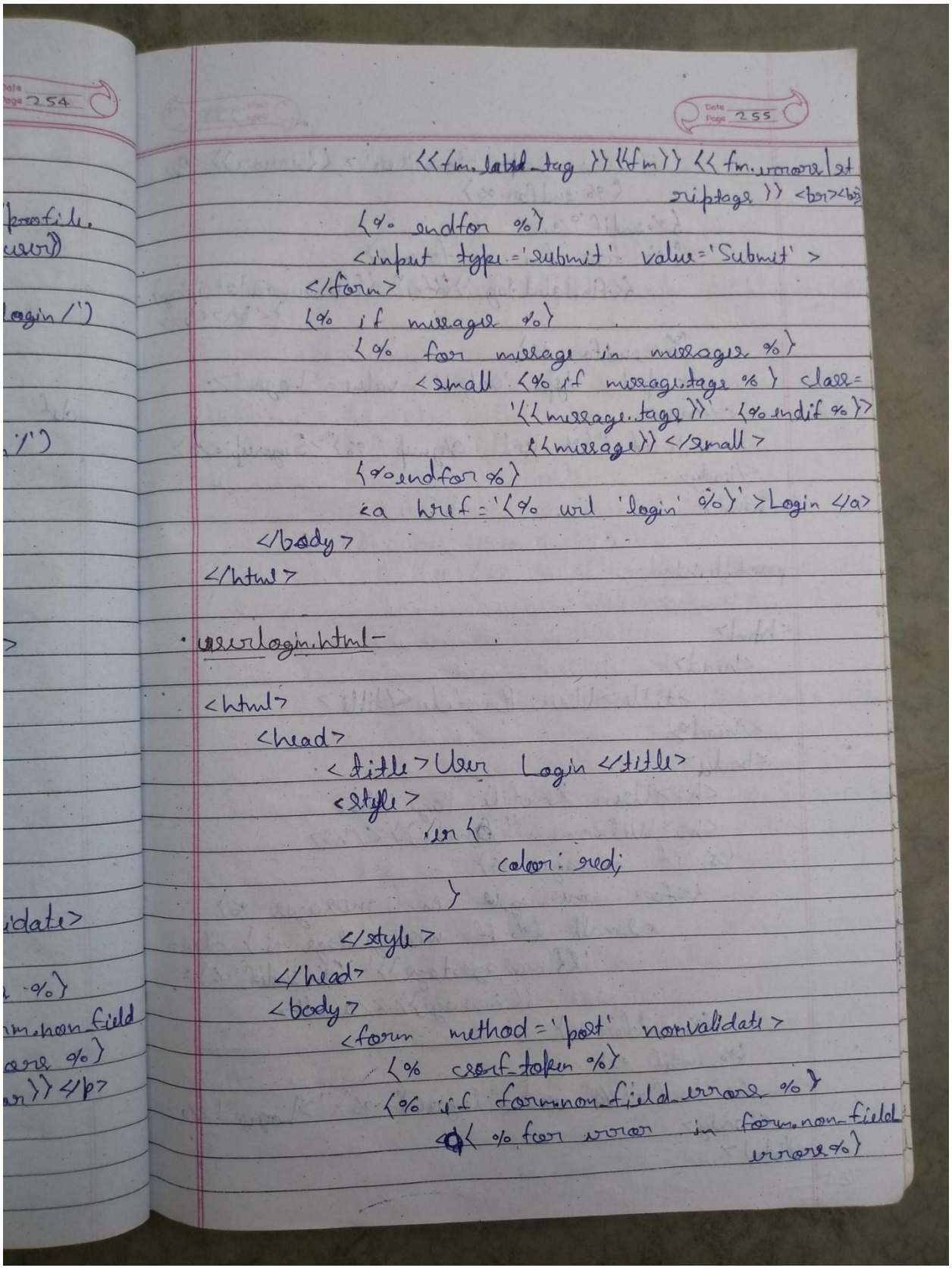
```
                messages.success(request, 'Account Created Successfully !!')
                fm.save()
        else:
            fm = SignUpForm()
        return render(request, 'enroll/signup.html', {'form':fm})

def user_login(request):
    if not request.user.is_authenticated:
        if request.method == 'POST':
            fm = AuthenticationForm(request=request, data=request.POST)
            if fm.is_valid():
                uname = fm.cleaned_data['username']
                upass = fm.cleaned_data['password']
                user = authenticate(username=uname, password=upass)
                if user is not None:
                    login(request, user)
                    messages.success(request, 'Logged in successfully !!')
                    return HttpResponseRedirect('/profile/')
        else:
            fm = AuthenticationForm()
        return render(request, 'enroll/userlogin.html', {'form':fm})
    else:
        return HttpResponseRedirect('/profile/')
```



```
def user_profile(request):
    if request.user.is_authenticated:
        return render(request, 'enroll/profile.
                html', {'name': request.user})
    else:
        return HttpResponseRedirect('/login/')

def user_logout(request):
    logout(request)
    return HttpResponseRedirect('/login/')
```

signup.html -

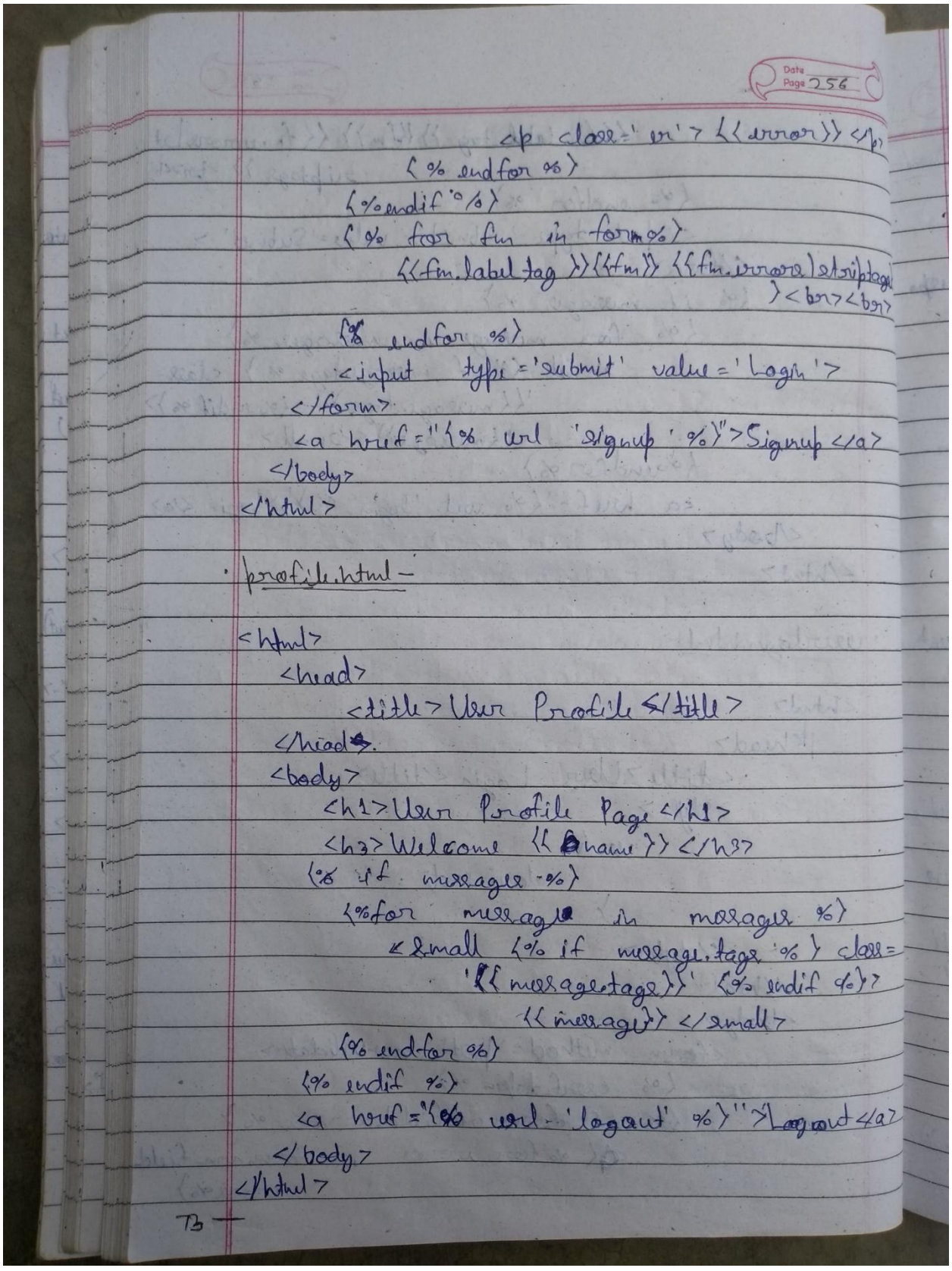
```
<html>
  <head>
    <title> User Registration </title>
    <style>
      .success { color: green; }
      .error { color: red; }
    </style>
  </head>
  <body>
    <form method='post' novalidate>
      {% csrf_token %}
      {% if form.non_field_errors %}
        {% for error in form.non_field
                _errors %}
          <p class="er">{{ error }}</p>
        {% endfor %}
      {% endif %}
      {% for fm in form %}
```



```
{{ fm.label_tag }} {{ fm }} {{ fm.errors|st
                                    ripttags }} <br><br>
        {% endfor %}
        <input type='submit' value='Submit' >
    </form>
    {% if messages %}
        {% for message in messages %}
            <small {% if message.tags %} class=
                '{{ message.tags }}' {% endif %}>
                {{ message }} </small>
        {% endfor %}
        <a href='{% url 'login' %}' >Login </a>
</body>
</html>
```

- userlogin.html -

```
<html>
    <head>
        <title> User Login </title>
        <style>
            .errorlist {
                color: red;
            }
        </style>
    </head>
    <body>
        <form method='post' novalidate>
            {% csrf_token %}
            {% if form.non_field_errors %}
                {% for error in form.non_field_
                                    errors %}
```

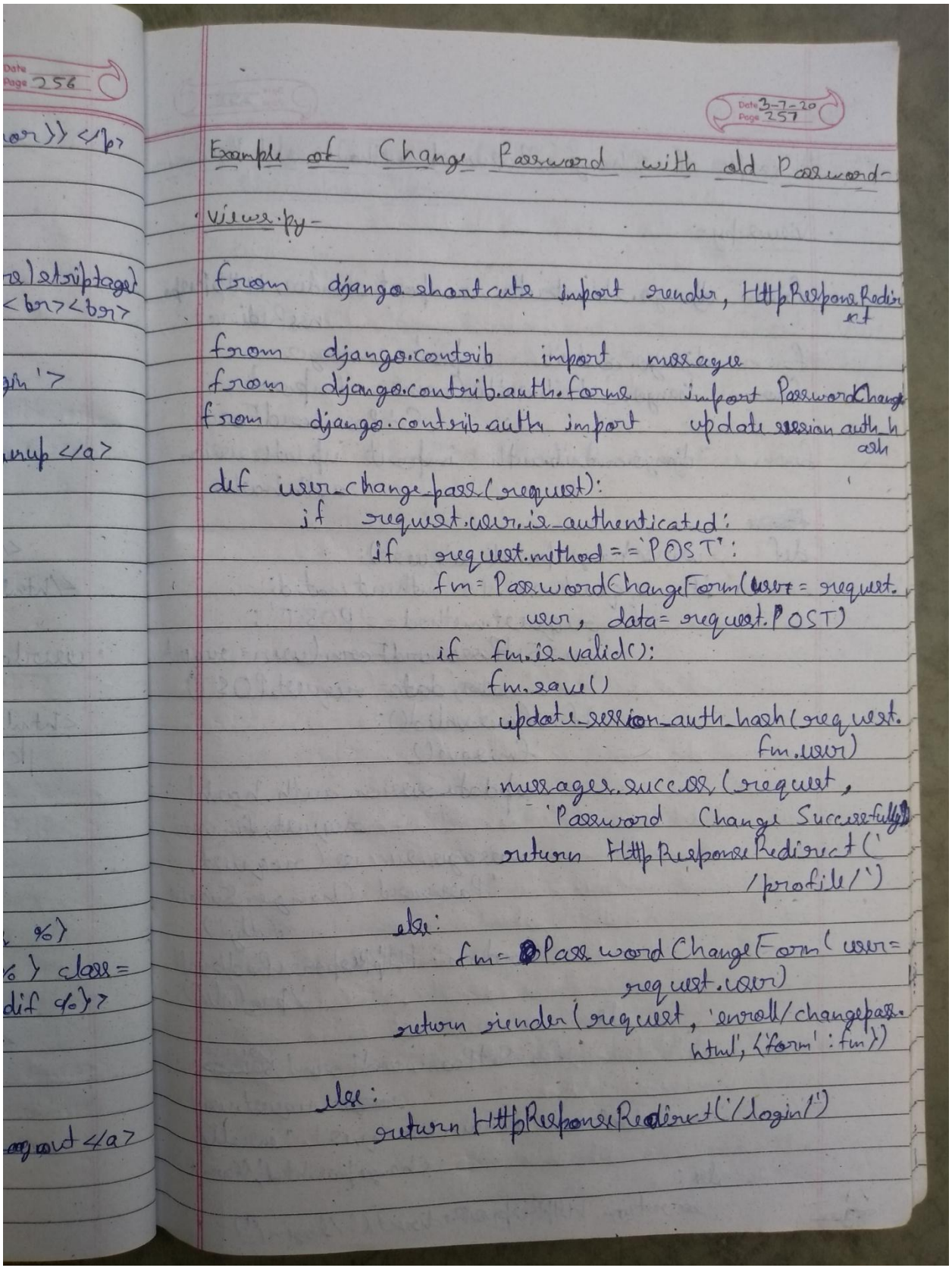


```
                <p class='er'> {{ error }} </p>
                {% endfor %}
            {% endif %}
            {% for fm in form %}
                {{ fm.label_tag }} {{ fm }} {{ fm.errors|striptags }} <br><br>
            {% endfor %}
            <input type='submit' value='Login'>
        </form>
        <a href="{% url 'signup' %}">Signup</a>
    </body>
</html>
```

- profile.html –

```
<html>
    <head>
        <title> User Profile </title>
    </head>
    <body>
        <h1> User Profile Page </h1>
        <h3> Welcome {{ name }} </h3>
        {% if messages %}
            {% for message in messages %}
                <small {% if message.tags %} class='{{ message.tags }}' {% endif %}>
                    {{ message }} </small>
            {% endfor %}
        {% endif %}
        <a href="{% url 'logout' %}">Logout</a>
    </body>
</html>
```

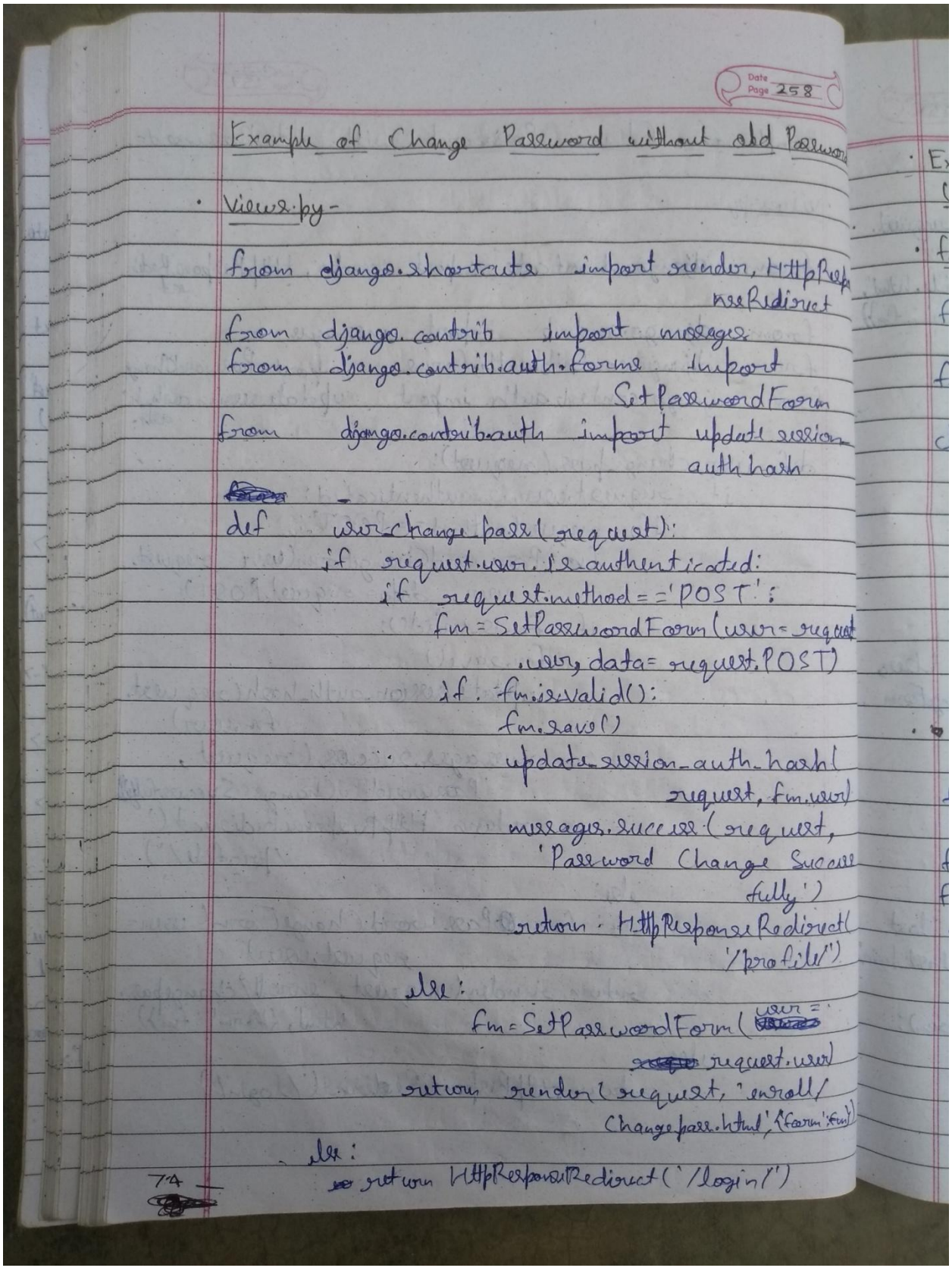
TB –



Example of Change Password with old Password-

views.py-

```
from django.shortcuts import render, HttpResponseRedirect
from django.contrib import messages
from django.contrib.auth.forms import PasswordChangeForm
from django.contrib.auth import update_session_auth_hash

def user_change_pass(request):
    if request.user.is_authenticated:
        if request.method == 'POST':
            fm = PasswordChangeForm(user = request.user, data = request.POST)
            if fm.is_valid():
                fm.save()
                update_session_auth_hash(request, fm.user)
                messages.success(request, 'Password Change Successfully')
                return HttpResponseRedirect('/profile/')
        else:
            fm = PasswordChangeForm(user = request.user)
        return render(request, 'enroll/changepass.html', {'form': fm})
    else:
        return HttpResponseRedirect('/login/')
```



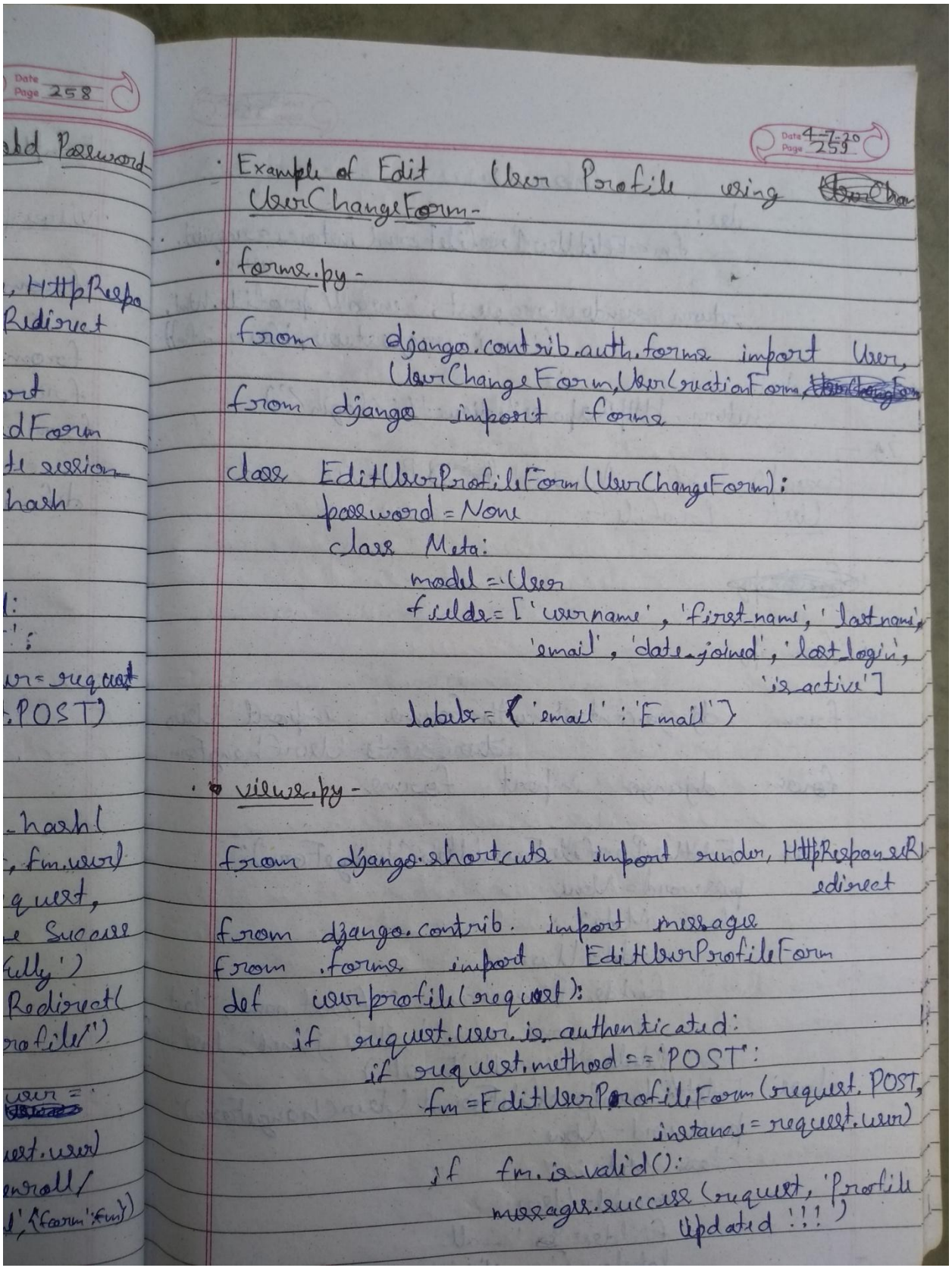
Date
Page 258

Example of Change Password without old Password

- Views.py -

```
from django.shortcuts import render, HttpResponseRedirect
from django.contrib import messages
from django.contrib.auth.forms import SetPasswordForm
from django.contrib.auth import update_session_auth_hash

def user_change_pass(request):
    if request.user.is_authenticated:
        if request.method=='POST':
            fm = SetPasswordForm(user=request.user, data=request.POST)
            if fm.is_valid():
                fm.save()
                update_session_auth_hash(request, fm.user)
                messages.success(request, 'Password Change Successfully')
                return HttpResponseRedirect('/profile/')
        else:
            fm = SetPasswordForm(user=request.user)
        return render(request, 'enroll/changepass.html', {'form':fm})
    else:
        return HttpResponseRedirect('/login/')
```

74



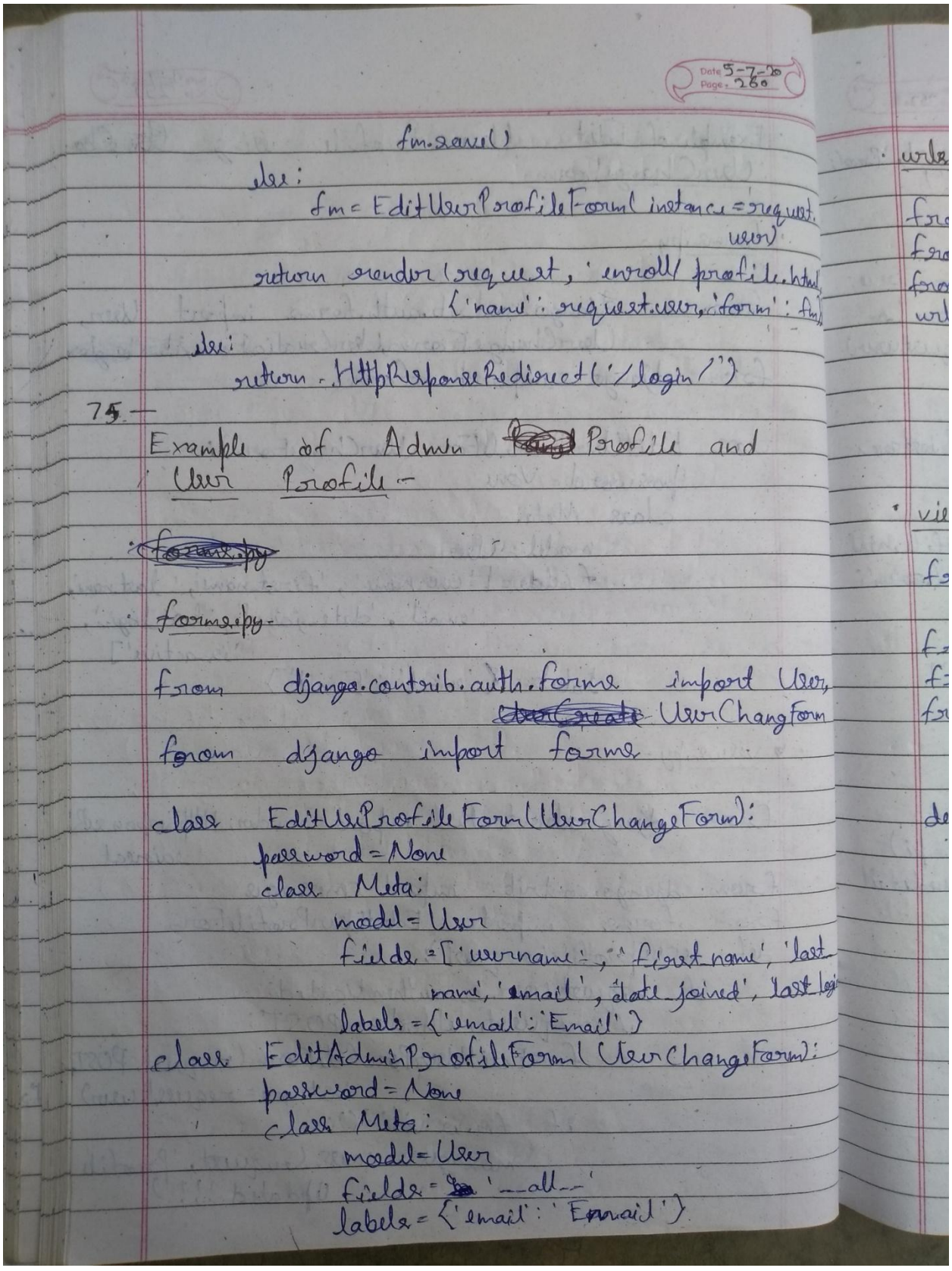
Example of Edit User Profile using UserChangeForm-

forms.py -

```
from django.contrib.auth.forms import User, UserChangeForm, UserCreationForm
from django import forms

class EditUserProfileForm(UserChangeForm):
    password = None
    class Meta:
        model = User
        fields = ['username', 'first_name', 'last_name',
                  'email', 'date_joined', 'last_login',
                  'is_active']
        labels = {'email': 'Email'}
```

views.py -

```
from django.shortcuts import render, HttpResponseRedirect
from django.contrib import messages
from .forms import EditUserProfileForm
def user_profile(request):
    if request.user.is_authenticated:
        if request.method == 'POST':
            fm = EditUserProfileForm(request.POST,
                                     instance=request.user)
            if fm.is_valid():
                messages.success(request, 'Profile
                                 Updated !!!')
```



fm.save()

else:

fm = EditUserProfileForm(instance=request.user)

return render(request, 'enroll/profile.html', {'name': request.user, 'form': fm})

else:

return HttpResponseRedirect('/login/')

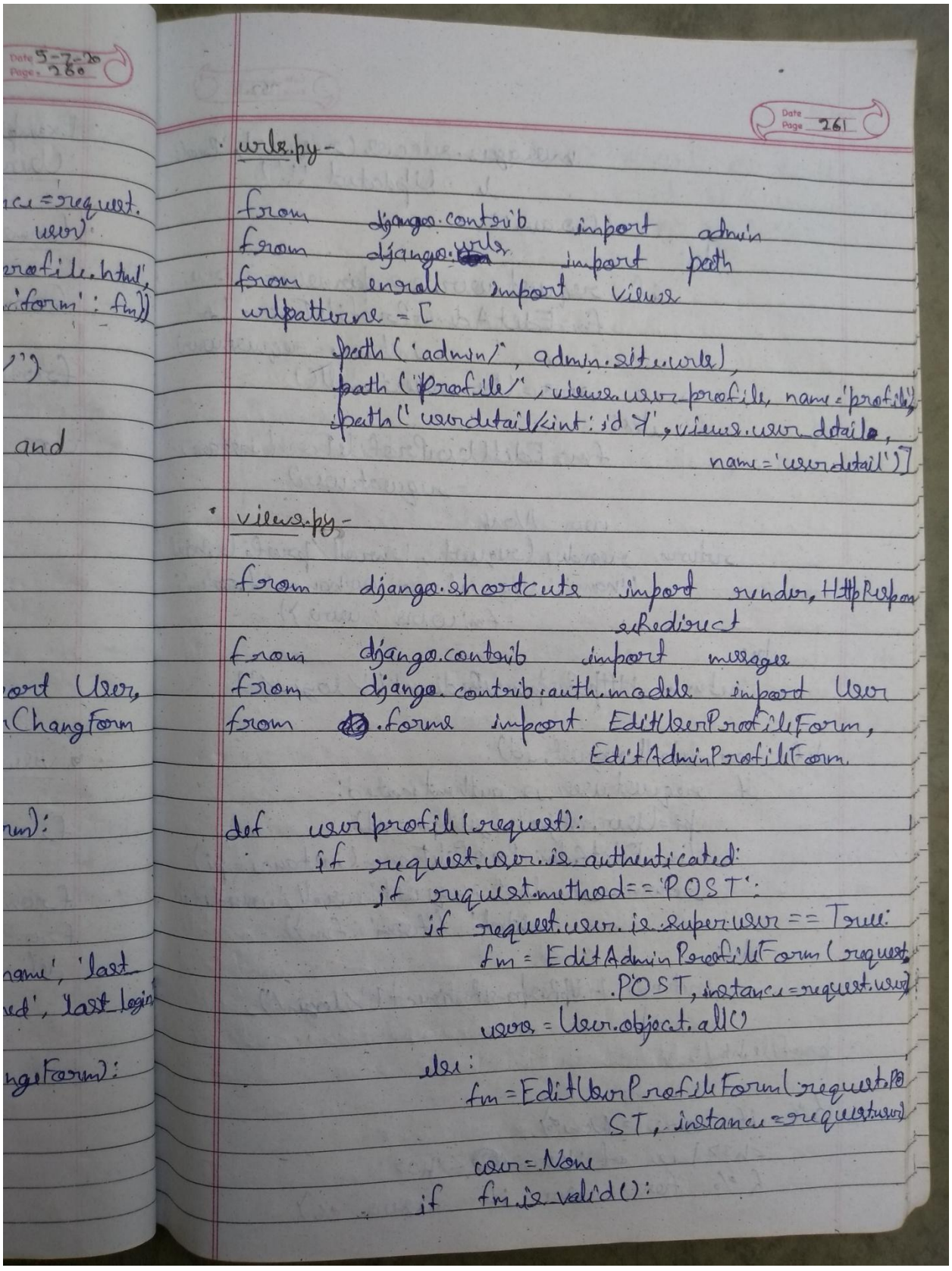
75.—

Example of Admin ~~Profile~~ Profile and User Profile —

~~forms.py~~

forms.py -

```
from django.contrib.auth.forms import UserChangeForm
from django import forms

class EditUserProfileForm(UserChangeForm):
    password = None
    class Meta:
        model = User
        fields = ['username', 'first_name', 'last_name', 'email', 'date_joined', 'last_login']
        labels = {'email': 'Email'}
class EditAdminProfileForm(UserChangeForm):
    password = None
    class Meta:
        model = User
        fields = '__all__'
        labels = {'email': 'Email'}
```

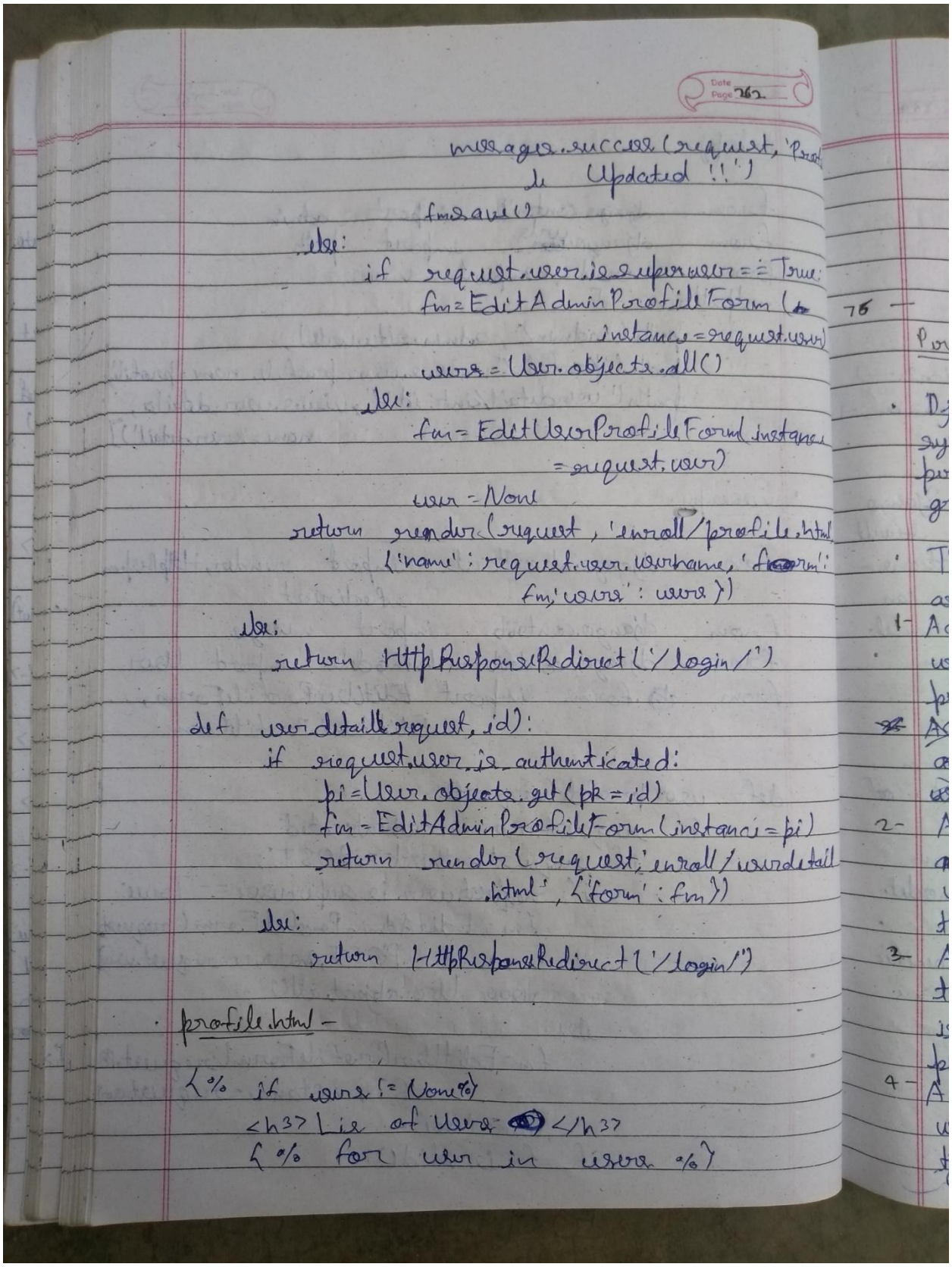


· urls.py -

```
from django.contrib import admin
from django.urls import path
from enroll import views
urlpatterns = [
    path('admin/', admin.site.urls),
    path('profile/', views.user_profile, name='profile'),
    path('userdetail/<int:id>', views.user_ddetails,
         name='userdetail')]
```

· views.py -

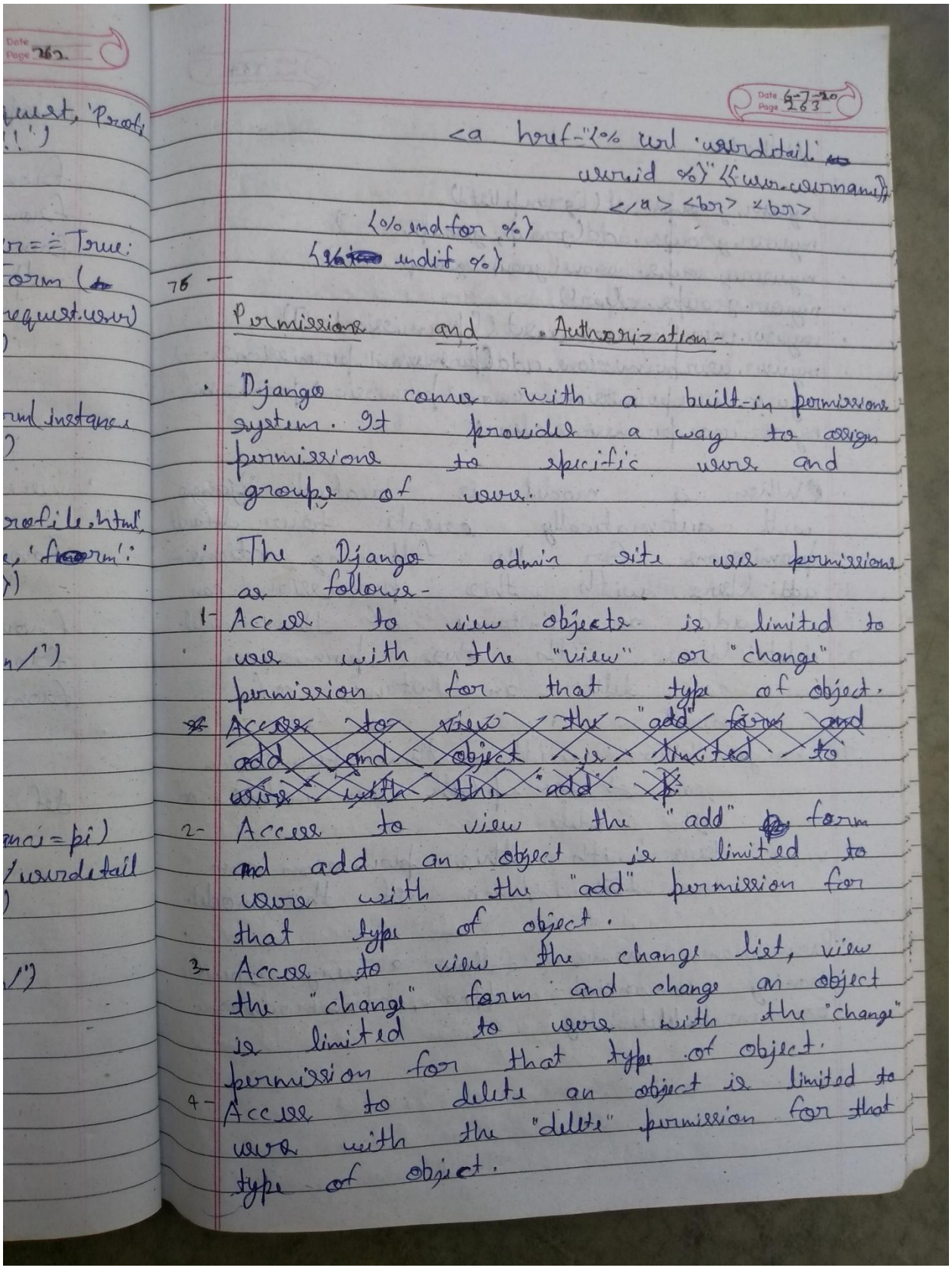
```
from django.shortcuts import render, HttpResponse
    ,Redirect
from django.contrib import messages
from django.contrib.auth.models import User
from .forms import EditUserProfileForm,
    EditAdminProfileForm

def user_profile(request):
    if request.user.is_authenticated:
        if request.method == 'POST':
            if request.user.is_superuser == True:
                fm = EditAdminProfileForm(request
                    .POST, instance=request.user)
                users = User.objects.all()
            else:
                fm = EditUserProfileForm(request.PO
                    ST, instance=request.user)
                users = None
            if fm.is_valid():
```



```
            messages.success(request, 'Profile Updated !!')
            fm.save()
        else:
            if request.user.is_superuser == True:
                fm = EditAdminProfileForm(instance=request.user)
                users = User.objects.all()
            else:
                fm = EditUserProfileForm(instance=request.user)
                users = None
            return render(request, 'enroll/profile.html', {'name': request.user.username, 'form': fm, 'users': users})
    else:
        return HttpResponseRedirect('/login/')

def user_detail(request, id):
    if request.user.is_authenticated:
        pi = User.objects.get(pk=id)
        fm = EditAdminProfileForm(instance=pi)
        return render(request, 'enroll/userdetail.html', {'form': fm})
    else:
        return HttpResponseRedirect('/login/')
```

- profile.html -

```
{% if users != None %}
    <h3> List of Users </h3>
    {% for user in users %}
```



{{user.username}}

{% endfor %}

{% endif %}

76 —

Permissions and Authorization -

- Django comes with a built-in permissions system. It provides a way to assign permissions to specific users and groups of users.
- The Django admin site uses permissions as follows -

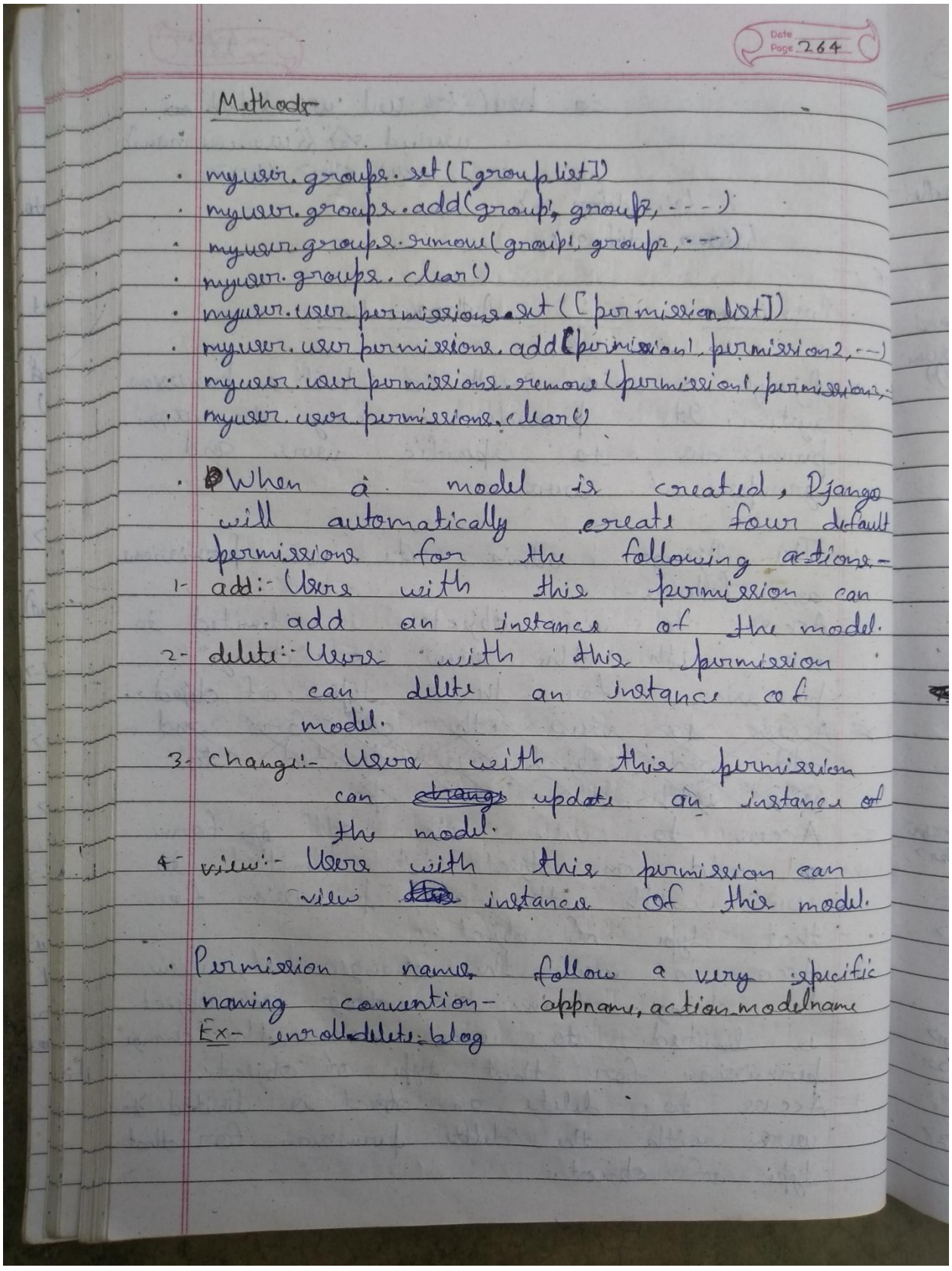
1- Access to view objects is limited to users with the "view" or "change" permission for that type of object.

~~* Access to view the "add" form and add an object is limited to users with this "add" *~~

2- Access to view the "add" form and add an object is limited to users with the "add" permission for that type of object.

3- Access to view the change list, view the "change" form and change an object is limited to users with the "change" permission for that type of object.

4- Access to delete an object is limited to users with the "delete" permission for that type of object.



Date Page 264

Methods-

- myuser.groups.set([grouplist])
- myuser.groups.add(group1, group2, ...)
- myuser.groups.remove(group1, group2, ...)
- myuser.groups.clear()
- myuser.user_permissions.set([permissionlist])
- myuser.user_permissions.add(permission1, permission2, ...)
- myuser.user_permissions.remove(permission1, permission2, ...)
- myuser.user_permissions.clear()

- When a model is created, Django will automatically create four default permissions for the following actions-

1- add:- Users with this permission can add an instance of the model.
2- delete:- Users with this permission can delete an instance of model.
3- change:- Users with this permission can ~~change~~ update an instance of the model.
4- view:- Users with this permission can view ~~the~~ instance of this model.

- Permission names follow a very specific naming convention- appname_action_modelname
Ex- enroll_delete_blog

Date Page 265

perms Template Variable -

The currently logged-in user's permissions are stored in the template variable {{ perms }}. This is an instance of django.contrib.auth.context_processors.PermWrapper, which is a template-friendly proxy of permissions.

Ex - (i) {% if perms.enroll.delete_blog %}
<input type="button" value="Delete">

{% endif %}

(ii) {% if perms.enroll %}
<input type="button" value="Delete">

{% endif %}

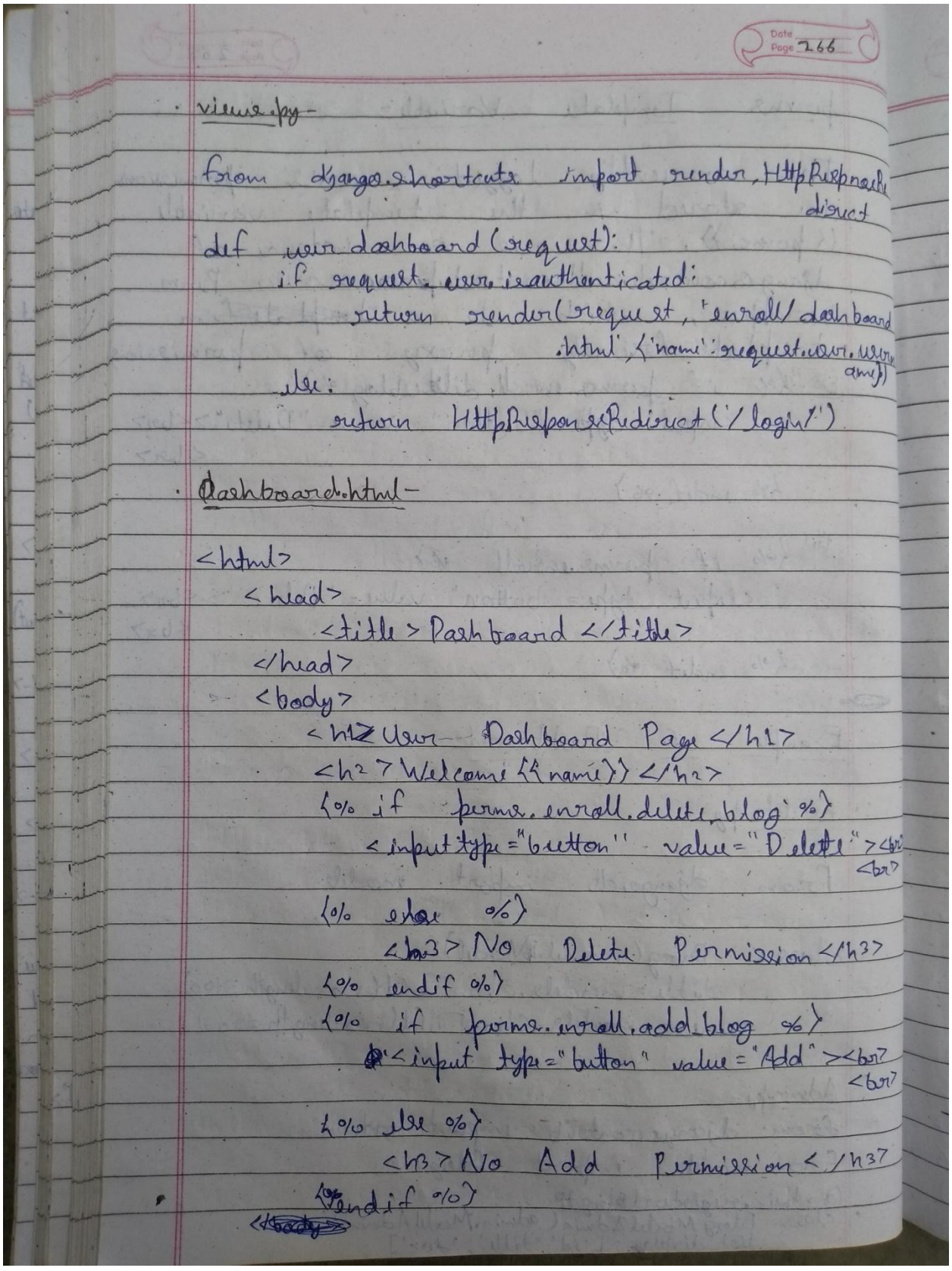
Example of Dashboard -

models.py -

```
from django.db import models

class Blog(models.Model):
    title = models.CharField(max_length=100)
    desc = models.CharField(max_length=200)
```

Admin.py -

```
from django.contrib import admin
from .models import Blog
@admin.register(Blog)
class BlogModelAdmin(admin.ModelAdmin):
    list_display = ['id', 'title', 'desc']
```



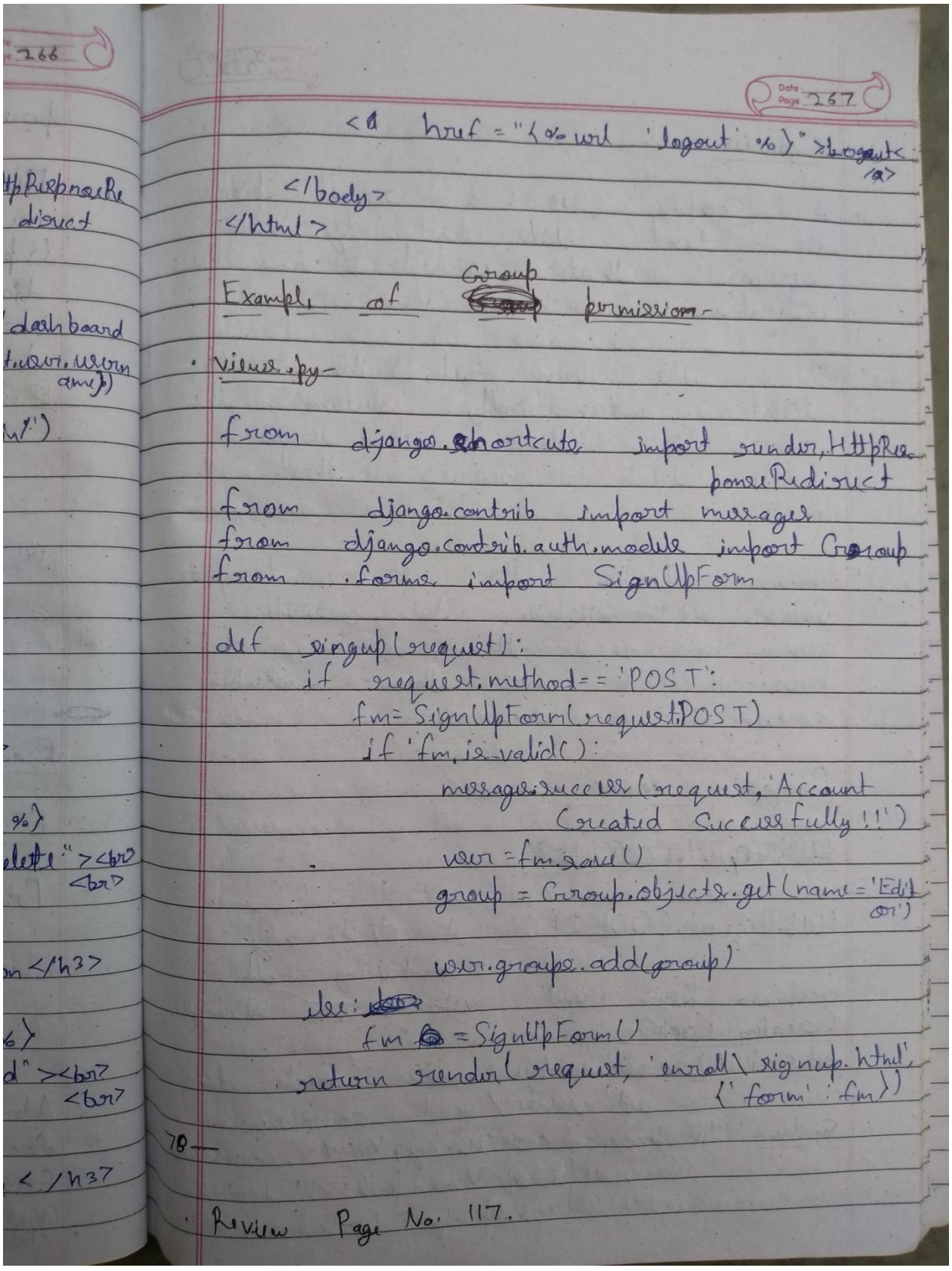
Date
Page 266

- views.py -

```
from django.shortcuts import render, HttpResponseRedirect

def user_dashboard(request):
    if request.user.is_authenticated:
        return render(request, 'enroll/dashboard.html', {'name': request.user.username})
    else:
        return HttpResponseRedirect('/login/')
```

- Dashboard.html -

```
<html>
  <head>
    <title> Dashboard </title>
  </head>
  <body>
    <h1> User Dashboard Page </h1>
    <h2> Welcome {{name}} </h2>
    {% if perms.enroll.delete_blog %}
      <input type="button" value="Delete"><br>
    {% else %}
      <h3> No Delete Permission </h3>
    {% endif %}
    {% if perms.enroll.add_blog %}
      <input type="button" value="Add"><br>
    {% else %}
      <h3> No Add Permission </h3>
    {% endif %}
  </body>
```



```
<a href="{% url 'logout' %}">Logout</a>
</body>
</html>
```

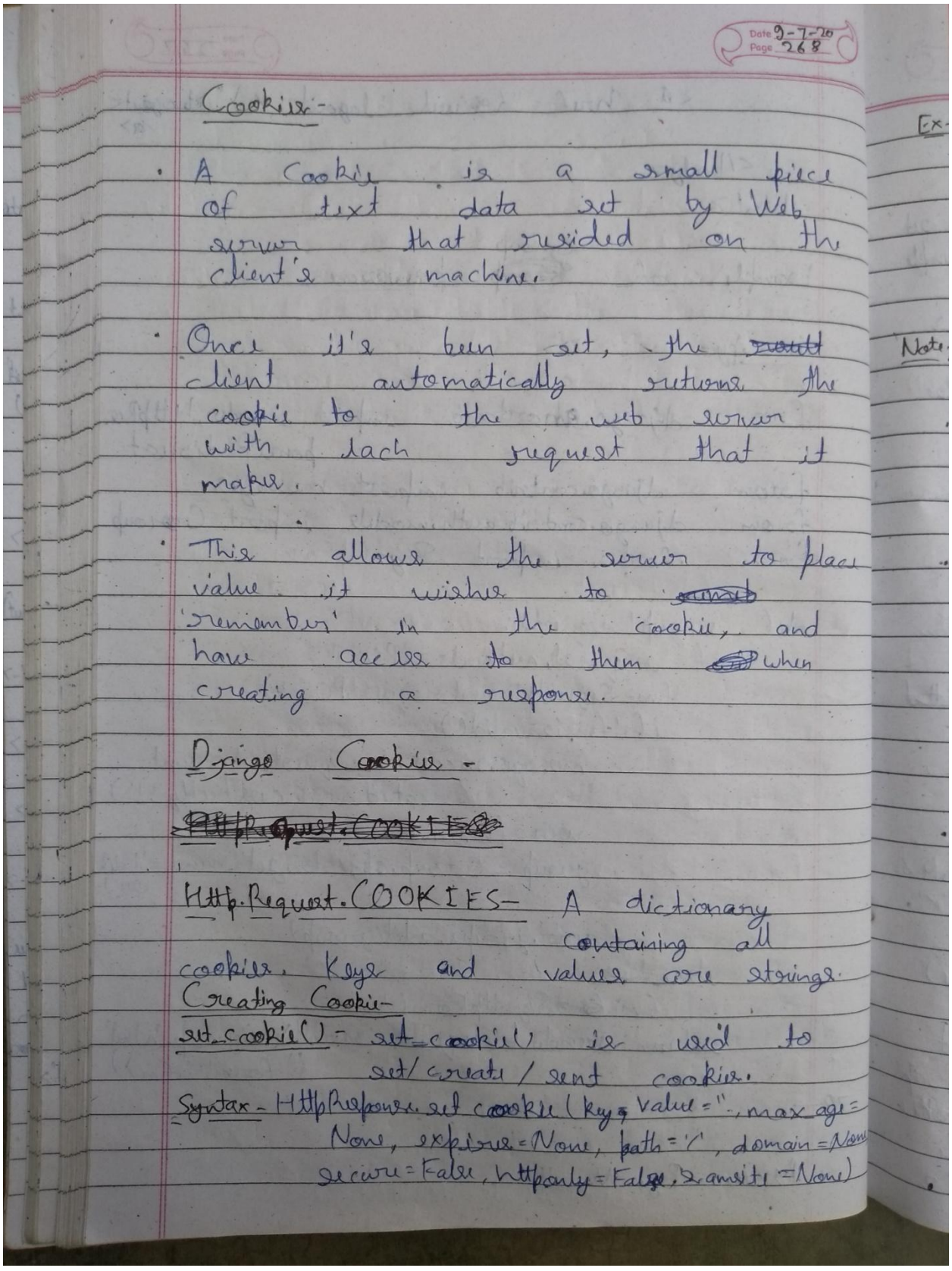
Example of Group permission -

- views.py -

```
from django.shortcuts import render, HttpResponseRedirect
from django.contrib import messages
from django.contrib.auth.models import Group
from .forms import SignUpForm

def signup(request):
    if request.method == 'POST':
        fm = SignUpForm(request.POST)
        if fm.is_valid():
            messages.success(request, 'Account Created Successfully !!')
            user = fm.save()
            group = Group.objects.get(name='Editor')
            user.groups.add(group)
    else:
        fm = SignUpForm()
    return render(request, 'enroll/signup.html', {'form': fm})
```

78—

- Review Page No. 117.



Date 9-7-20
Page 268

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.

Django Cookies -

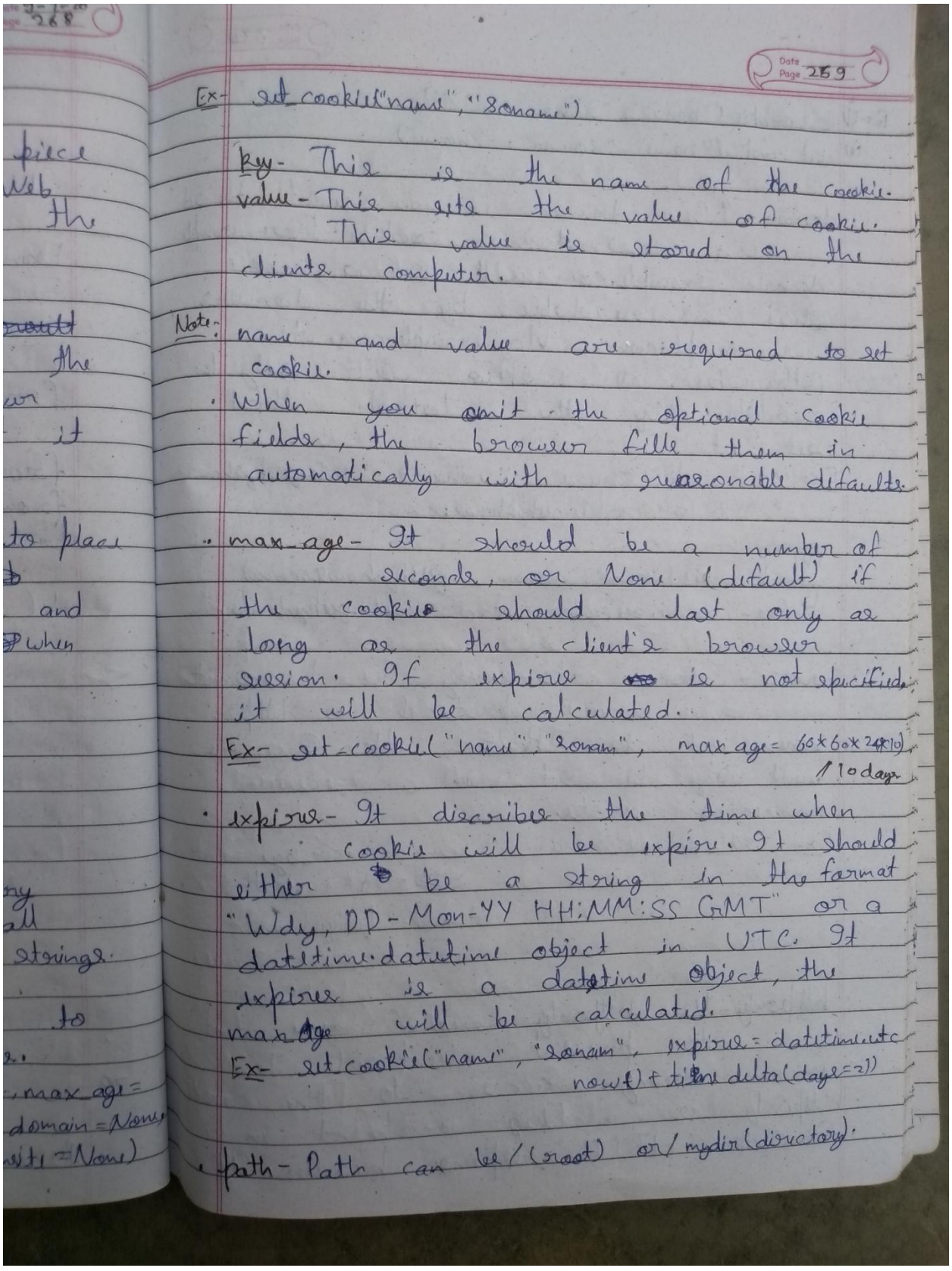
~~HttpRequest.COOKIES~~

HttpRequest.COOKIES - A dictionary containing all cookies. Keys and values are strings.

Creating Cookie -

set_cookie() - set_cookie() is used to set/create/sent cookies.

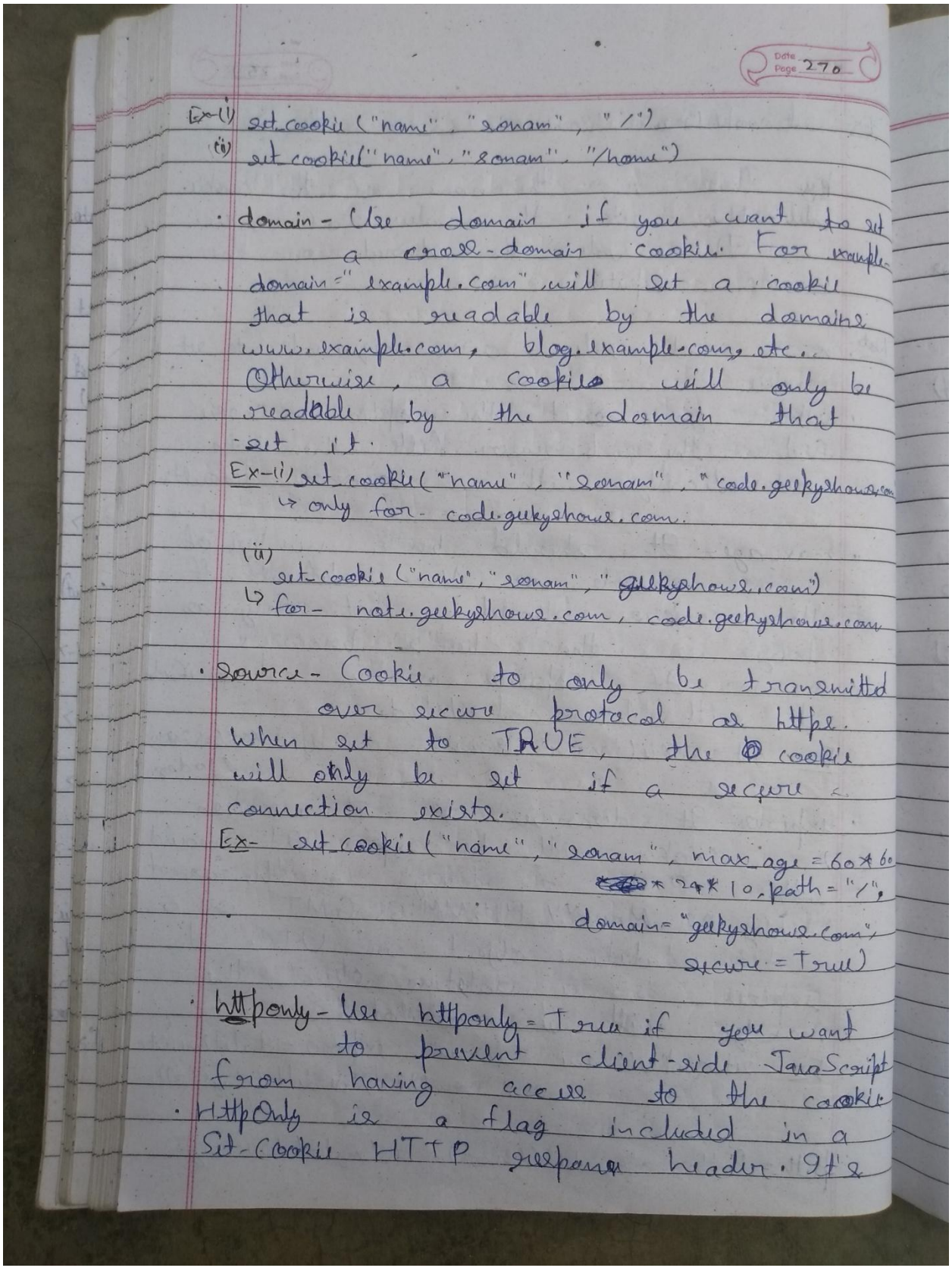
Syntax - HttpResponse.set_cookie(key, value='', max_age=None, expires=None, path='/', domain=None, secure=False, httponly=False, samesite=None)



Ex- set_cookie("name", "sonam")

key- This is the name of the cookie.
value- This sets the value of cookie. This value is stored on the clients computer.

Note:
- name and value are required to set cookie.
- When you omit the optional cookie fields, the browser fills them in automatically with reasonable defaults.
- max_age- It should be a number of seconds, or None (default) if the cookie should last only as long as the client's browser session. If expires is not specified, it will be calculated.
  Ex- set_cookie("name", "sonam", max_age= 60*60*24*10) // 10 days
- expires- It describes the time when cookie will be expire. It should either be a string in the format "Wdy, DD-Mon-YY HH:MM:SS GMT" or a datetime.datetime object in UTC. If expires is a datetime object, the max_age will be calculated.
  Ex- set_cookie("name", "sonam", expires= datetime.utcnow() + timedelta(days=2))
- path- Path can be / (root) or /mydir (directory).


Date Page 270

Ex-(i) set_cookie("name", "sonam", "/")
(ii) set_cookie("name", "sonam", "/home")

- domain - Use domain if you want to set a cross-domain cookie. For example, domain=".example.com" will set a cookie that is readable by the domains www.example.com, blog.example.com, etc. Otherwise, a cookie will only be readable by the domain that set it.

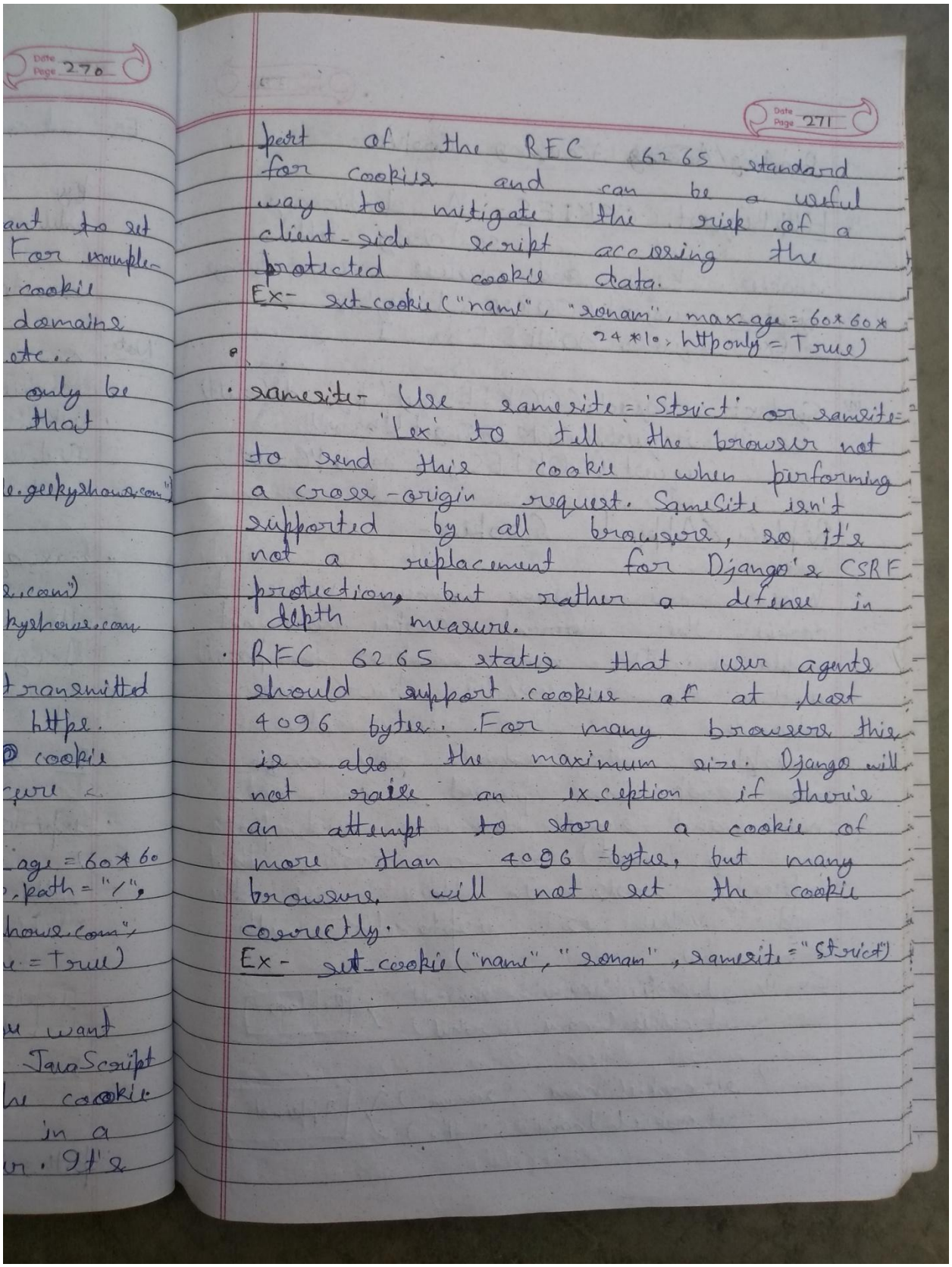
Ex-(i) set_cookie("name", "sonam", "code.geekyshows.com")
↳ only for - code.gukyshows.com.

(ii) set_cookie("name", "sonam", "geekyshows.com")
↳ for - note.geekyshows.com, code.geekyshows.com

- Source - Cookie to only be transmitted over secure protocol as https. When set to TRUE, the cookie will only be set if a secure connection exists.

Ex- set_cookie("name", "sonam", max_age = 60*60*24*10, path="/", domain="geekyshows.com", secure=True)

- httponly - Use httponly=True if you want to prevent client-side JavaScript from having access to the cookie.
- HttpOnly is a flag included in a Set-Cookie HTTP response header. It's

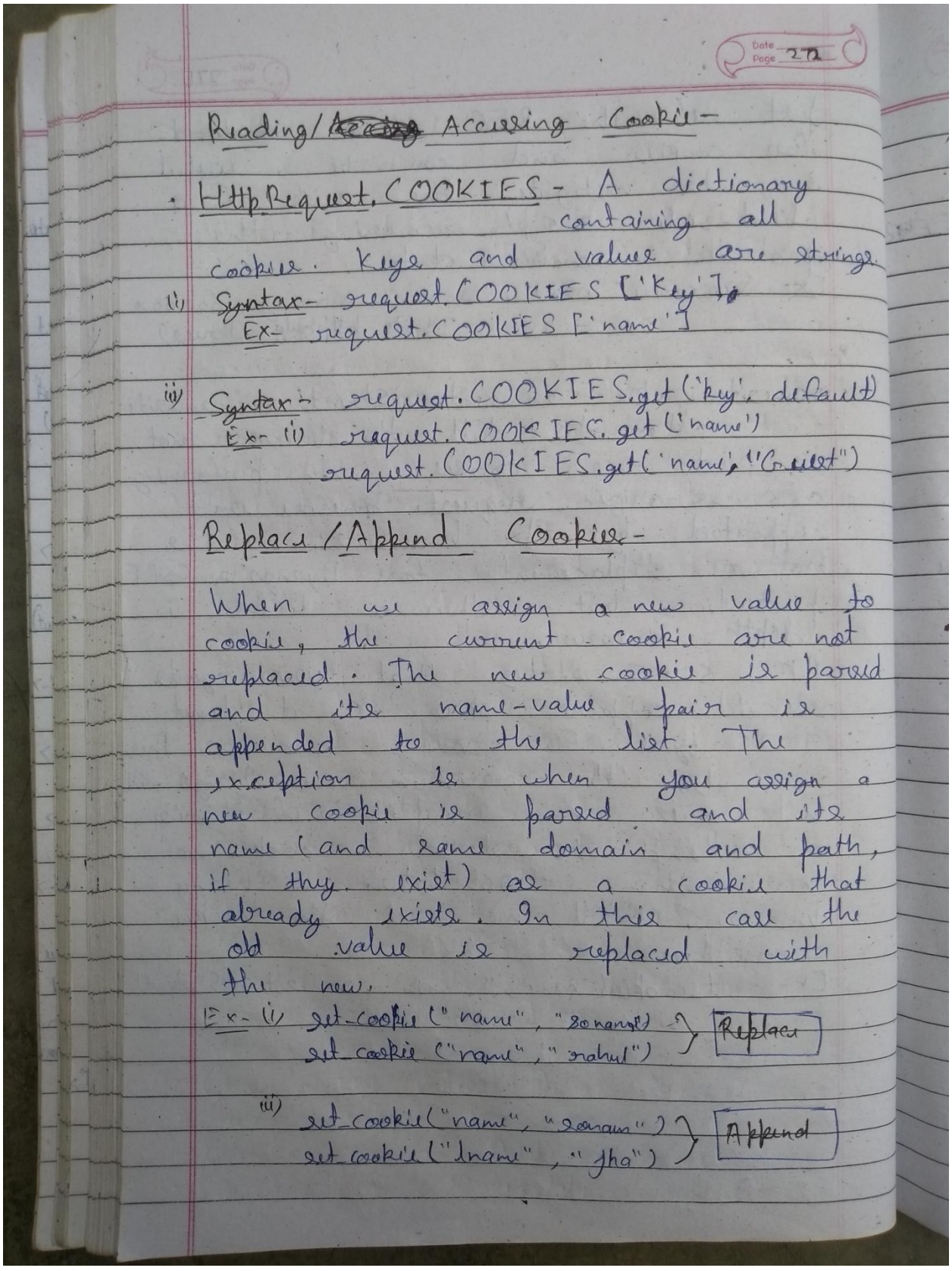


part of the REC 6265 standard for cookies and can be a useful way to mitigate the risk of a client-side script accessing the protected cookie data.

Ex- set_cookie("name", "sonam", max_age = 60*60*24*10, httponly = True)

- samesite- Use samesite = 'Strict' or samesite = 'Lax' to tell the browser not to send this cookie when performing a cross-origin request. SameSite isn't supported by all browsers, so it's not a replacement for Django's CSRF protection, but rather a defense in depth measure.
- REC 6265 states that user agents should support cookies of at least 4096 bytes. For many browsers this is also the maximum size. Django will not raise an exception if there's an attempt to store a cookie of more than 4096 bytes, but many browsers will not set the cookie correctly.

Ex- set_cookie("name", "sonam", samesite = "Strict")



Reading/Accessing Cookie -

- HttpRequest.COOKIES - A dictionary containing all cookie. Keys and values are strings.

i) Syntax - request.COOKIES['key']
Ex - request.COOKIES['name']

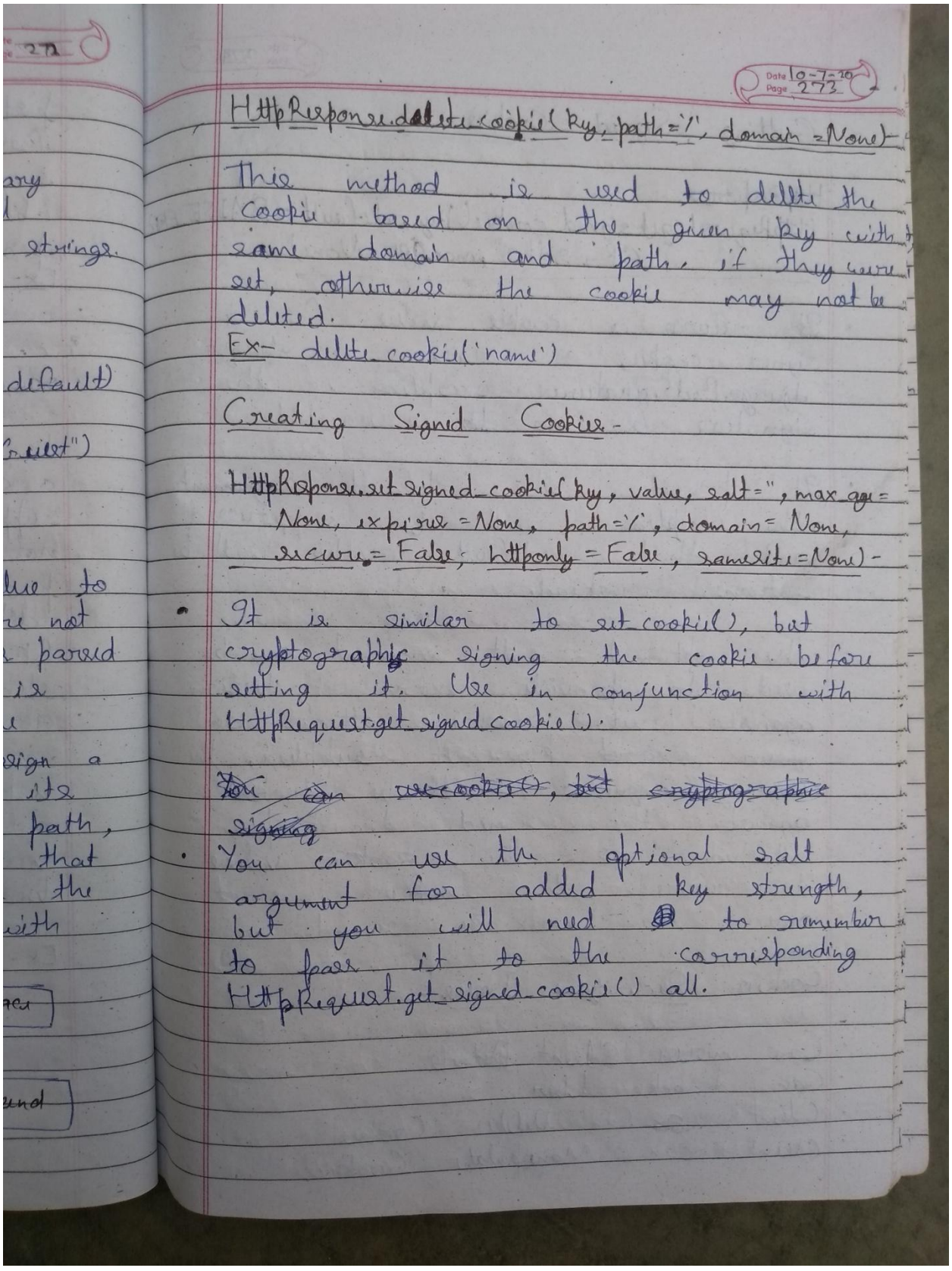
ii) Syntax - request.COOKIES.get('key', default)
Ex - i) request.COOKIES.get('name')
request.COOKIES.get('name', "Guest")

Replace / Append Cookies -

When we assign a new value to cookie, the current cookie are not replaced. The new cookie is parsed and its name-value pair is appended to the list. The exception is when you assign a new cookie is parsed and its name (and same domain and path, if they exist) as a cookie that already exists. In this case the old value is replaced with the new.

Ex - i) set_cookie("name", "sonam") } Replace
set_cookie("name", "rahul")

ii) set_cookie("name", "sonam") } Append
set_cookie("lname", "jha")



HttpResponse.delete_cookie(key, path='/', domain=None)

This method is used to delete the cookie based on the given key with the same domain and path, if they were set, otherwise the cookie may not be deleted.

Ex- delete_cookie('name')

Creating Signed Cookies -

HttpResponse.set_signed_cookie(key, value, salt='', max_age=None, expires=None, path='/', domain=None, secure=False, httponly=False, samesite=None)

- It is similar to set_cookie(), but cryptographic signing the cookie before setting it. Use in conjunction with HttpRequest.get_signed_cookie().

~~You can set_cookie(), but cryptographic signing~~

- You can use the optional salt argument for added key strength, but you will need to remember to pass it to the corresponding HttpRequest.get_signed_cookie() call.

Date Page 274

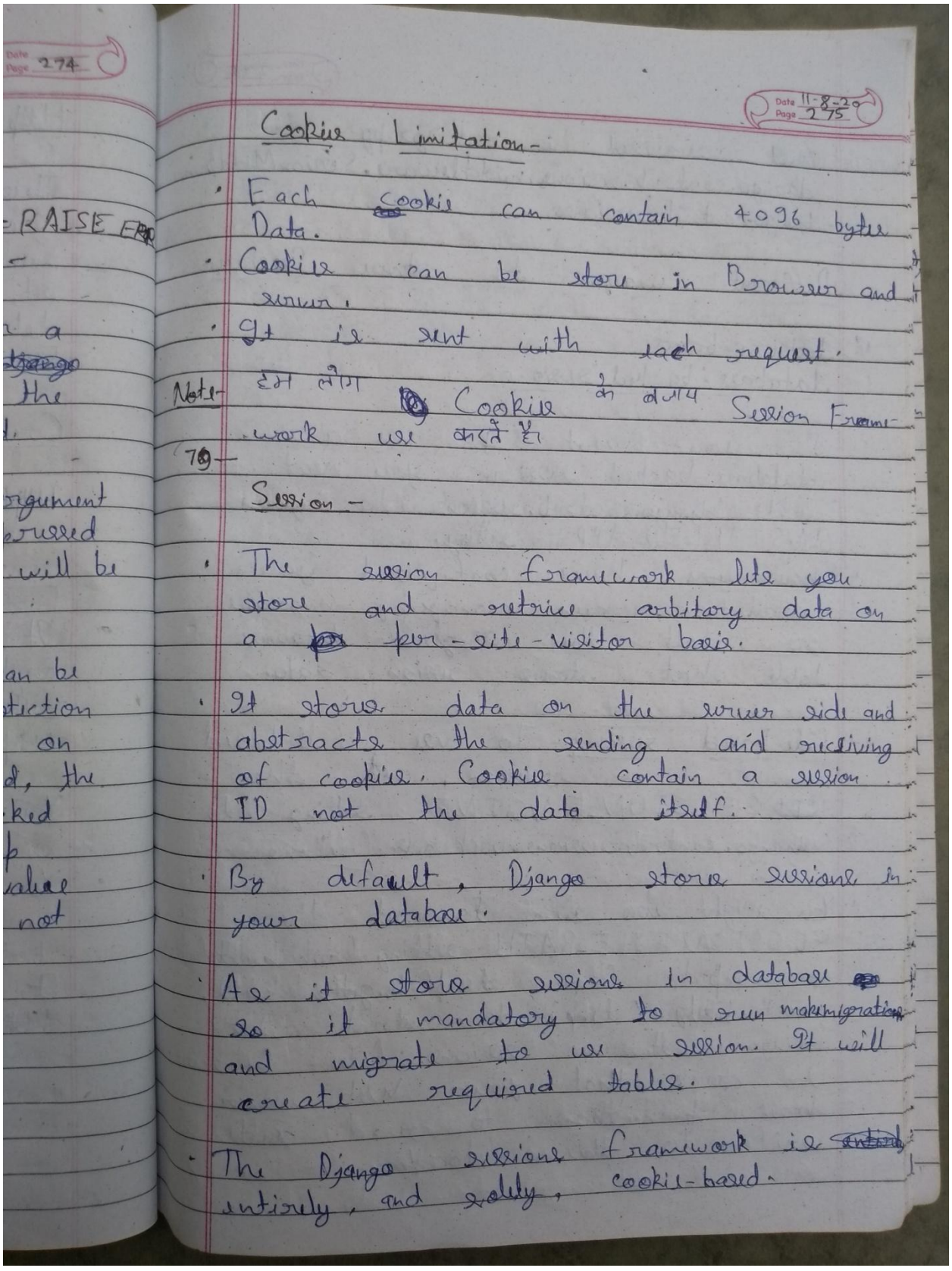
Getting Signed Cookies -

HttpRequest.get_signed_cookie(key, default=RAISE_E
OR, salt='', max_age=None) -

- It returns a cookie value for a signed cookie, or raise a django.BadSignature exception if the signature is no longer valid.
- It provide the default argument the exception will be suppressed and that default value will be returned instead.
- The optional salt argument can be used to provide extra protection against brute force attacks on your secret key. If supplied, the max_age argument will be checked against the signed timestamp attached to the cookie value to ensure the cookie is not older than max_age seconds.

Cookies Security Issues -

- Can misuse Client Details
- Can track User
- Client Can Delete Cookies
- Client can Manipulate Cookies



Cookie Limitation -

- Each cookie can contain 4096 bytes Data.
- Cookies can be store in Browser and server.
- It is sent with each request.

Note- हम लोग Cookie के बजाय Session Framework use करते है

79 -

Session -

- The session framework lets you store and retrive arbitary data on a per-site-visitor basis.
- It stores data on the server side and abstracts the sending and recieving of cookies. Cookies contain a session ID not the data itself.
- By default, Django stores sessions in your database.
- As it stores sessions in database so it mandatory to run makemigration and migrate to use session. It will create required tables.
- The Django sessions framework is entirely, and solely, cookie-based.

Date Page 276

. Must required in settings.py -
- django.contrib.sessions.middleware.SessionMiddleware
- django.contrib.sessions

Different ways to store Session -

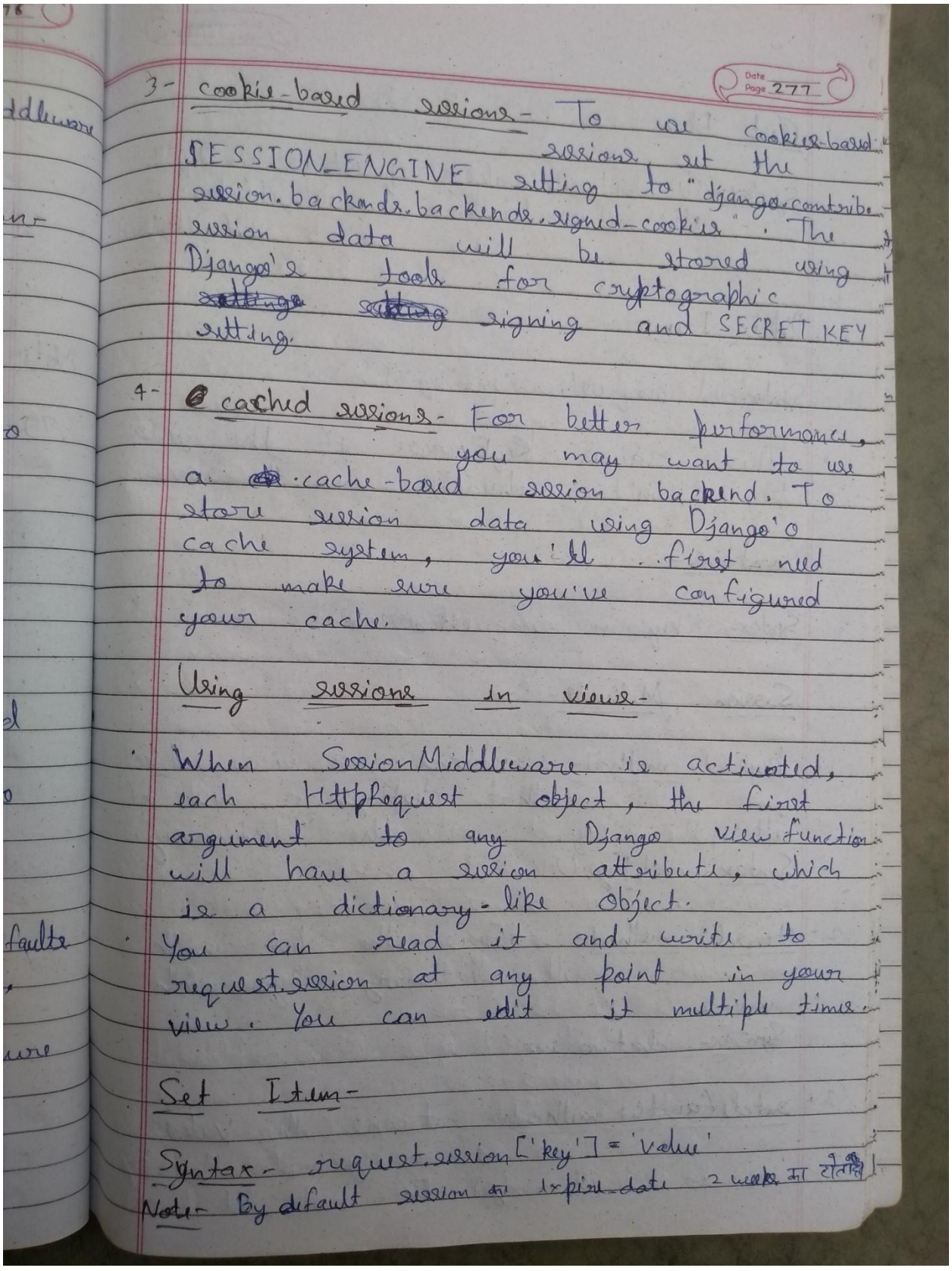
~~1. database-based~~

1- database-backed sessions

. If you want to use a database-backed session, you need to add 'django.contrib.sessions' to your INSTALLED_APPS setting.
. Once you have configured your installation, run manage.py migrate to install the single database table that stores session data.

2- file-based sessions - To use file-based session, set the SESSION_ENGINE setting to "django.contrib.sessions.backends.file".

. You might also want to set the SESSION_FILE_PATH setting (which defaults to input from tempfile.gettempdir(), most likely /tmp) to control where Django stores session files. Be sure to check that your Web server has permissions to read and write to this location.



3- cookie-based sessions- To use cookie-based sessions, set the SESSION_ENGINE setting to "django.contrib.sessions.backends.signed_cookies". The session data will be stored using Django's tools for cryptographic signing and SECRET KEY setting.

4- cached sessions- For better performance, you may want to use a cache-based session backend. To store session data using Django's cache system, you'll first need to make sure you've configured your cache.

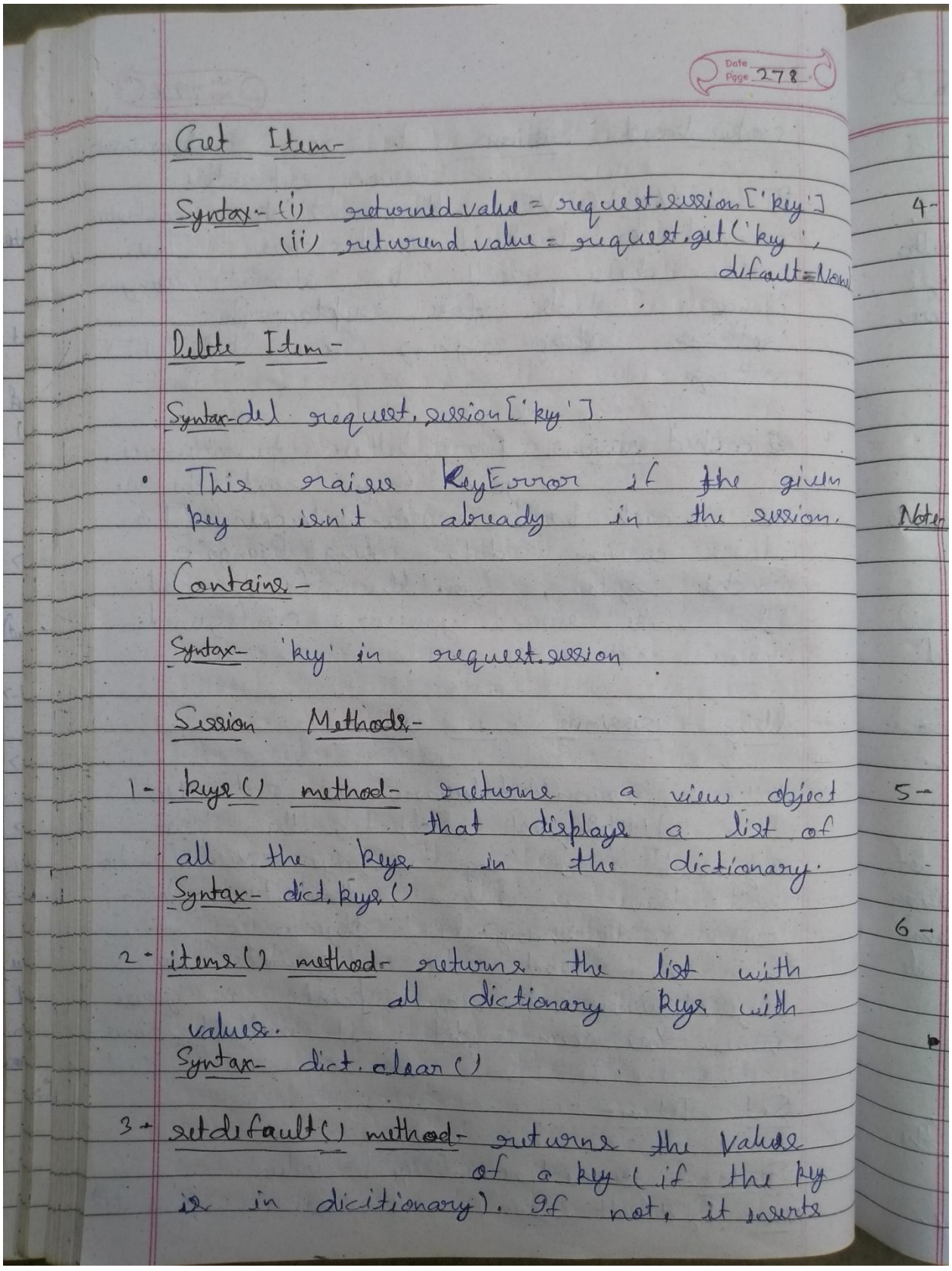
Using sessions in views-

- When SessionMiddleware is activated, each HttpRequest object, the first argument to any Django view function will have a session attribute, which is a dictionary-like object.
- You can read it and write to request.session at any point in your view. You can edit it multiple times.

Set Item-

Syntax- request.session['key'] = 'value'

Note- By default session का Expiry date 2 weeks का होता है।



Date
Page 278

Get Item-

Syntax- (i) returned_value = request.session['key']
(ii) returned value = request.get('key', default=None)

Delete Item-

Syntax- del request.session['key']

- This raises KeyError if the given key isn't already in the session.

Contains-

Syntax- 'key' in request.session

Session Methods-

1- keys() method- returns a view object that displays a list of all the keys in the dictionary.
Syntax- dict.keys()

2- items() method- returns the list with all dictionary keys with values.
Syntax- dict.clear()

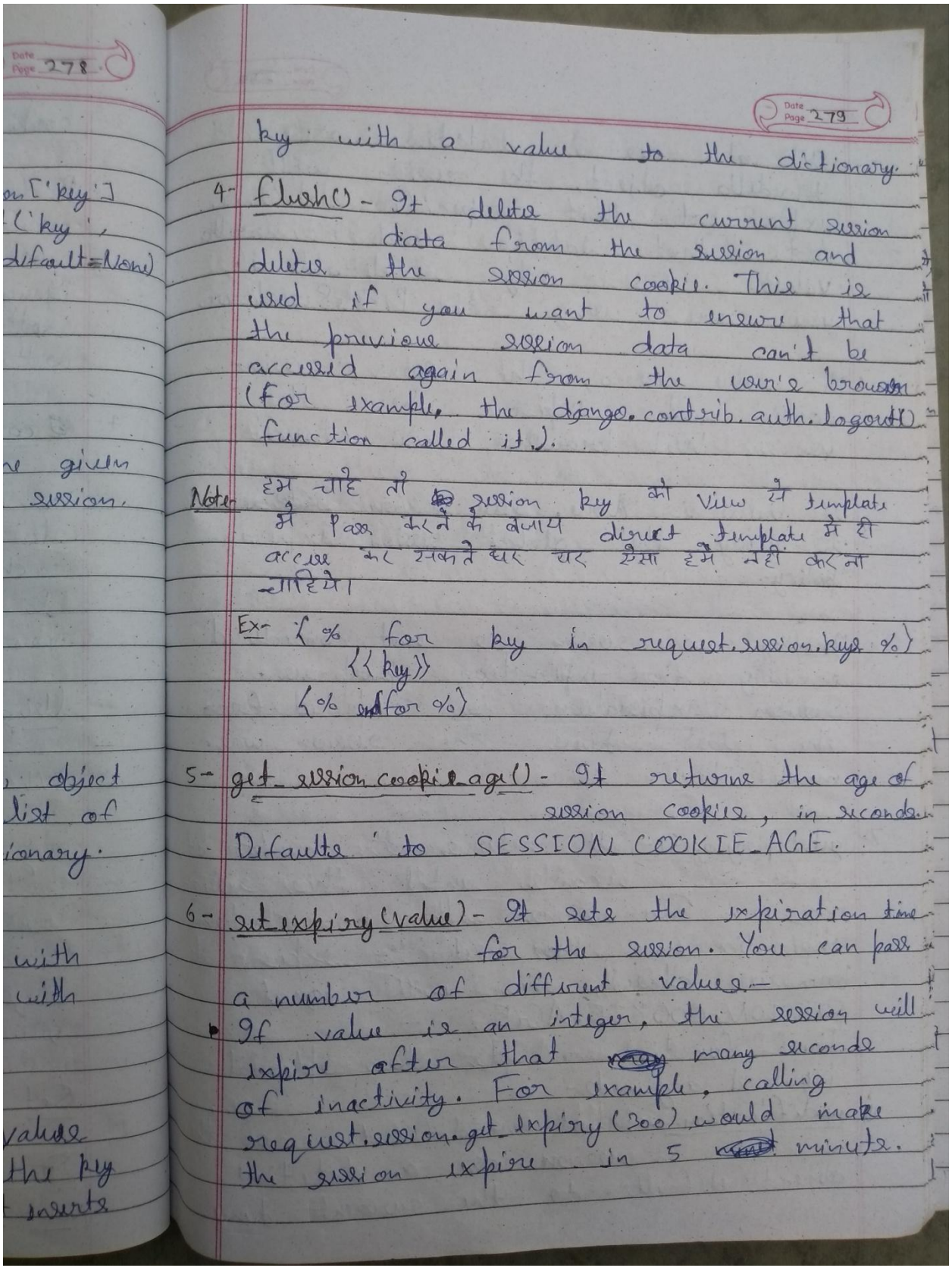
3- setdefault() method- returns the values of a key (if the key is in dicitionary). If not, it inserts

4-

Note

5-

6-



key with a value to the dictionary.

4- flush() - It delete the current session data from the session and delete the session cookie. This is used if you want to ensure that the previous session data can't be accessed again from the user's browser (For example, the django.contrib.auth.logout() function called it.).

Note- हम चाहे तो session key को view से template में pass करने के बजाय direct template में ही access कर सकते पर पर ऐसा हमें नहीं करना चाहिये।

Ex- {% for key in request.session.keys %}
{{key}}
{% endfor %}

5- get_session_cookie_age() - It returns the age of session cookie, in seconds. Defaults to SESSION_COOKIE_AGE.

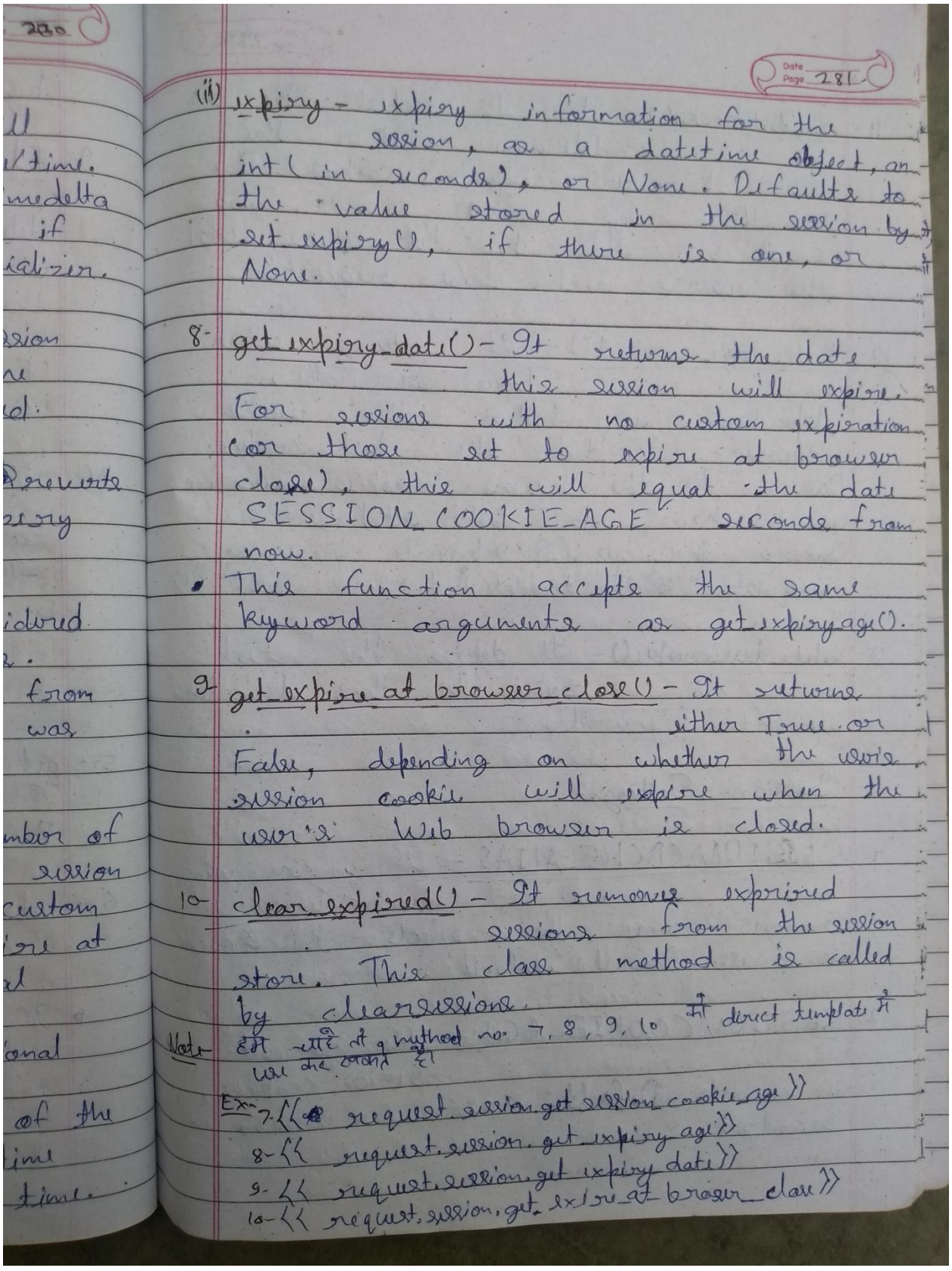
6- set_expiry(value) - It sets the expiration time for the session. You can pass a number of different values -
- If value is an integer, the session will expire after that many seconds of inactivity. For example, calling request.session.get_expiry(300) would make the session expire in 5 minute.

- If value is a datetime or timedelta object, the session will expire at that specific date/time. Note that datetime and timedelta values are only serializable if you are using the PickleSerializer.
- If value is 0, the user's session cookie will expire when the user's Web browser is closed.
- If value is None, the session reverts to using the global session expiry policy.
- Reading a session is not considered activity for expiration purposes. Session expiration is computed from the last time the session was modified.

7- get_expiry_age() - It returns the number of seconds until this session expires. For sessions with no custom expiration (or those set to expire at browser close), this will equal SESSION_COOKIE_AGE.

- This function accepts two optional keyword arguments-

(i) modification- last modification of the session, as a datetime object. Defaults to the current time.



(iii) expiry - expiry information for the session, as a datetime object, an int (in seconds), or None. Defaults to the value stored in the session by set_expiry(), if there is one, or None.

8- get_expiry_date() - It returns the date this session will expire. For sessions with no custom expiration (or those set to expire at browser close), this will equal the date SESSION_COOKIE_AGE seconds from now.

- This function accepts the same keyword arguments as get_expiry_age().

9- get_expire_at_browser_close() - It returns either True or False, depending on whether the user's session cookie will expire when the user's Web browser is closed.

10- clear_expired() - It removes expired sessions from the session store. This class method is called by clearsessions.

Note - हम यहाँ पर 7, 8, 9, 10 नं method को direct template में use कर सकते हैं।

Ex- 7. {{ request.session.get_session_cookie_age }}
8- {{ request.session.get_expiry_age }}
9- {{ request.session.get_expiry_date }}
10- {{ request.session.get_expire_at_browser_close }}

11- set_test_cookie() - It sets a test cookie to determine whether the user's browser supports cookies. Due to the way cookies work, you won't be able to test this until the user's next page request.

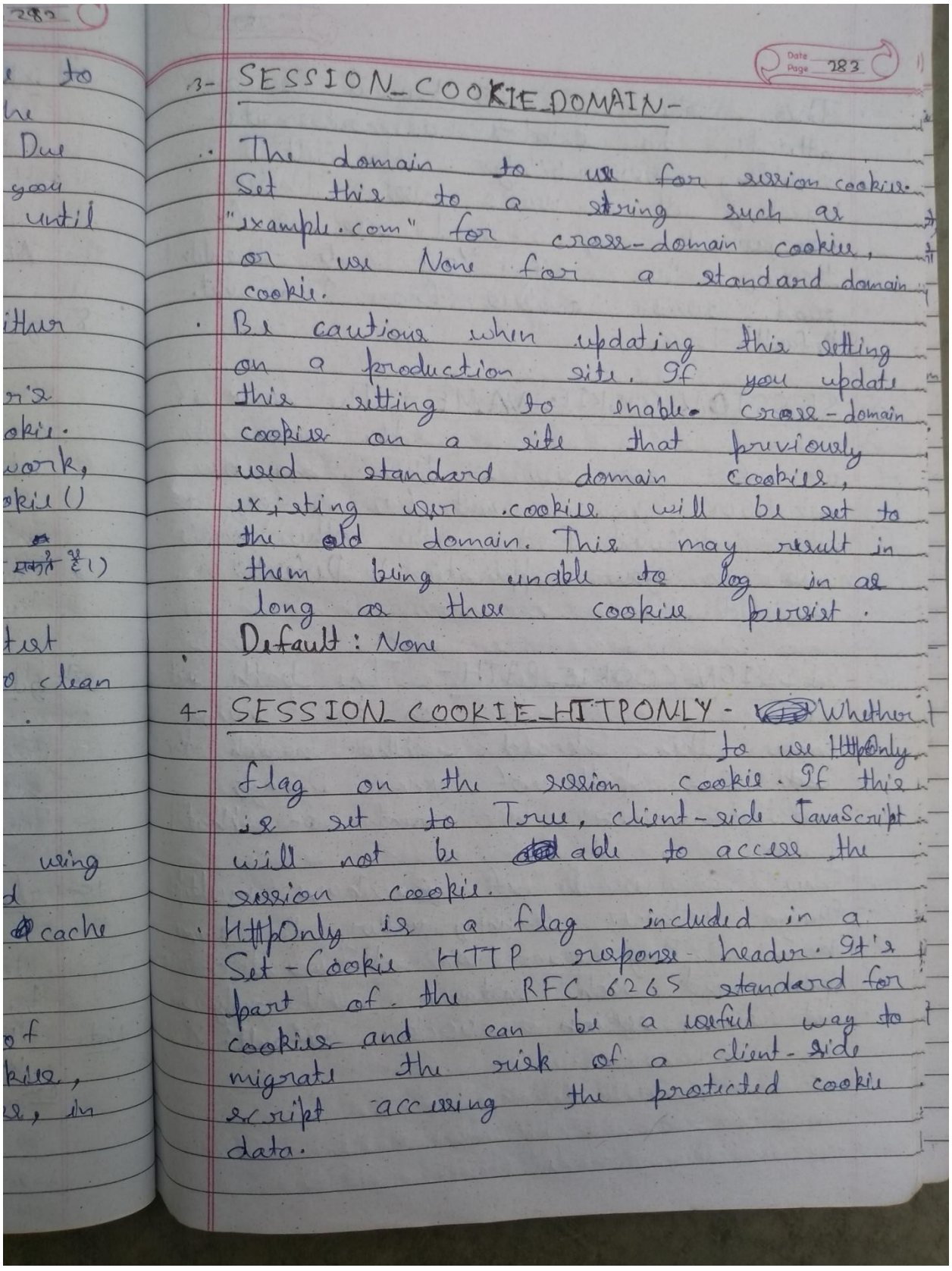
12- test_cookie_worked() - It returns either True or False, depending on whether the user's browser accepted the test cookie. Due to the way cookies work, you'll have to call set_test_cookie() on a previous, separate page request.

13- delete_test_cookie() - It deletes the test cookie. Use this to clean up after yourself.

Session Settings-

1- SESSION_CACHE_ALIAS - If you're using cache-based session storage, this selects the cache to use. Default: 'default'.

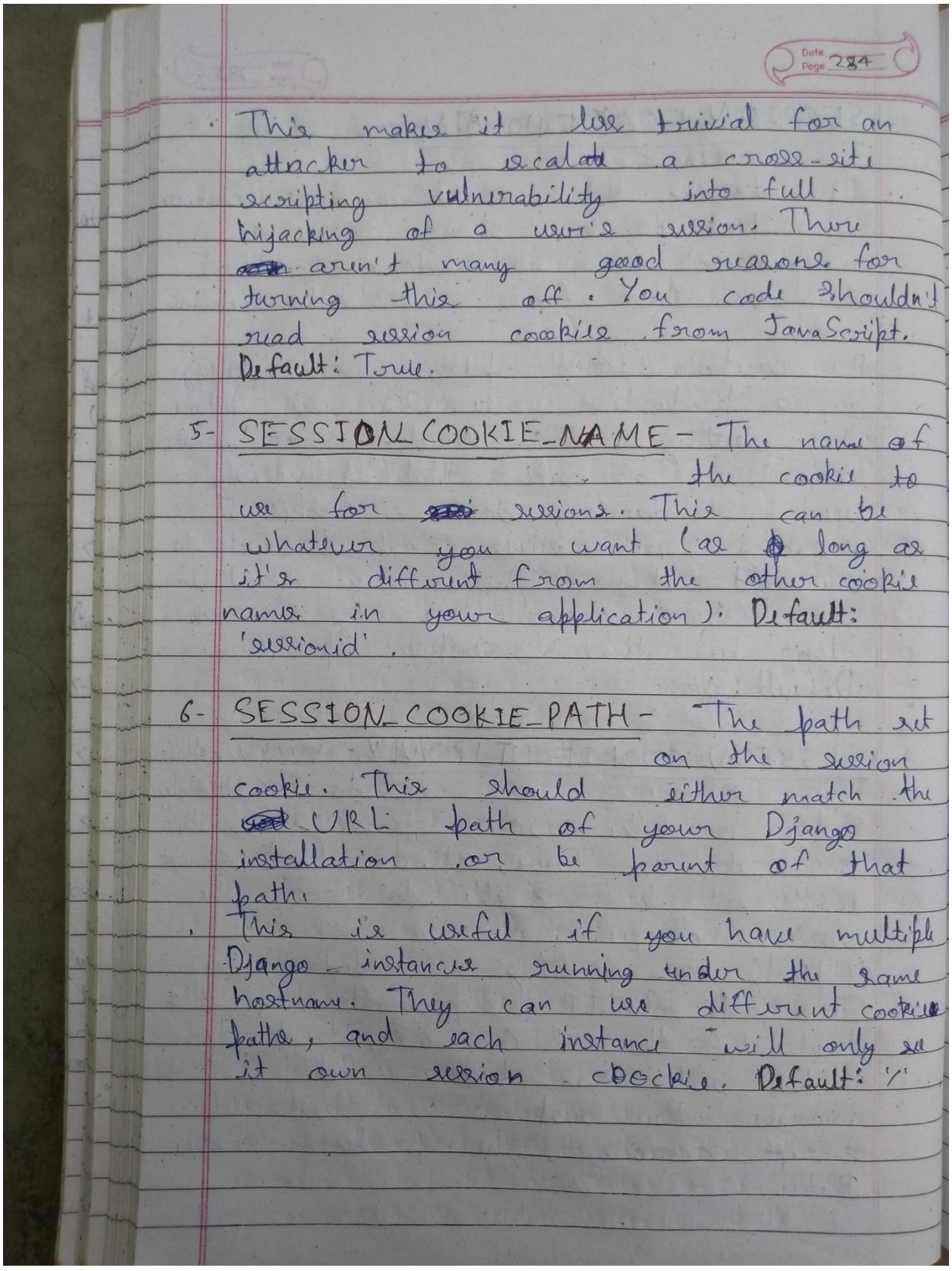
2- SESSION_COOKIE_AGE - The age of session cookies, in seconds. Default: 1209600 (2 weeks, in seconds).



3- SESSION_COOKIE_DOMAIN-

- The domain to use for session cookies. Set this to a string such as ".example.com" for cross-domain cookies, or use None for a standard domain cookie.
- Be cautious when updating this setting on a production site. If you update this setting to enable cross-domain cookies on a site that previously used standard domain cookies, existing user cookies will be set to the old domain. This may result in them being unable to log in as long as these cookies persist.
- Default: None

4- SESSION_COOKIE_HTTPONLY - Whether to use HttpOnly flag on the session cookie. If this is set to True, client-side JavaScript will not be able to access the session cookie.
- HttpOnly is a flag included in a Set-Cookie HTTP response header. It's part of the RFC 6265 standard for cookies and can be a useful way to mitigate the risk of a client-side script accessing the protected cookie data.



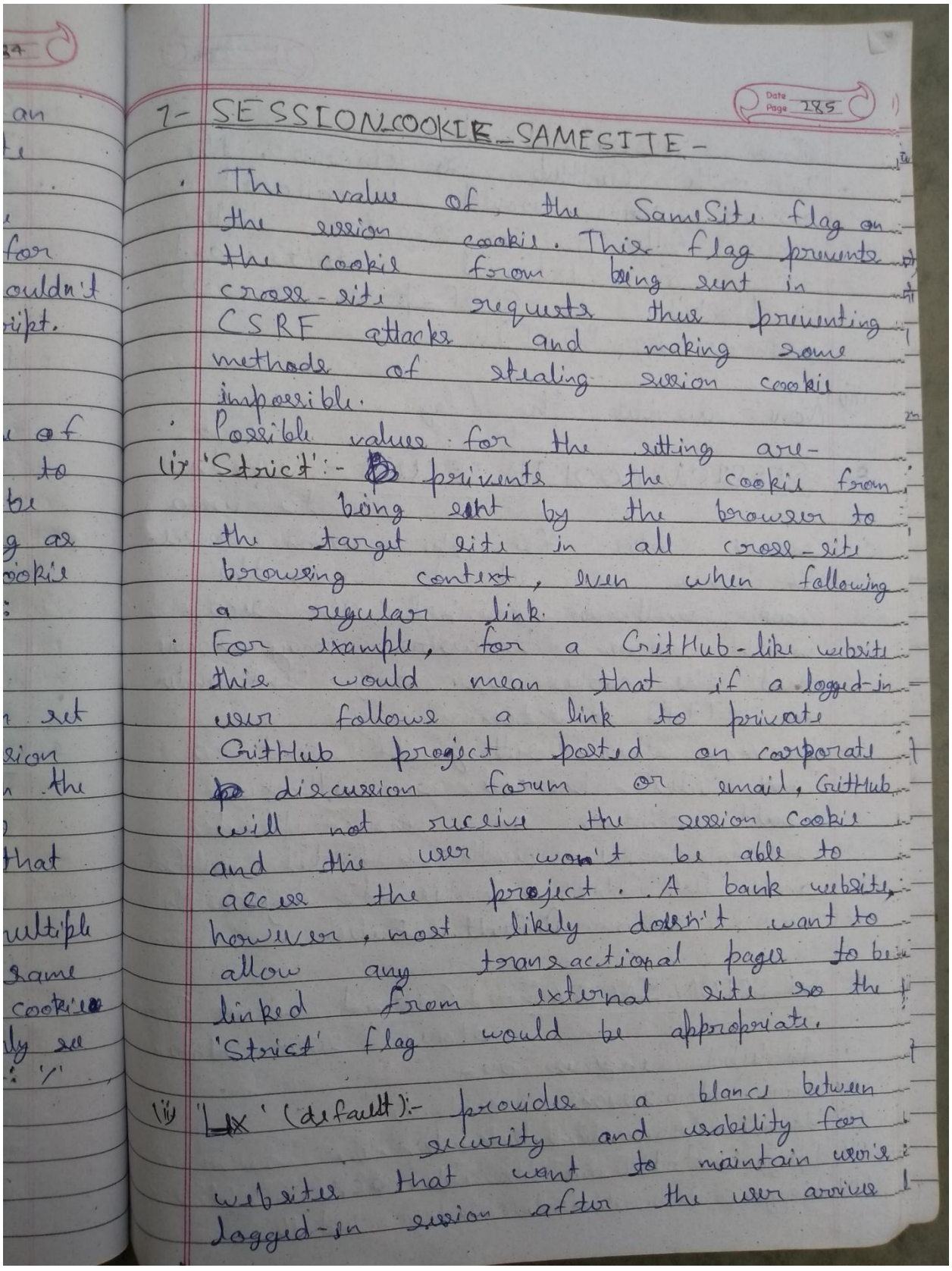
Date Page 284

This makes it less trivial for an attacker to escalate a cross-site scripting vulnerability into full hijacking of a user's session. There aren't many good reasons for turning this off. You code shouldn't read session cookies from JavaScript. Default: True.

5- SESSION_COOKIE_NAME - The name of the cookie to use for sessions. This can be whatever you want (as long as it's different from the other cookie names in your application). Default: 'sessionid'.

6- SESSION_COOKIE_PATH - The path set on the session cookie. This should either match the URL path of your Django installation or be parent of that path.

This is useful if you have multiple Django instances running under the same hostname. They can use different cookie paths, and each instance will only see it own session cookie. Default: '/'

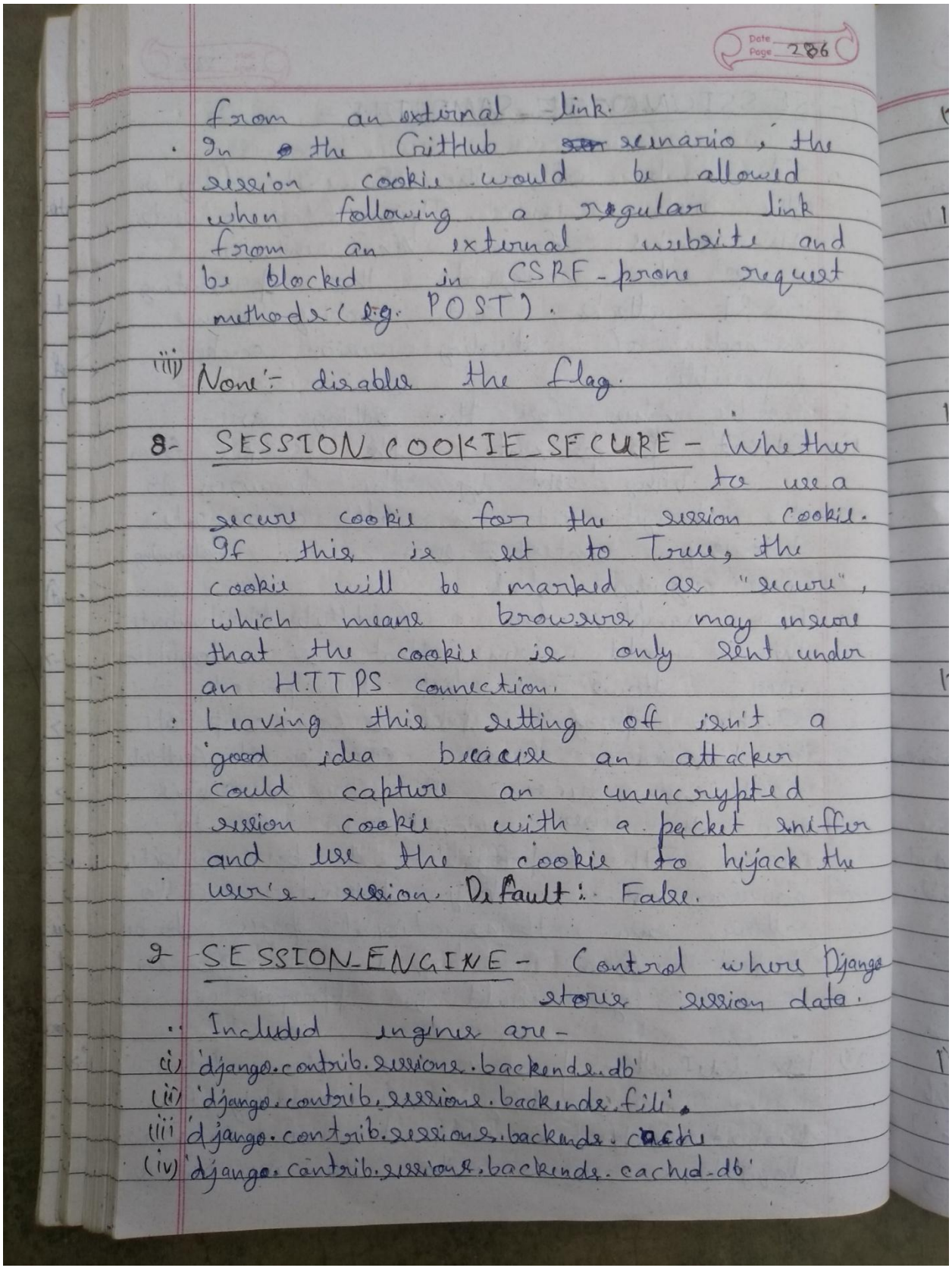


7- SESSION_COOKIE_SAMESITE -

- The value of the SameSite flag on the session cookie. This flag prevents the cookie from being sent in cross-site requests thus preventing CSRF attacks and making some methods of stealing session cookie impossible.
- Possible values for the setting are -

(i) 'Strict' :- prevents the cookie from being sent by the browser to the target site in all cross-site browsing context, even when following a regular link.
- For example, for a GitHub-like website this would mean that if a logged-in user follows a link to private GitHub project posted on corporate discussion forum or email, GitHub will not receive the session cookie and the user won't be able to access the project. A bank website, however, most likely doesn't want to allow any transactional page to be linked from external site so the 'Strict' flag would be appropriate.

(ii) 'Lax' (default) :- provides a blance between security and usability for websites that want to maintain user's logged-in session after the user arrives



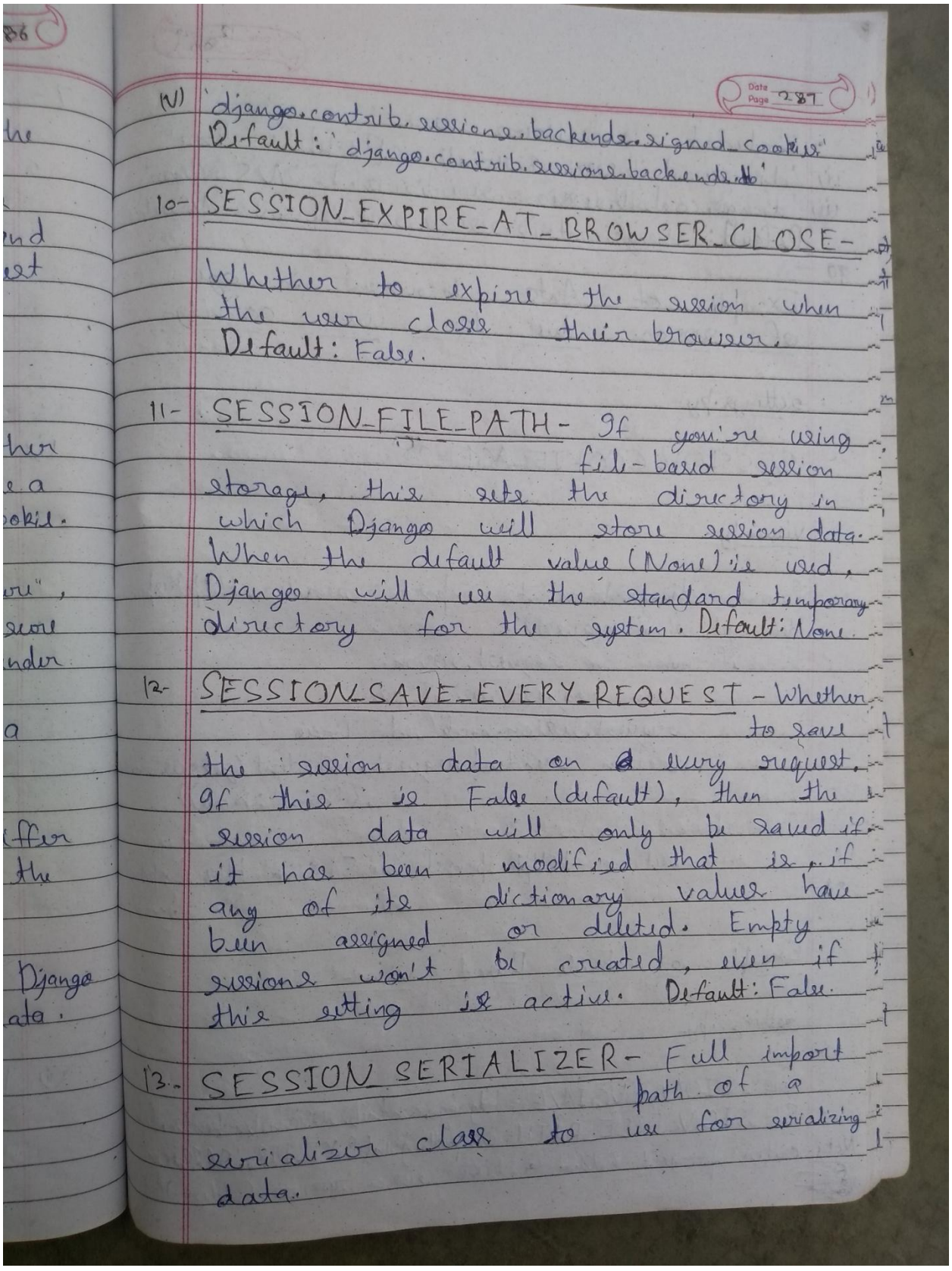
from an external link.
- In the GitHub scenario, the session cookie would be allowed when following a regular link from an external website and be blocked in CSRF-prone request methods (e.g. POST).

(iii) None:- disable the flag.

8- SESSION_COOKIE_SECURE - Whether to use a secure cookie for the session cookie. If this is set to True, the cookie will be marked as "secure", which means browsers may ensure that the cookie is only sent under an HTTPS connection.
- Leaving this setting off isn't a good idea because an attacker could capture an unencrypted session cookie with a packet sniffer and use the cookie to hijack the user's session. Default: False.

9- SESSION_ENGINE - Control where Django stores session data.
- Included engines are -

(i) 'django.contrib.sessions.backends.db'
(ii) 'django.contrib.sessions.backends.file'.
(iii) 'django.contrib.sessions.backends.cache'
(iv) 'django.contrib.sessions.backends.cached_db'



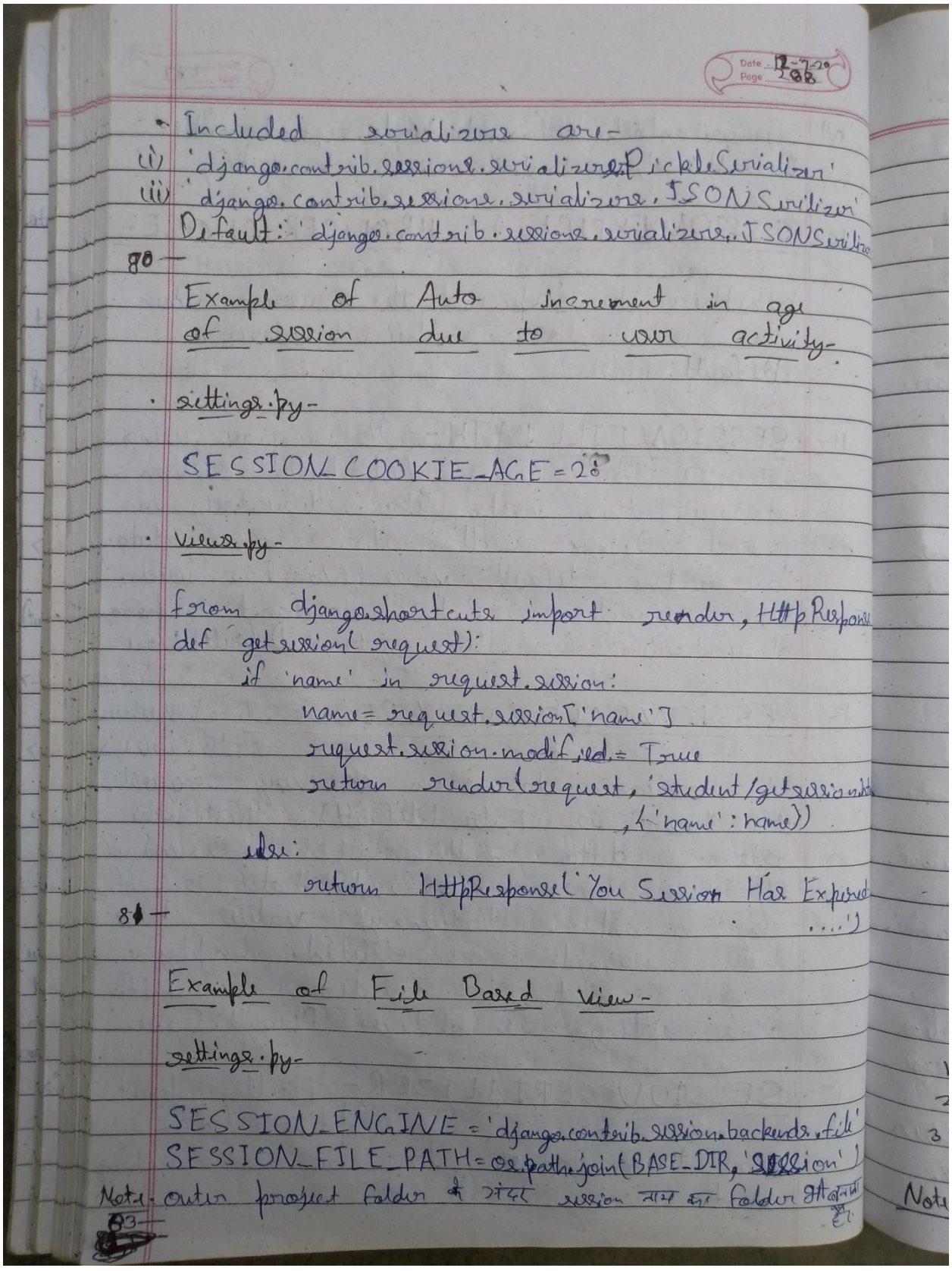
(v) 'django.contrib.sessions.backends.signed_cookies'
Default: 'django.contrib.sessions.backends.db'

10- SESSION_EXPIRE_AT_BROWSER_CLOSE- Whether to expire the session when the user closes their browser. Default: False.

11- SESSION_FILE_PATH- If you're using file-based session storage, this sets the directory in which Django will store session data. When the default value (None) is used, Django will use the standard temporary directory for the system. Default: None.

12- SESSION_SAVE_EVERY_REQUEST- Whether to save the session data on every request. If this is False (default), then the session data will only be saved if it has been modified that is, if any of its dictionary values have been assigned or deleted. Empty sessions won't be created, even if this setting is active. Default: False.

13- SESSION_SERIALIZER- Full import path of a serializer class to use for serializing data.



- Included serializers are-
(i) 'django.contrib.sessions.serializers.PickleSerializer'
(ii) 'django.contrib.sessions.serializers.JSONSerializer'
Default: 'django.contrib.sessions.serializers.JSONSerializer'

80 —

Example of Auto increment in age of session due to user activity-

- settings.py-

SESSION_COOKIE_AGE = 20

- views.py-

```
from django.shortcuts import render, HttpResponse
def getsession(request):
    if 'name' in request.session:
        name = request.session['name']
        request.session.modified = True
        return render(request, 'student/getsession.html'
                      ,{'name': name})
    else:
        return HttpResponse('Your Session Has Expired
                            ....')
```

81 —

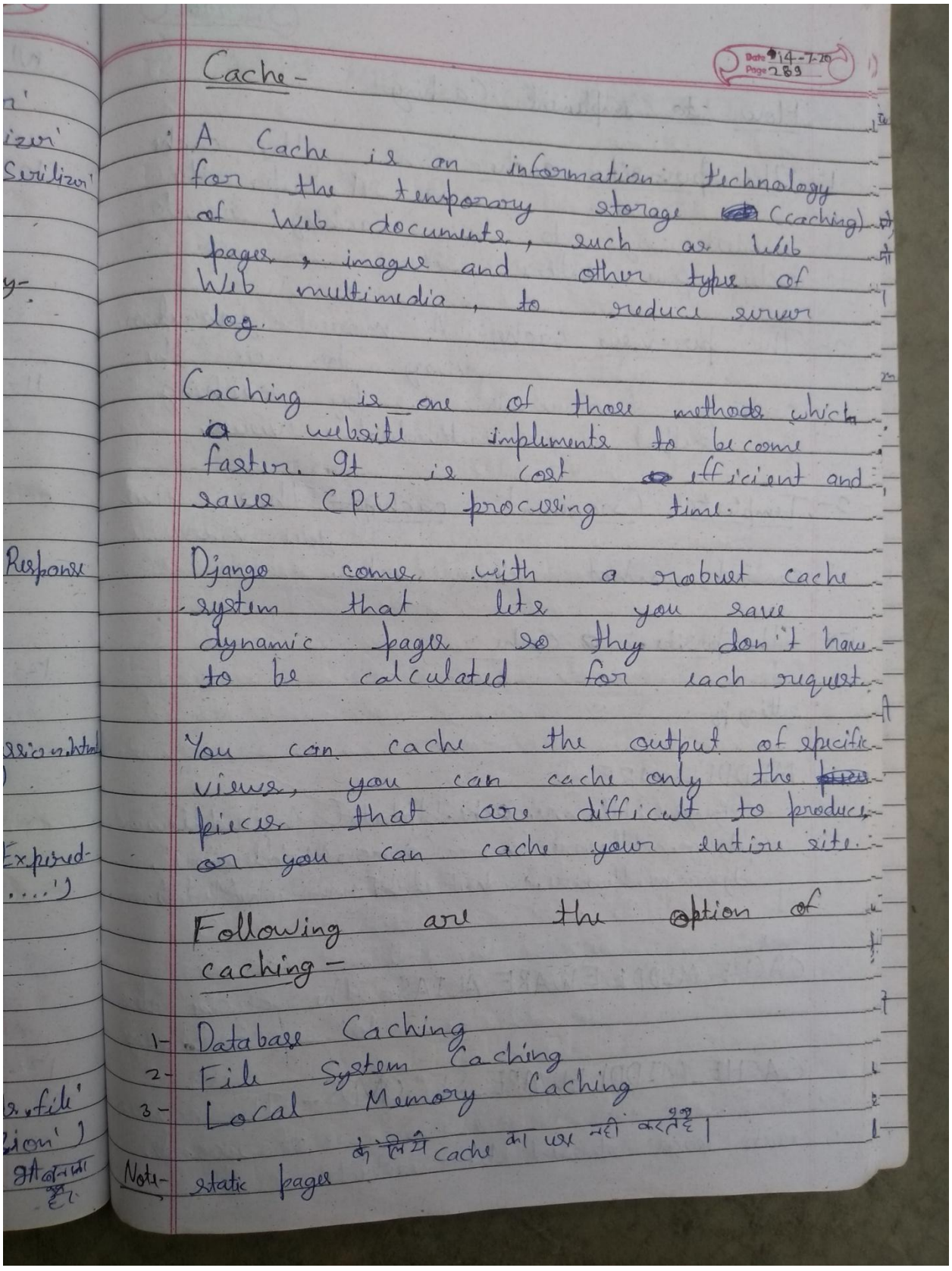
Example of File Based View-

settings.py-

```
SESSION_ENGINE = 'django.contrib.sessions.backends.file'
SESSION_FILE_PATH = os.path.join(BASE_DIR, 'session')
```

Note- outer project folder में session नाम का folder होना चाहिए

83 —



Cache -

A Cache is an information technology for the temporary storage (caching) of Web documents, such as Web pages, image and other type of Web multimedia, to reduce server log.

Caching is one of those methods which a website implements to become faster. It is cost efficient and saves CPU processing time.

Django comes with a robust cache system that lets you save dynamic pages so they don't have to be calculated for each request.

You can cache the output of specific views, you can cache only the pieces that are difficult to produce, or you can cache your entire site.

Following are the option of caching -

1- Database Caching
2- File System Caching
3- Local Memory Caching

Note- static pages को लिये cache का use नहीं करते हैं।

Date Page 290

How to implement Caching-

1- The per-site cache- Once the cache is set up, the simplest way to use caching is to cache your entire site.

2- The per-view cache- A more granular way to use the caching framework is by caching the output of individual views.

3- Template fragment caching- This gives you more control what to cache.

1-> The per site cache -

setting.py-

```
MIDDILWARE = [
    'django.middleware.cache.UpdateCacheMiddleware',
    'django.middleware.common.CommonMiddleware',
    'django.middleware.cache.FetchFromCacheMiddleware',
]
```

CACHE_MIDDLEWARE_ALIAS- The cache alias to use for storage

CACHE_MIDDLEWARE_SECONDS- The number of seconds each page should be cached.

CACHE_MIDDLEWARE_KEY_PREFIX - If the cache is shared across multiple sites using the same Django installation, set this to the name of the site, or some other string that is unique to this Django instance, to prevent key collisions. Use an empty string if you don't care.

Note - setting.py में MIDDLWARE = [ ] की value का order same रहना चाहिये।

Options of caching -

(i) Database Caching -

Django can store its cached data in your database. This works best if you've got a fast, well-indexed database server.

settings.py -

```
CACHES = {
    'default': {
        'BACKED': 'django.core.cache.backends.db.DatabaseCache',
        'LOCATION': 'my_cache_table',
    }
}
```

The name of the database table. This name can be whatever you want, as long as it's a valid table name that's not already being used in your database.

Date Page 292

- Before using the database cache, you must create the cache table with this command -
  python manage.py createcachetable

- This creates a table in your database that is in the proper format that Django's database-cache system expects.

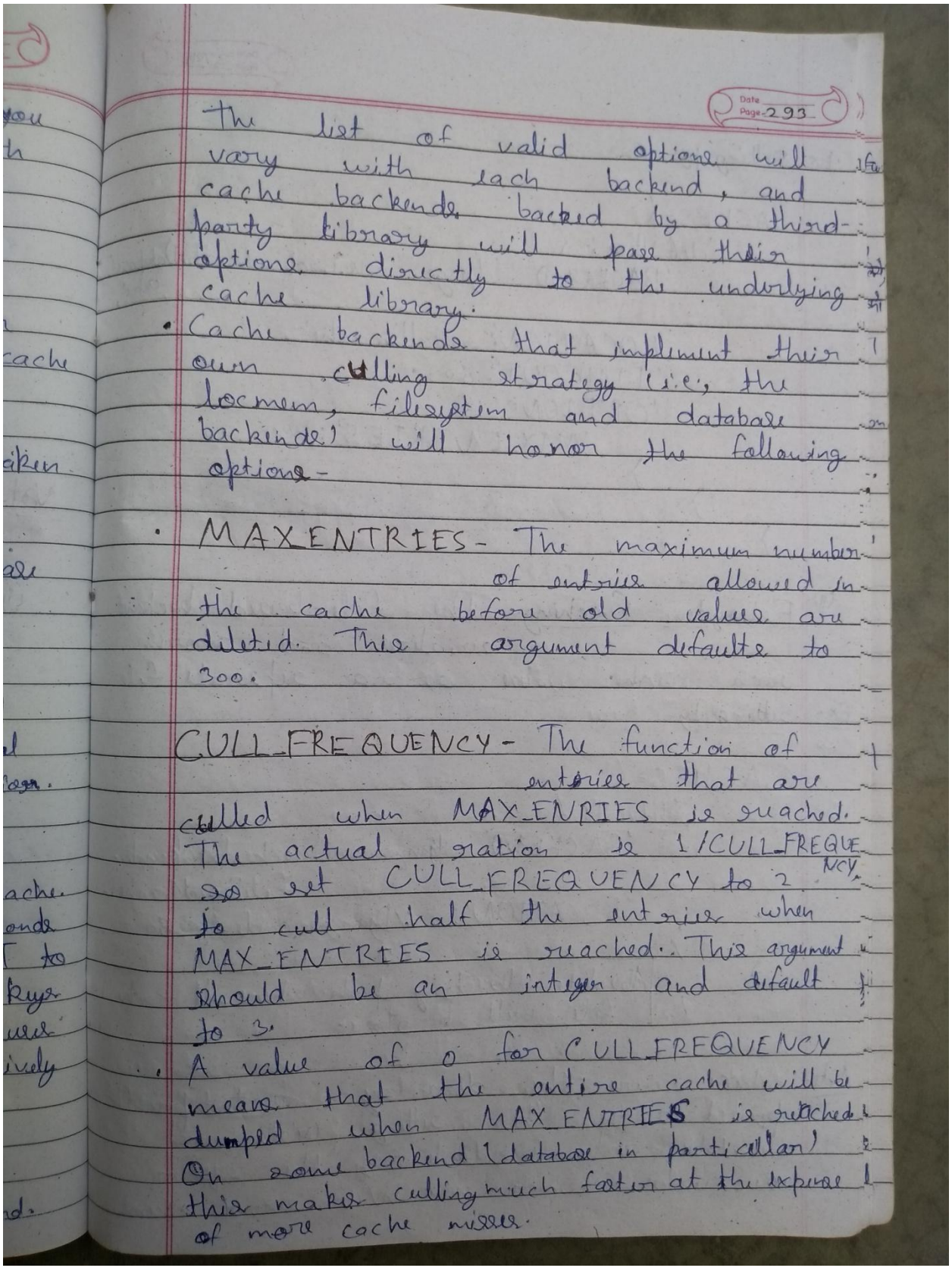
- The name of the table is taken from LOCATION.

- If you are using multiple database caches, createcachetable creates one table for each cache.

Cache Arguments - Each cache backend can be given additional arguments to control caching behavior.

- TIMEOUT - The default timeout, in seconds, to use for the cache. This argument defaults to 300 seconds (5 minutes). You can set TIMEOUT to None so that, by default, cache keys never expire. A value of 0 causes keys to immediately expire (effectively "don't cache").

- OPTIONS - Any options that should be passed to the cache backend.



The list of valid options will vary with each backend, and cache backends backed by a third-party library will pass their options directly to the underlying cache library.

- Cache backends that implement their own culling strategy (i.e., the locmem, filesystem and database backends) will honor the following options -

- MAX_ENTRIES - The maximum number of entries allowed in the cache before old values are deleted. This argument defaults to 300.

CULL_FREQUENCY - The function of entries that are culled when MAX_ENRIES is reached. The actual ration is 1/CULL_FREQUENCY. So set CULL_FREQUENCY to 2 to cull half the entries when MAX_ENTRIES is reached. This argument should be an integer and default to 3.

- A value of 0 for CULL_FREQUENCY means that the entire cache will be dumped when MAX_ENTRIES is reached. On some backend (database in particular) this makes culling much faster at the expense of more cache misses.

Date Page 294

Ex- settings.py

```
CACHES={
    'default': {
        'BACKEND': 'django.core.backends.db.DatabaseCache',
        'LOCATION': 'enroll_cache',
        'TIMEOUT': 60,
        'OPTIONS':{
            'MAX_ENTRIES': 1000
        }
    }
}
```

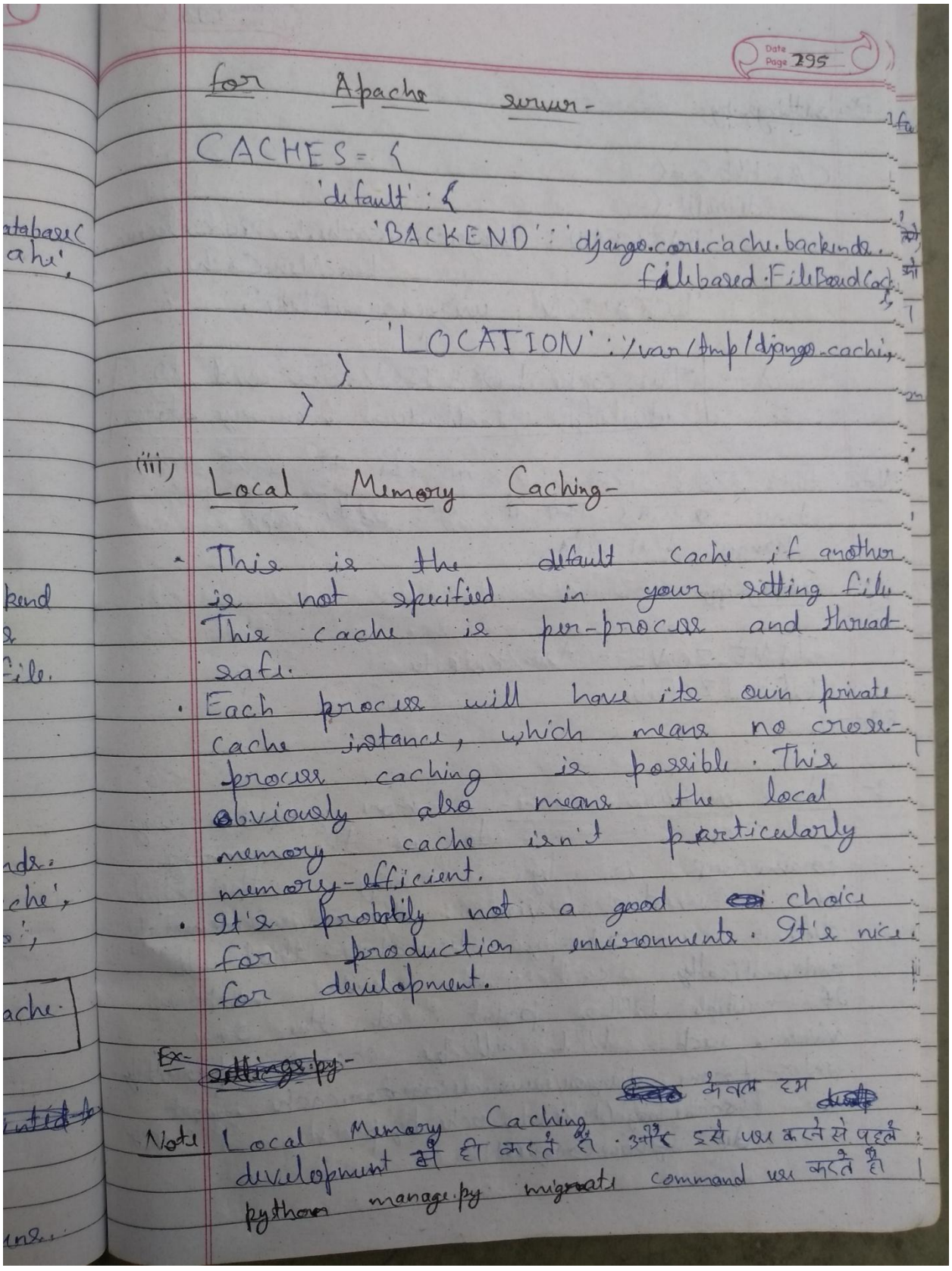
(ii) Filesystem Caching- The file-based backend serializes and stores each cache value as a separate file.

Ex- settings.py

```
CACHES={
    'default': {
        'BACKEND': 'django.core.cache.backends.filiBaudCache',
        'LOCATION': 'c:/Django code/go.80',
    }
}
```

Absolute directory path where all cache file will be store.

Note- Make sure the directory ~~[illegible]~~ pointed to by this setting exists and readable and writable by the system user under which your Web server runs.



for Apache server -

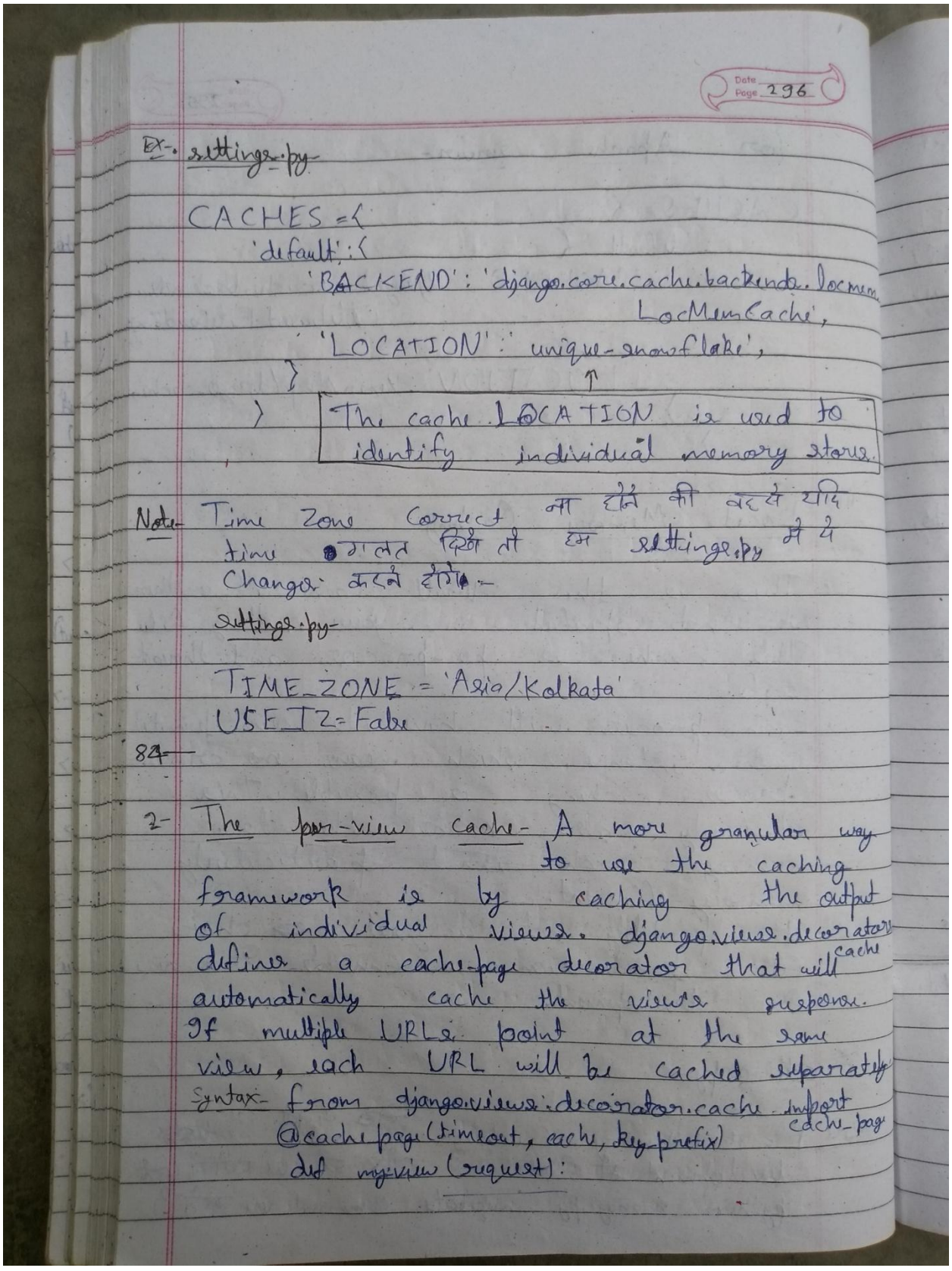
```
CACHES = {
    'default': {
        'BACKEND': 'django.core.cache.backends.
                    filebased.FileBasedCache',
        'LOCATION': '/var/tmp/django_cache',
    }
}
```

(iii) Local Memory Caching -

- This is the default cache if another is not specified in your setting file. This cache is per-process and thread-safe.
- Each process will have its own private cache instance, which means no cross-process caching is possible. This obviously also means the local memory cache isn't particularly memory-efficient.
- It's probably not a good choice for production environments. It's nice for development.

Ex- ~~settings.py~~ -

Note Local Memory Caching development के ही करते हैं। और इसे use करने से पहले python manage.py migrate command use करते हैं।



Date Page 296

Ex- settings.py

```
CACHES = {
    'default': {
        'BACKEND': 'django.core.cache.backends.locmem.
                    LocMemCache',
        'LOCATION': 'unique-snowflake',
    }
}
```

The cache LOCATION is used to identify individual memory stores.

Note- Time Zone Correct न होने की दशा में time गलत दिखे तो हम settings.py में ये Changes करने होंगे -

settings.py

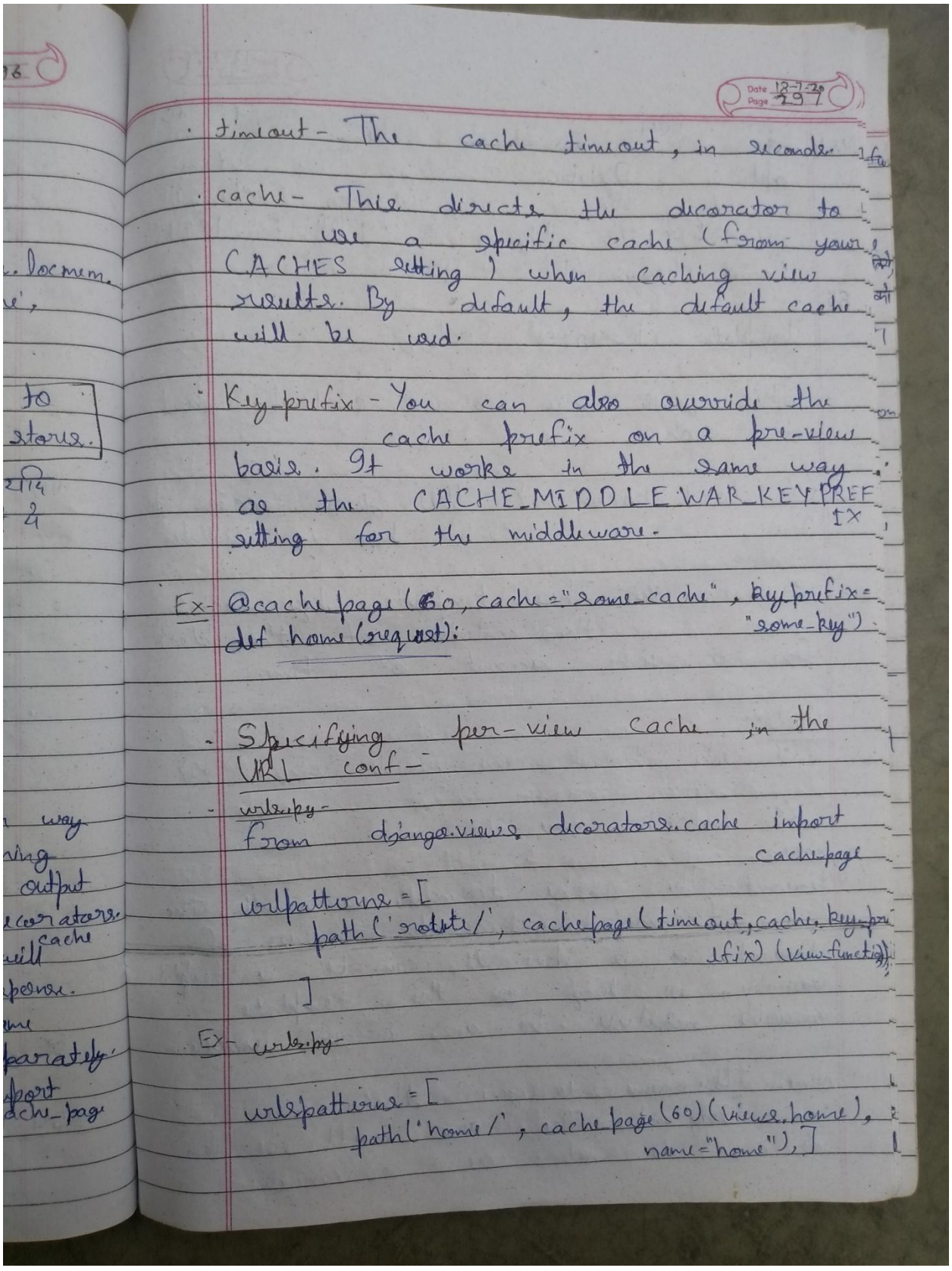
```
TIME_ZONE = 'Asia/Kolkata'
USE_TZ = False
```

84-

2- The per-view cache- A more granular way to use the caching framework is by caching the output of individual views. django.views.decorators.cache define a cache_page decorator that will automatically cache the views response. If multiple URLs point at the same view, each URL will be cached separately

Syntax- from django.views.decorator.cache import cache_page

```
@cache_page(timeout, cache, key_prefix)
def my_view(request):
```



- timeout - The cache timeout, in seconds.
- cache - This directs the decorator to use a specific cache (from your CACHES setting) when caching view results. By default, the default cache will be used.
- Key_prefix - You can also override the cache prefix on a per-view basis. It works in the same way as the CACHE_MIDDLEWARE_KEY_PREFIX setting for the middleware.

Ex- @cache_page(60, cache="some_cache", key_prefix="some_key")
def home(request):

- Specifying per-view cache in the URL conf -
- urls.py -

```
from django.views.decorators.cache import cache_page

urlpatterns = [
    path('route/', cache_page(timeout, cache, key_prefix)(view_function)),
]
```

Ex- urls.py -

```
urlspatterns = [
    path('home/', cache_page(60)(views.home), name="home"),
]
```

Date 19-7-20
Page 298

Note- इसके लिये हमें caching के वही तीन options Database caching, File System caching और Local Memory Caching! इस तरीके में हमे settings.py में MIDDILEWARE=[] में कोई changes नहीं करना पड़ता है।

85-

Template Fragment Caching-

This gives you more control what to cache.

{% load cache %} - This gives access to cache tag in template.

{% cache %} - This template tag caches the contents of the block for a given amount of time.

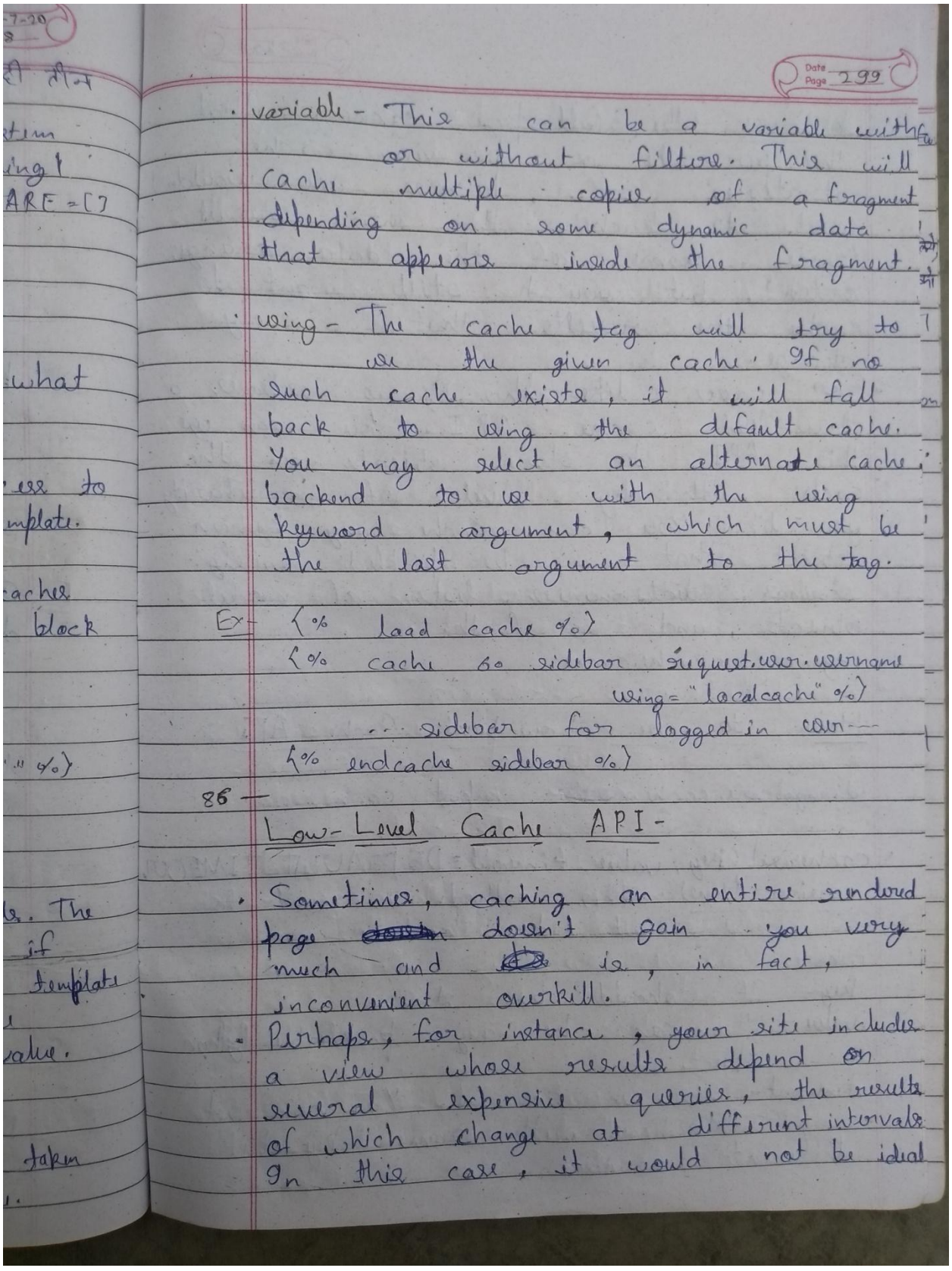
Syntax-

{% cache timeout name variable using="." %}

{% endcache name %}

where-

- timeout - The cache timeout, in seconds. The fragment is cached forever if timeout is None. It can be a template variable, as long as the template variable resolves to an integer value.
- name - The name to give the cache fragment. The name will be taken as it, do not use a variable.



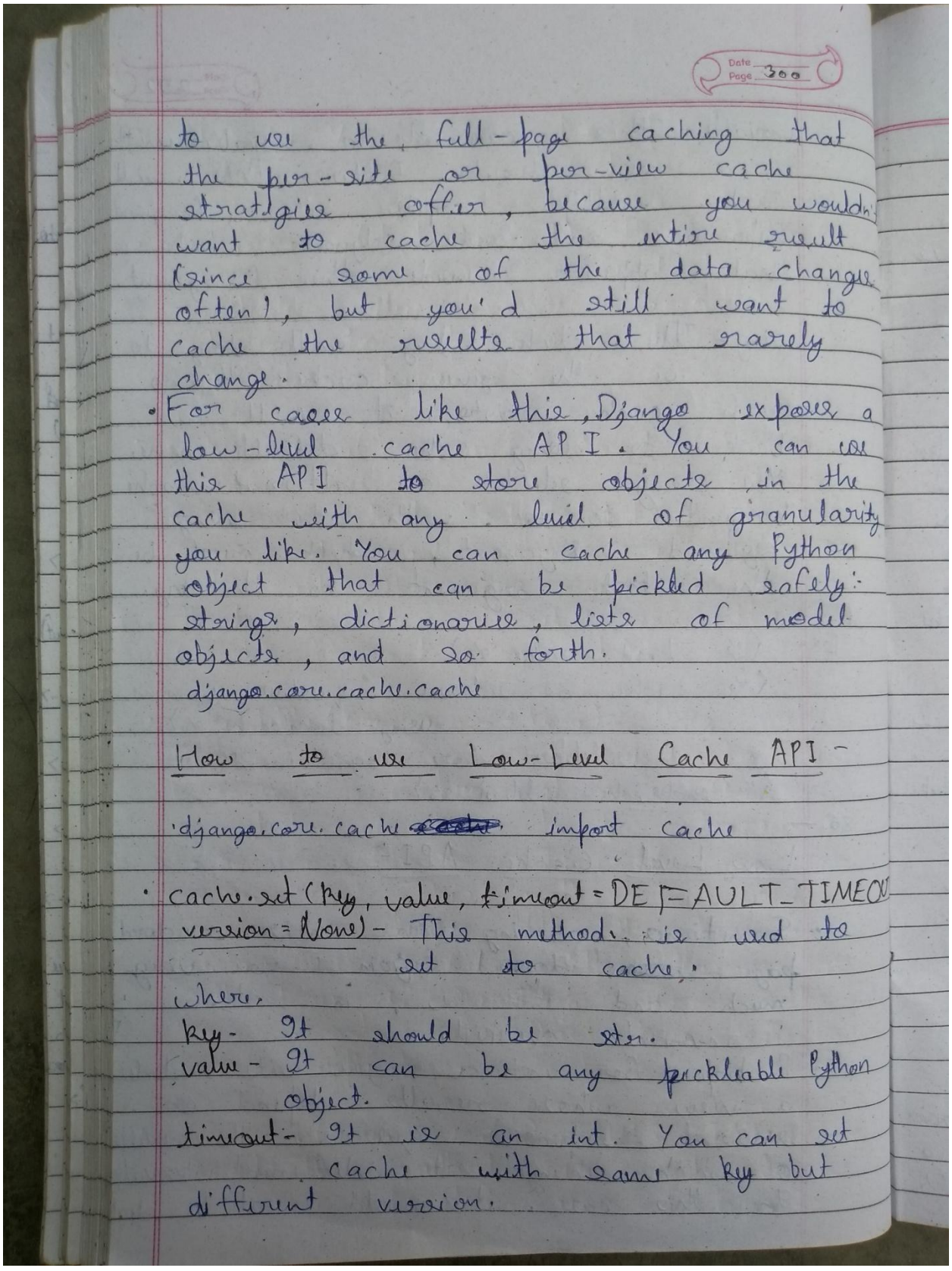
- variable - This can be a variable with or without filters. This will cache multiple copies of a fragment depending on some dynamic data that appears inside the fragment.
- using - The cache tag will try to use the given cache. If no such cache exists, it will fall back to using the default cache. You may select an alternate cache backend to use with the using keyword argument, which must be the last argument to the tag.

Ex- {% load cache %}
{% cache 60 sidebar request.user.username using="localcache" %}
... sidebar for logged in user ...
{% endcache sidebar %}

86 —

Low-Level Cache API-

- Sometimes, caching an entire rendered page doesn't gain you very much and is, in fact, inconvenient overkill.
- Perhaps, for instance, your site includes a view whose results depend on several expensive queries, the results of which change at different intervals. In this case, it would not be ideal



to use the full-page caching that the per-site or per-view cache strategies offer, because you wouldn't want to cache the entire result (since some of the data changes often), but you'd still want to cache the results that rarely change.

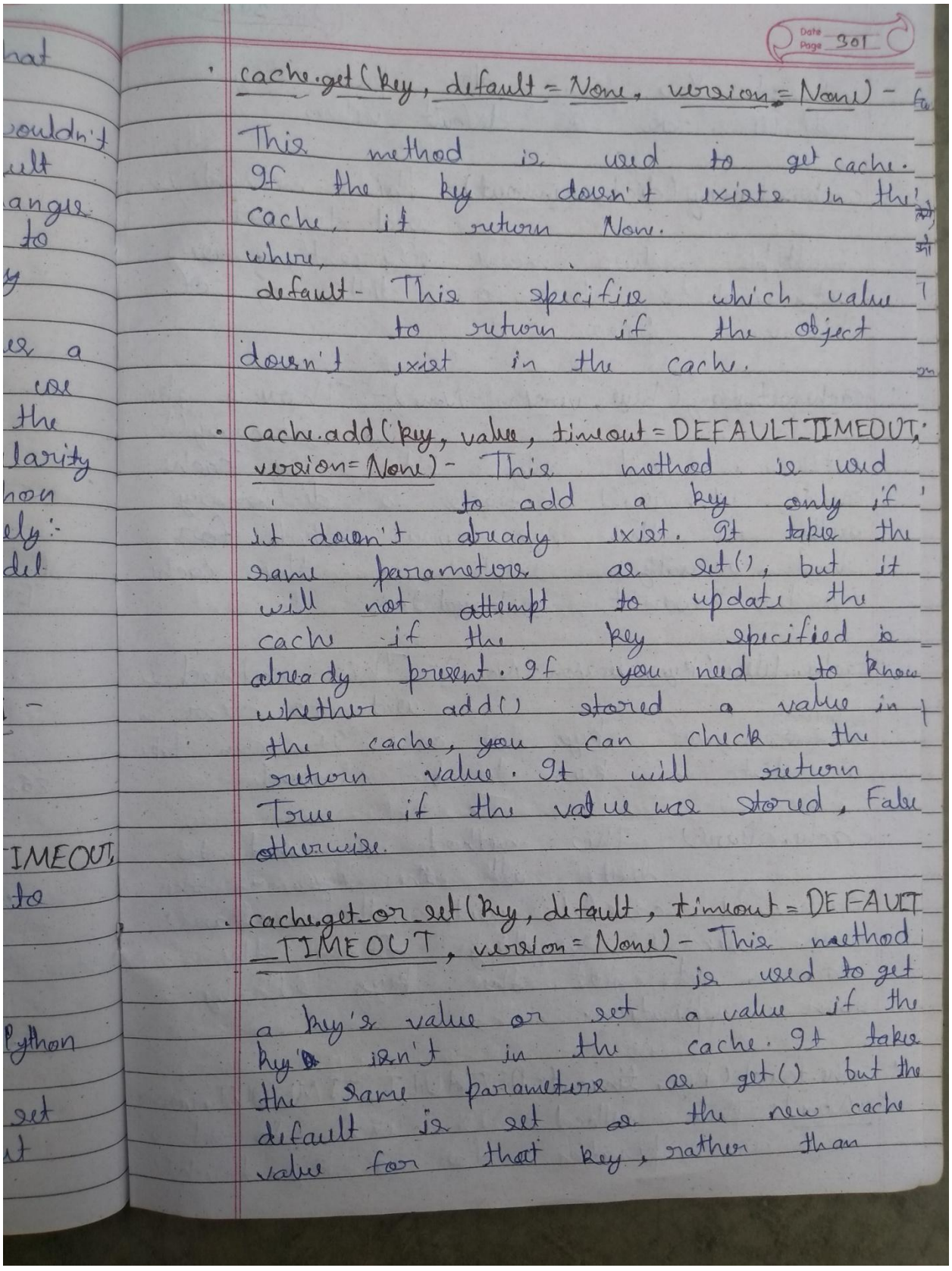
- For cases like this, Django exposes a low-level cache API. You can use this API to store objects in the cache with any level of granularity you like. You can cache any Python object that can be pickled safely: strings, dictionaries, lists of model objects, and so forth.
  django.core.cache.cache

How to use Low-Level Cache API -

'django.core.cache import cache

- cache.set(key, value, timeout=DEFAULT_TIMEOUT, version=None) - This method is used to set to cache.

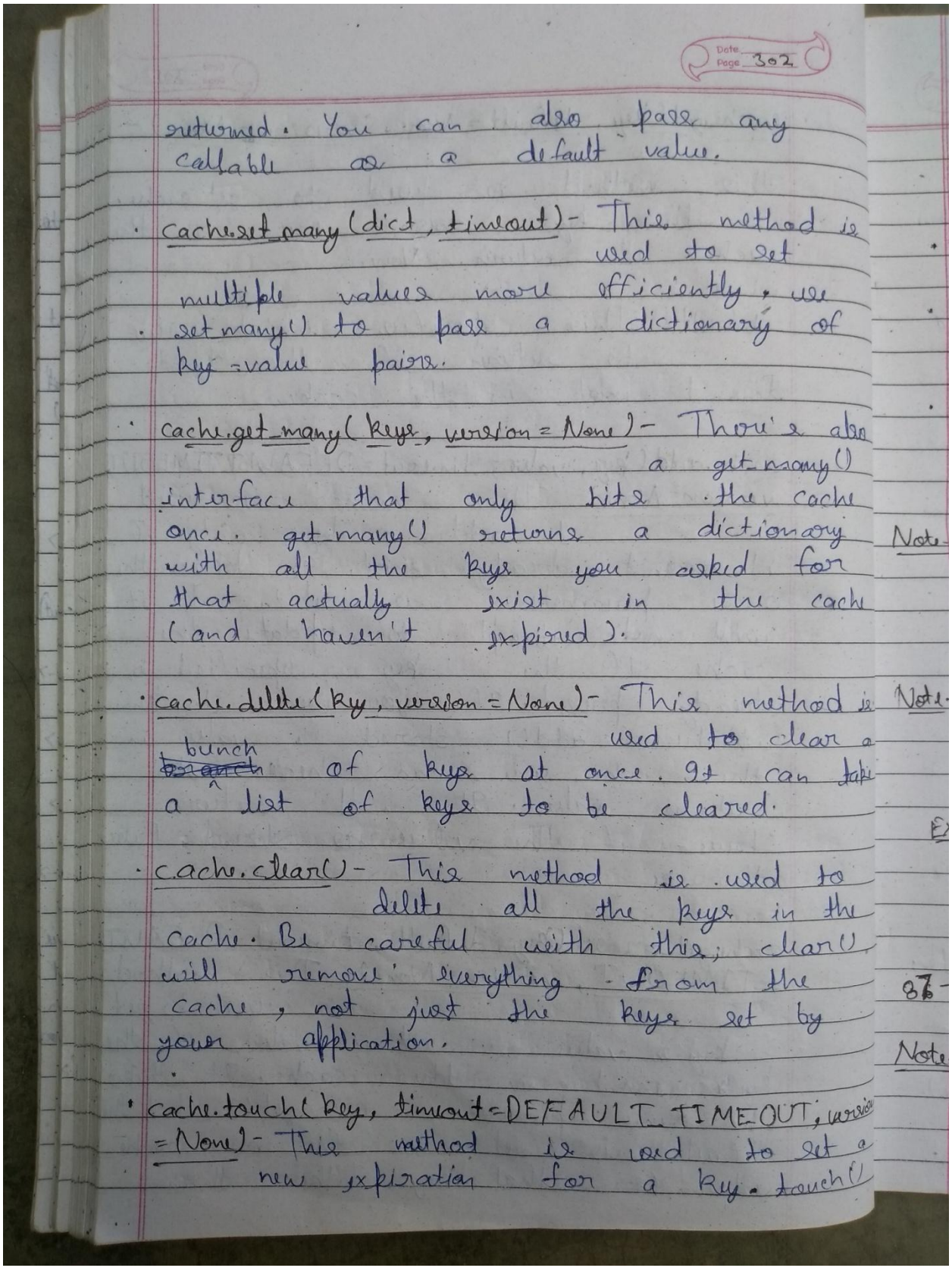
where,
key - It should be str.
value - It can be any picklable Python object.
timeout - It is an int. You can set cache with same key but different version.



- cache.get(key, default=None, version=None) - This method is used to get cache. If the key doesn't exists in the cache, it return None.
where,
default - This specifies which value to return if the object doesn't exist in the cache.

- cache.add(key, value, timeout=DEFAULT_TIMEOUT, version=None) - This method is used to add a key only if it doesn't already exist. It takes the same parameters as set(), but it will not attempt to update the cache if the key specified is already present. If you need to know whether add() stored a value in the cache, you can check the return value. It will return True if the value was stored, False otherwise.

- cache.get_or_set(key, default, timeout=DEFAULT_TIMEOUT, version=None) - This method is used to get a key's value or set a value if the key isn't in the cache. It takes the same parameters as get() but the default is set as the new cache value for that key, rather than



Date Page 302

returned. You can also pass any callable as a default value.

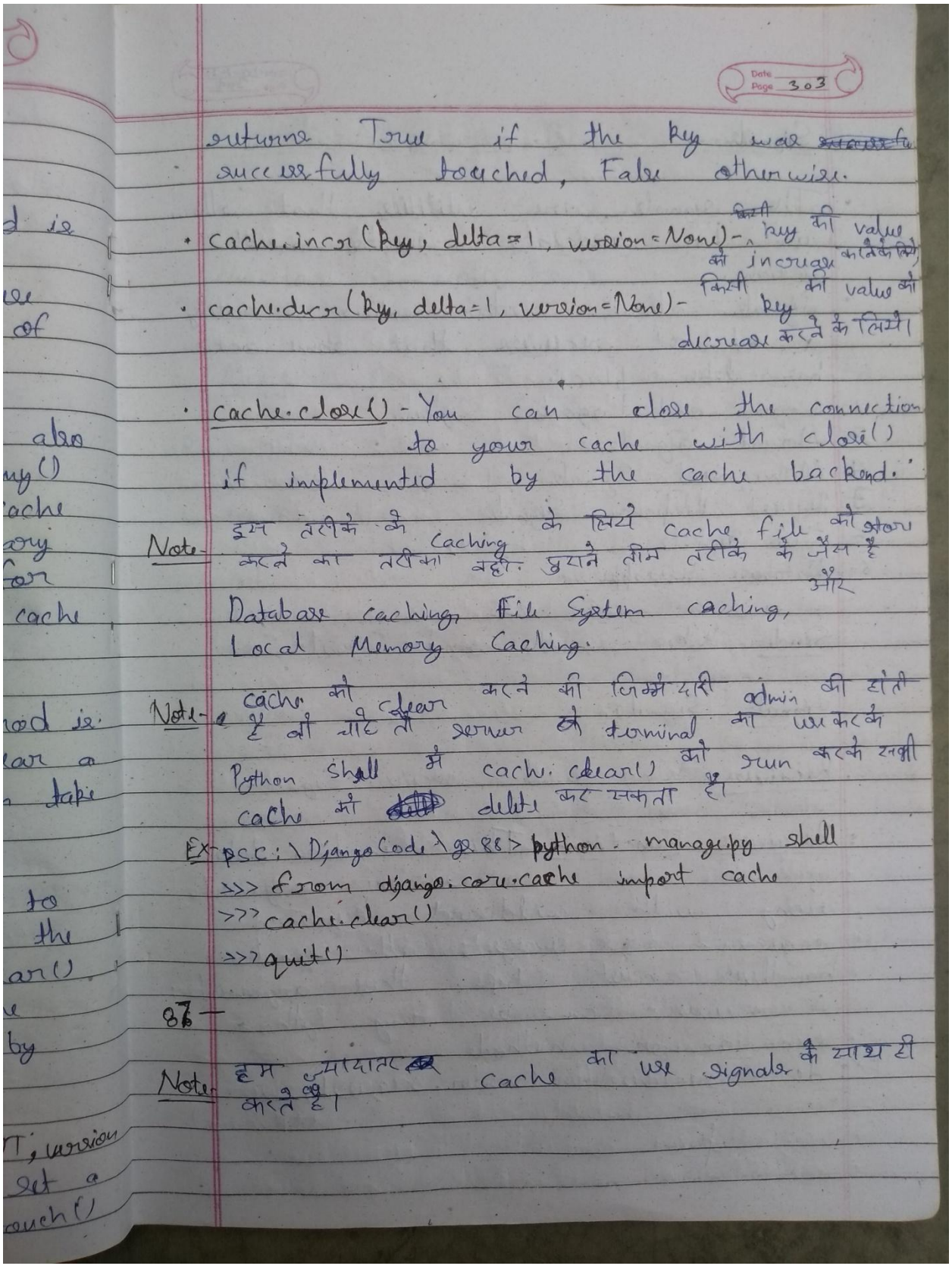
- cache.set_many(dict, timeout) - This method is used to set multiple values more efficiently. use set_many() to pass a dictionary of key-value pairs.

- cache.get_many(keys, version=None) - There's also a get_many() interface that only hits the cache once. get_many() returns a dictionary with all the keys you asked for that actually exist in the cache (and haven't expired).

- cache.delete(key, version=None) - This method is used to clear a ~~branch~~ bunch of keys at once. It can take a list of keys to be cleared.

- cache.clear() - This method is used to delete all the keys in the cache. Be careful with this; clear() will remove everything from the cache, not just the keys set by your application.

- cache.touch(key, timeout=DEFAULT_TIMEOUT, version=None) - This method is used to set a new expiration for a key. touch()



returns True if the key was successfully touched, False otherwise.

- cache.incr(key, delta=1, version=None) - किसी key की value को increase करने के लिये।
- cache.decr(key, delta=1, version=None) - किसी key की value को decrease करने के लिये।
- cache.close() - You can close the connection to your cache with close() if implemented by the cache backend.

Note- इस तरीके के Caching के लिये cache file को store करने का तरीका बहोत पुराने तीन तरीके के जैसा है और Database caching, File System caching, Local Memory Caching.

Note- cache को clear करने की जिम्मेदारी admin की होती है वो चाहे तो server को terminal का use करके Python Shell में cache.clear() को run करके सभी cache को delete कर सकता है।

Ex- PS C:\Django Code\ge 88> python manage.py shell
>>> from django.core.cache import cache
>>> cache.clear()
>>> quit()

87 -

Note- हम ज्यादातर cache का use signals के साथ ही करते है।

Date 20-7-20
Page 304

Signals -

- The signals are utilities that allow us to associate events with actions.
- Signals allow certain senders to notify a set of receivers that some action has taken place.

1- Login and Logout Signals
2- Model Signals
4- Management Signals
3- Request / Response Signals
5- Test Signals
6- Database Wrappers.

Sender- Who will send Signal.

Signal- Signal.

Receiver- Who will receive Signal.

- Receiver Function- This function takes a sender argument, along with wildcard keyword arguments (**kwargs); all signal handlers must take these arguments. A receiver can be any Python function or method.

Ex- def receiver_func(sender, request, user, **kwargs):
        pass

Connecting/Registering Receiver Function - There are two ways you can connect a receiver to a signal -

1. Manual Connect Route
2. Decorator.

1- Manual Connect Route - To receive a signal, register a receiver function using the Signal.connect() method. The receiver function is called when the signal is sent. All of the signal's receiver functions are called one at a time, in the order they were registered.

Syntax - Signal.connect(receiver_func, sender=None, weak=True, dispatch_uid=None)

where,

receiver_func - The callback function which will be connected to signal.

sender - Specifies a particular sender to receive signals from.

weak - Django stores signal handlers as weak references by default. Thus, if your receiver is a local function, it may be garbage collected. To prevent this, pass weak=False when you call the signal's connect() method.

dispatch_uid - A unique identifier for a signal receiver in case where duplicate signals may be sent.

Date Page 306

2- Decorator - इसमें हमें इस receiver को import करना होगा।
Syntax- from django.dispatch import receiver
@receiver (signal or list of signal, sender)

Built-in Signals- Django provide a set of built-in signals that let user code get notified by Django itself of certain actions. ~~For use signals you import that- django.contrib.auth.signals~~

1- Login and Logout Signals- The auth framework uses the following signals that can be used for notification when a user login in or out. for use this signals we import that from django.contrib.auth.signals

(i) user_logged_in (sender, request, user) - Sent when a user login in successfully.
where,
- sender- The class of the user that just logged in.
- request- The current HttpRequest instance.
- user- The user instance that just logged

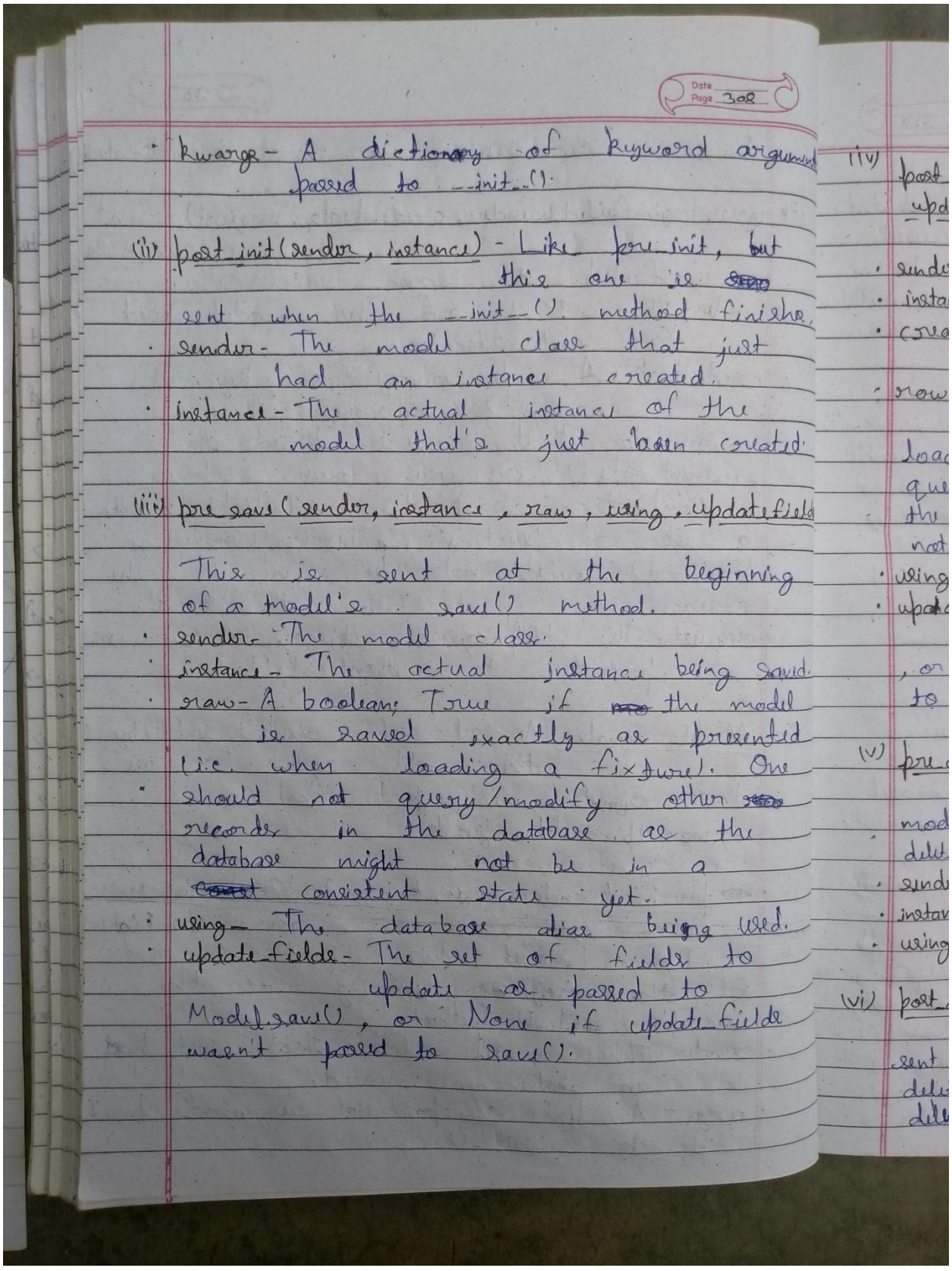
(ii) user_logged_out (sender, request, user) - Sent when the logout method is called.
- sender- The class of the user that just logged out or None if the user was not authenticated.
- request- The current HttpRequest instance.
- user- The user instance that just logged out or

None. if the user was not authenticated

(iii) user_login_failed (sender, credentials, request) - Sent when the user failed to login successfully.
- sender - The name of the module used for authentication.
- credentials - A dicitionary of keyword arguments containing the user credentials that were passed to authenticate() or your own custom authentication backend. Credentials matching a set of 'sensitive' patterns, (including password) will not be sent in the clear as part of the signal.
- request - The HttpRequest object, if one was provided to authenticate().

2- Model signals - A set of signals sent by the model system. For use these signals we import that from- django.db.models.signals.

(i) pre_init (sender, args, kwargs) - Whenever you instantiate a Django model, this signal is sent at the beginning of the model's __init__() method.
- sender - The model class that just had an instance created.
- args - A list of positional arguments passed to __init__().

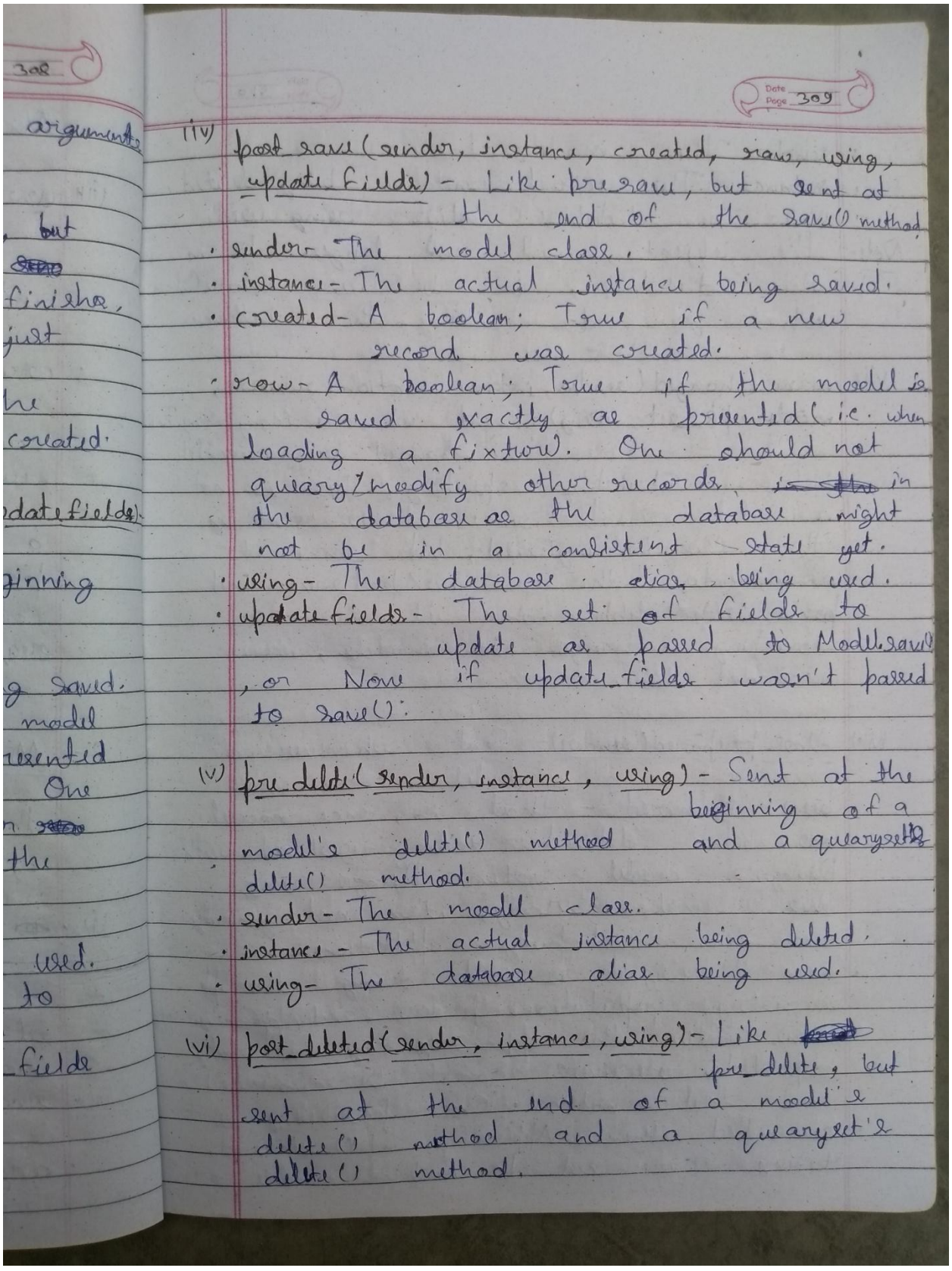


- kwargs - A dictionary of keyword arguments passed to __init__().

(ii) post_init(sender, instance) - Like pre_init, but this one is sent when the __init__() method finishes.
- sender - The model class that just had an instance created.
- instance - The actual instance of the model that's just been created.

(iii) pre_save(sender, instance, raw, using, update_fields)

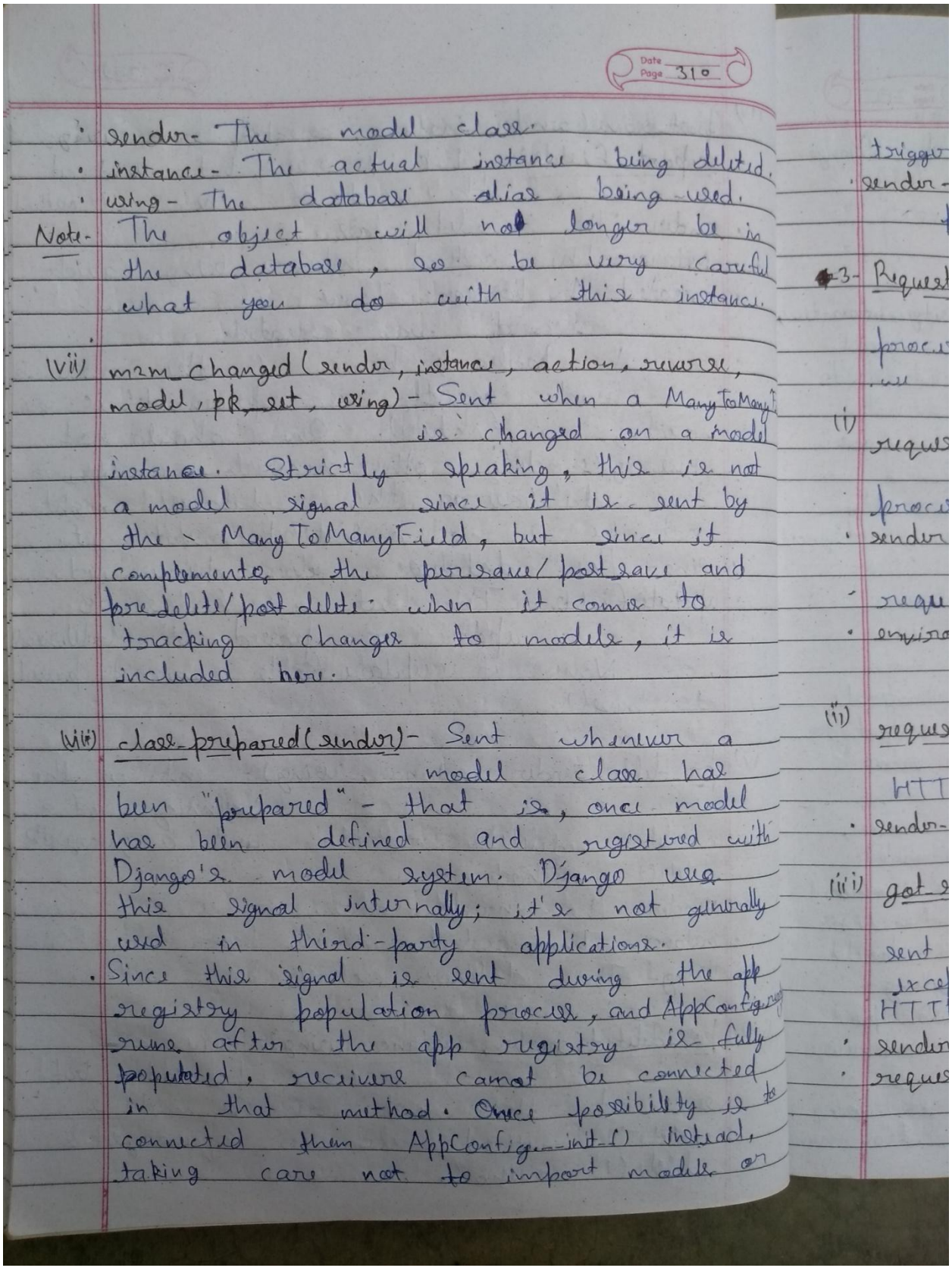
This is sent at the beginning of a model's save() method.
- sender - The model class.
- instance - The actual instance being saved.
- raw - A boolean; True if the model is saved exactly as presented (i.e. when loading a fixture). One should not query/modify other records in the database as the database might not be in a consistent state yet.
- using - The database alias being used.
- update_fields - The set of fields to update as passed to Model.save(), or None if update_fields wasn't passed to save().



(iv) post_save(sender, instance, created, raw, using, update_fields) - Like pre_save, but sent at the end of the save() method.
- sender - The model class.
- instance - The actual instance being saved.
- created - A boolean; True if a new record was created.
- raw - A boolean; True if the model is saved exactly as presented (i.e. when loading a fixture). One should not query/modify other records in the database as the database might not be in a consistent state yet.
- using - The database alias being used.
- update_fields - The set of fields to update as passed to Model.save(), or None if update_fields wasn't passed to save().

(v) pre_delete(sender, instance, using) - Sent at the beginning of a model's delete() method and a queryset's delete() method.
- sender - The model class.
- instance - The actual instance being deleted.
- using - The database alias being used.

(vi) post_deleted(sender, instance, using) - Like pre_delete, but sent at the end of a model's delete() method and a queryset's delete() method.



Date Page 310

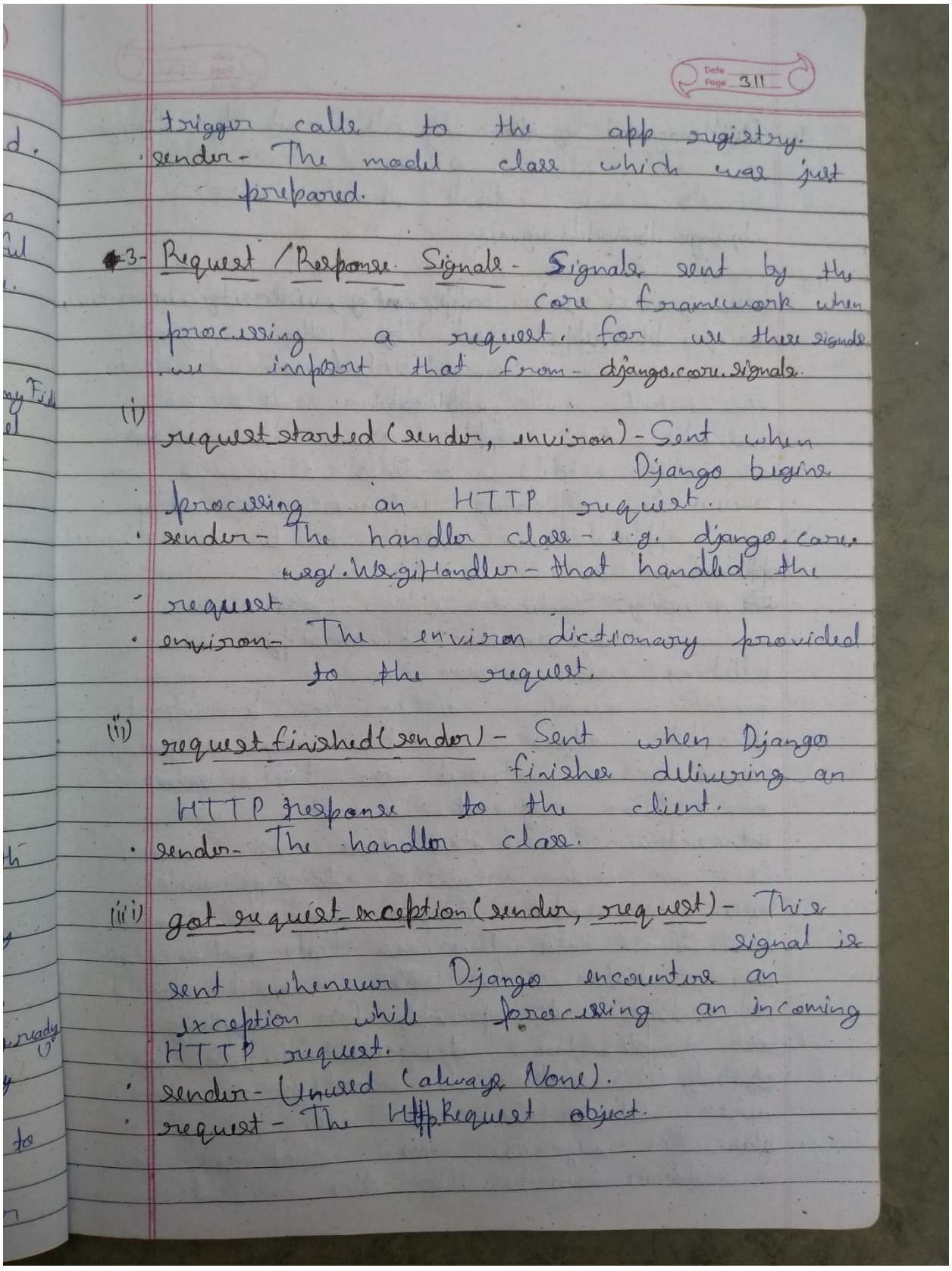
- sender - The model class.
- instance - The actual instance being deleted.
- using - The database alias being used.

Note - The object will not longer be in the database, so be very careful what you do with this instance.

(vii) m2m_changed(sender, instance, action, reverse, model, pk_set, using) - Sent when a ManyToManyField is changed on a model instance. Strictly speaking, this is not a model signal since it is sent by the ManyToManyField, but since it complements the pre_save/post_save and pre_delete/post_delete when it comes to tracking changes to models, it is included here.

(viii) class_prepared(sender) - Sent whenever a model class has been "prepared" - that is, once model has been defined and registered with Django's model system. Django uses this signal internally; it's not generally used in third-party applications.

- Since this signal is sent during the app registry population process, and AppConfig.ready() runs after the app registry is fully populated, receivers cannot be connected in that method. Once possibility is to connected them AppConfig.__init__() instead, taking care not to import models or



trigger calls to the app registry.
- sender - The model class which was just prepared.

3- Request / Response Signals - Signals sent by the core framework when processing a request. for we these signals we import that from - django.core.signals.

(i) request_started (sender, environ) - Sent when Django begins processing an HTTP request.
- sender - The handler class - e.g. django.core.wsgi.WsgiHandler - that handled the request
- environ - The environ dictionary provided to the request.

(ii) request_finished (sender) - Sent when Django finishes delivering an HTTP response to the client.
- sender - The handler class.

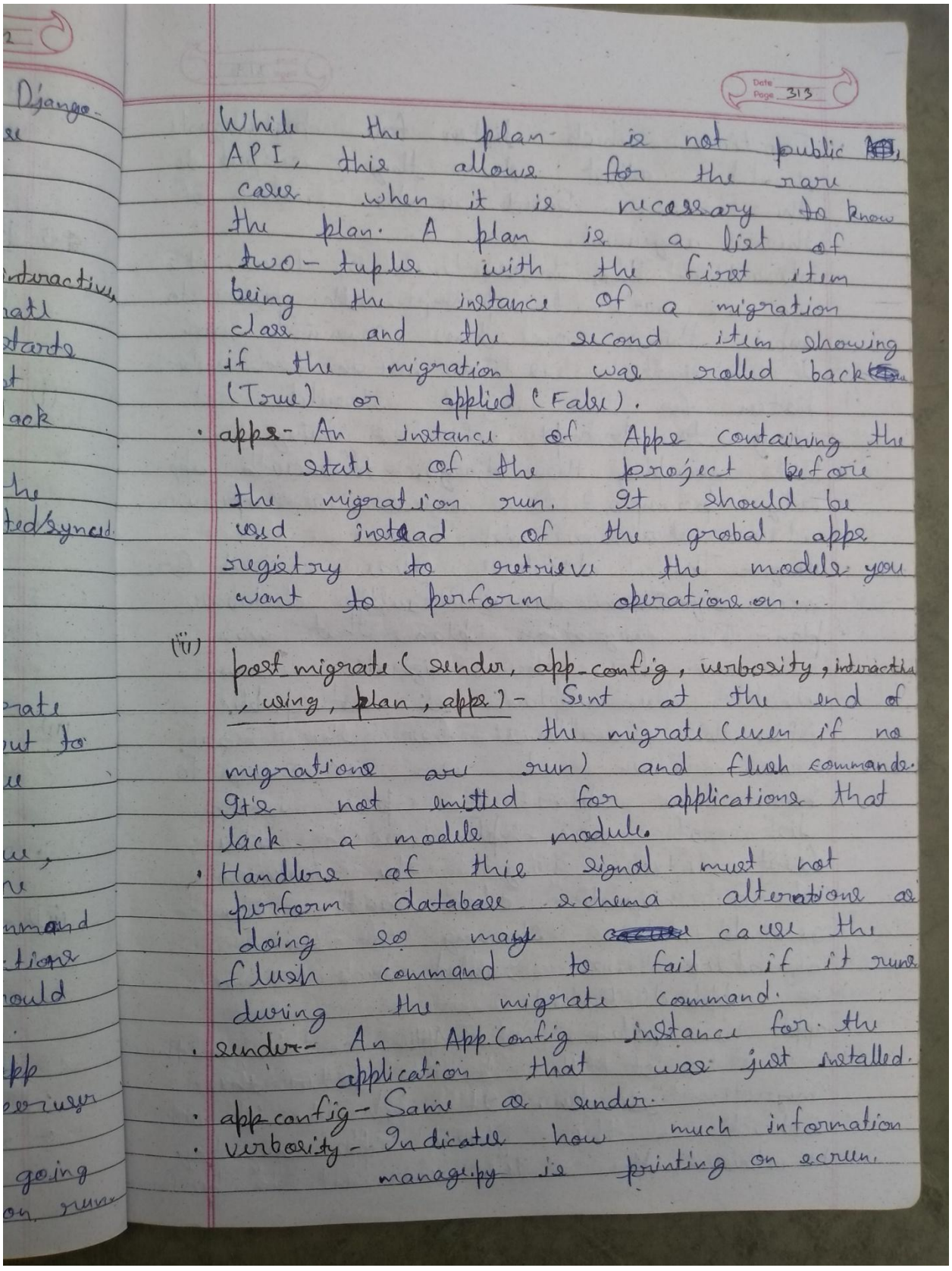
(iii) got_request_exception (sender, request) - This signal is sent whenever Django encounters an exception while processing an incoming HTTP request.
- sender - Unused (always None).
- request - The HttpRequest object.

4- Management signals - Signals sent by Django admin for use these signals we import that from - django.db.models.signals.

(i) pre_migrate(sender, app_config, verbosity, interactive, using, plan, apps) - Sent by the migrate command before it starts to install an application. It's not emitted for applications that lack a models module.

- sender - An AppConfig instance for the application about to be migrated/synced.
- app_config - Same as sender.
- verbosity - Indicates how much information manage.py is printing on screen.
- Function which listen for pre_migrate should adjust what they output to the screen based on the value of this argument.
- interactive - If interactive is True, it's safe to prompt the user to input things on the command line. If interactive is False, functions which listen for this signal should not try to prompt for anything.
- For example, the django.contrib.auth app only prompts to create a superuser when interactive is True.
- plan - The migration plan that is going to be used for the migration run

(ii)

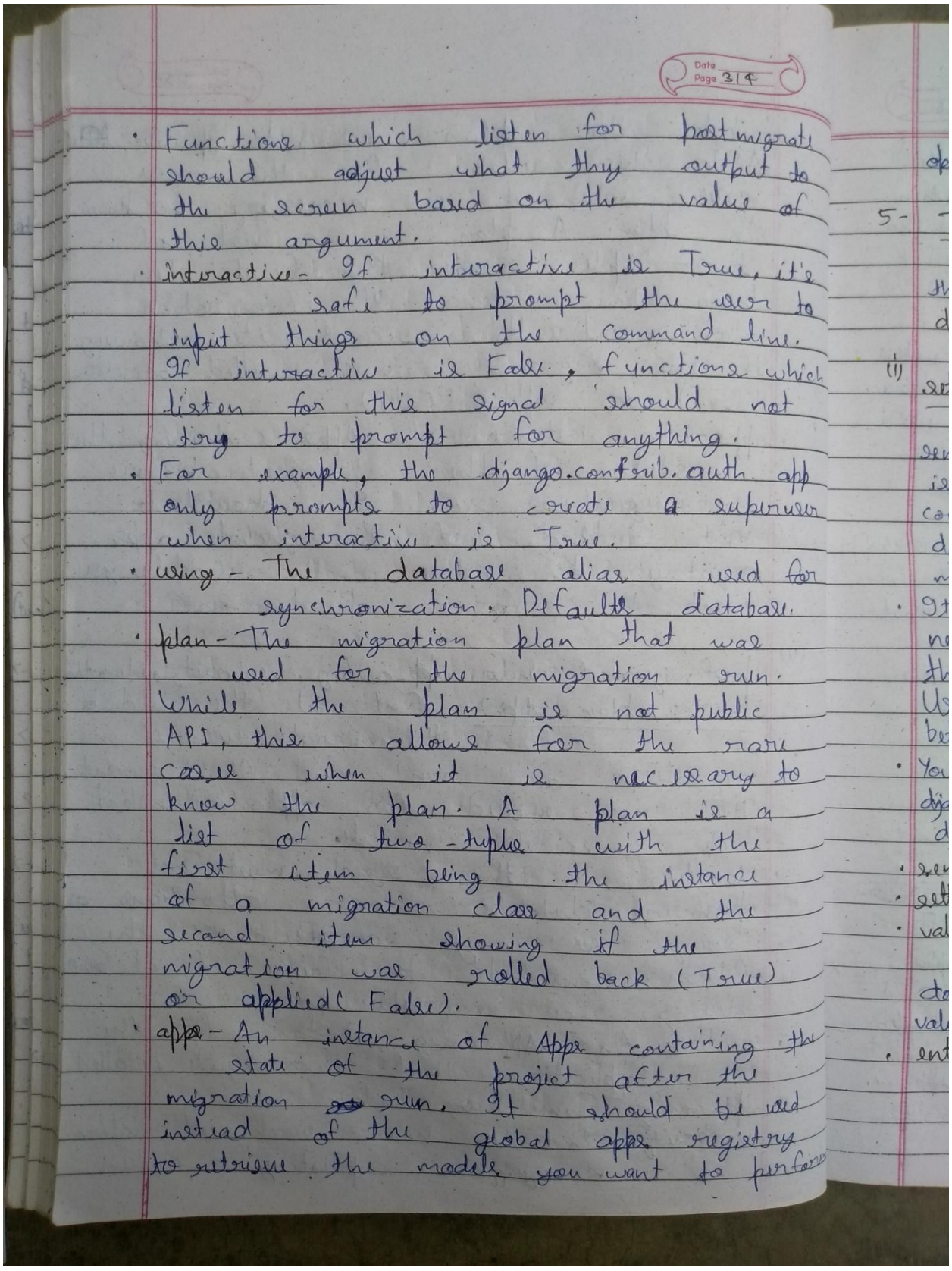


While the plan is not public API, this allows for the rare cases when it is necessary to know the plan. A plan is a list of two-tuples with the first item being the instance of a migration class and the second item showing if the migration was rolled back (True) or applied (False).

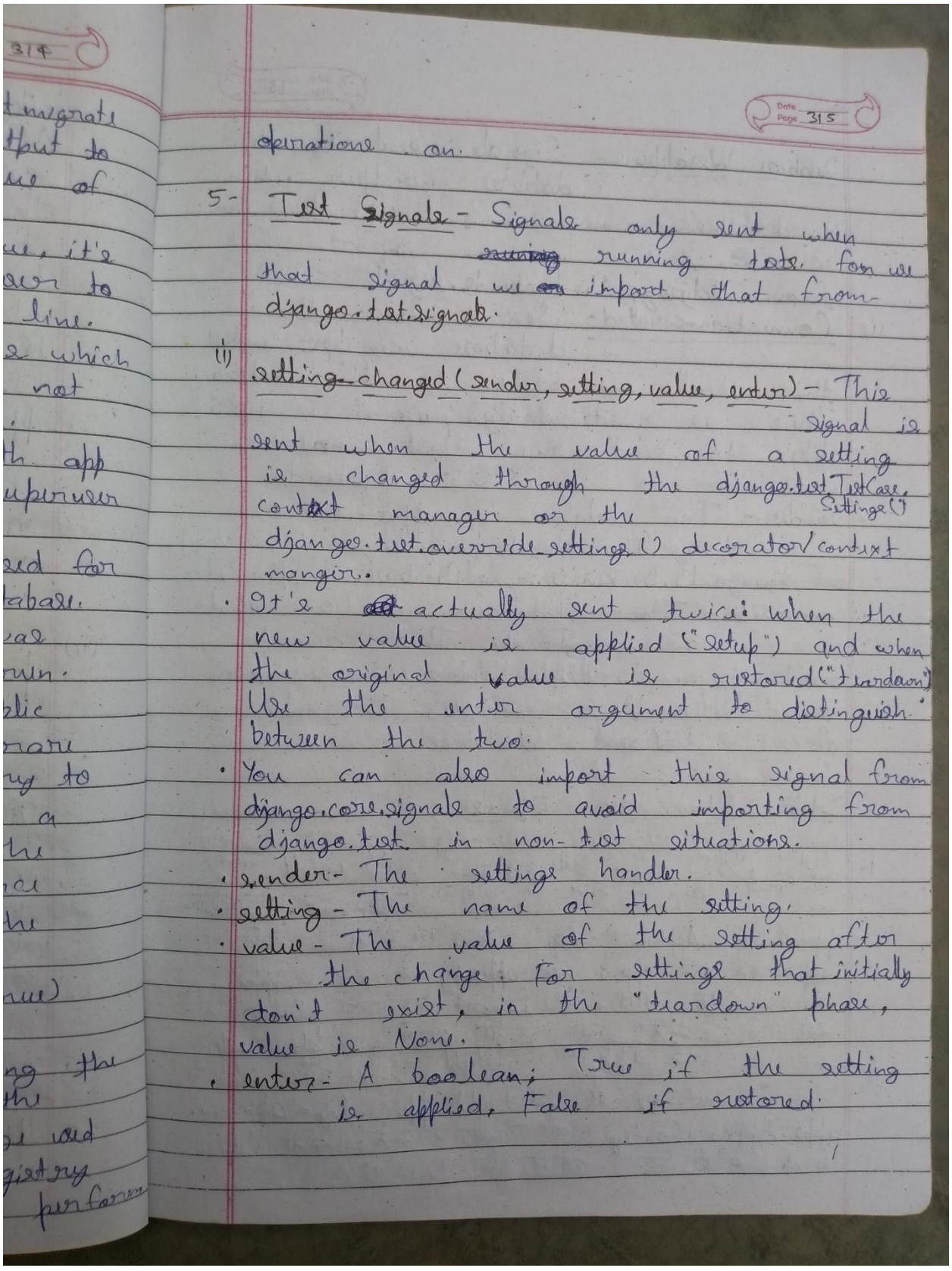
- apps- An instance of Apps containing the state of the project before the migration run. It should be used instead of the global apps registry to retrieve the models you want to perform operations on.

(iii) post_migrate(sender, app_config, verbosity, interactive, using, plan, apps)- Sent at the end of the migrate (even if no migrations are run) and flush commands. It's not emitted for applications that lack a models module.

- Handlers of this signal must not perform database schema alterations as doing so may cause the flush command to fail if it runs during the migrate command.
- sender- An AppConfig instance for the application that was just installed.
- app_config- Same as sender.
- verbosity- Indicates how much information manage.py is printing on screen.



- Functions which listen for post_migrate should adjust what they output to the screen based on the value of this argument.
- interactive - If interactive is True, it's safe to prompt the user to input things on the command line. If interactive is False, functions which listen for this signal should not try to prompt for anything.
- For example, the django.contrib.auth app only prompts to create a superuser when interactive is True.
- using - The database alias used for synchronization. Default database.
- plan - The migration plan that was used for the migration run. While the plan is not public API, this allows for the rare case when it is necessary to know the plan. A plan is a list of two-tuples with the first item being the instance of a migration class and the second item showing if the migration was rolled back (True) or applied (False).
- apps - An instance of Apps containing the state of the project after the migration run. It should be used instead of the global apps registry to retrieve the models you want to perform



operations on.

5- Test Signals - Signals only sent when running tests. For us that signal we import that from- django.test.signals.

i) setting_changed(sender, setting, value, enter) - This signal is sent when the value of a setting is changed through the django.test.TestCase.setting() context manager or the django.test.override_settings() decorator/context mangir.

- It's actually sent twice: when the new value is applied ("setup") and when the original value is restored ("teardown"). Use the enter argument to distinguish between the two.
- You can also import this signal from django.core.signals to avoid importing from django.test in non-test situations.
- sender - The settings handler.
- setting - The name of the setting.
- value - The value of the setting after the change. For settings that initially don't exist, in the "teardown" phase, value is None.
- enter - A boolean; True if the setting is applied, False if restored.

Date Page 316

Dabbase Wrappers- Signals sent by the database wrapper when a database connection is initiated. For use that we import that from- django.db.backends.signals.

ii) Connection_created- Sent when the database wrapper makes the initial connection to the database. This is particularly useful if you'd like to send any post connection commands to the SQL backend.

- sender- The database wrapper - i.e. django.db.backends.postgresql.DatabaseWrapper or django.db.backends.mysql.DatabaseWrapper etc.
- connection- The database connection that was opened. This can be used in a multiple-database configration to differentiate connection signals from different databases.

Steps to Use Signals-

1- सबसे पहले हम settings.py में app install करने के लिये app के नाम के जगह ~~appname.apps.Appname~~ "appname.apps.AppnameConfig" का use करते हैं जैसे ~~app का नाम~~ app के अंदर apps.py के अंदर AppnameConfig class का नाम को import करने का path होता है या फिर ऐसा ना कर ना हो तो simple appname का use करके app को install करते और app के अंदर __init__.py में default_app_config = "Appname.apps.AppnameConfig" को लिख सकते हैं दोनों option में से कोई एक काम करना होगा।

Ex-

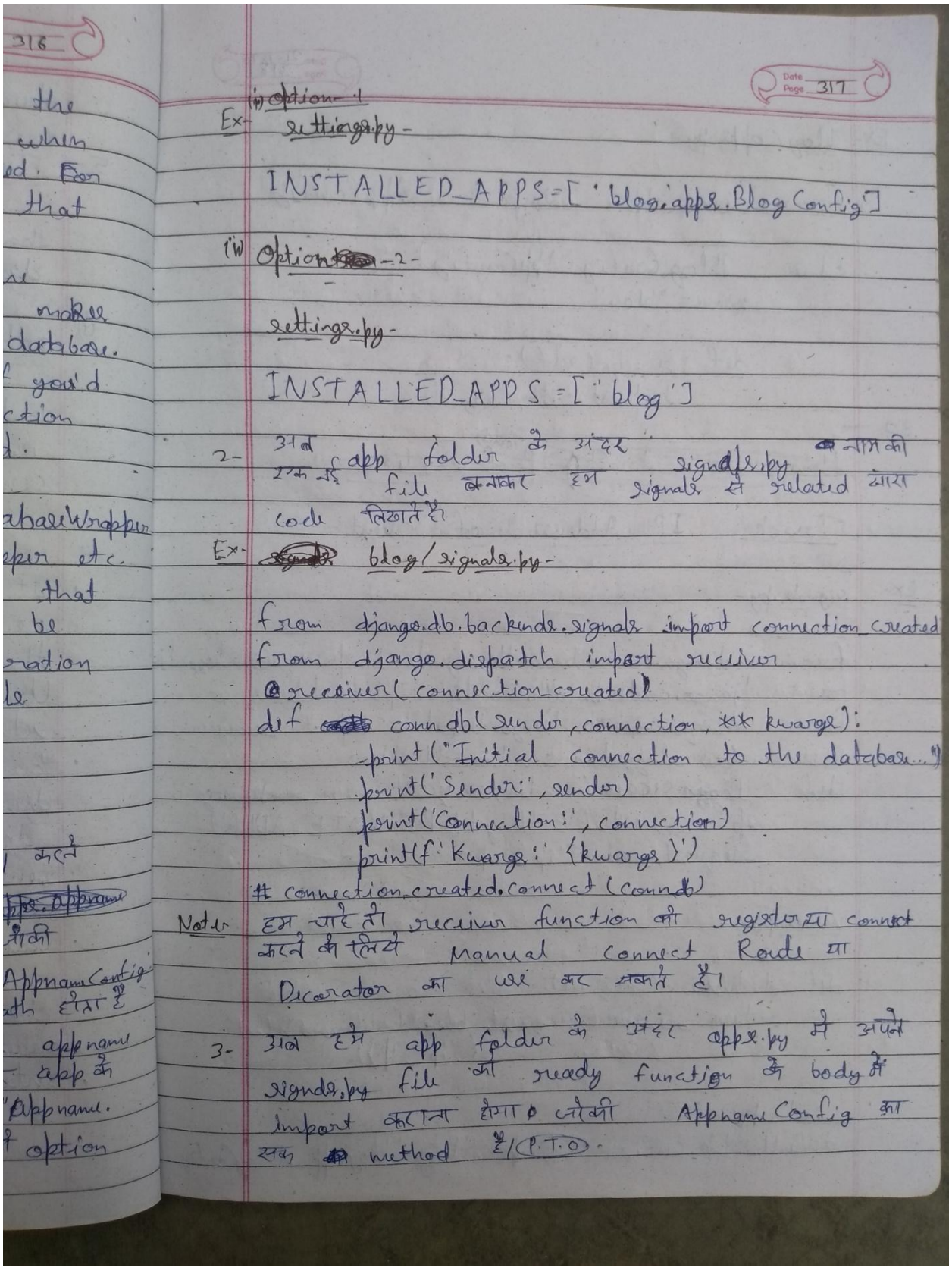
(iv)

2-

Ex-

Note-

3-



316

the
when
ed. For
that

make
database.
you'd
ction
t.

abaseWrapper
pper etc.
that
be
ration
le

करते

Appname.Appname
की

AppnameConfig
ath होता है

appname
app के
'Appname.
t option

(ii) option-1

Ex- settings.py -

```
INSTALLED_APPS=['blog.apps.BlogConfig']
```

(ii) Option-2-

settings.py -

```
INSTALLED_APPS=['blog']
```

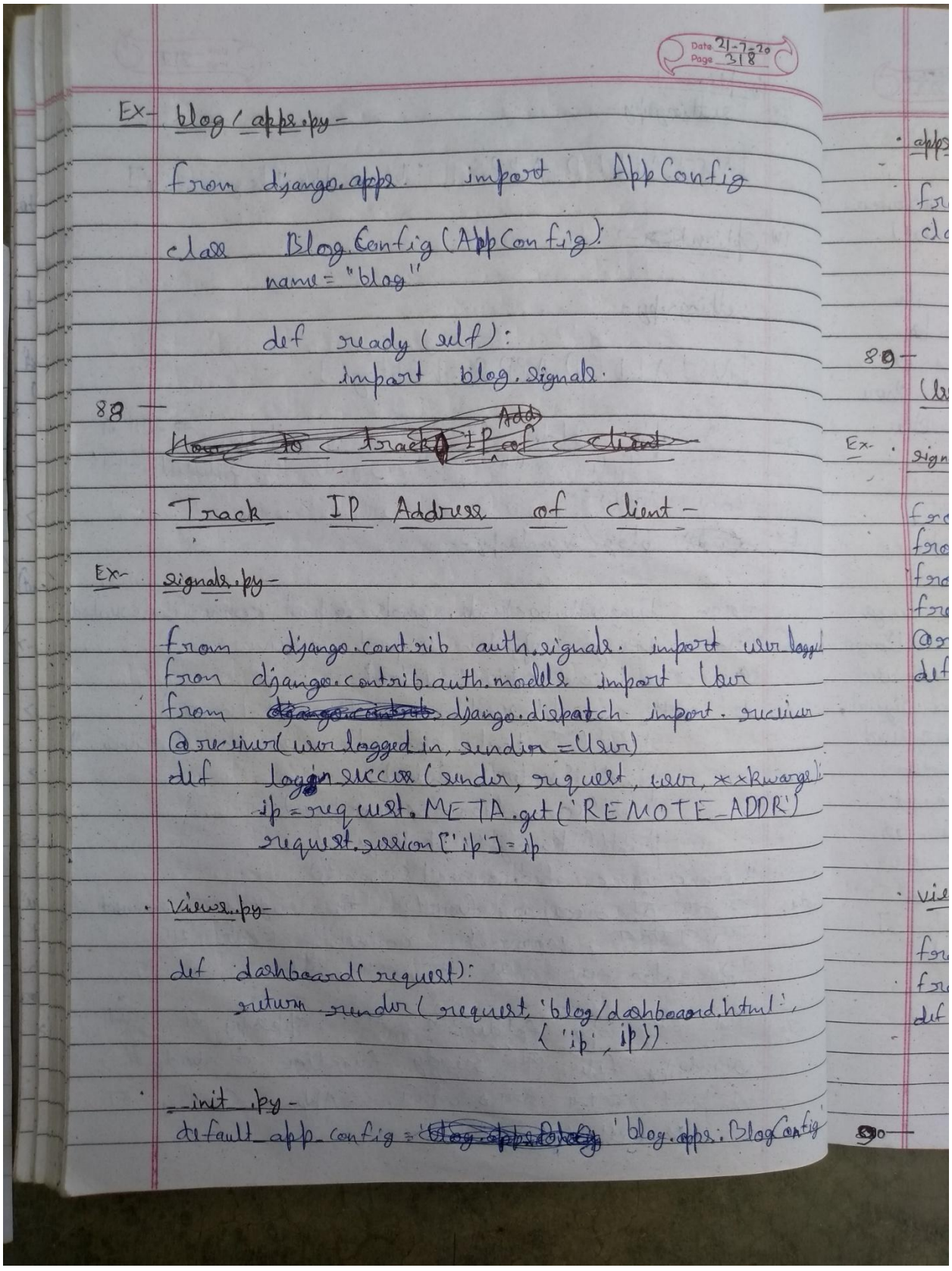
2- अब app folder के अंदर signals.py नाम की एक नई file बनाकर हम signals से related कोड लिखते है

Ex- blog/signals.py -

```
from django.db.backends.signals import connection_created
from django.dispatch import receiver
@receiver(connection_created)
def conn_db(sender, connection, **kwargs):
    print("Initial connection to the database...")
    print('Sender:', sender)
    print('Connection:', connection)
    print(f'Kwargs: {kwargs}')
# connection_created.connect(conn_db)
```

Note- हम चाहे तो receiver function को register या connect करने के लिये Manual connect Route या Decorator का use कर सकते है।

3- अब हमे app folder के अंदर apps.py में अपने signals.py file को ready function के body में import कराना होगा। चोंकी AppnameConfig का एक method है (P.T.O).



Date 21-7-20
Page 318

Ex- blog/apps.py-

```
from django.apps import AppConfig

class BlogConfig(AppConfig):
    name = "blog"

    def ready(self):
        import blog.signals
```

88 —

~~How to track IP of client~~ Add

Track IP Address of client -

Ex- signals.py-

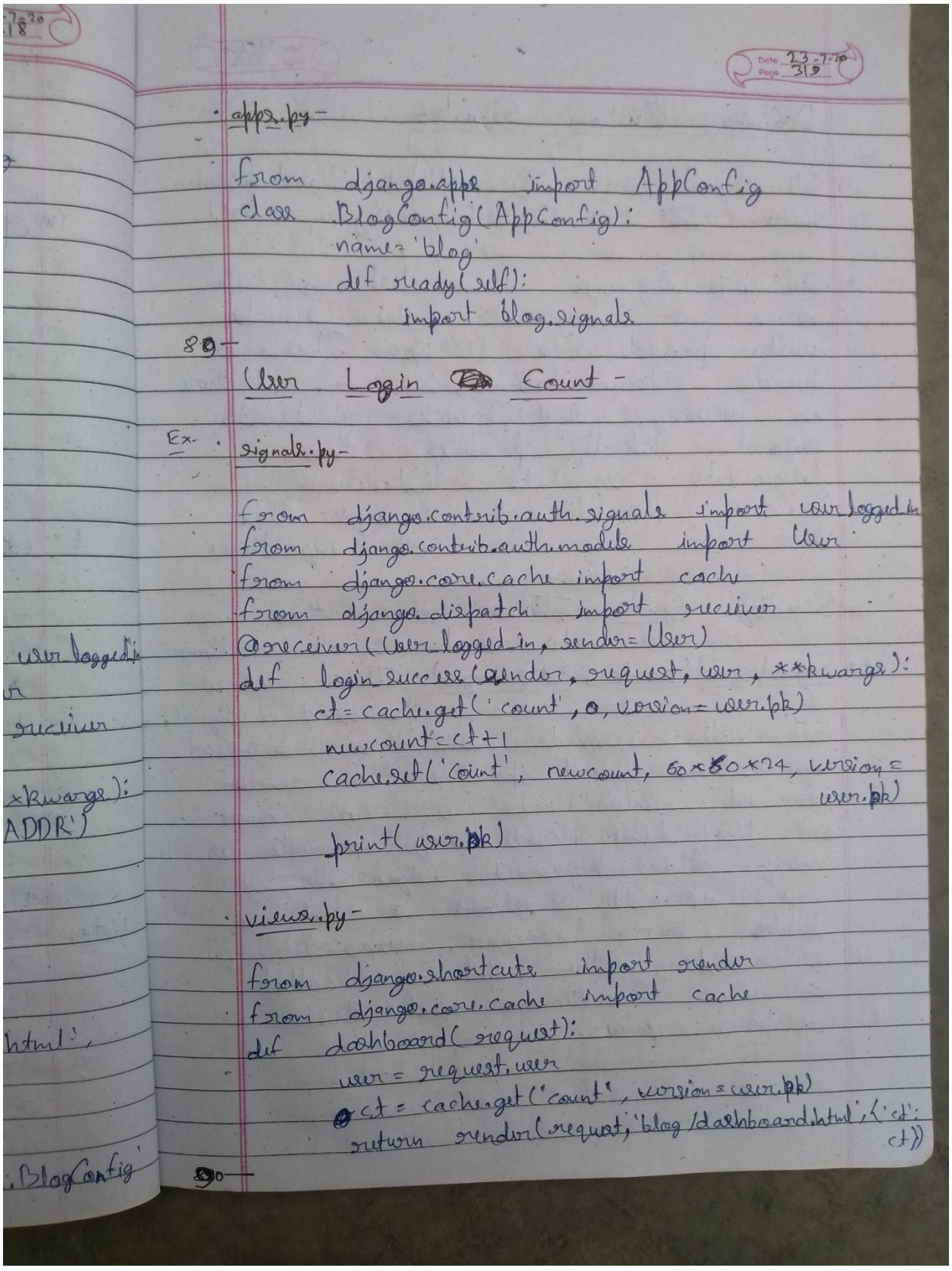
```
from django.contrib.auth.signals import user_logged
from django.contrib.auth.models import User
from django.dispatch import receiver
@receiver(user_logged_in, sender=User)
def login_success(sender, request, user, **kwargs):
    ip = request.META.get('REMOTE_ADDR')
    request.session['ip'] = ip
```

- views.py-

```
def dashboard(request):
    return render(request, 'blog/dashboard.html',
                  {'ip': ip})
```

- __init__.py-

```
default_app_config = 'blog.apps.BlogConfig'
```



apps.py -

```
from django.apps import AppConfig
class BlogConfig(AppConfig):
    name = 'blog'
    def ready(self):
        import blog.signals
```

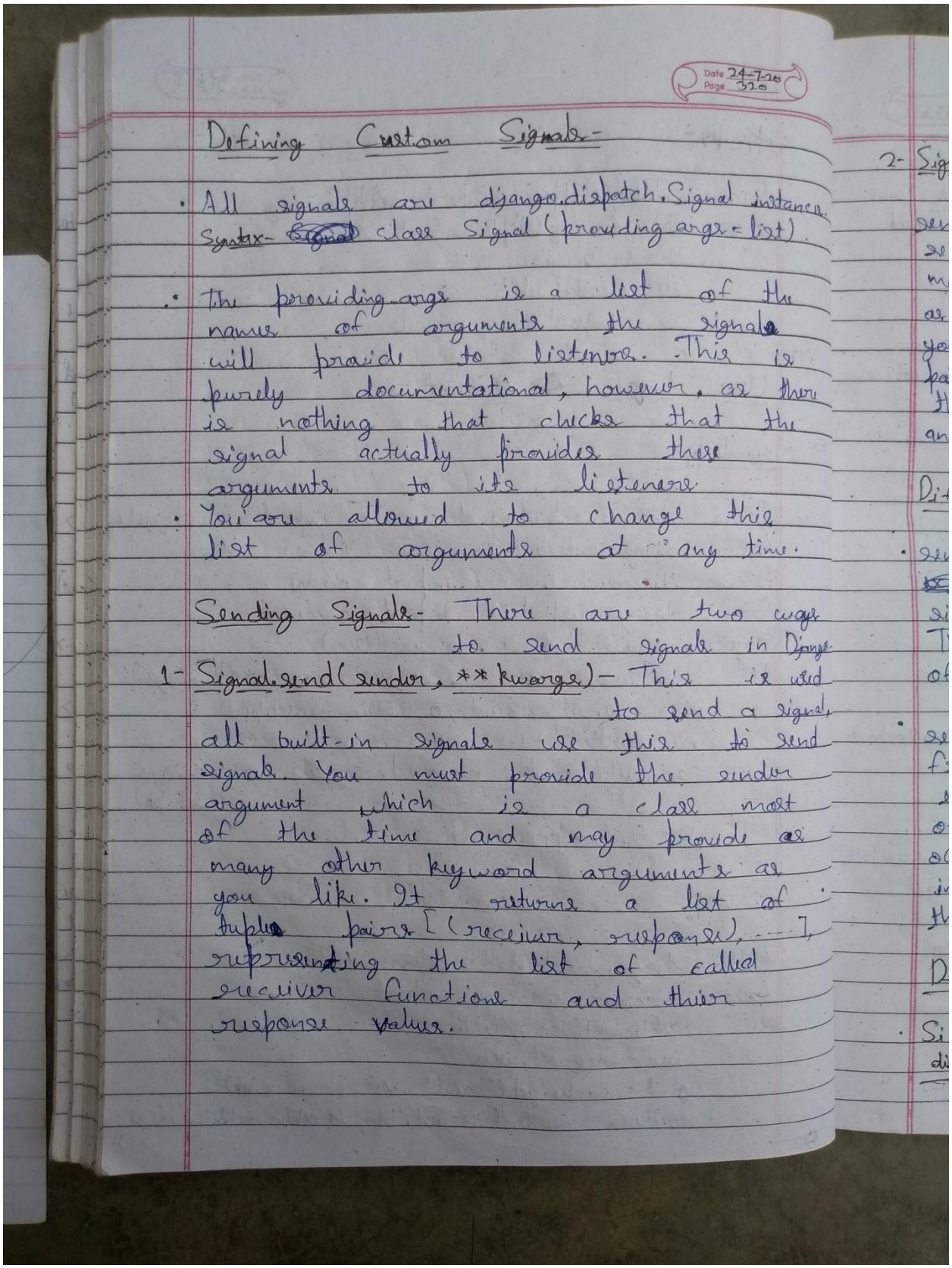
User Login Count -

Ex. signals.py -

```
from django.contrib.auth.signals import user_logged_in
from django.contrib.auth.models import User
from django.core.cache import cache
from django.dispatch import receiver
@receiver(user_logged_in, sender=User)
def login_success(sender, request, user, **kwargs):
    ct = cache.get('count', 0, version=user.pk)
    newcount = ct+1
    cache.set('count', newcount, 60*60*24, version=
                                                user.pk)
    print(user.pk)
```

views.py -

```
from django.shortcuts import render
from django.core.cache import cache
def dashboard(request):
    user = request.user
    ct = cache.get('count', version=user.pk)
    return render(request,'blog/dashboard.html',{'ct':
                                                ct})
```



Date 24-7-20
Page 320

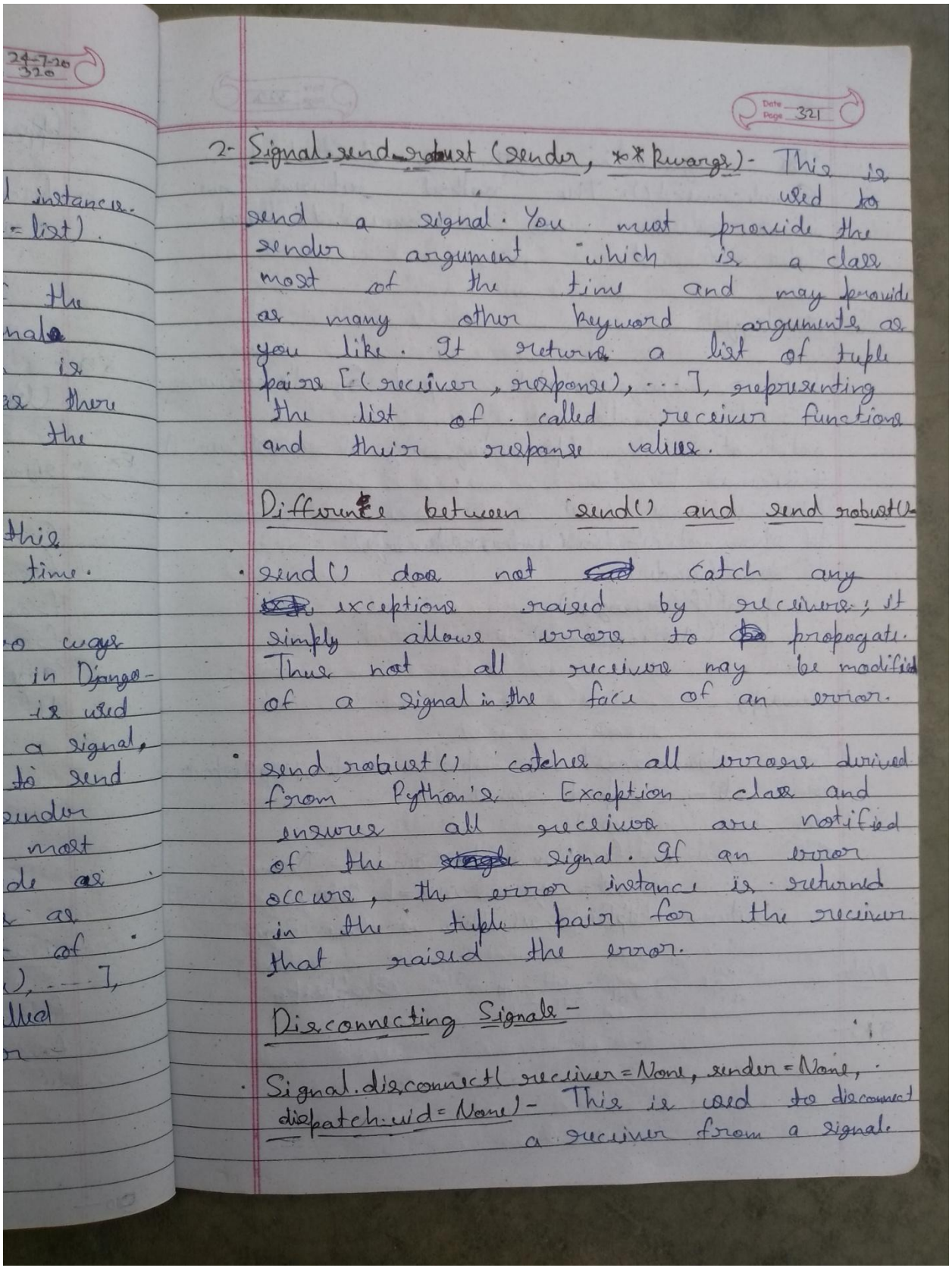
Defining Custom Signals-

- All signals are django.dispatch.Signal instances.
Syntax- class Signal(providing_args=list)

- The providing_args is a list of the names of arguments the signal will provide to listeners. This is purely documentational, however, as there is nothing that checks that the signal actually provides these arguments to its listeners.
- You are allowed to change this list of arguments at any time.

Sending Signals- There are two ways to send signals in Django.

1- Signal.send(sender, **kwargs)- This is used to send a signal. all built-in signals use this to send signals. You must provide the sender argument which is a class most of the time and may provide as many other keyword arguments as you like. It returns a list of tuple pairs [(receiver, response), ...], representing the list of called receiver functions and their response values.



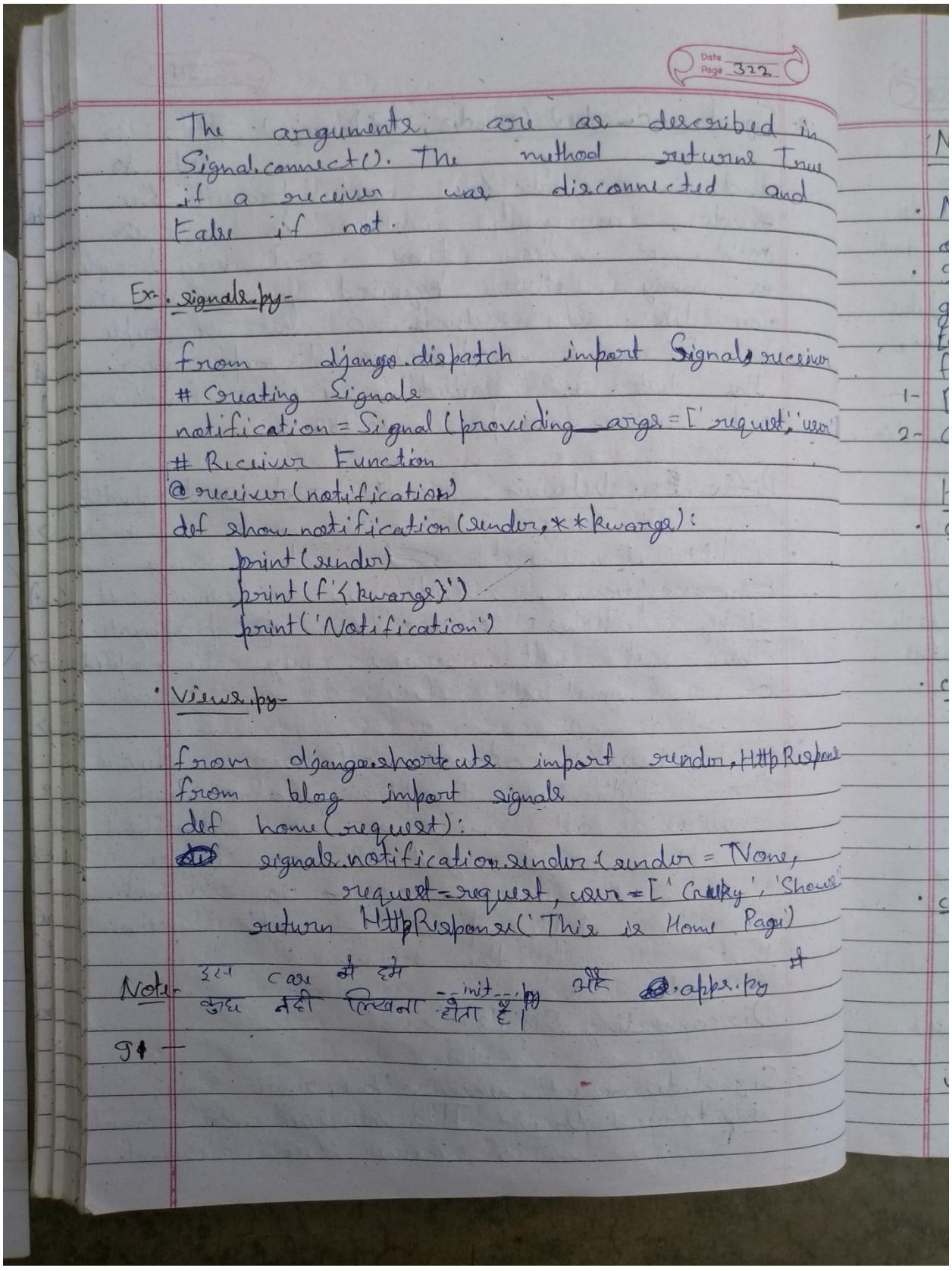
2- Signal.send_robust (sender, **kwargs)- This is used to send a signal. You must provide the sender argument which is a class most of the time and may provide as many other keyword arguments as you like. It returns a list of tuple pairs [(receiver, response), ...], representing the list of called receiver functions and their response values.

Differences between send() and send_robust()-

- send() does not catch any exceptions raised by receivers; it simply allows errors to propagate. Thus not all receivers may be notified of a signal in the face of an error.
- send_robust() catches all errors derived from Python's Exception class and ensures all receivers are notified of the signal. If an error occurs, the error instance is returned in the tuple pair for the receiver that raised the error.

Disconnecting Signals-

- Signal.disconnect(receiver=None, sender=None, dispatch_uid=None)- This is used to disconnect a receiver from a signal.



Date Page 322

The arguments are as described in Signal.connect(). The method returns True if a receiver was disconnected and False if not.

Ex. • signals.py-

```
from django.dispatch import Signal, receiver
# Creating Signals
notification = Signal(providing_args=['request','user'])
# Receiver Function
@receiver(notification)
def show_notification(sender,**kwargs):
    print(sender)
    print(f'{kwargs}')
    print('Notification')
```

• Views.py-

```
from django.shortcuts import render, HttpResponse
from blog import signals
def home(request):
    signals.notification.send(sender=None,
        request=request, user=['Chetty','Shaw'])
    return HttpResponse('This is Home Page')
```

Note- इस case में हमें __init__.py और apps.py में कुछ नहीं लिखना होता है।

91 -

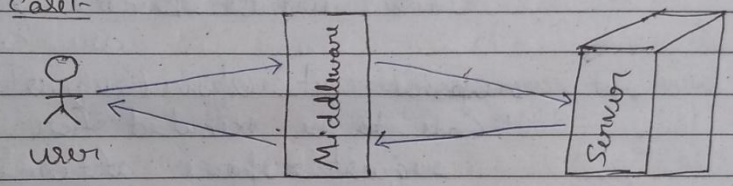
Middleware-

- Middleware is a framework of hooks of into Django's request/response processing.
- It's a light, low-level "plug-in" system for globally altering Django's input or output. Each middleware component is responsible for doing some specific function.

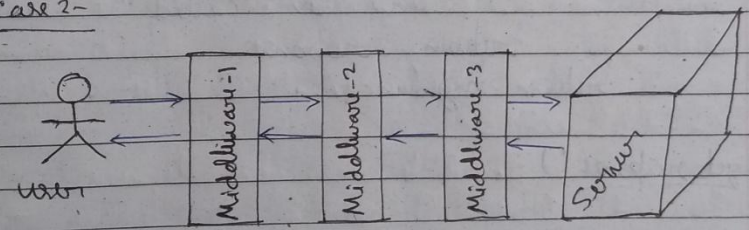
1- Built-in Middleware
2- Custom Middleware

How Middleware Works-

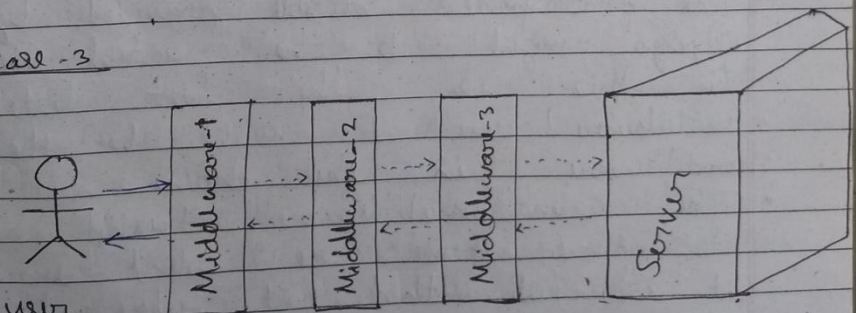
- Case1-

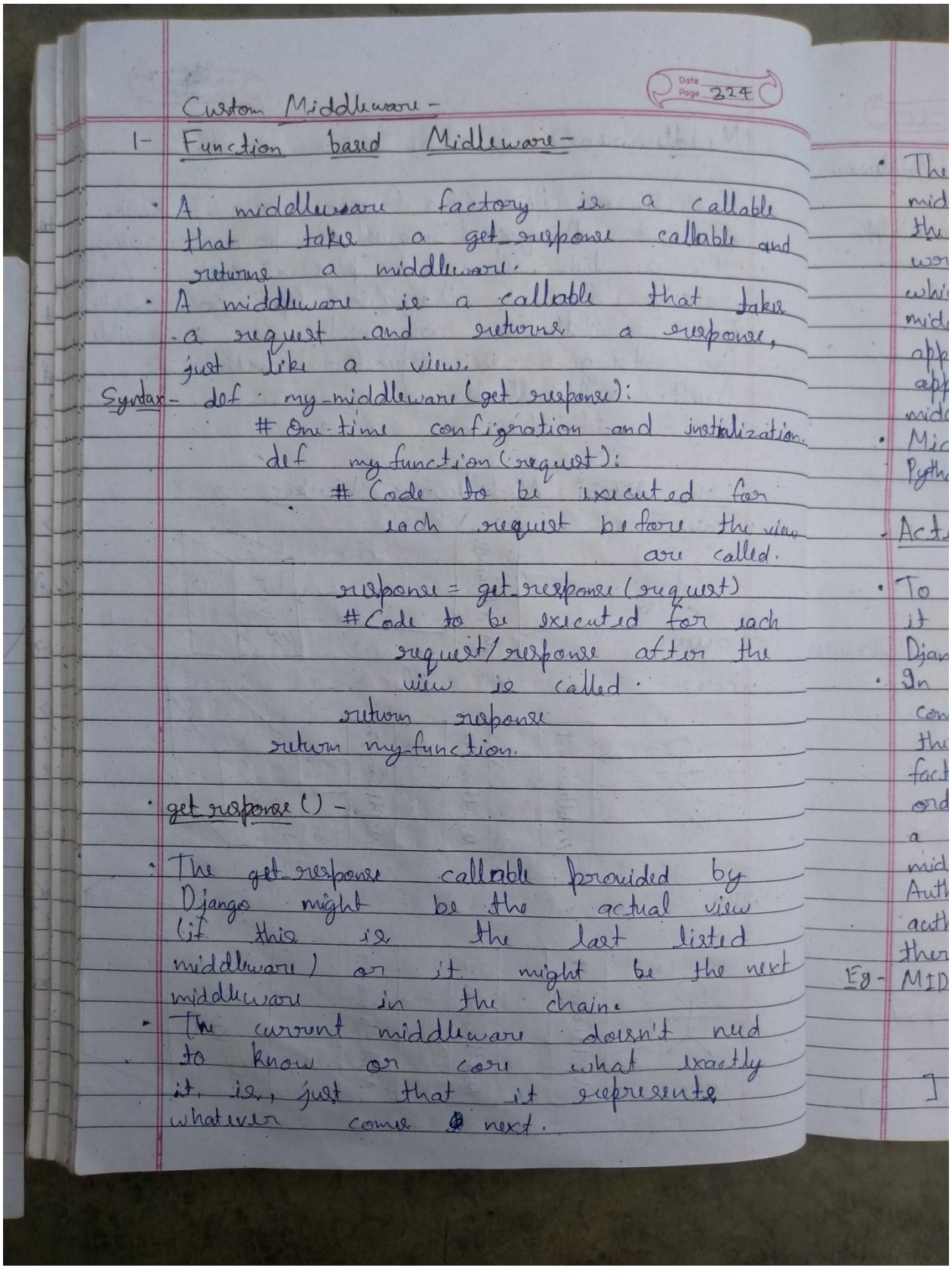


- Case 2-



- Case-3





Custom Middleware -

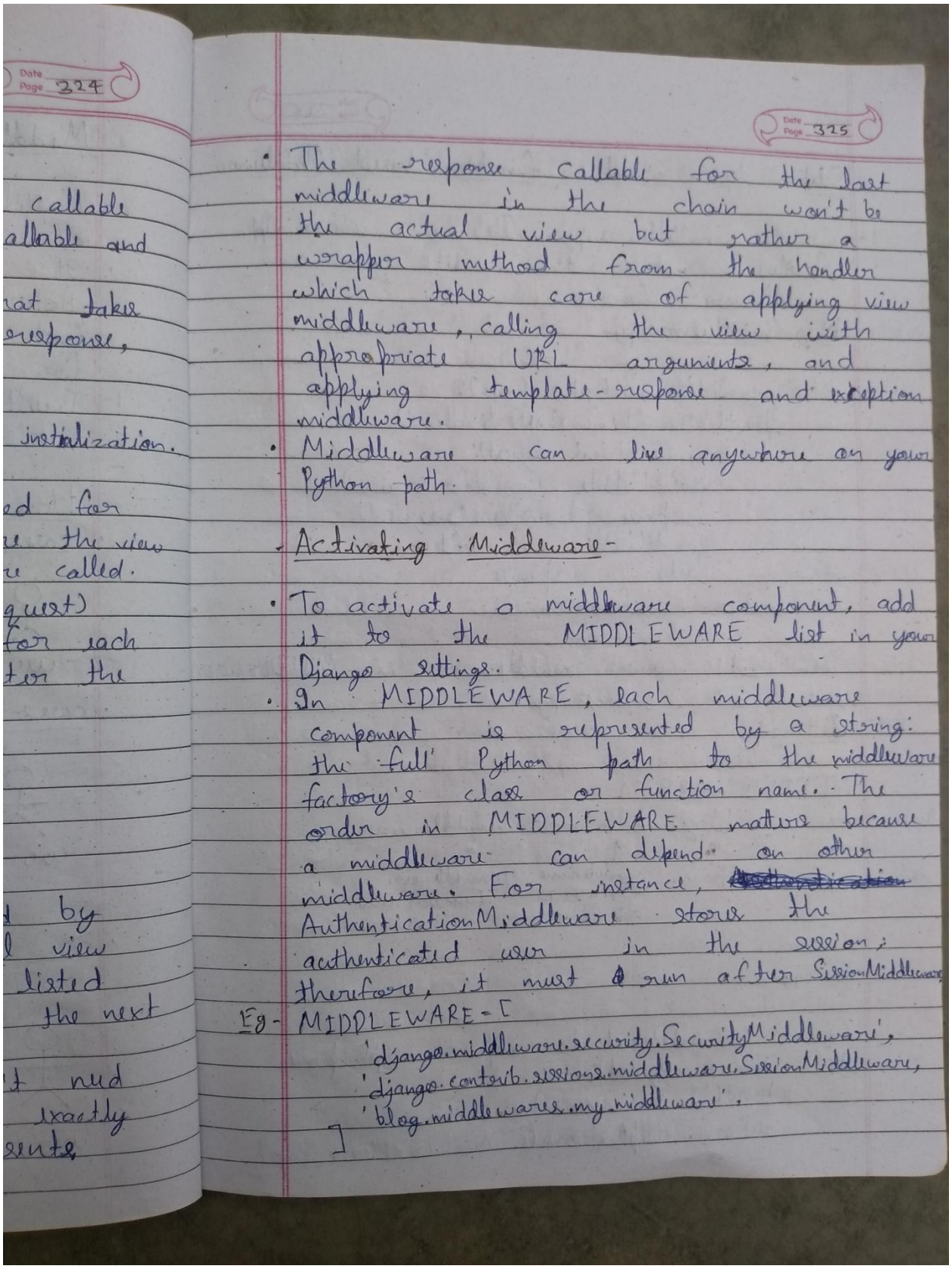
1- Function based Middleware -

- A middleware factory is a callable that take a get_response callable and returns a middleware.
- A middleware is a callable that take a request and returns a response, just like a view.

Syntax - def my_middleware(get_response):
    # One-time configration and initialization.
    def my_function(request):
        # Code to be executed for each request before the view are called.
        response = get_response(request)
        # Code to be executed for each request/response after the view is called.
        return response
    return my_function.

- get_response() -

- The get_response callable provided by Django might be the actual view (if this is the last listed middleware) or it might be the next middleware in the chain.
- The current middleware doesn't need to know or care what exactly it is, just that it represents whatever comes next.



Date Page 324

callable
allable and
at takes
response,
initialization.
ed for
e the view
e called.
quest)
for each
ter the
d by
l view
listed
the next
t nud
exactly
ente

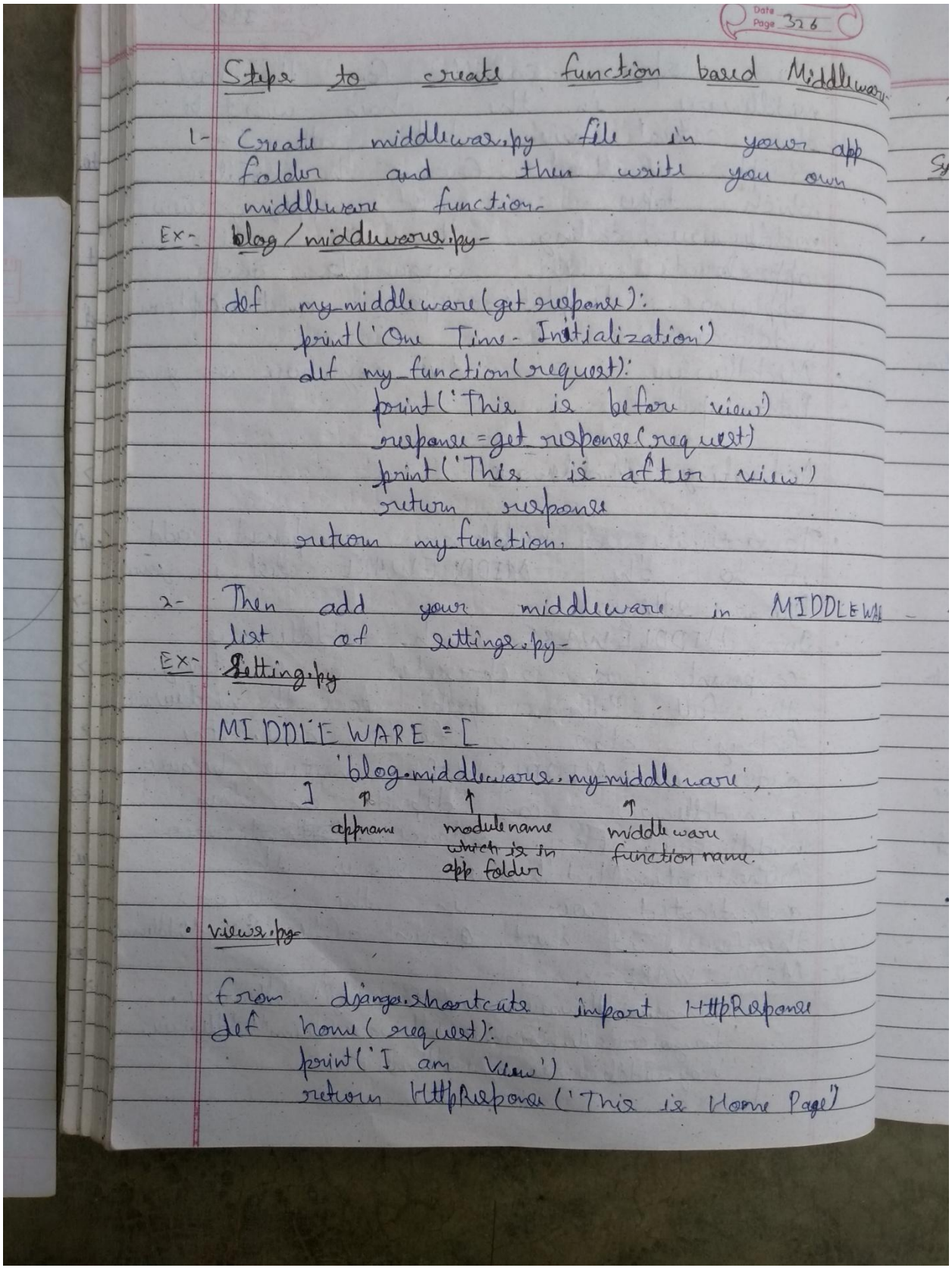
Date Page 325

- The response callable for the last middleware in the chain won't be the actual view but rather a wrapper method from the handler which takes care of applying view middleware, calling the view with appropriate URL arguments, and applying template-response and exception middleware.
- Middleware can live anywhere on your Python path.

Activating Middleware-

- To activate a middleware component, add it to the MIDDLEWARE list in your Django settings.
- In MIDDLEWARE, each middleware component is represented by a string: the full Python path to the middleware factory's class or function name. The order in MIDDLEWARE matters because a middleware can depend on other middleware. For instance, ~~Authentication~~ AuthenticationMiddleware stores the authenticated user in the session; therefore, it must run after SessionMiddleware.

Eg- MIDDLEWARE = [
'django.middleware.security.SecurityMiddleware',
'django.contrib.sessions.middleware.SessionMiddleware',
'blog.middlewares.my_middleware',
]



Date Page 326

Steps to create function based Middleware

1- Create middleware.py file in your app folder and then write you own middleware function.

Ex- blog/middleware.py-

```
def my_middleware(get_response):
    print('One Time - Initialization')
    def my_function(request):
        print('This is befor view')
        response = get_response(request)
        print('This is after view')
        return response
    return my_function.
```

2- Then add your middleware in MIDDLEWARE list of settings.py-

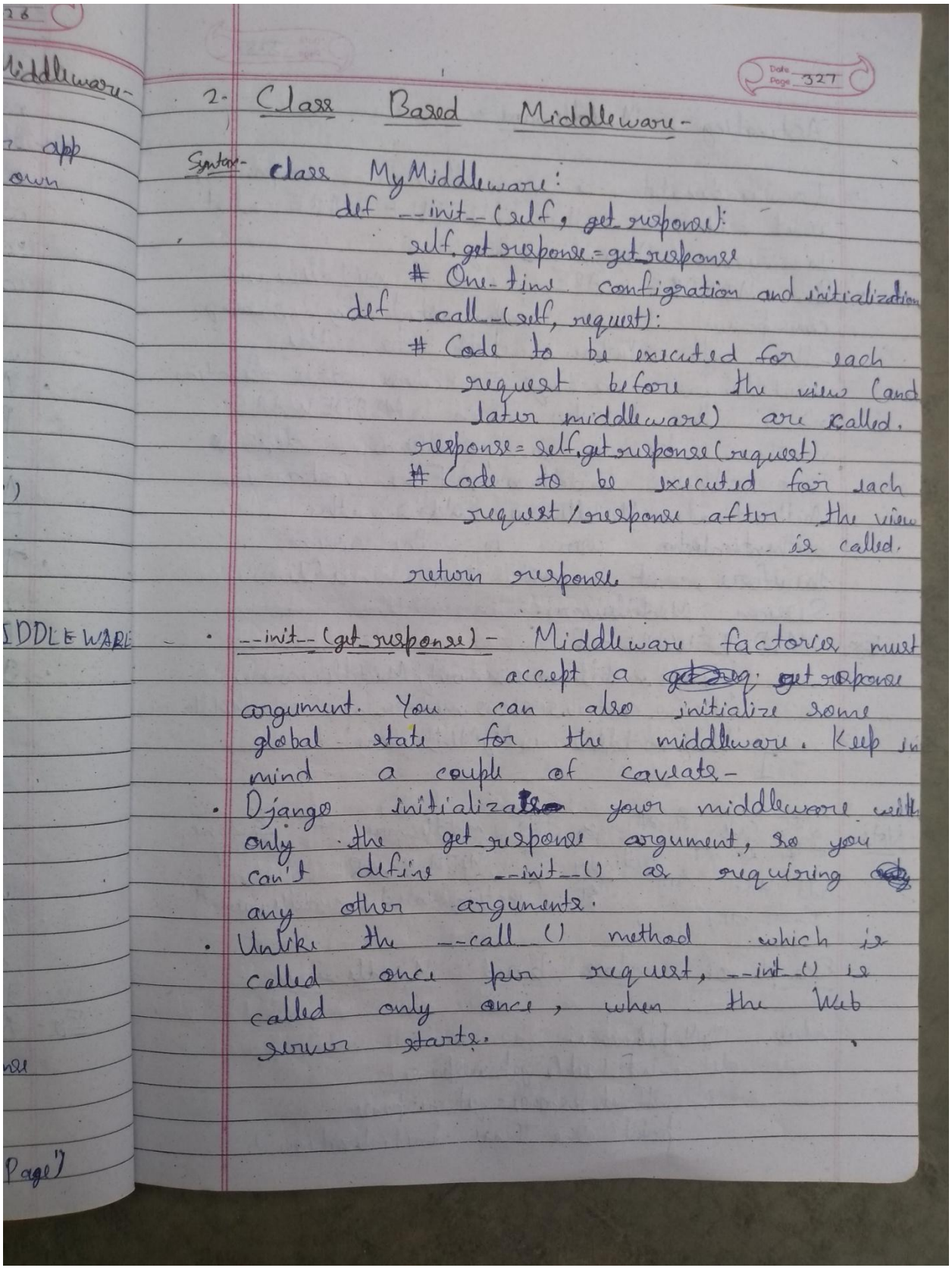
Ex- Setting.py

```
MIDDLEWARE = [
    'blog.middleware.my_middleware',
]
```

↑ appname ↑ module name which is in app folder ↑ middleware function name.

- views.py

```
from django.shortcuts import HttpResponse
def home(request):
    print('I am View')
    return HttpResponse('This is Home Page')
```

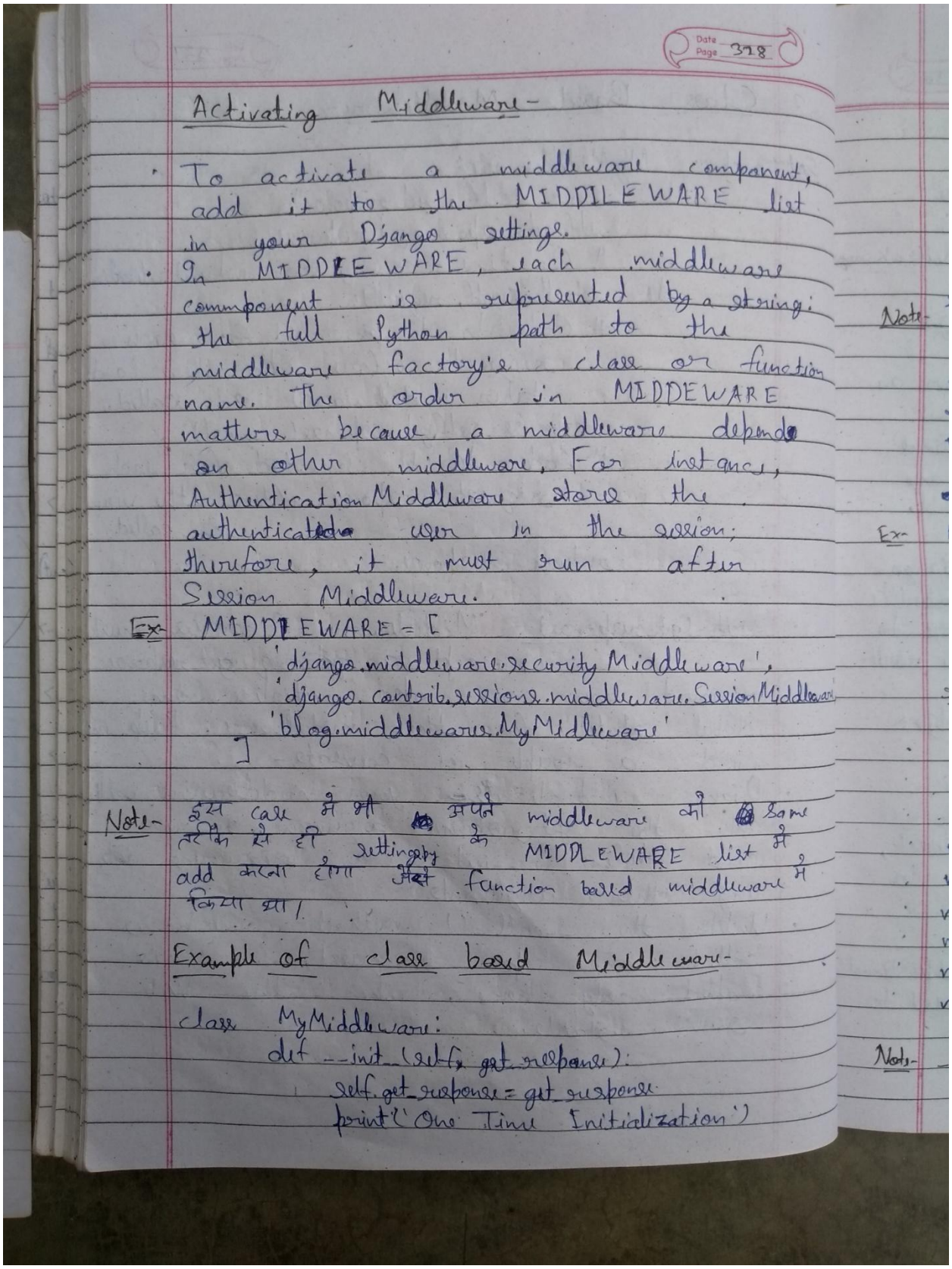


2. Class Based Middleware -

Syntax -

```
class MyMiddleware:
    def __init__(self, get_response):
        self.get_response = get_response
        # One-time configuration and initialization
    def __call__(self, request):
        # Code to be executed for each
          request before the view (and
          later middleware) are called.
        response = self.get_response(request)
        # Code to be executed for each
          request/response after the view
                                  is called.
        return response
```

- __init__(get_response) - Middleware factories must accept a get_response argument. You can also initialize some global state for the middleware. Keep in mind a couple of caveats -
- Django initializes your middleware with only the get_response argument, so you can't define __init__() as requiring any other arguments.
- Unlike the __call__() method which is called once per request, __init__() is called only once, when the Web server starts.



Date Page 378

Activating Middleware -

- To activate a middleware component, add it to the MIDDLEWARE list in your Django settings.
- In MIDDLEWARE, each middleware component is represented by a string: the full Python path to the middleware factory's class or function name. The order in MIDDLEWARE matters because a middleware depends on other middleware. For instance, AuthenticationMiddleware stores the authenticated user in the session; therefore, it must run after SessionMiddleware.

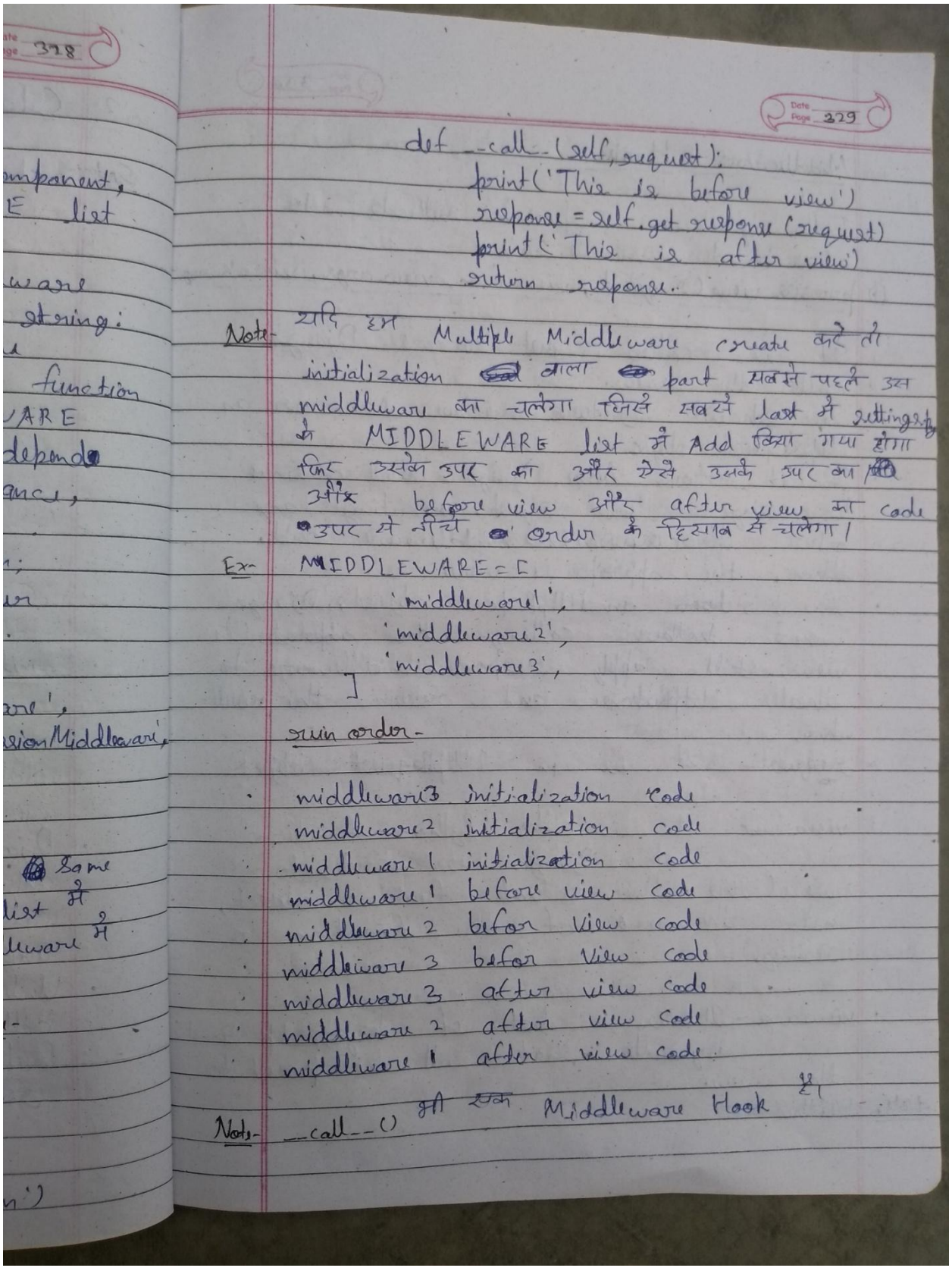
Ex-
```
MIDDLEWARE = [
    'django.middleware.security.SecurityMiddleware',
    'django.contrib.sessions.middleware.SessionMiddleware',
    'blog.middlewares.MyMiddleware'
]
```

Note- इस case में मैं अपने middleware को same तरीके से ही settings के MIDDLEWARE list में add करना होगा जैसे function based middleware में किया था।

Example of class based Middleware -

```
class MyMiddleware:
    def __init__(self, get_response):
        self.get_response = get_response
        print('One Time Initialization')
```

Note-



def __call__(self, request):
    print('This is before view')
    response = self.get_response(request)
    print('This is after view')
    return response.

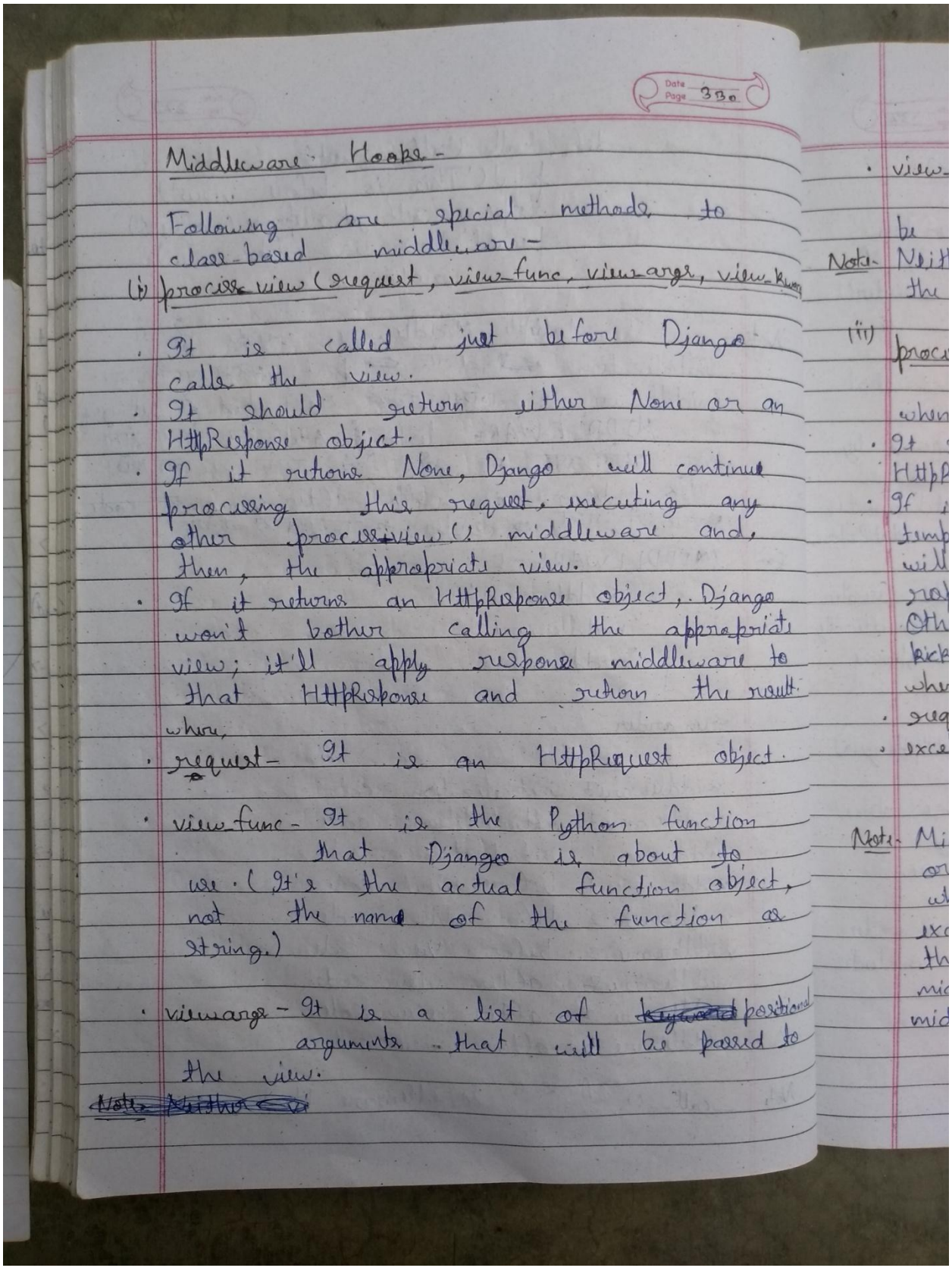
Note- यदि हम Multiple Middleware create करें तो initialization वाला part सबसे पहले उस middleware का चलेगा जिसे सबसे last में settings के MIDDLEWARE list में Add किया गया होगा फिर उसके ऊपर का और ऐसे उसके ऊपर का और before view और after view का code ऊपर से नीचे order के हिसाब से चलेगा।

Ex- MIDDLEWARE = [
    'middleware1',
    'middleware2',
    'middleware3',
]

Run order-

middleware3 initialization code
middleware2 initialization code
middleware1 initialization code
middleware1 before view code
middleware2 befor view code
middleware3 befor view code
middleware3 after view code
middleware2 after view code
middleware1 after view code

Note- __call__() is a Middleware Hook



Middleware Hooks -

Following are special methods to class based middleware -

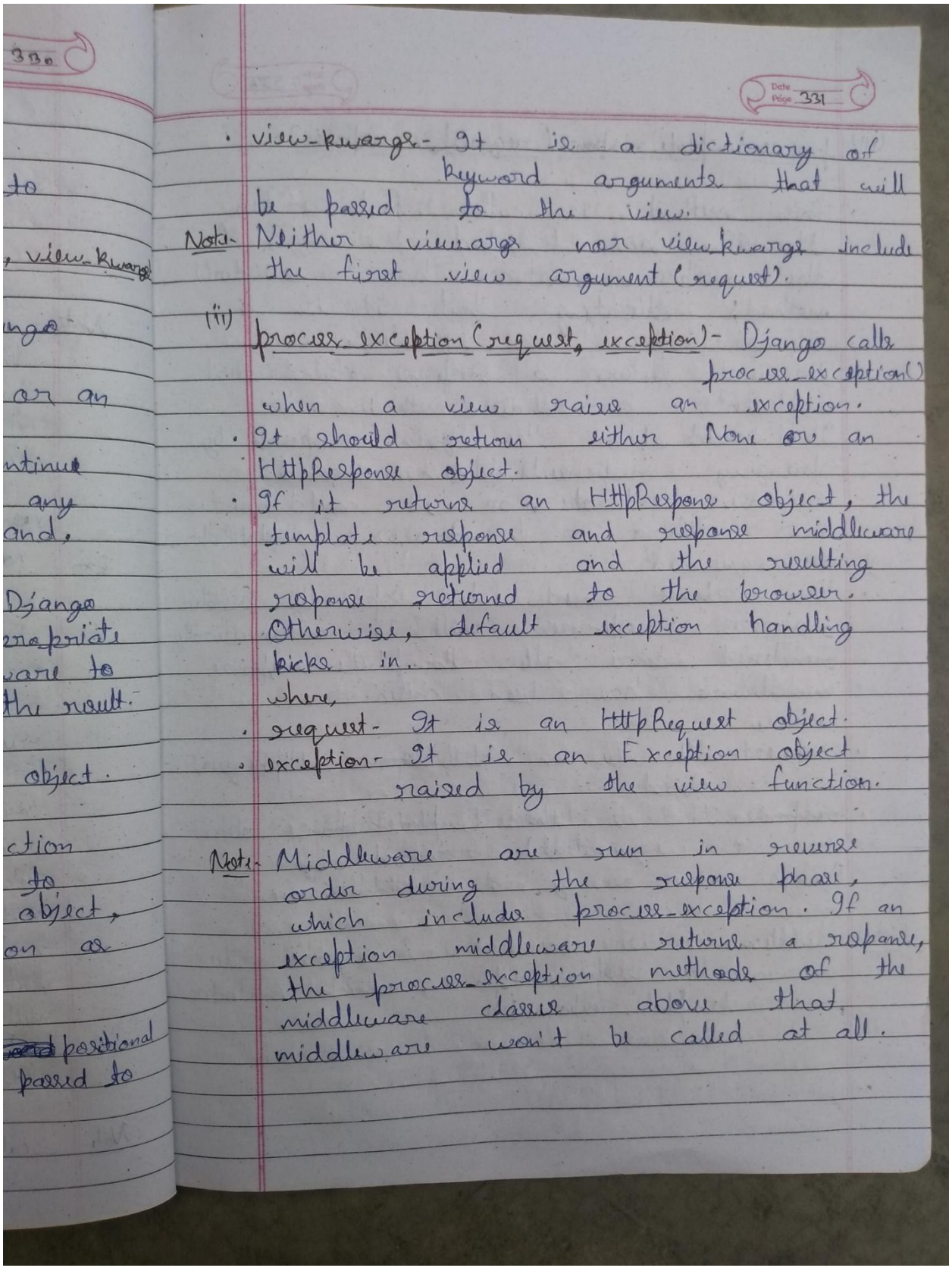
(i) process_view(request, view_func, view_args, view_kwargs)

- It is called just before Django calls the view.
- It should return either None or an HttpResponse object.
- If it returns None, Django will continue processing this request, executing any other process_view() middleware and, then, the appropriate view.
- If it returns an HttpResponse object, Django won't bother calling the appropriate view; it'll apply response middleware to that HttpResponse and return the result.

where,

- request - It is an HttpRequest object.
- view_func - It is the Python function that Django is about to use. (It's the actual function object, not the name of the function as string.)
- view_args - It is a list of ~~keyword~~ positional arguments that will be passed to the view.

~~Note: Neither~~



- view_kwargs - It is a dictionary of keyword arguments that will be passed to the view.

Note- Neither view_args nor view_kwargs include the first view argument (request).

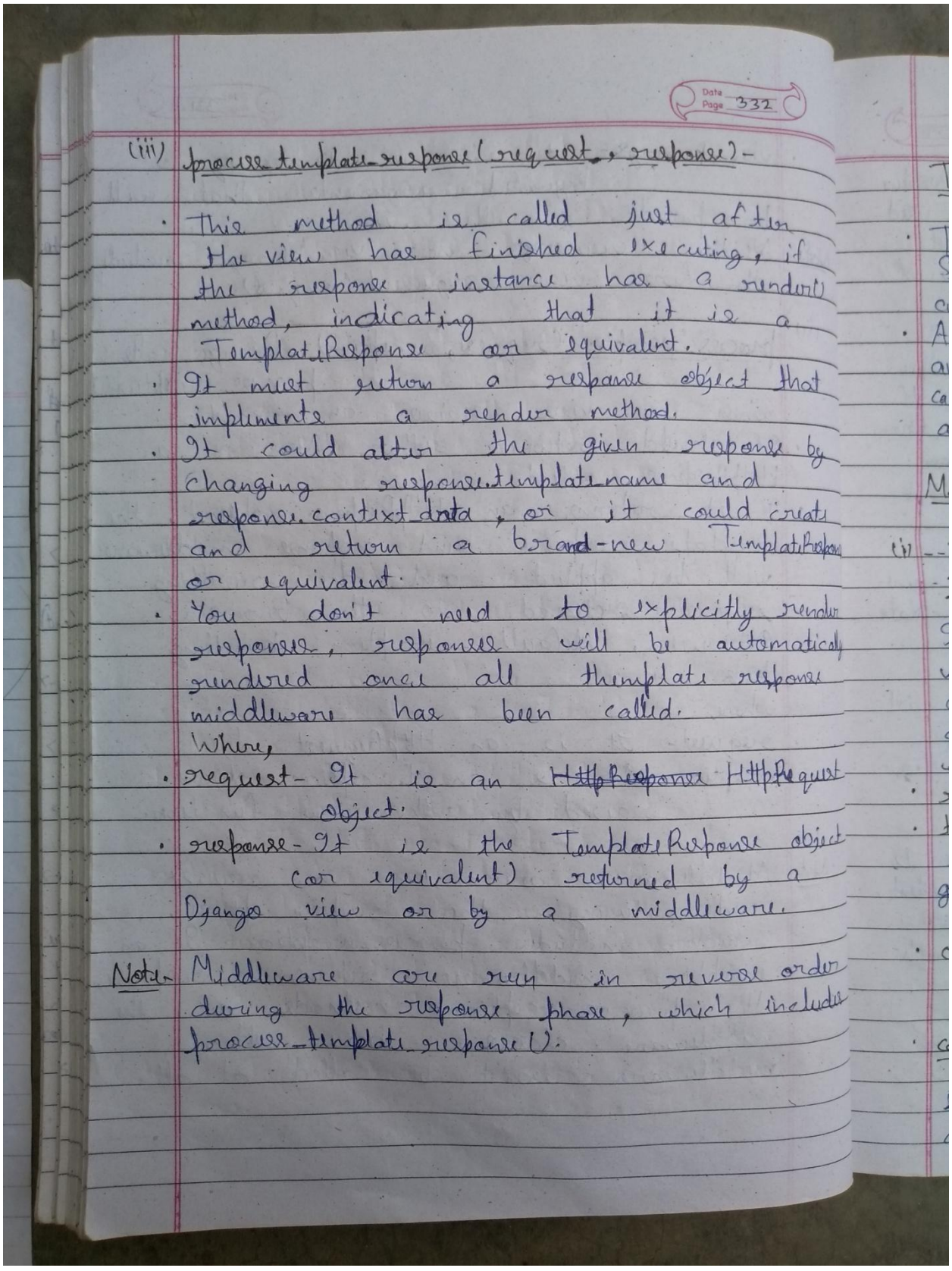
(iii) process_exception(request, exception) - Django calls process_exception() when a view raises an exception.

- It should return either None or an HttpResponse object.
- If it returns an HttpResponse object, the template response and response middleware will be applied and the resulting response returned to the browser. Otherwise, default exception handling kicks in.

where,

- request - It is an HttpRequest object.
- exception - It is an Exception object raised by the view function.

Note- Middleware are run in reverse order during the response phase, which includes process_exception. If an exception middleware returns a response, the process_exception methods of the middleware classes above that middleware won't be called at all.



Date Page 331

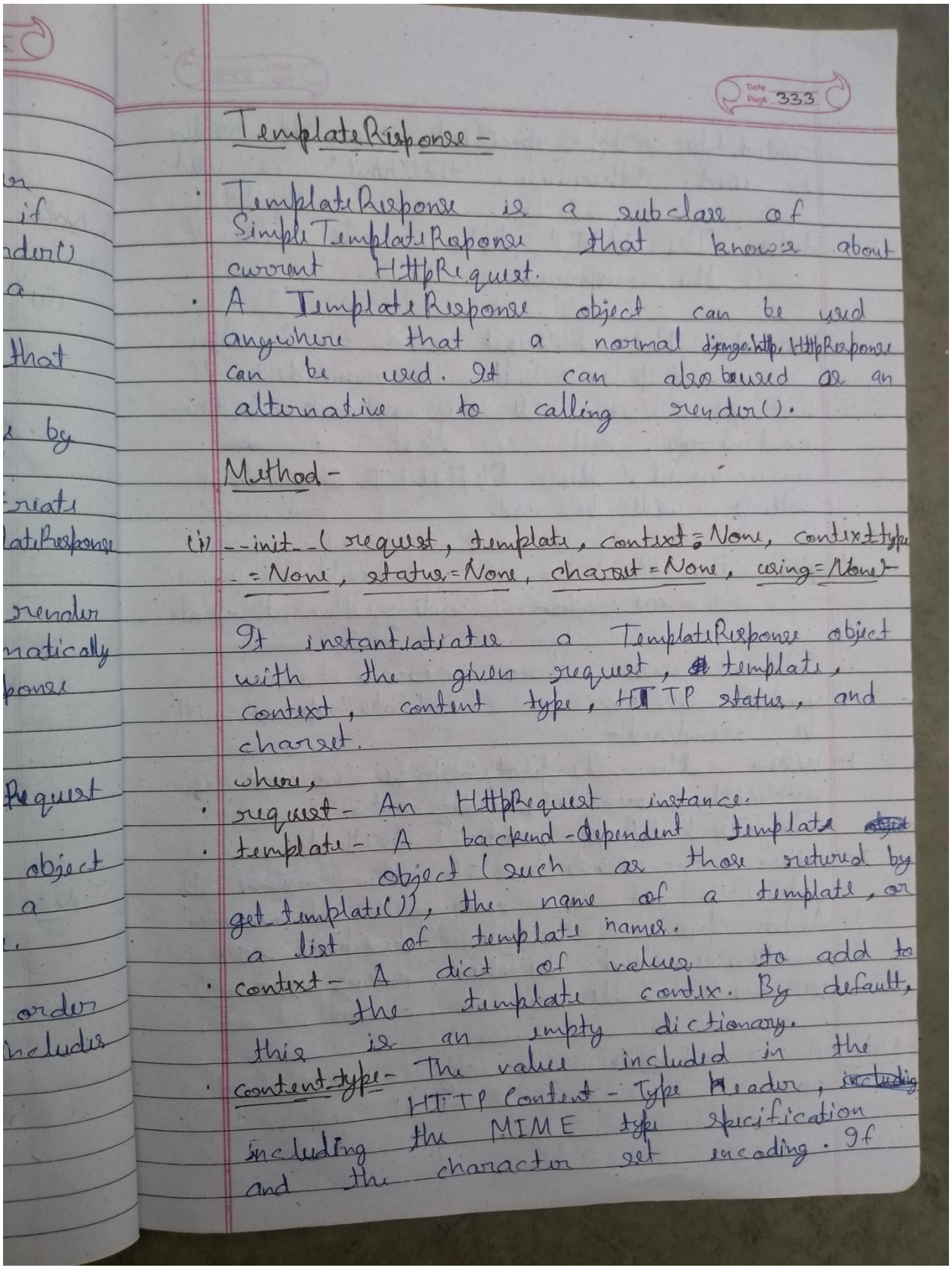
(iii) process_template_response(request, response) -

- This method is called just after the view has finished executing, if the response instance has a render() method, indicating that it is a TemplateResponse or equivalent.
- It must return a response object that implements a render method.
- It could alter the given response by changing response.template_name and response.context_data, or it could create and return a brand-new TemplateResponse or equivalent.
- You don't need to explicitly render responses, responses will be automatically rendered once all the template response middleware has been called.

Where,

- request - It is an ~~HttpResponse~~ HttpRequest object.
- response - It is the TemplateResponse object (or equivalent) returned by a Django view or by a middleware.

Note - Middleware are run in reverse order during the response phase, which includes process_template_response().



Date
Page 333

TemplateResponse -

- TemplateResponse is a subclass of SimpleTemplateResponse that knows about current HttpRequest.
- A TemplateResponse object can be used anywhere that a normal django.http.HttpResponse can be used. It can also be used as an alternative to calling render().

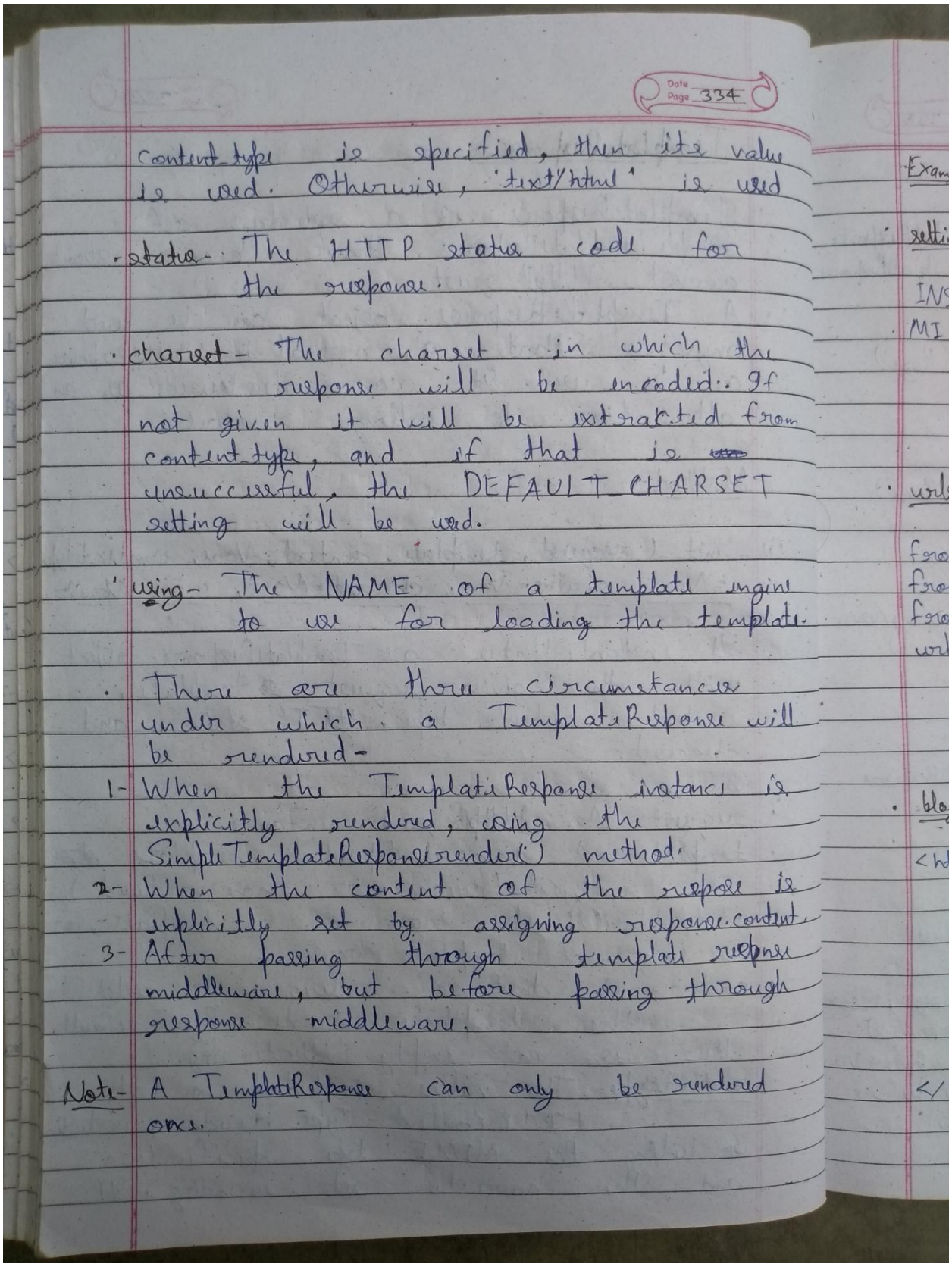
Method -

(i) __init__(request, template, context=None, context_type=None, status=None, charset=None, using=None)

It instantiates a TemplateResponse object with the given request, template, context, content type, HTTP status, and charset.

where,

- request - An HttpRequest instance.
- template - A backend-dependent template object (such as those returned by get_template()), the name of a template, or a list of template names.
- context - A dict of values to add to the template context. By default, this is an empty dictionary.
- content_type - The value included in the HTTP Content-Type header, including the MIME type specification and the character set encoding. If



content_type is specified, then its value is used. Otherwise, 'text/html' is used.

- status - The HTTP status code for the response.

- charset - The charset in which the response will be encoded. If not given it will be extracted from content_type, and if that is unsuccessful, the DEFAULT_CHARSET setting will be used.

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

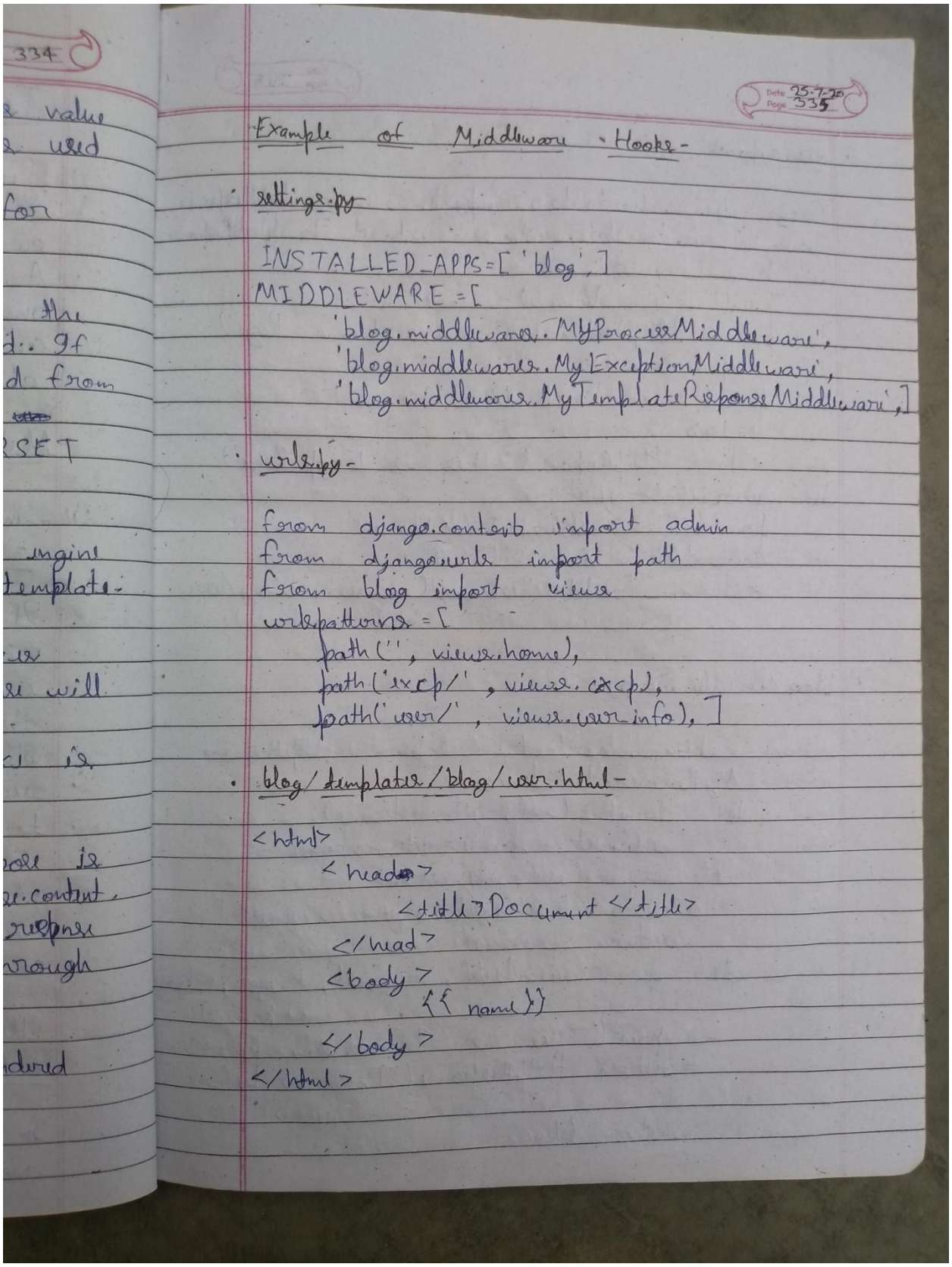
- There are three circumstances under which a TemplateResponse will be rendered -

1- When the TemplateResponse instance is explicitly rendered, using the SimpleTemplateResponse.render() method.

2- When the content of the response is explicitly set by assigning response.content.

3- After passing through template response middleware, but before passing through response middleware.

Note- A TemplateResponse can only be rendered once.



Example of Middleware -Hooks-

- settings.py

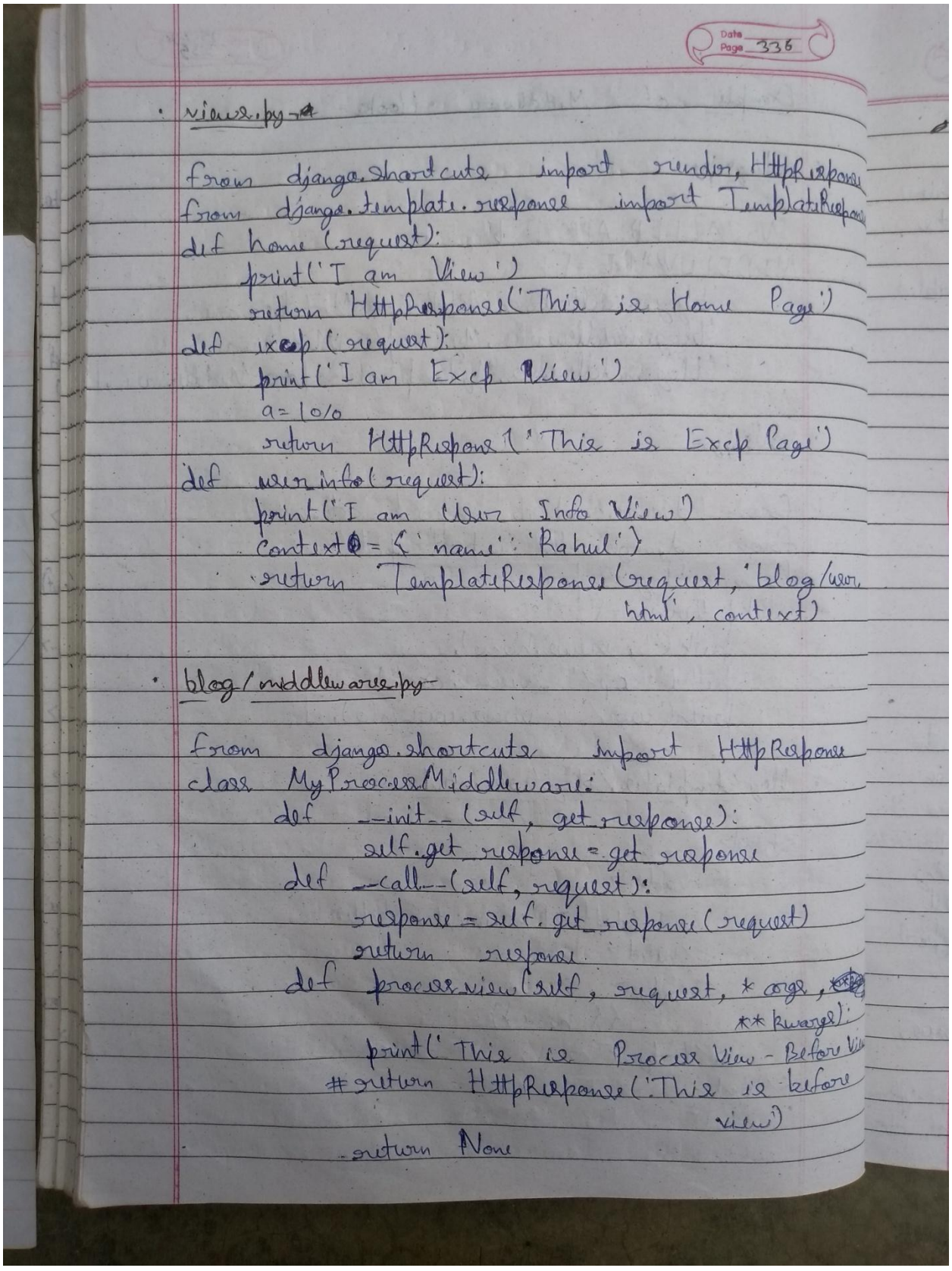
```
INSTALLED_APPS=[ 'blog', ]
MIDDLEWARE =[
        'blog.middlewares.MyProcessMiddleware',
        'blog.middlewares.MyExceptionMiddleware',
        'blog.middlewares.MyTemplateResponseMiddleware',]
```

- urls.py -

```
from django.contrib import admin
from django.urls import path
from blog import views
urlpatterns = [
    path('', views.home),
    path('excp/', views.excp),
    path('user/', views.user_info), ]
```

- blog/templates/blog/user.html -

```
<html>
    <head>
        <title>Document</title>
    </head>
    <body>
        {{ name }}
    </body>
</html>
```



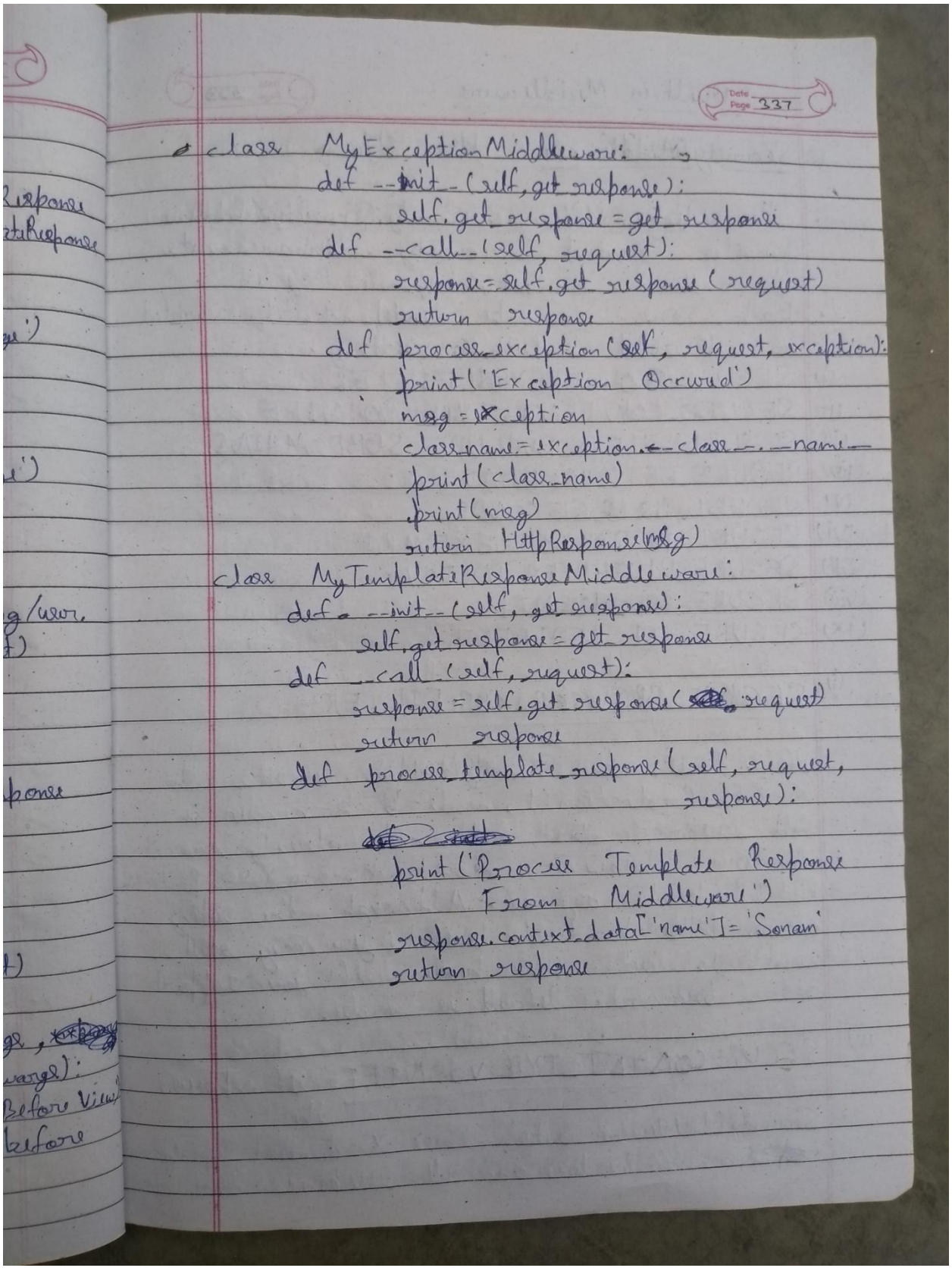
Date
Page 336

- views.py

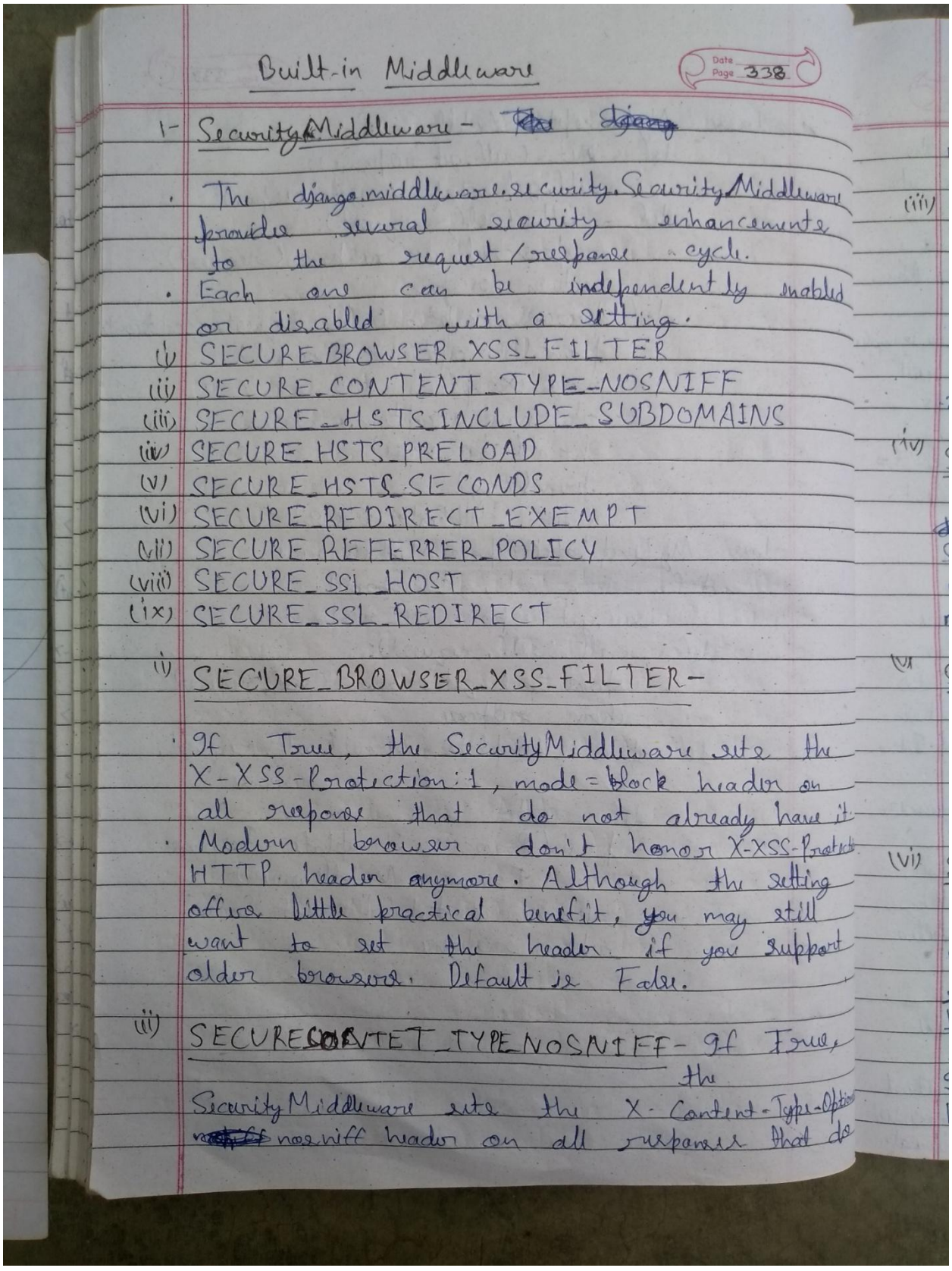
```
from django.shortcuts import render, HttpResponse
from django.template.response import TemplateResponse
def home(request):
    print('I am View')
    return HttpResponse('This is Home Page')
def excp(request):
    print('I am Excp View')
    a=10/0
    return HttpResponse('This is Excp Page')
def user_info(request):
    print('I am User Info View')
    context={'name':'Rahul'}
    return TemplateResponse(request,'blog/user.
                            html', context)
```

- blog/middlewares.py

```
from django.shortcuts import HttpResponse
class MyProcessMiddleware:
    def __init__(self, get_response):
        self.get_response=get_response
    def __call__(self, request):
        response=self.get_response(request)
        return response
    def process_view(self, request, *args,
                     **kwargs):
        print('This is Process View - Before Vie
        #return HttpResponse('This is before
                             view')
        return None
```



```
class MyExceptionMiddleware:
    def __init__(self, get_response):
        self.get_response = get_response
    def __call__(self, request):
        response = self.get_response(request)
        return response
    def process_exception(self, request, exception):
        print('Exception Occured')
        msg = exception
        class_name = exception.__class__.__name__
        print(class_name)
        print(msg)
        return HttpResponse(msg)
class MyTemplateResponseMiddleware:
    def __init__(self, get_response):
        self.get_response = get_response
    def __call__(self, request):
        response = self.get_response(request)
        return response
    def process_template_response(self, request,
                                  response):
        print('Process Template Response
              From Middleware')
        response.context_data['name'] = 'Sonam'
        return response
```



Built-in Middleware

Date Page 338

1- SecurityMiddleware -

- The django.middleware.security.SecurityMiddleware provides several security enhancements to the request/response cycle.
- Each one can be independently enabled or disabled with a setting.

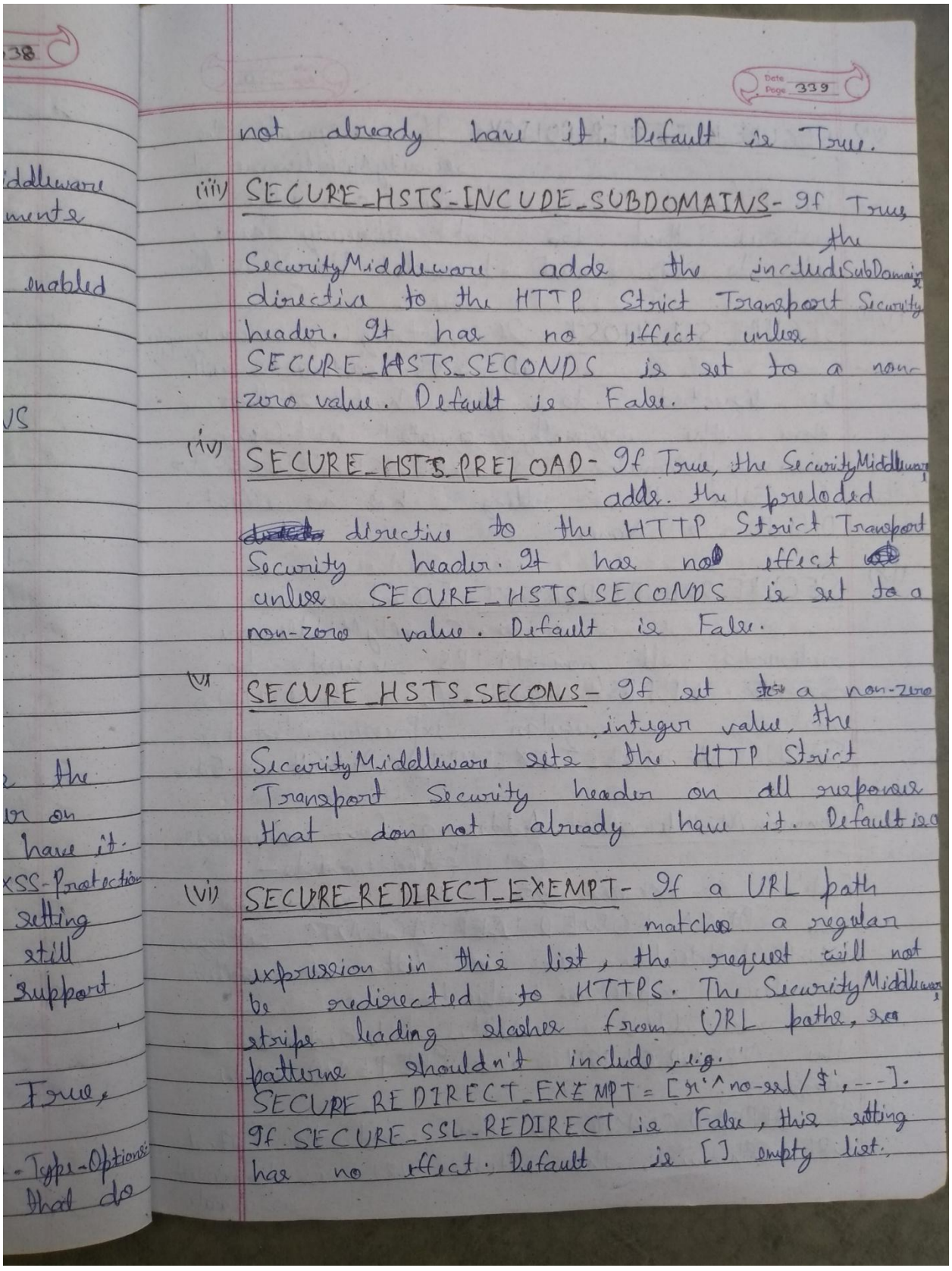
(i) SECURE_BROWSER_XSS_FILTER
(ii) SECURE_CONTENT_TYPE_NOSNIFF
(iii) SECURE_HSTS_INCLUDE_SUBDOMAINS
(iv) SECURE_HSTS_PRELOAD
(v) SECURE_HSTS_SECONDS
(vi) SECURE_REDIRECT_EXEMPT
(vii) SECURE_REFERRER_POLICY
(viii) SECURE_SSL_HOST
(ix) SECURE_SSL_REDIRECT

(i) SECURE_BROWSER_XSS_FILTER-

- If True, the SecurityMiddleware sets the X-XSS-Protection: 1, mode=block header on all response that do not already have it.
- Modern browser don't honor X-XSS-Protection HTTP header anymore. Although the setting offers little practical benefit, you may still want to set the header if you support older browsers. Default is False.

(ii) SECURE_CONTENT_TYPE_NOSNIFF- If True, the

SecurityMiddleware sets the X-Content-Type-Options nosniff header on all responses that do



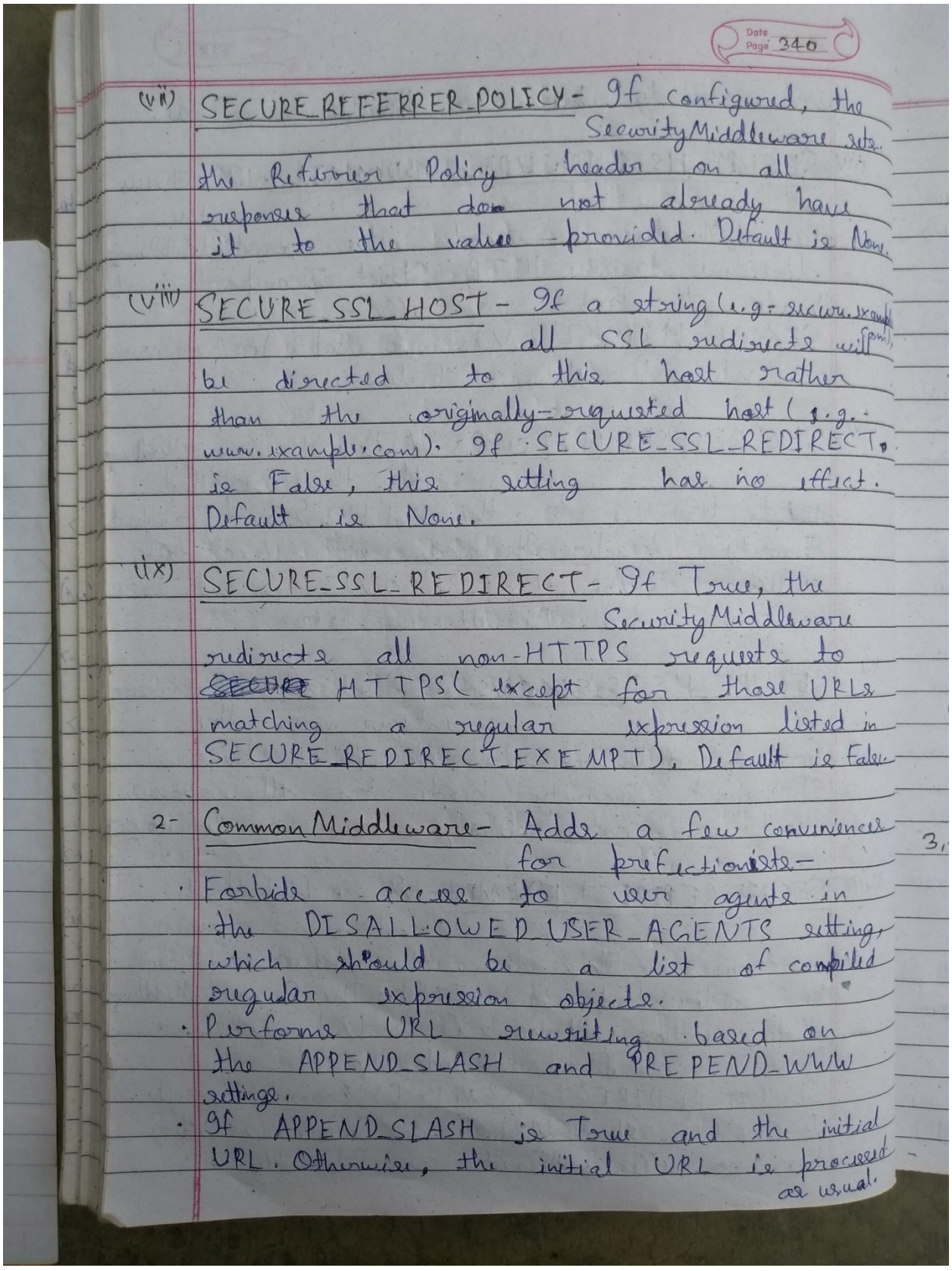
not already have it. Default is True.

(iii) SECURE_HSTS_INCUDE_SUBDOMAINS - If True, the SecurityMiddleware adds the includeSubDomains directive to the HTTP Strict Transport Security header. It has no effect unless SECURE_HSTS_SECONDS is set to a non-zero value. Default is False.

(iv) SECURE_HSTS_PRELOAD - If True, the SecurityMiddleware adds the preloaded directive to the HTTP Strict Transport Security header. It has no effect unless SECURE_HSTS_SECONDS is set to a non-zero value. Default is False.

(v) SECURE_HSTS_SECONS - If set to a non-zero integer value, the SecurityMiddleware sets the HTTP Strict Transport Security header on all responses that don not already have it. Default is 0

(vi) SECURE_REDIRECT_EXEMPT - If a URL path matches a regular expression in this list, the request will not be redirected to HTTPS. The SecurityMiddleware strips leading slashes from URL paths, so patterns shouldn't include /. e.g. SECURE_REDIRECT_EXEMPT = [r'^no-ssl/$', ...]. If SECURE_SSL_REDIRECT is False, this setting has no effect. Default is [] empty list.



Date Page 340

(vii) SECURE_REFERRER_POLICY - If configured, the SecurityMiddleware sets the Referrer Policy header on all responses that don't already have it to the value provided. Default is None.

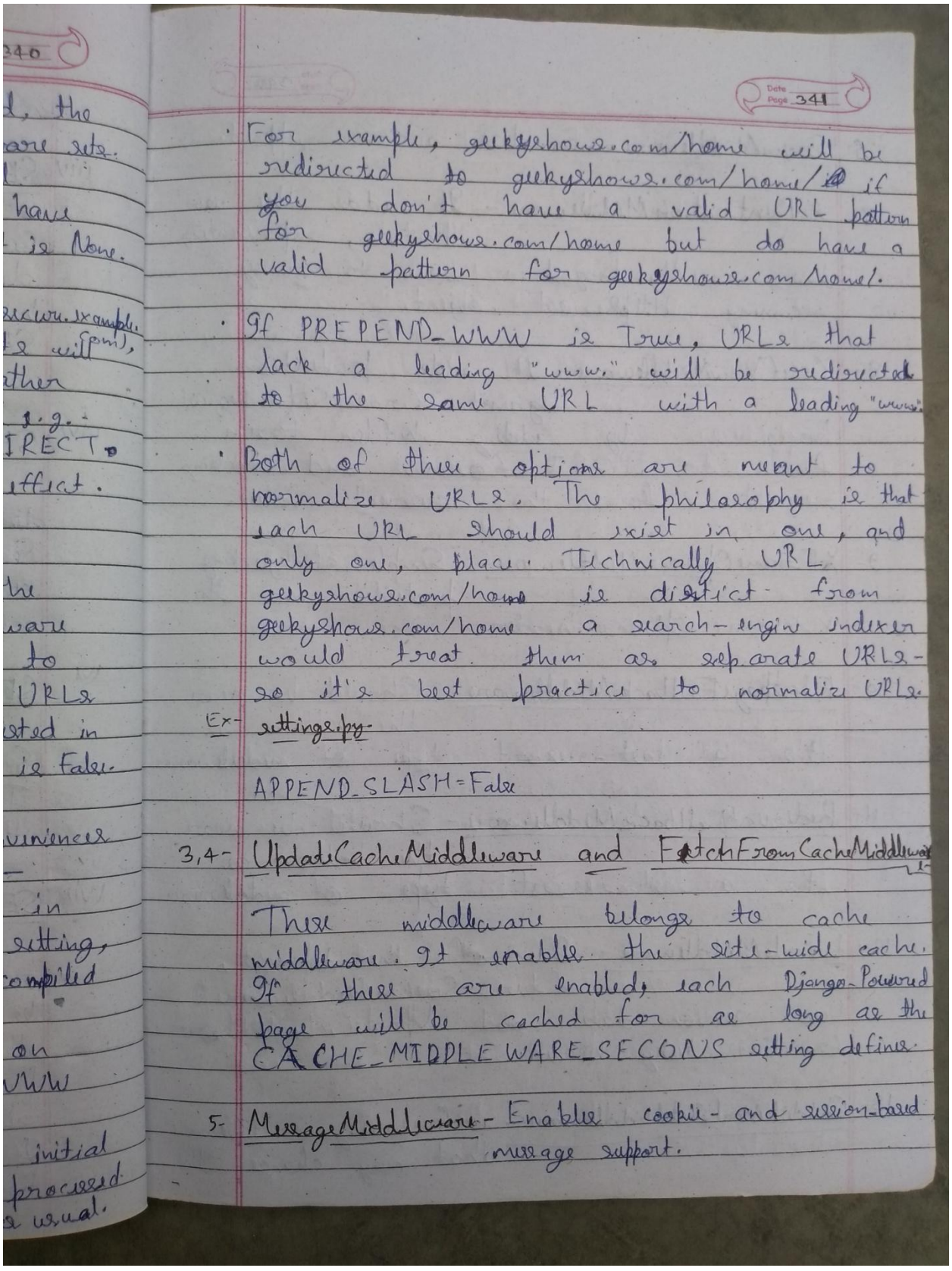
(viii) SECURE_SSL_HOST - If a string (e.g. secure.example.com), all SSL redirects will be directed to this host rather than the originally-requested host (e.g. www.example.com). If SECURE_SSL_REDIRECT is False, this setting has no effect. Default is None.

(ix) SECURE_SSL_REDIRECT - If True, the SecurityMiddleware redirects all non-HTTPS requests to ~~SECURE~~ HTTPS (except for those URLs matching a regular expression listed in SECURE_REDIRECT_EXEMPT). Default is False.

2- CommonMiddleware - Adds a few conveniences for perfectionists -

- Forbids access to user agents in the DISALLOWED_USER_AGENTS setting, which should be a list of compiled regular expression objects.
- Performs URL rewriting based on the APPEND_SLASH and PREPEND_WWW settings.
- If APPEND_SLASH is True and the initial URL. Otherwise, the initial URL is processed as usual.

3,



340

d, the
are site.
l
have
is None.

www.example.
le will [com],
ther
e.g.
IRECT
effect.

the
ware
to
URLs
ted in
is False.

viniences
in
setting,
compiled

on
WW

initial
processed
s usual.

Date
Page 341

· For example, geekyshows.com/home will be redirected to geekyshows.com/home/ if you don't have a valid URL pattern for geekyshows.com/home but do have a valid pattern for geekyshows.com/home/.

· If PREPEND_WWW is True, URLs that lack a leading "www." will be redirected to the same URL with a leading "www."

· Both of these options are meant to normalize URLs. The philosophy is that each URL should exist in one, and only one, place. Technically URL geekyshows.com/home is distinct from geekyshows.com/home a search-engine indexer would treat them as separate URLs - so it's best practice to normalize URLs.

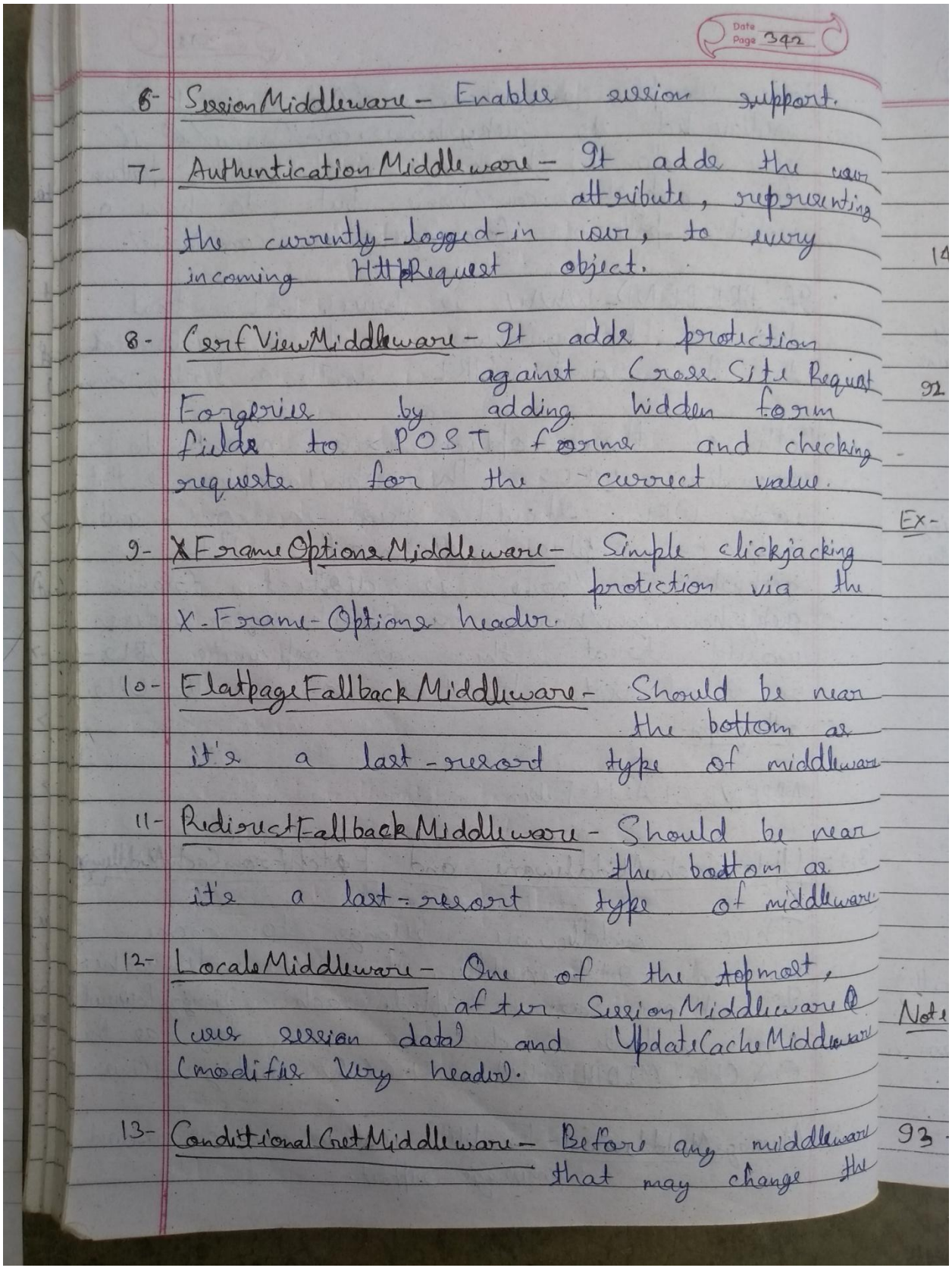
Ex- settings.py

APPEND_SLASH=False

3,4- UpdateCacheMiddleware and FetchFromCacheMiddleware

These middleware belongs to cache middleware. It enable the site-wide cache. If these are enabled, each Django-Powered page will be cached for as long as the CACHE_MIDDLEWARE_SECONDS setting defines.

5- MessageMiddleware - Enable cookie- and session-based message support.



6- SessionMiddleware - Enables session support.

7- AuthenticationMiddleware - It adds the user attribute, representing the currently-logged-in user, to every incoming HttpRequest object.

8- CsrfViewMiddleware - It adds protection against Cross Site Request Forgeries by adding hidden form fields to POST forms and checking requests for the correct value.

9- XFrameOptionsMiddleware - Simple clickjacking protection via the X-Frame-Options header.

10- FlatpageFallbackMiddleware - Should be near the bottom as it's a last-resort type of middleware

11- RedirectFallbackMiddleware - Should be near the bottom as it's a last-resort type of middleware

12- LocaleMiddleware - One of the topmost, after SessionMiddleware (uses session data) and UpdateCacheMiddleware (modifies Vary header).

13- ConditionalGetMiddleware - Before any middleware that may change the

response (it sets the ETag header). After GZipMiddleware so it won't calculate an ETag header on gzipped contents.

14- GZipMiddleware - Before any middleware that may change or use the response body. After UpdateCacheMiddleware: Modifies Vary header.

92 —

Example to Show Under Construction Page - Using Middleware -

Ex- . mysite/middleware.py -

```
from django.shortcuts import render, HttpResponse
class UnderConstructionMiddleware:
    def __init__(self, get_response):
        self.get_response = get_response
    def __call__(self, request):
        print('Call From Middleware Before View')
        #response = HttpResponse('This is Under Construction')
        response = render(request, 'mysite/site.html')
        print('Call Form Middleware After View')
        return response.
```

Note- जब हमें ये Under Construction का Page नहीं दिखाना होता हम settings.py में जाकर अपने custom Middleware को comment या हटा देंगे और जब दिखाना है uncomment या add कर देंगे।

93 —