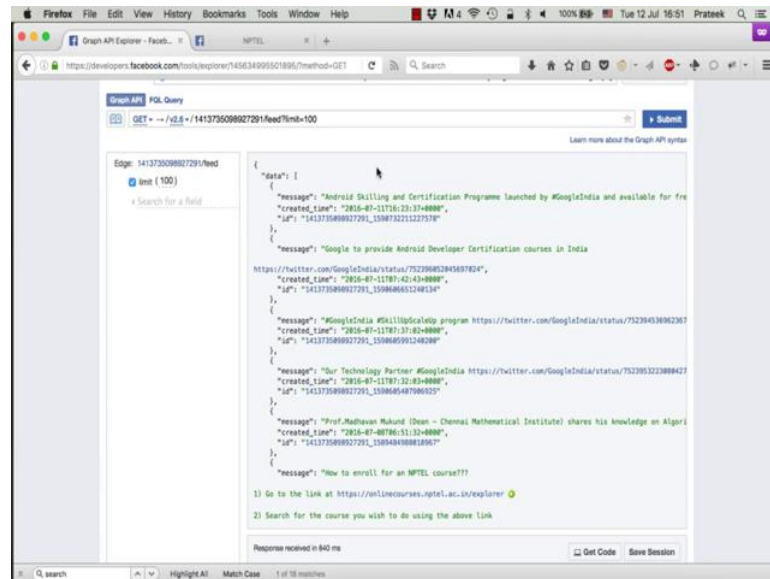**Privacy and Security in Online Social Networks**
Department of Computer Science and Engineering
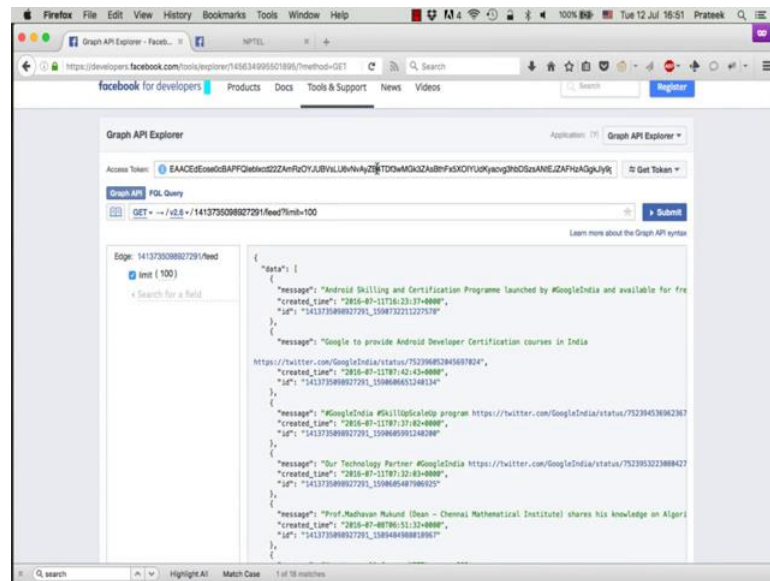**Indian Institute of Technology, Madras**

Lecture – 08
**Tutorial 2, Part 2, Facebook API**

(Refer Slide Time: 00:12)

So, now that we have learned the basics of the graph API using this graph API explorer,

we will now learn how to collect data programmatically. Now one problem in doing this is the lifetime of the access tokens that we are using. If you recall in the last video we discussed that it is essential to have an access token in order to collect data from the graph API. One thing we did not really pay attention to was that these tokens have a very short life span.
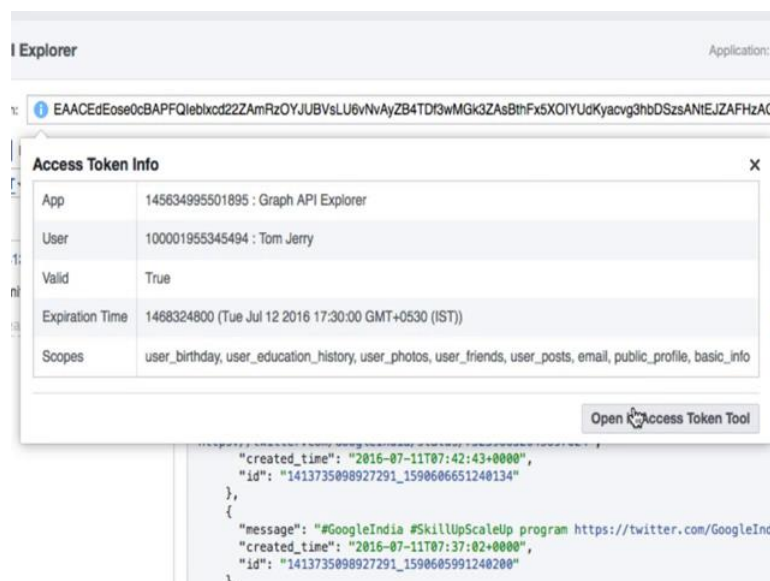
For example, if you open this access token in the access token debugger tool,
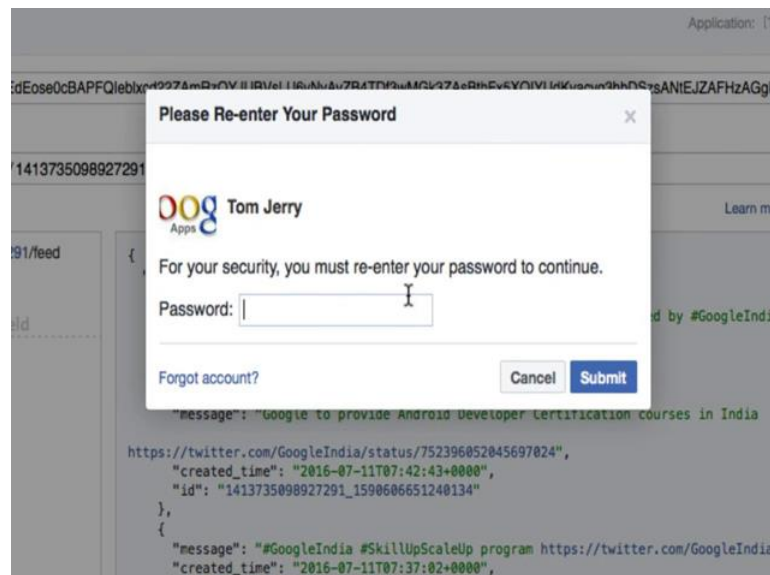
(Refer Slide Time: 00:45)
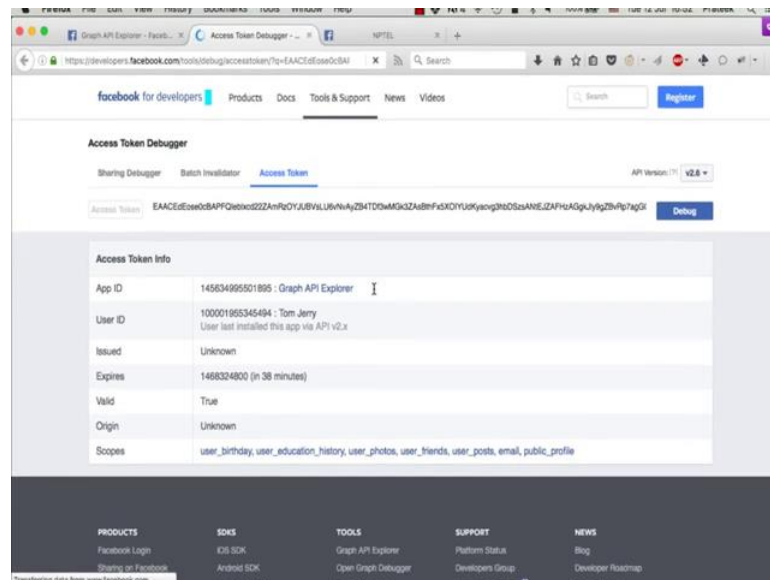
you will notice that this token expires in 38 minutes, which means that if you make a request to the Graph API using this particular token after 38 minutes, the API will not return any data, instead it will return an error saying that this token is invalid or expired. If you intend to collect data programmatically for a prolonged period of time, you cannot keep on generating a new token manually after every one or two hours and put it in the code. So, Facebook provides a way to extend the validity of these tokens, but for that you need to create an app of your own.

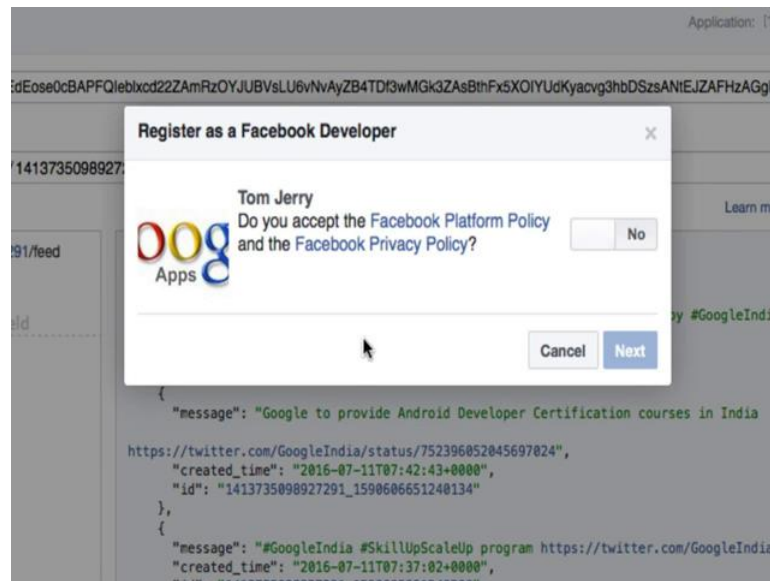(Refer Slide Time: 01:20)
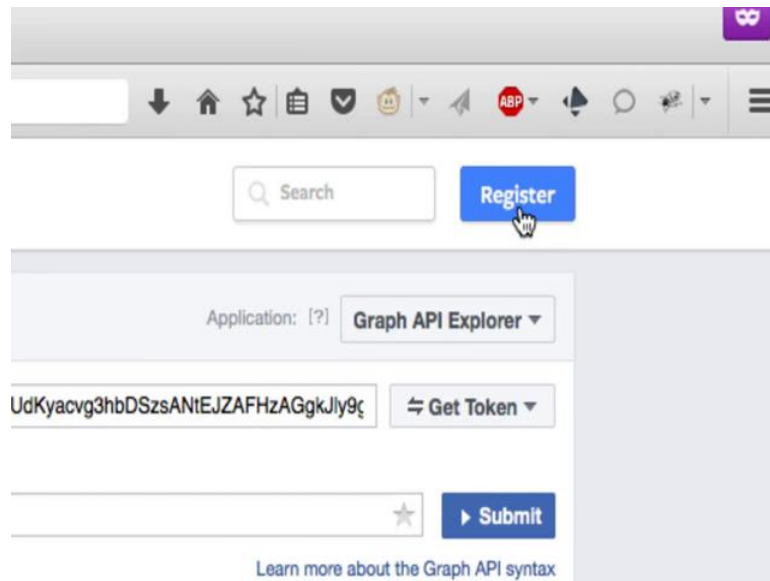
To create an app, you need to first register yourself as a developer; click on the blue register button on the top right of the page.
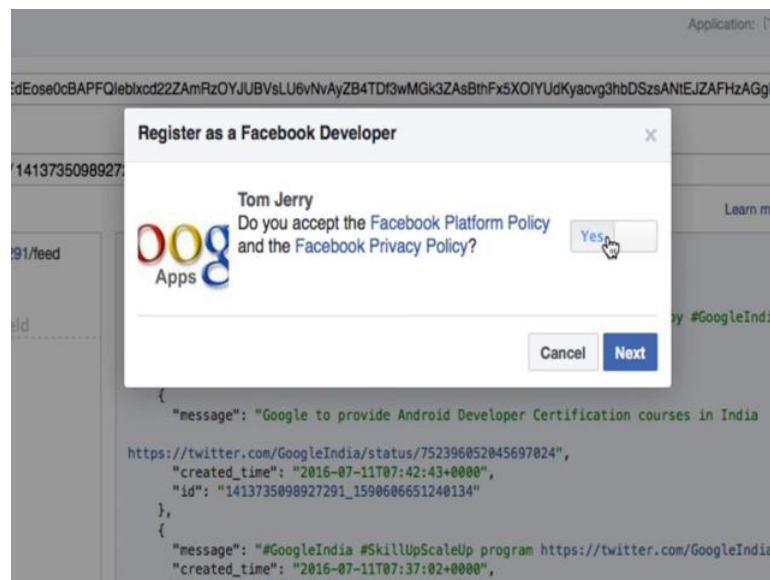
(Refer Slide Time: 01:22)

Facebook will ask you to reenter your password.
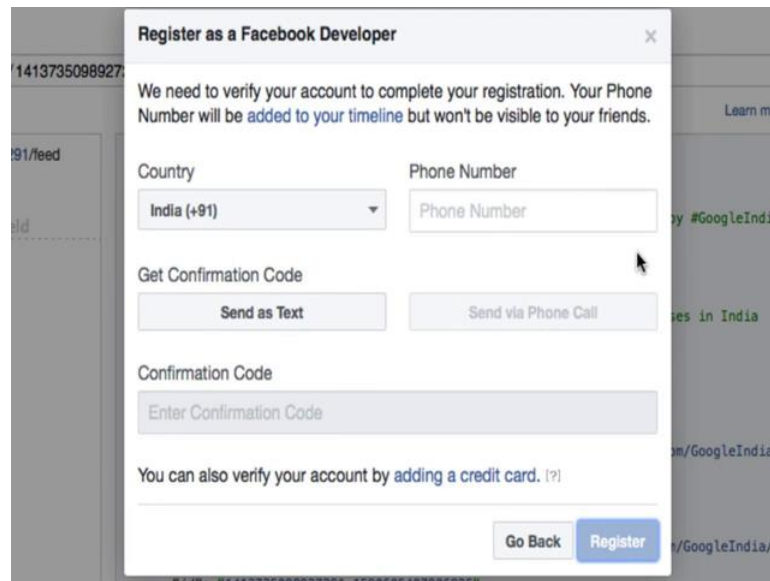
(Refer Slide Time: 01:30)

So, enter your password.
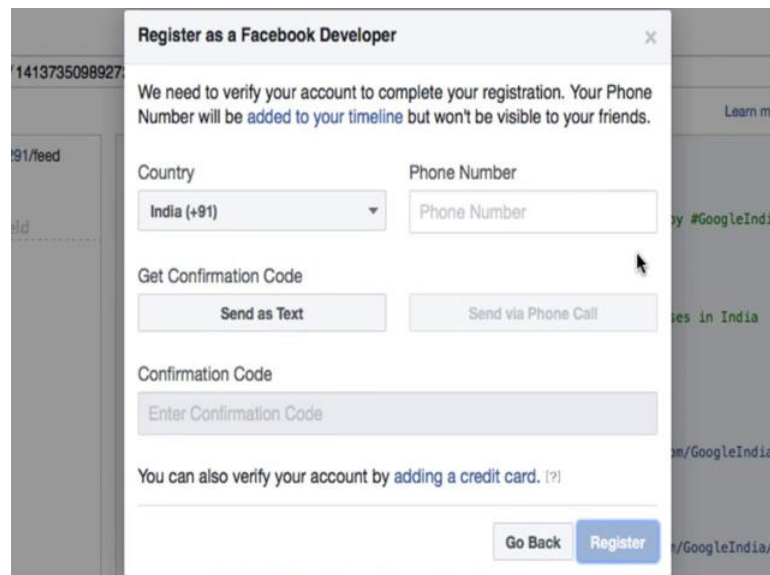
(Refer Slide Time: 01:33)

 Click on this button to accept the privacy policy and click next.

(Refer Slide Time: 01:36)

Now in order to successfully register yourself as a developer, you need to register a phone number with your account. So put in your phone number.

(Refer Slide Time: 01:43)

Then click on the 'Send as Text' button, you will receive a SMS with a ==confirmation== code.
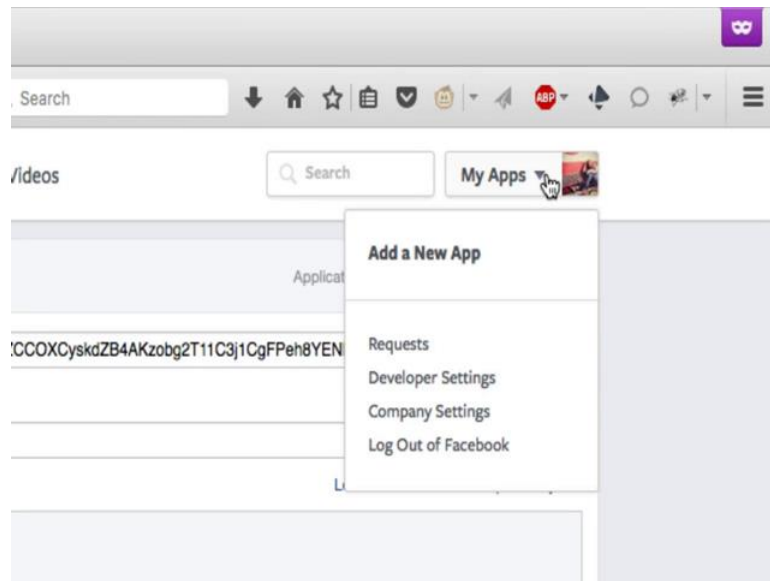
(Refer Slide Time: 01:49)

Put in this <mark>confirmation</mark> code in the designated box, and click 'Register'.

(Refer Slide Time: 01:54)
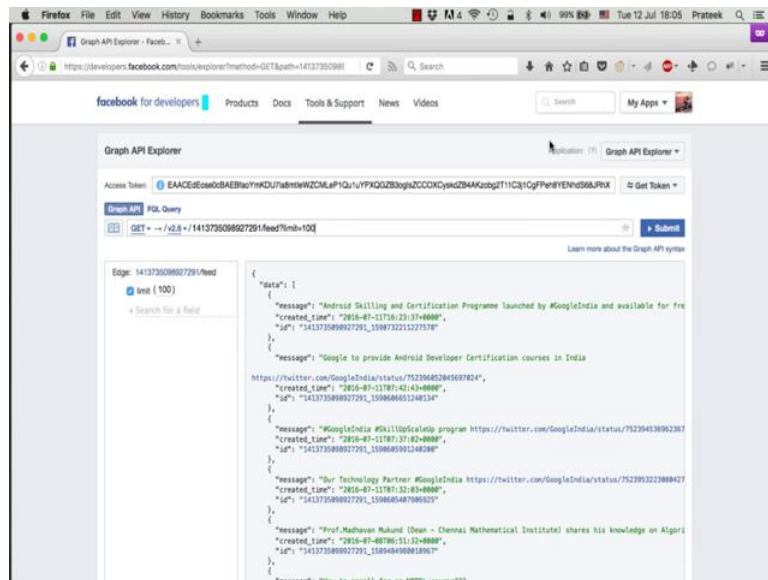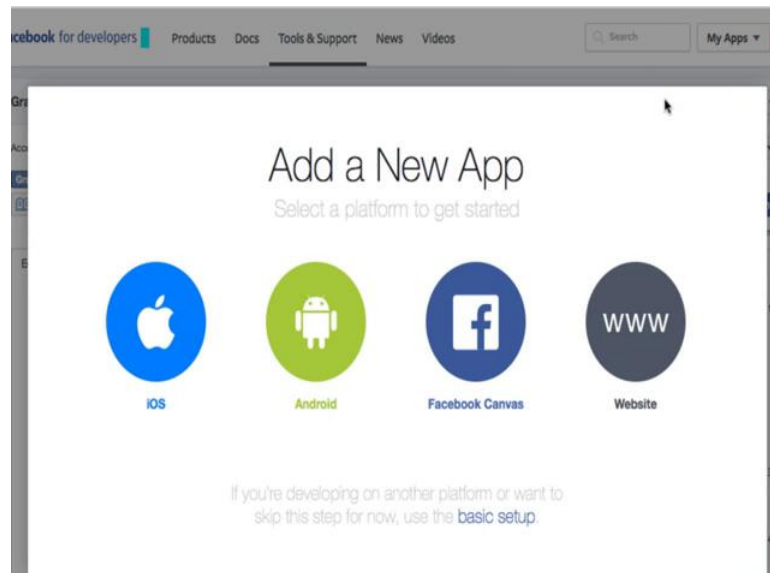

Once you register successfully, you will notice that the blue register button is now gone.

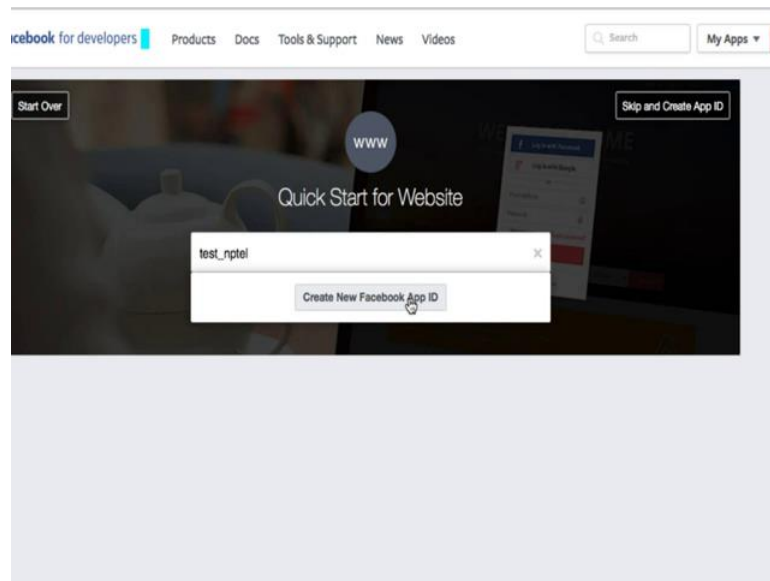And there is My Apps drop down menu in place of that register button. Click on this drop down and select 'Add a New App'.
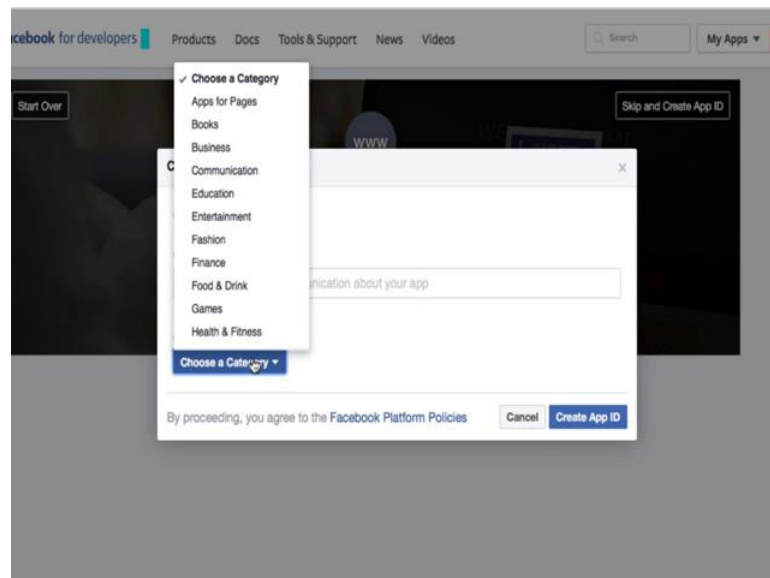
Select the platform to be 'Website' option with the www.
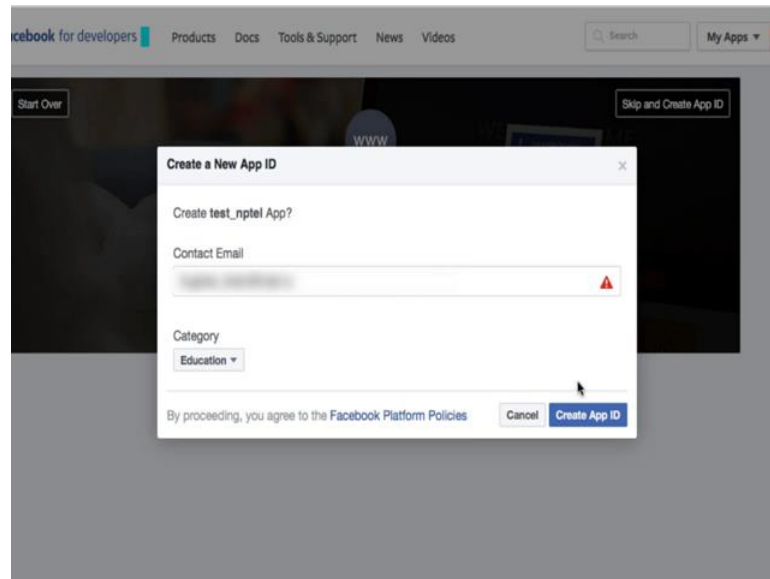
(Refer Slide Time: 02:11)

Add a name for the app you want to create, let us say, test underscore nptel.

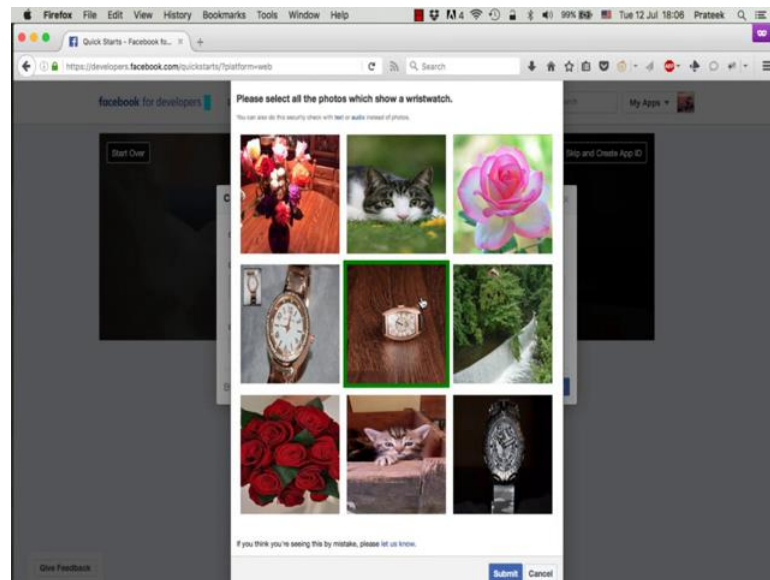(Refer Slide Time: 02:20)

And click create new Facebook App ID. You need to assign a category to this app.
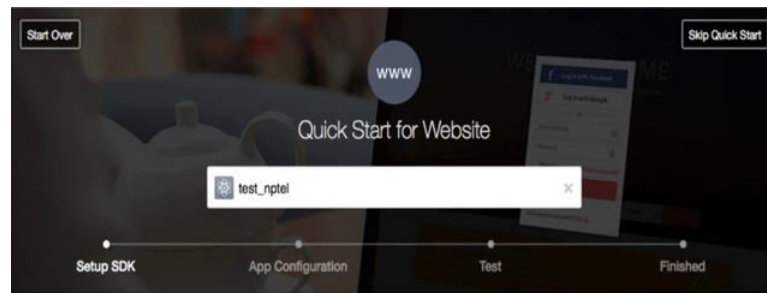
(Refer Slide Time: 02:28)

Let us say education, then put in your email id and click 'Create App ID'.

(Refer Slide Time: 02:31)

Our security check will pop up here. So, follow the instructions on the top. So, it says 'select all wristwatches' in this case. Select the appropriate images and submit.

(Refer Slide Time: 02:44)

So, the APP has now been created.

(Refer Slide Time: 02:47)

Scroll down a bit. The page will ask you for your website. It does not really matter. Just putting any valid URL will work.

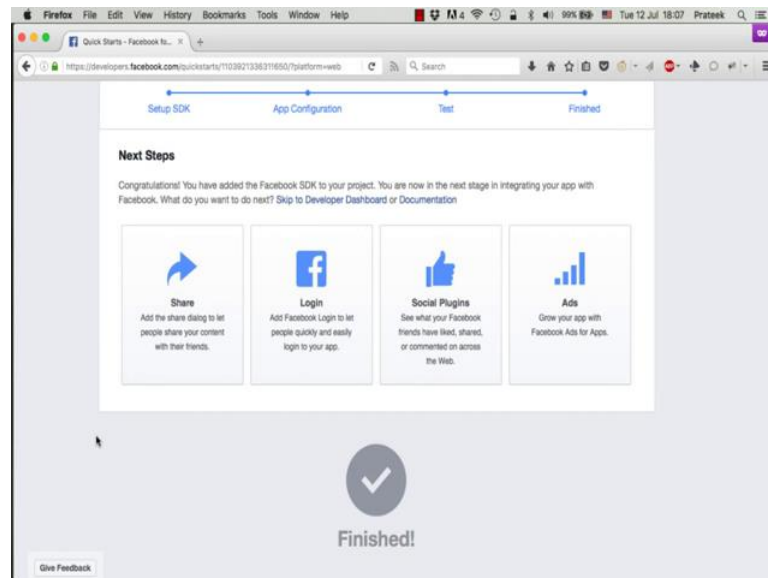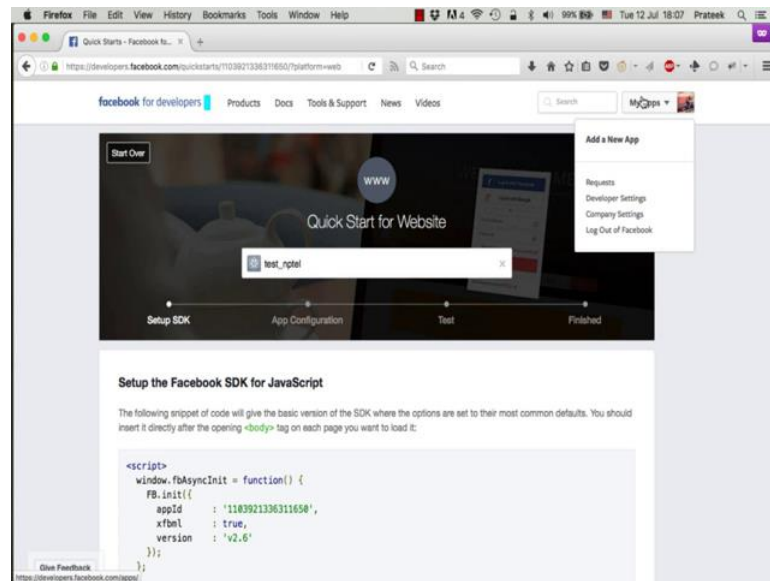(Refer Slide Time: 02:55)

Let us say http www.google.com. Then click next and scroll down.

(Refer Slide Time: 03:03)

And that is it. We have completed the APP creation process.

(Refer Slide Time: 03:14)

Now, when you click on 'My Apps' on the top right corner, you will see this page showing your APP name and ID.

(Refer Slide Time: 03:16)
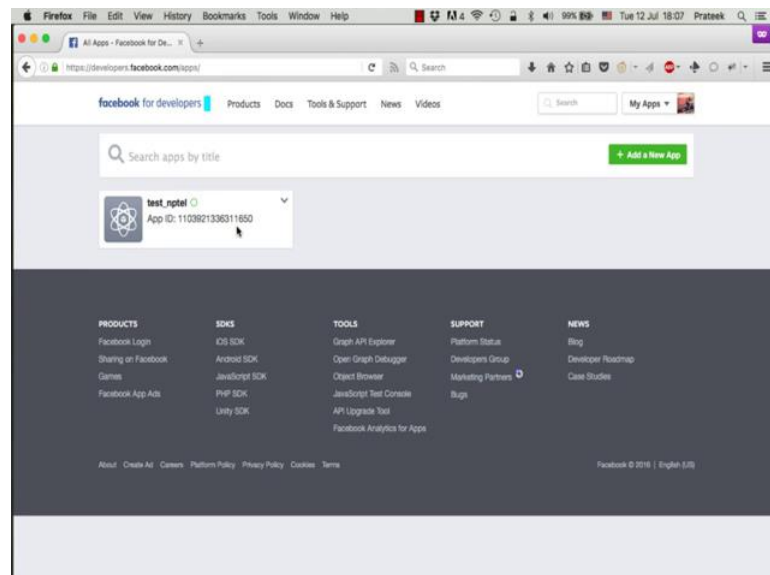
Click on the APP name to open the apps settings page.

(Refer Slide Time: 03:21)

Now click on this 'Show' button to reveal the app secret code. Now, we will use these two objects 'App ID' and 'App Secret' to extend the validity of our access token.

(Refer Slide Time: 03:56)

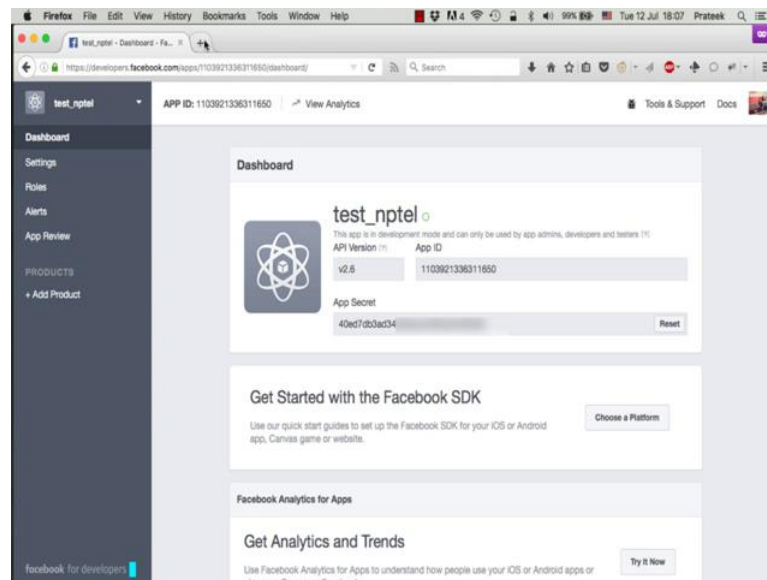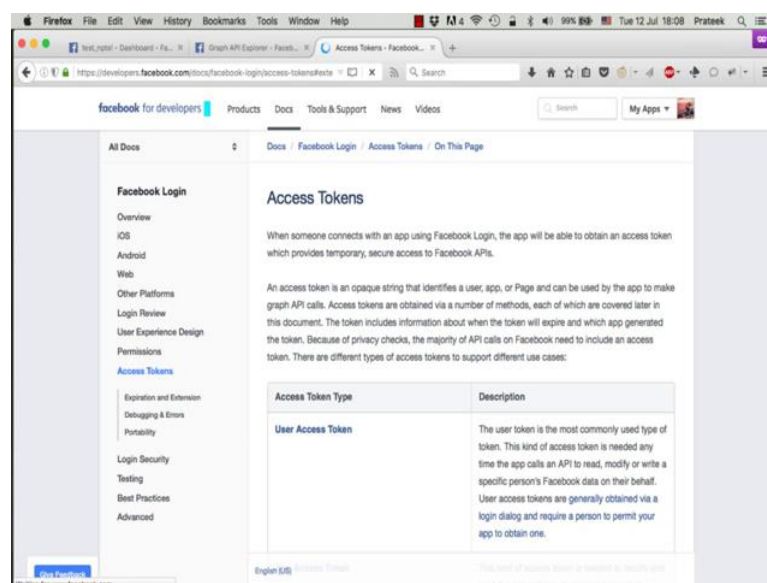Open a new tab and open the Graph API explorer again. In another tab, open the Graph API documentation page which talks about access tokens. If you just search for access token in the documentation, you will easily find this page.

(Refer Slide Time: 03:58)
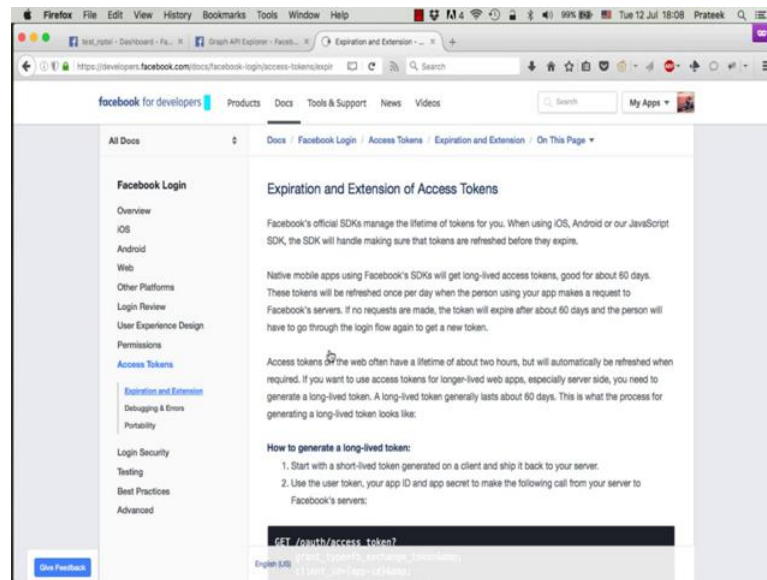
Go to this 'Expiration and Extension of Access Tokens' section, scroll down a bit.

(Refer Slide Time: 04:08)

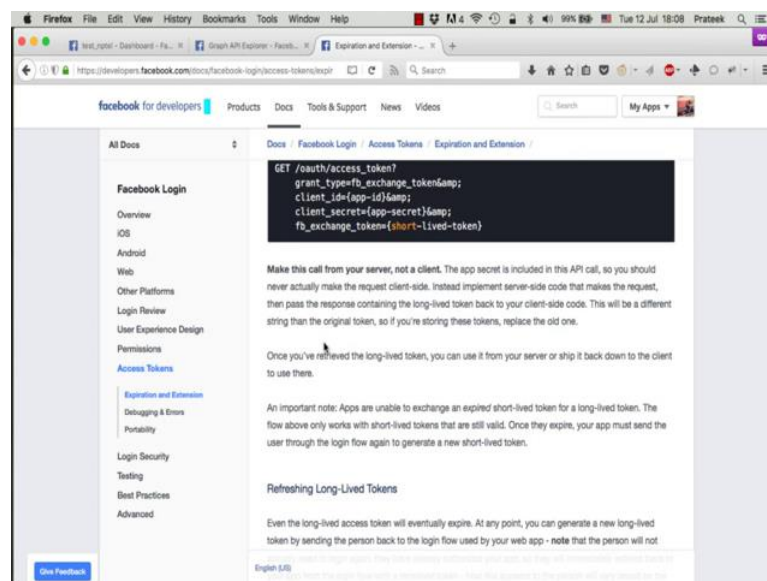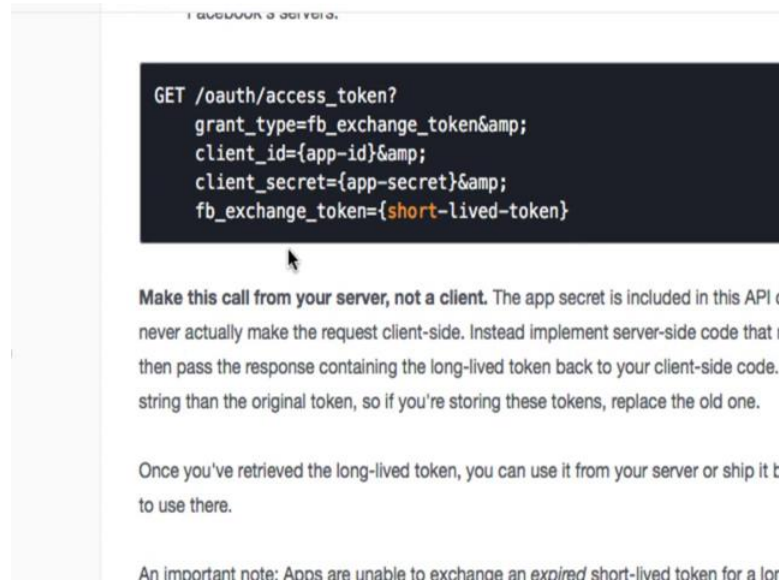And you will see the format of the GET request that you need to make to extend the access token.

(Refer Slide Time: 04:12)


```
GET /oauth/access_token?
    grant_type=fb_exchange_token&amp;
    client_id={app-id}&amp;
    client_secret={app-secret}&amp;
    fb_exchange_token={short-lived-token}
```
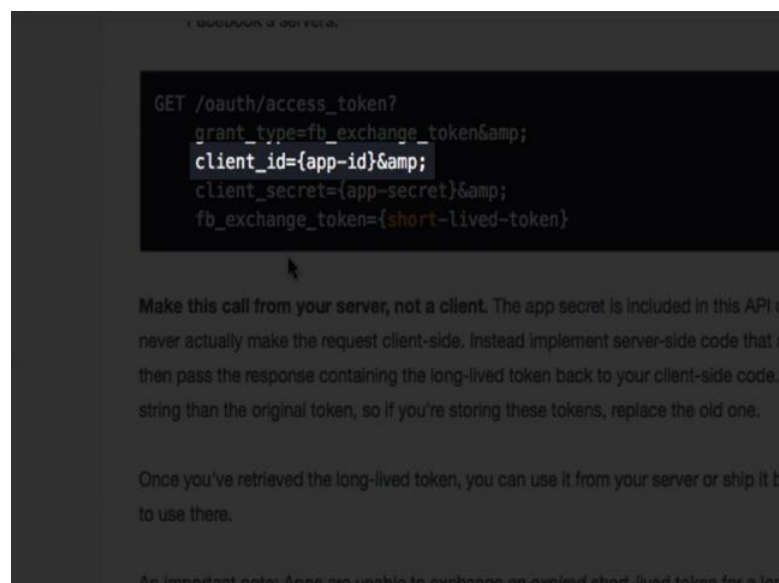
Make this call from your server, not a client. The app secret is included in this API o
never actually make the request client-side. Instead implement server-side code that r
then pass the response containing the long-lived token back to your client-side code.
string than the original token, so if you're storing these tokens, replace the old one.

Once you've retrieved the long-lived token, you can use it from your server or ship it b
to use there.

An important note: Apps are unable to exchange an *expired* short-lived token for a lor

Now, if you notice, we need three things to make this request.

(Refer Slide Time: 04:15)


```
GET /oauth/access_token?
    grant_type=fb_exchange_token&amp;
    client_id={app-id}&amp;
    client_secret={app-secret}&amp;
    fb_exchange_token={short-lived-token}
```
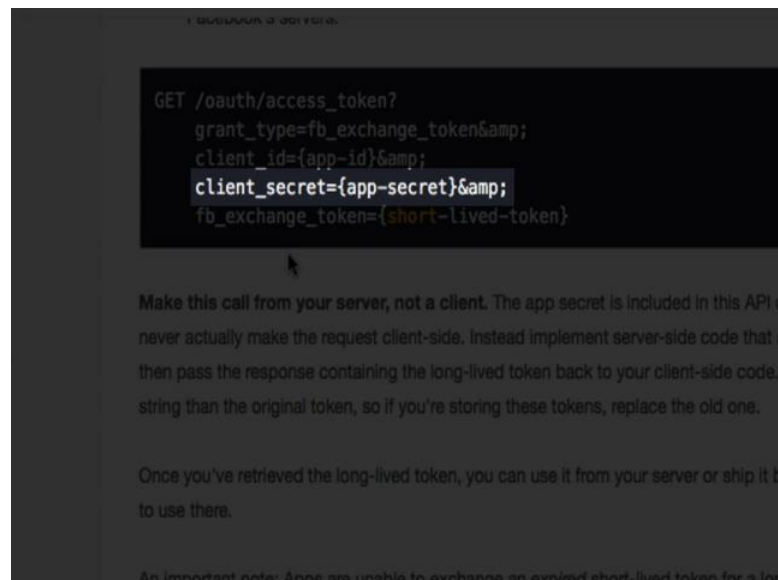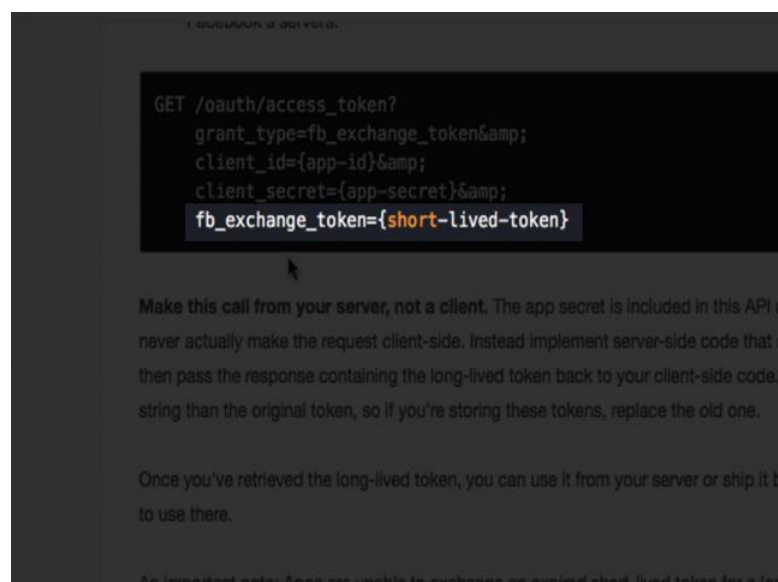
Make this call from your server, not a client. The app secret is included in this API o
never actually make the request client-side. Instead implement server-side code that
then pass the response containing the long-lived token back to your client-side code.
string than the original token, so if you're storing these tokens, replace the old one.

Once you've retrieved the long-lived token, you can use it from your server or ship it b
to use there.

An important note: Apps are unable to exchange an *expired* short-lived token for a lor

The client id.

Client secret.

And a short-lived token.

(Refer Slide Time: 04:19)

The client id and client secret are the same as the APP ID and APP Secret that we just obtained in the last screen when we created the APP test nptel.

(Refer Slide Time: 04:30)

Now, let us get the short-lived token. Go back to the Graph API explorer, select the test underscore nptel app that you just created from the applications drop down menu.

(Refer Slide Time: 04:39)
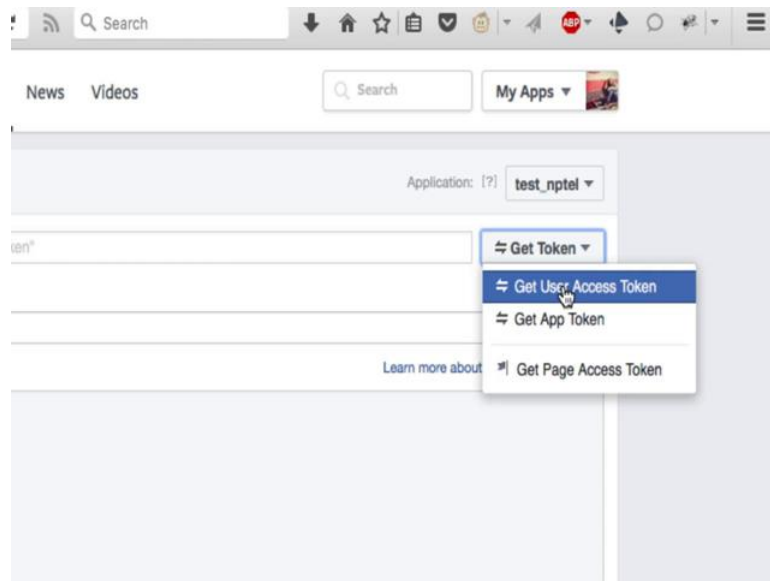
Click on get token, get user token.

(Refer Slide Time: 04:46)
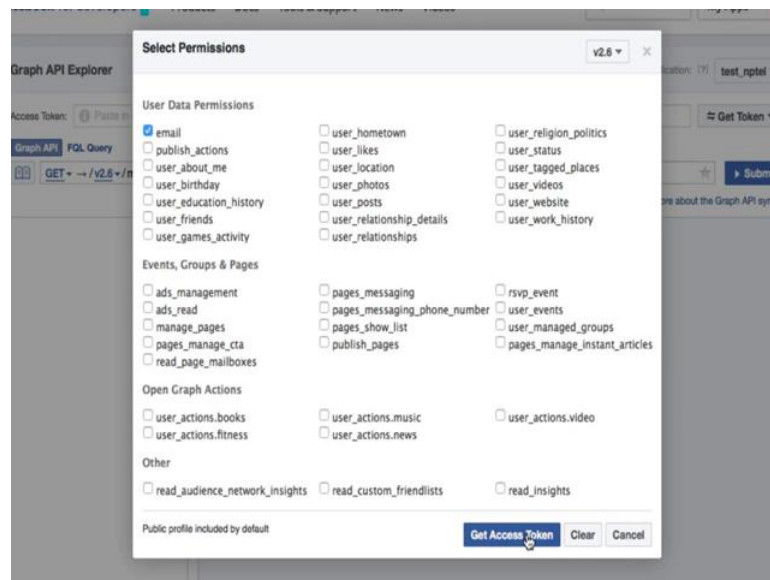
Again select any permission, say email, and click 'Get Access Token'.

(Refer Slide Time: 04:56)
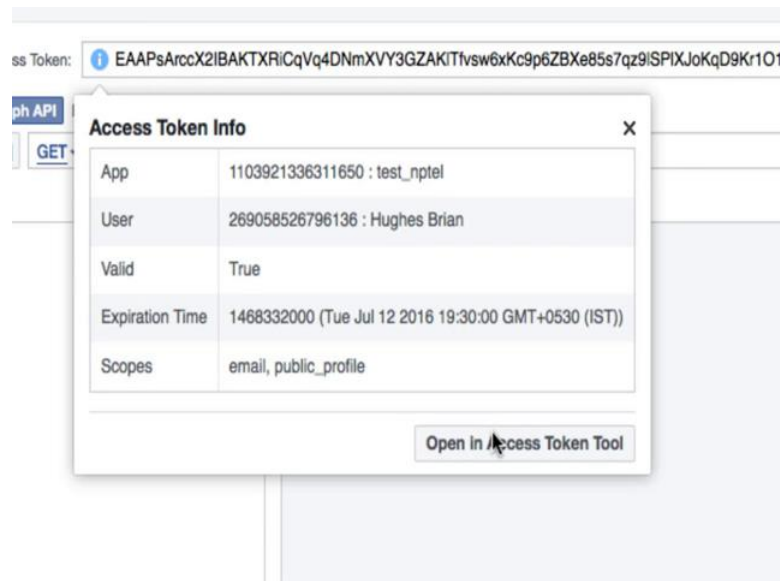
And you have this short lived access token generated using your own test underscore nptel APP. Again if you open this access token in the access token tool.

(Refer Slide Time: 05:04)
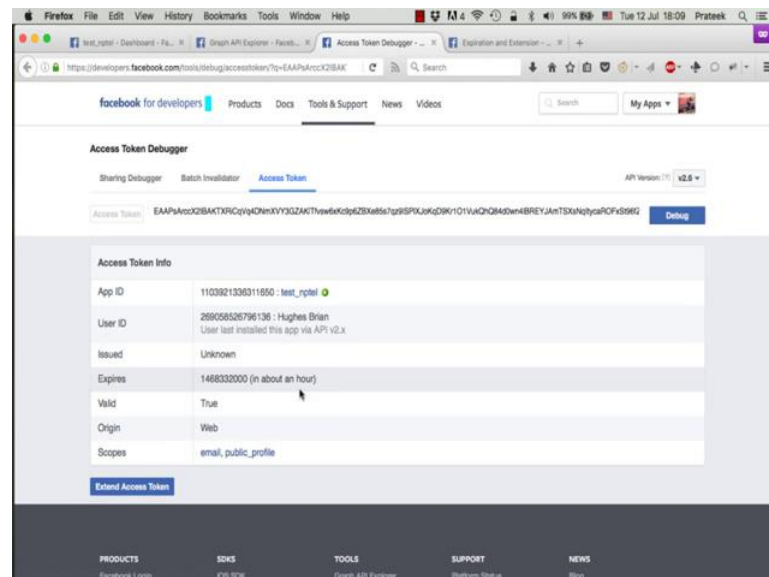
You will see that this token also expires in about an hour.

(Refer Slide Time: 05:10)
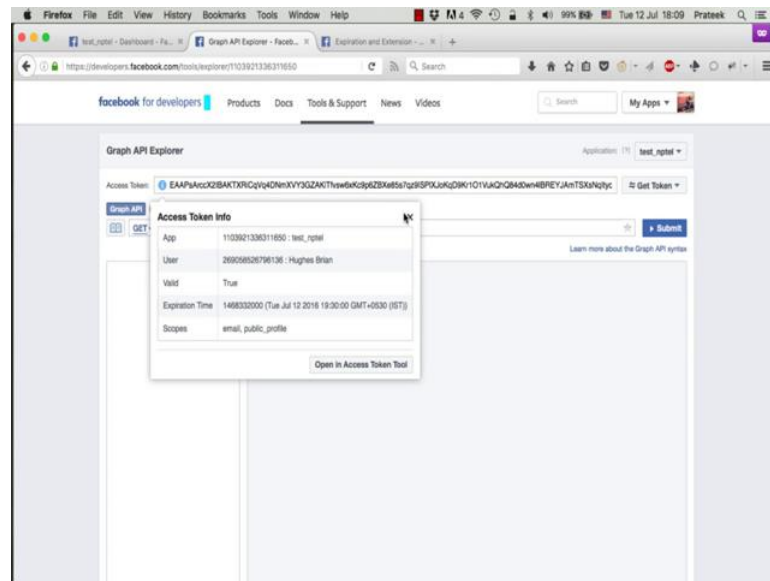
So, now we have all the three components that are needed to get an extended access token.

(Refer Slide Time: 05:34)
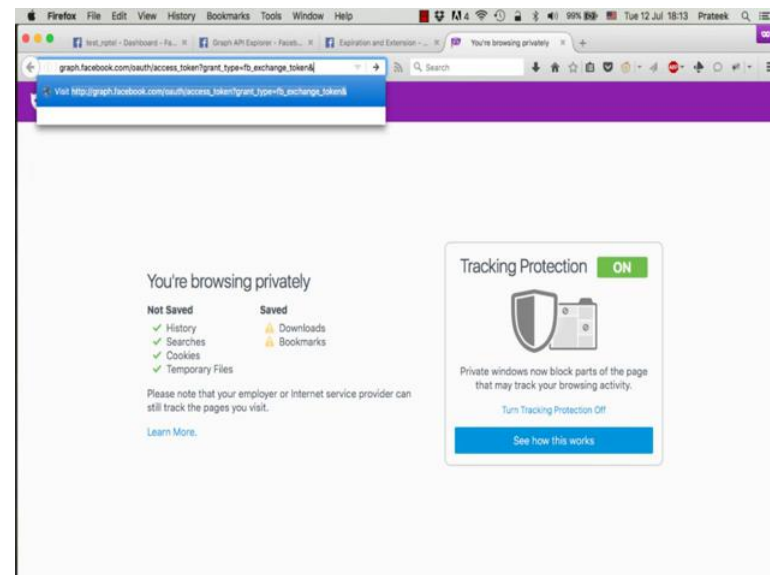
So now, open a new tab and type in http graphs.facebook.com oauth slash access underscore token question mark grant underscore type is equal to fb underscore

exchange underscore token and client underscore id is equal to the App ID.

(Refer Slide Time: 05:57)
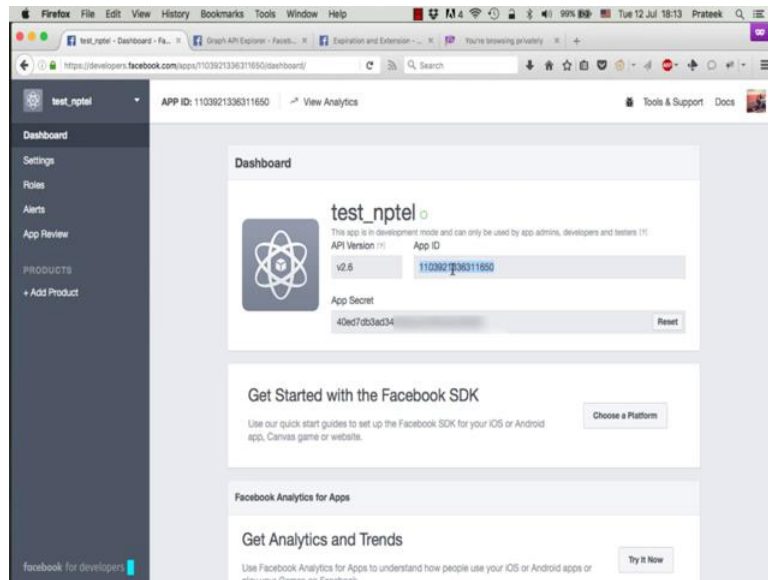
And client underscore secret is equal to the APP secret and fb underscore exchange underscore token is equal to the short token that we just generated.

(Refer Slide Time: 06:40)
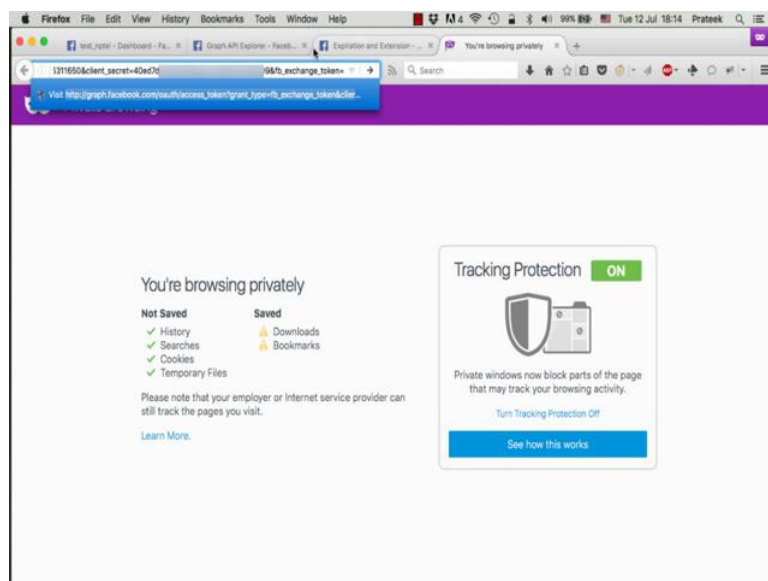
(Refer Slide Time: 06:45)
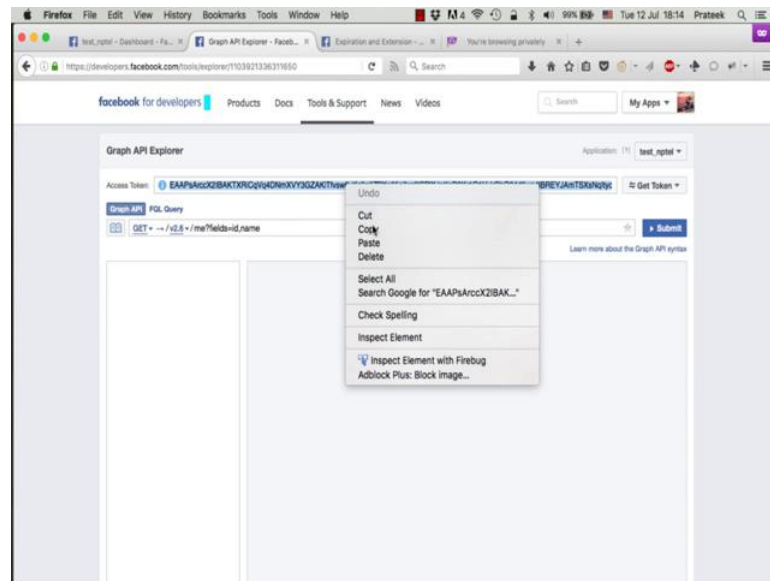
And press enter.

(Refer Slide Time: 06:54)
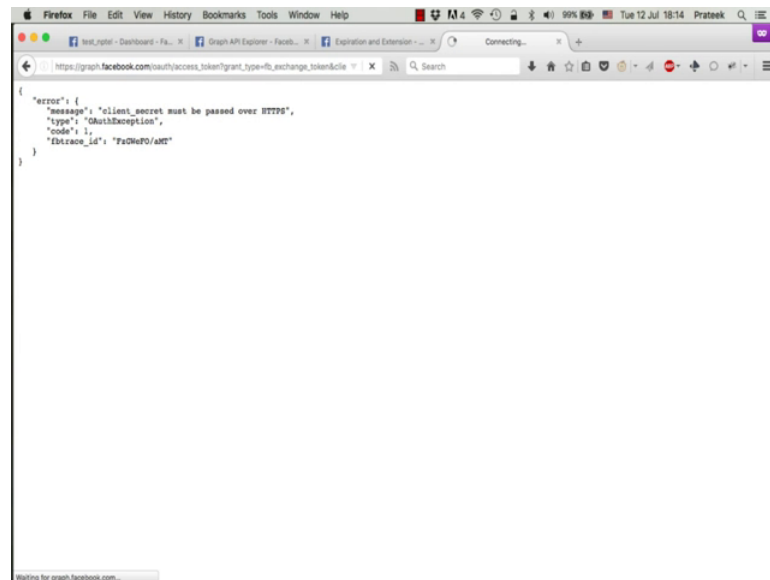
So this request needs to be made using https, so just add https in the beginning in the address bar and press enter and here you go. We have successfully generated an extended access token.

(Refer Slide Time: 07:01)
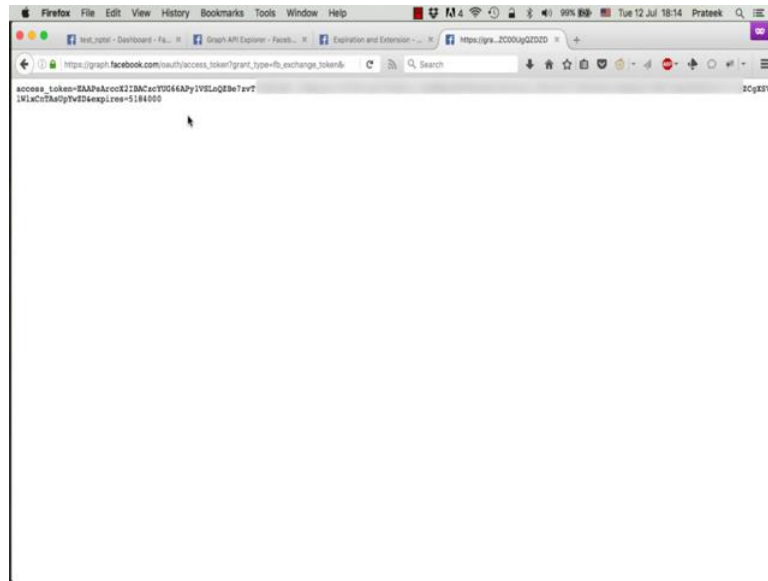
Now copy this access token, starting after the access underscore token is equal to part, until just before the ampersand sign.

(Refer Slide Time: 07:18)
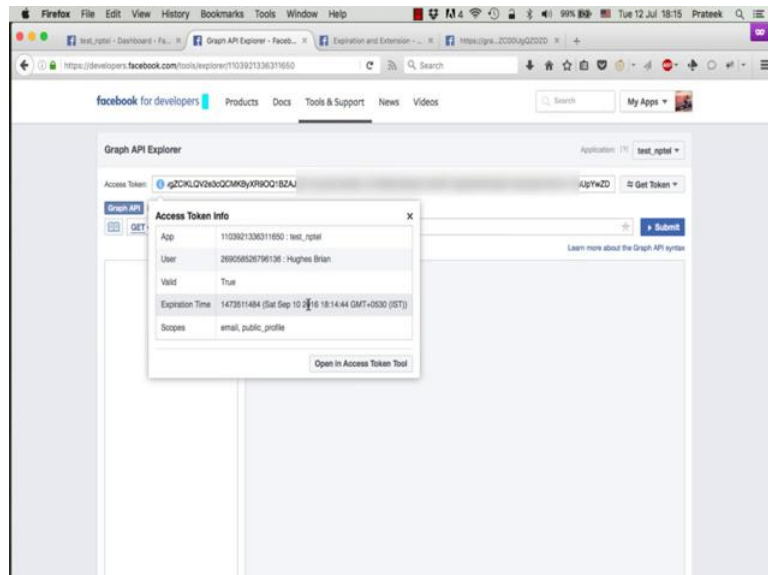
Copy this token. Paste it in the Graph API explorer, and open the access token tool.

(Refer Slide Time: 07:34)
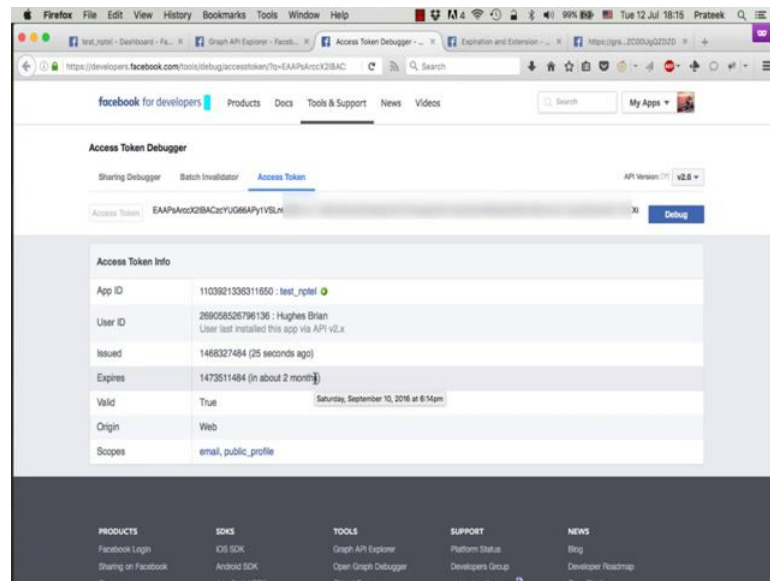
So, you see here that this token is valid for two months

(Refer Slide Time: 07:40)
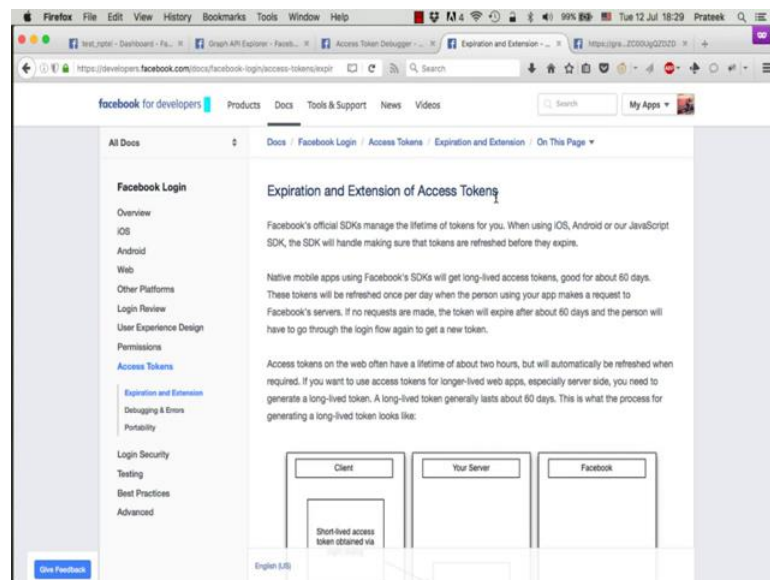
If you go back to the documentation, this is where the documentation page also says that the extended token is good for about 60 days. So, now that we have a token that does not expire for 60 days, we are all set to write our program to collect data from the Graph

API.

(Refer Slide Time: 08:04)
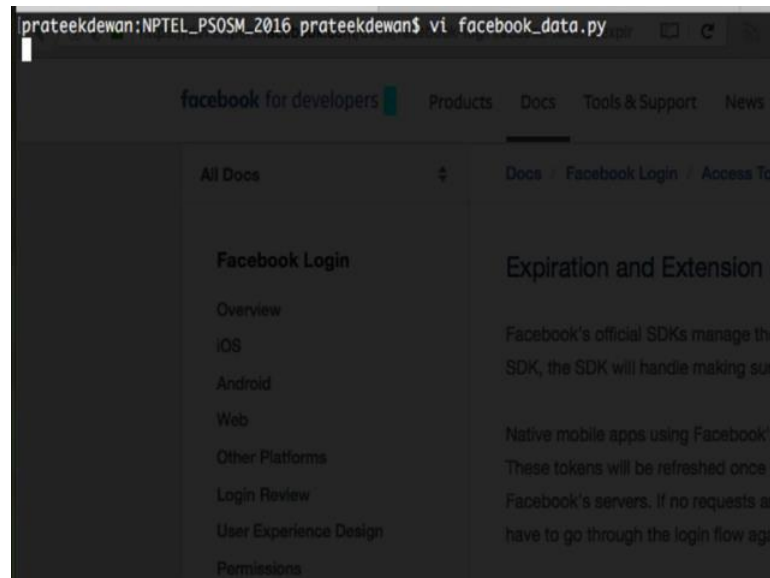
To do this we will use the terminal in python, which we learnt in the previous tutorial. So open the terminal and create a new python script using the vi editor. Let us say facebook underscore data dot py.

(Refer Slide Time: 08:21)

We start by importing the requests library that we saw in the previous tutorial. If you recall, requests is the python library, which is used to make http requests. Now we define a new function, say, get underscore page underscore data, which takes one parameter, which is the page id. The beginning of the function body is indicated by a colon sign. Now python is an indentation base language, so to define a code block within a function, we will add an indentation level to the entire body of the function. This is equivalent to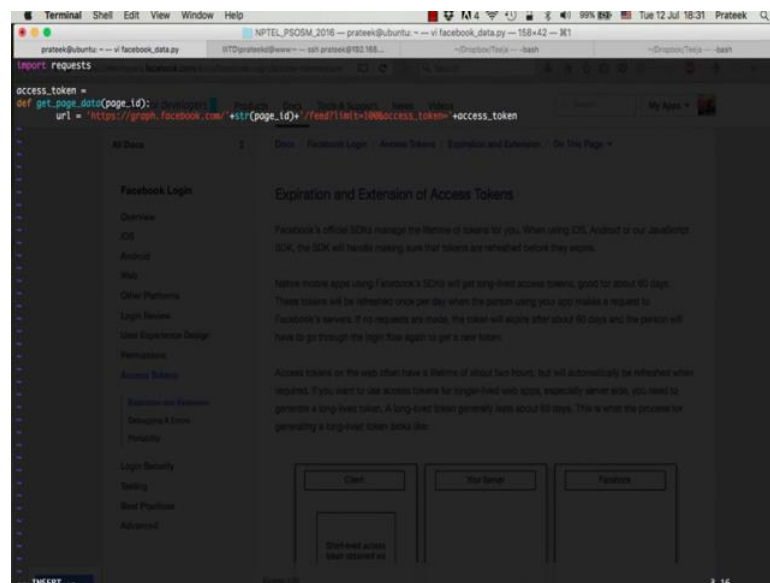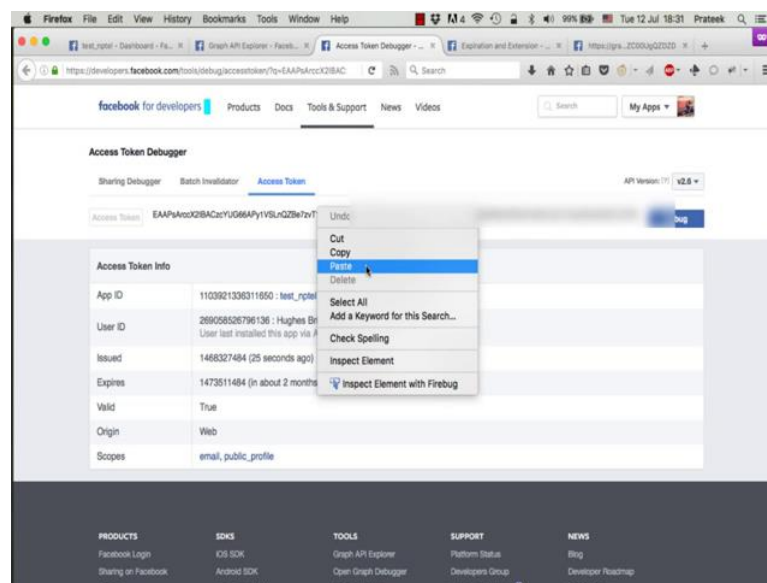 the curly braces in C or C++. Just like the body of a function is defined within curly braces in C; in python, the function body is defined within an indentation level. Once you get back to a lower indentation level, you exit the function body. So we press tab to create an indentation level, and start defining the function body.
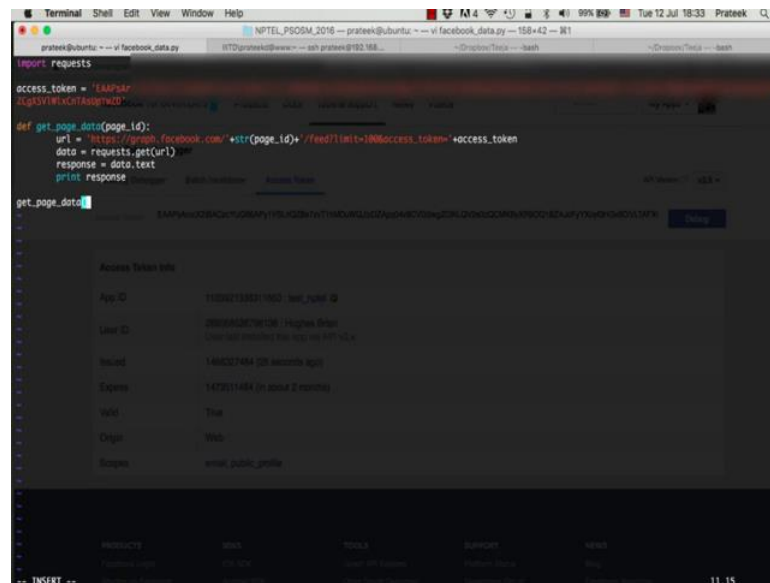
We first define a variable which will contain the URL that we would send the request to. So, we say URL is equal to http graph, no, https graph dot facebook dot com slash the page id converted into string format - you can simply use plus sign to concatenate strings in python - slash feet question mark limit is equal to one hundred and access token is equal to access underscore token. Now we need to define this access underscore token variable before we can use it. So, we say access underscore token is equal to

(Refer Slide Time: 10:21)

this extended token that we just generated.

Now we say data is equal to requests dot get URL. So, we are just sending a get request to this URL and storing the response in a variable name data. Now we need a text part of this response, which will contain the data returned by the Graph API. So, we store the text part of the data in another variable called the response. And we say, print response. And we get back to our original indentation level indicating the end of the function body. Now, to call this function, we type the function name followed by parenthesis.

(Refer Slide Time: 11:39)
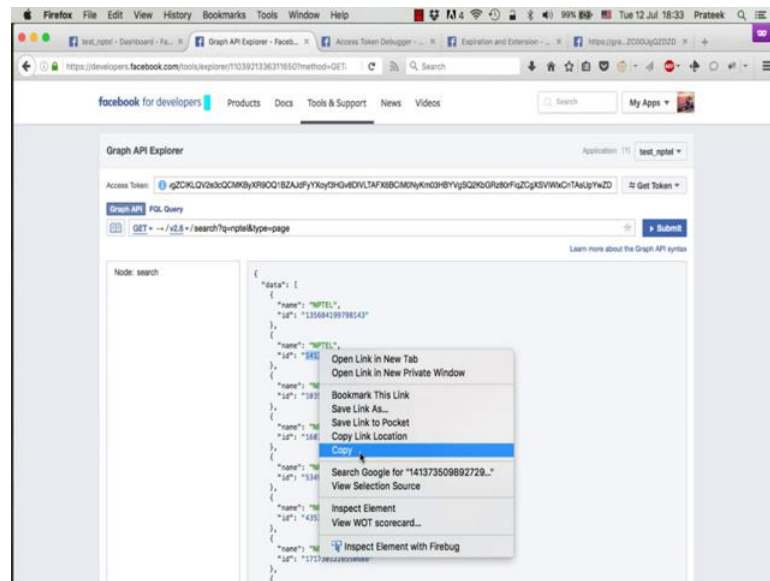
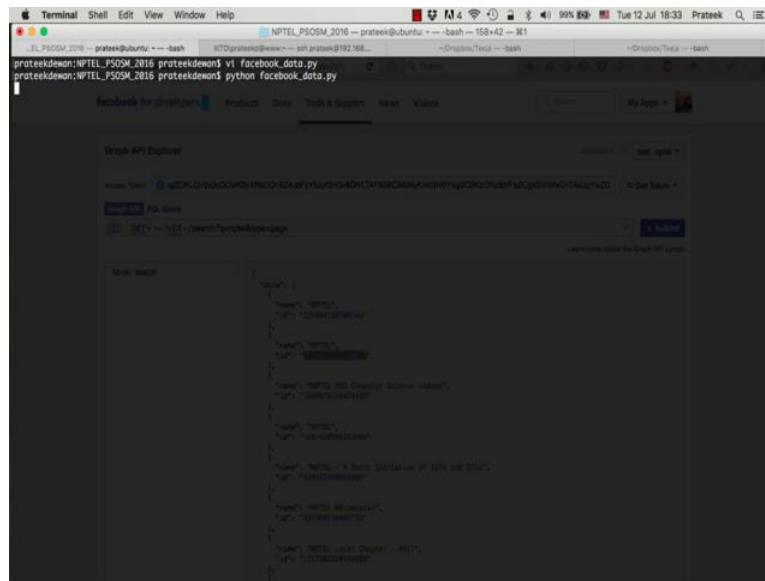And will add the page id parameter in the parenthesis, that we are supposed to pass to this function. So we go back to the explorer, search for NPTEL pages again, get the page ID of the second result, copy it, and pasted in the quote.
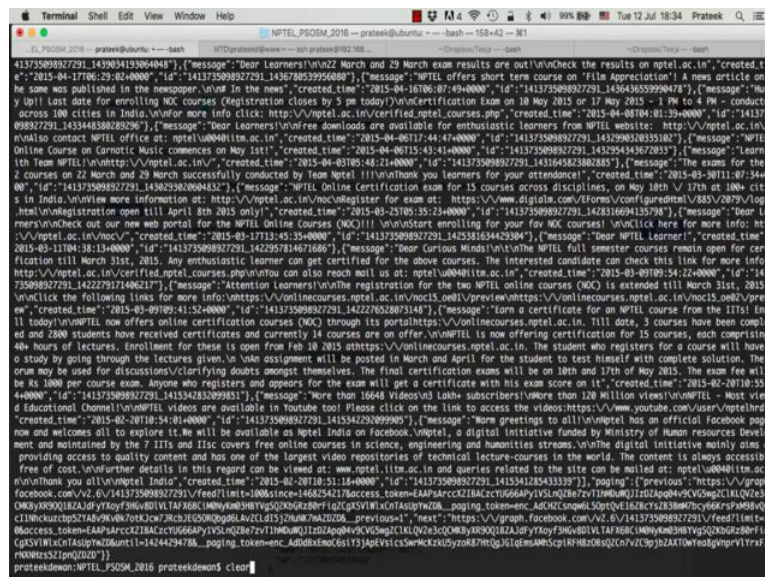
(Refer Slide Time: 12:11)
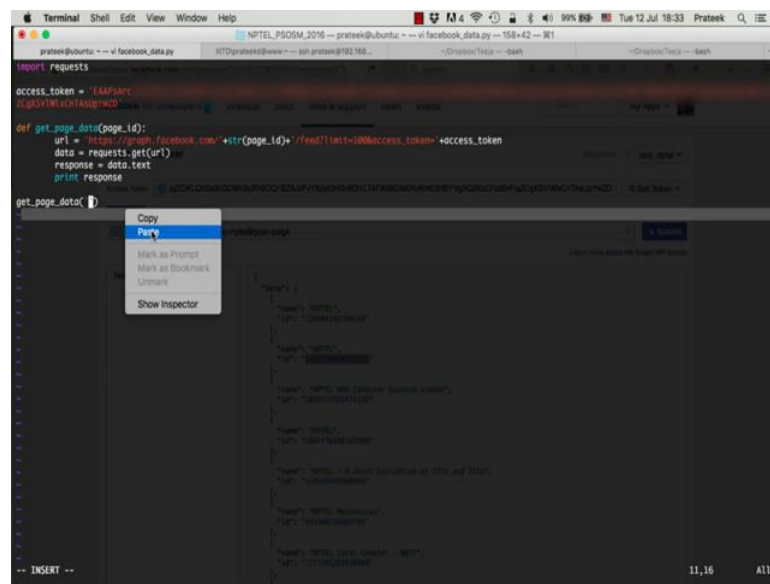
Now we write this file and quit the editor by pressing escape colon wq enter.

(Refer Slide Time: 12:23)

Now to execute this code, type python space facebook underscore data dot p y and press enter.
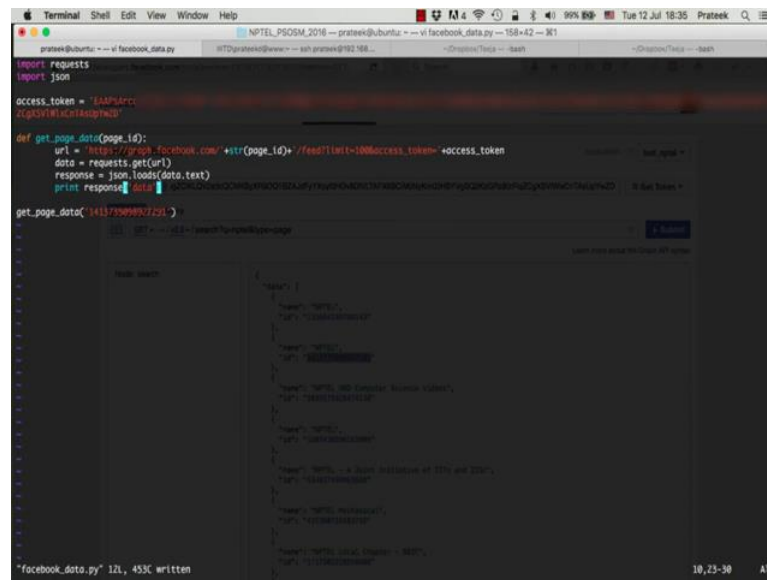
(Refer Slide Time: 12:35)

And you get the entire data written in a JSON format. This is exactly the same response you saw from the Graph API explorer, except that this has no line breaks, so it is harder

to understand in this format.

(Refer Slide Time: 12:56)

Now to be able to read this data more efficiently, we will make use of the JSON library for python, which is used to pass JSON objects, if we remember we discuss that the graph API written data in JSON format. So, you type import j s o n. In most cases, this library comes preinstalled in python; if you do not have this library installed, you will see an error message saying no module name 'json' when you run the code.

You can install this library using pip that we discussed in the first tutorial. Just type sudo pip install json in the terminal and the library will be installed. So, we now load response text in JSON format using json dot loads data dot text and print the information present in the data field of the JSON response. So, the JSON format is essentially a key value pair based format, where the key is the name of the field and the value is the information present in this field. So, when we say response with this string data in square braces, the data here is the key and the values is all the post preset in this field.

(Refer Slide Time: 14:08)
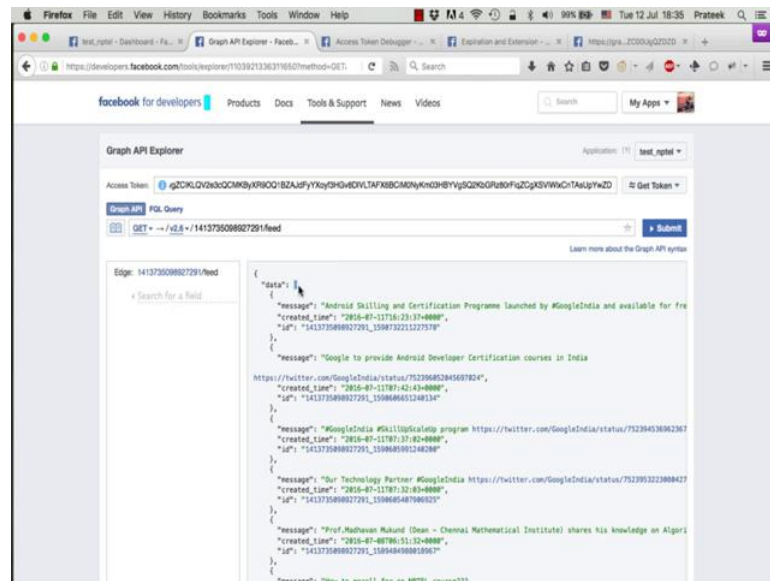
Now notice that the content present in the data field is a list which is similar to an array in C or C++.

(Refer Slide Time: 14:23)
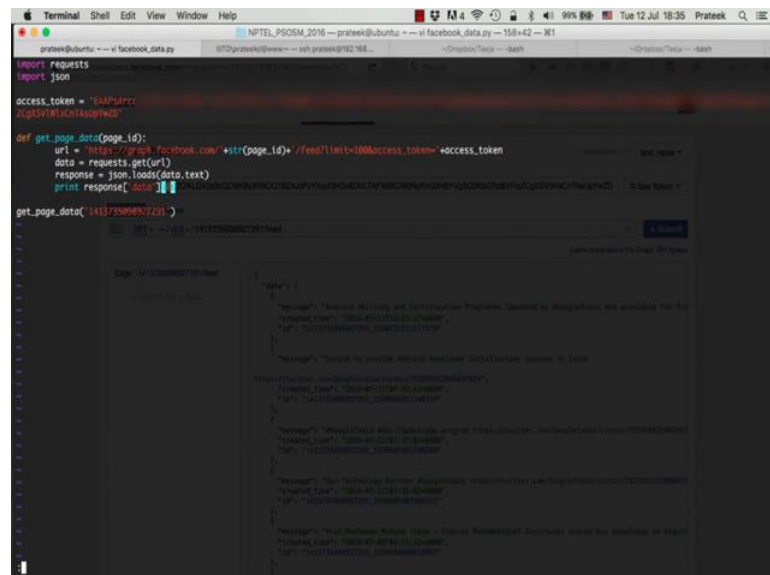
So let us print the 0th element of this list, which is the post. So, there you go.

(Refer Slide Time: 14:33)

This prints the first post exactly as we just saw in the Graph API explorer. Now, let us try to query the version 2.0 of the API to get more details as we did using the explorer.

(Refer Slide Time: 14:52)

So, we add v 2.0 slash just before the page id in the URL.
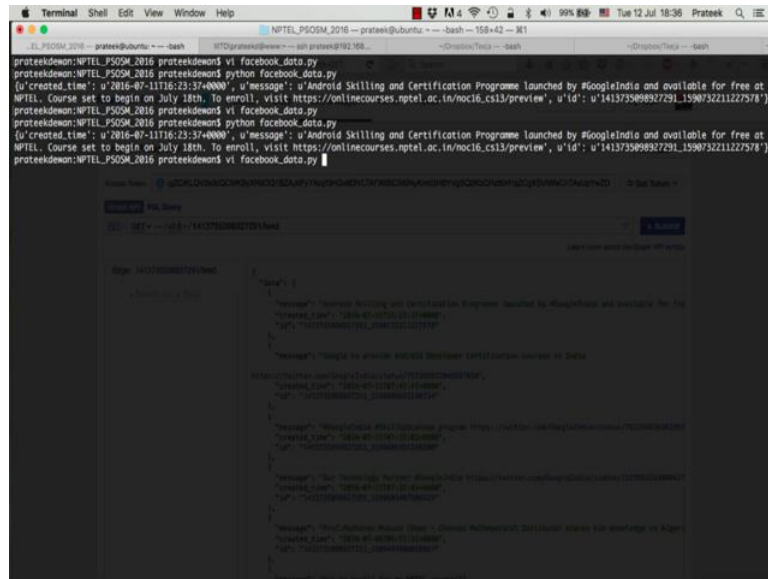
(Refer Slide Time: 15:05)
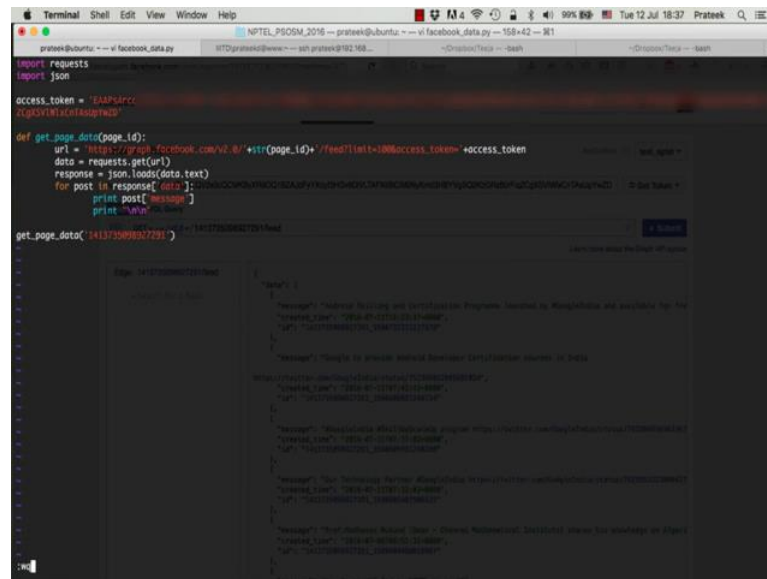

And run the code again. So, the response received is exactly the same as you got in version 2.6, no extra information was returned.

(Refer Slide Time: 15:19)
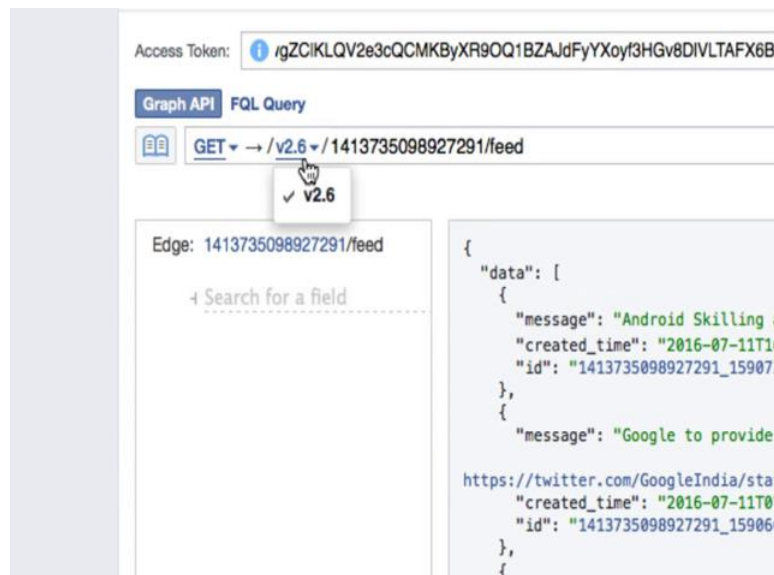
Now why is that? If you go back to the explorer, you notice that now in the version drop down there are no old versions of the API available.

(Refer Slide Time: 15:30)
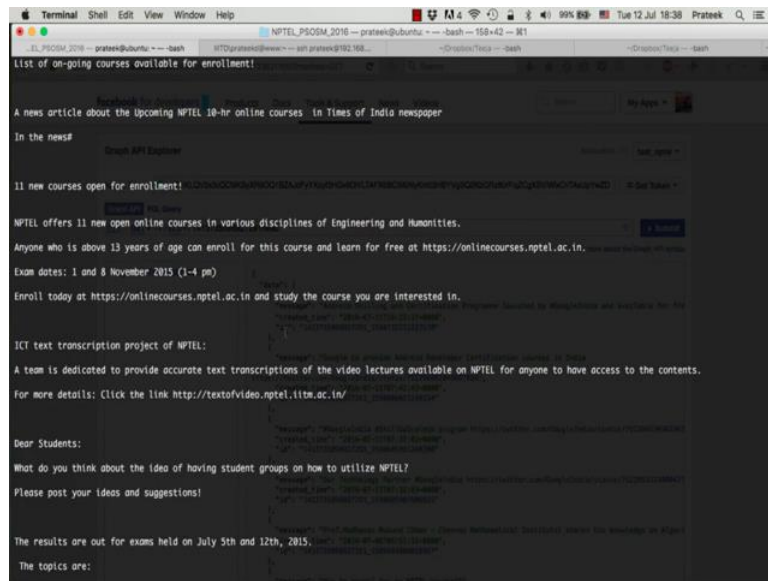

This is because when you create a new app, this new app does not have access to any older versions of the API beyond the current version. So, in this case, since version 2.6 is the current version and our APP was created after the version 2.6 was released, our APP does not have access to any API version before version 2.6. Remember that the APP you were using in the beginning was the Graph API explorer APP which was created long ago by Facebook itself, so it has access to all the older versions of the API.

So, let us try to iterate over of the posts in the data list returned by the API. We will use the for loop in python for doing this. So, we say for post in response, square braces, data colon. So, now, we create a new indentation level to define the body of for loop, just like we did for the function. And we say print post square braces message.

Now what this loop will do is it will go over every element or post in the data list one by one and assign the element to this variable named post. So, when we are inside the loop, this post variable will be storing the current element being iterated by the loop. So, the contents of this post variable will be updated upon each iteration of the 'for' loop. And we will print the value corresponding to the message key present in the post in each iteration until all the post has been iterated over. We print a couple of new line characters to differentiate between the different posts.

(Refer Slide Time: 16:58)

Now let us run this code. So, you see all the messages printed separated by two blank lines.

(Refer Slide Time: 17:30)


Now let us learn how to handle the API response, when the results are returned in multiple pages. If you remember, we saw in the first tutorial that if the number of results

are greater than a certain limit, the API is also returns a URL to the next page of the results which you need to visit in order to get the complete results. So, let us reduce this limit to say 20, so that we will get 20 results in each page. We already saw that the NPTEL page

we are querying had about 65 posts in total. So, if we set the limit to 20, we will get results in four pages, the first three pages containing 20 results each, that is 60, and the fourth page containing five posts, totaling to 65. So what we need to do is to create a loop which will keep a track of the next page URL, and keep on querying the next page as long as needed; until the point where the API response does not contain a next page URL anymore. So, we say next underscore page is equal to response paging next.

(Refer Slide Time: 18:39)
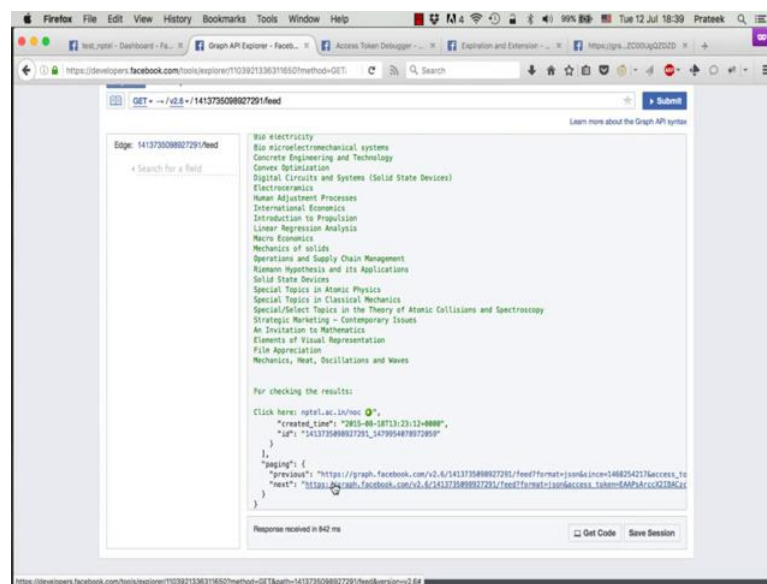
So, if you notice in the response, there is this key called paging. Inside the paging there are again two keys previous and next. And the value corresponding to the next key is what we need. So, we keep on visiting this next page until the API response does not contain any paging key.

(Refer Slide Time: 18:59)
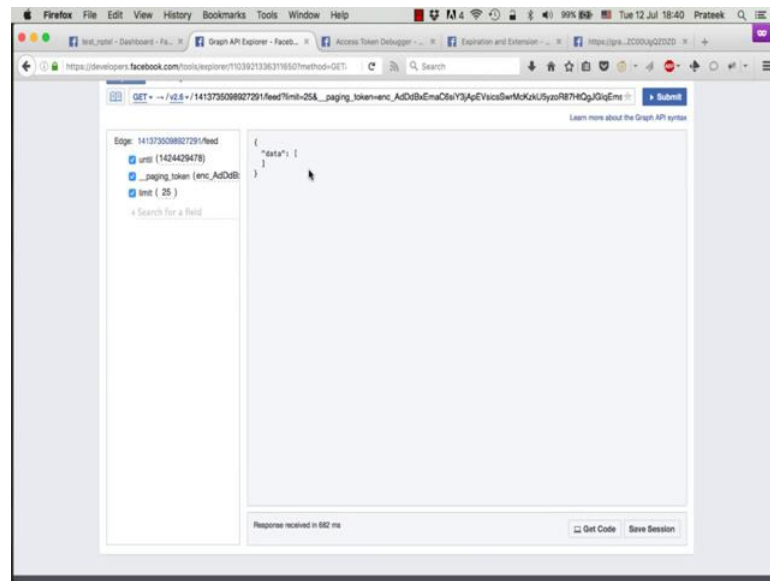
Notice that the data key is still present, but it is an empty list now, meaning we have run out of responses. We saw only three pages here since a limit was set to 25 by default. So, the first two pages contained 25, 25 posts that is 50, and the third page contained the remaining fifteen total into 65, and this last page is empty.
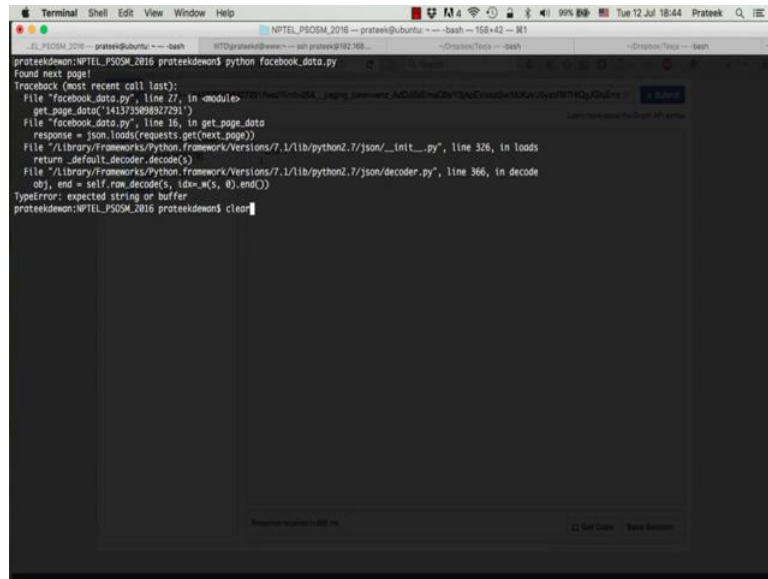
(Refer Slide Time: 19:31)
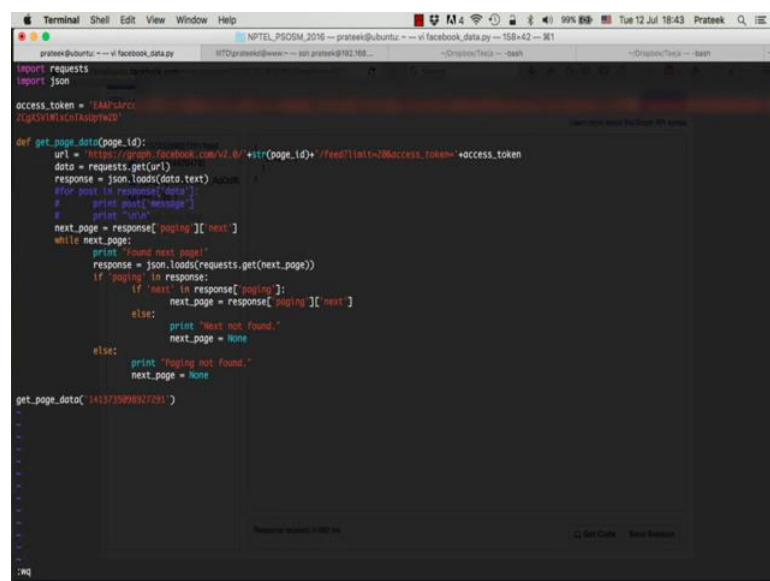
So, we pick up the next page URL in the first page of the results; let us move this after the print statement. And we say while next underscore page colon and we begin another loop, a while loop this time. So, this loop will keep running as long as the next page variable has something in it, meaning as long as it is not blank or null. And inside this loop we send a get request to this next URL, and store the response in a variable named response. So, we are just over writing the response with the contents of this next page. Let's just add a debug print statement, when the loop begins, found next page. So, every time this loop executes, this message saying that we found a next page will be printed. Let us comment out the earlier print statements to avoid confusion.

And now we will check, if this new response that we got has a key named paging in it. So, we say if paging in response, now inside this if, we again need to check if there is a next key present in paging. Sometimes, it may happen that there is a paging key, but it only has a previous page and no next page. So, we cannot just rely on the paging key alone. So, inside this if block, we again check if next in response paging; and if both the above conditions hold true, we know that there is a next page. So, we update the value of this next underscore page variable with the next page URL. So, next underscore page is equal to response paging next.

Now what happens if there is no next page, so we say else print next not found. And we say next underscore page is equal to none with a capital N. We update the value of the next underscore page variable to None, which is the same as null in C or C++. So, if this happens, the condition in the while loop will become false and as we discussed, the loop will break in the next iteration if the next underscore page field is blank or null, as we just defined. Similarly, if there is no paging key in response, we create an else block corresponding to the outer if block. And print paging not found, and again update the next underscore page variable to none.

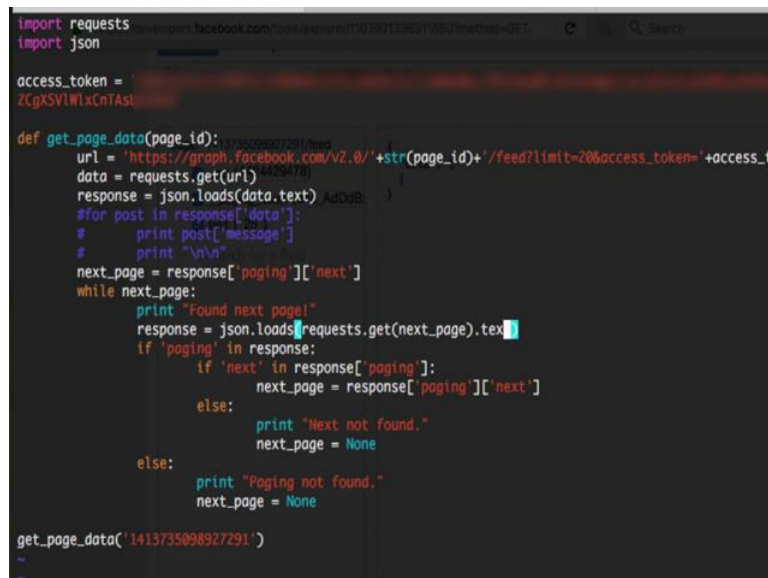(Refer Slide Time: 22:37)

So, let us save this file, exit the editor and run this code. Clear the terminal. python space facebook underscore data dot py, enter, OK. So, there is an error on line 16, so we forgot to add the dot text part in the get request inside the while loop.

(Refer Slide Time: 23:00)

So we add dot text, save.

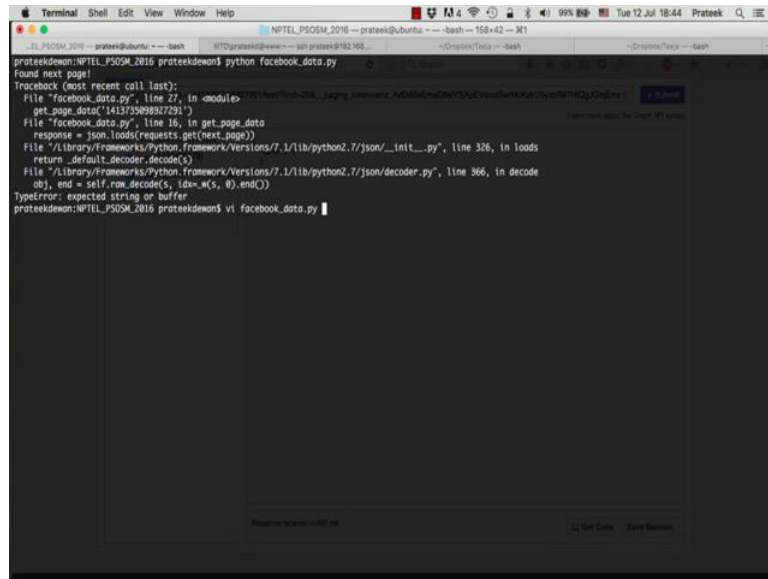(Refer Slide Time: 23:11)

And run the code again. Clear buffer, python space facebook underscore data dot py, enter. So, there you go, we found four pages as we just discussed, and the fifth page did not contain any paging key as we expected which made the while loop <mark>break</mark>. So, now, we have also learnt how to handle paging using python.

(Refer Slide Time: 23:43)

As one last exercise let us see how to collect likes on a post through python. So, we pick up the id of the first post that appears in the results post underscore id is equal to response, data, 0, id. And we define a variable containing the URL for getting likes, likes underscore URL is equal to https graph dot facebook dot com slash post id slash likes question mark limit is equal to 100 and access underscore token is equal to the access token. Let us print the post id too. And let us just directly print the response. We say print requests dot get likes underscore URL dot text. And let us add an exit statement just after this; we do not need the rest of the code to execute. So, the program will just exit after this print statement.
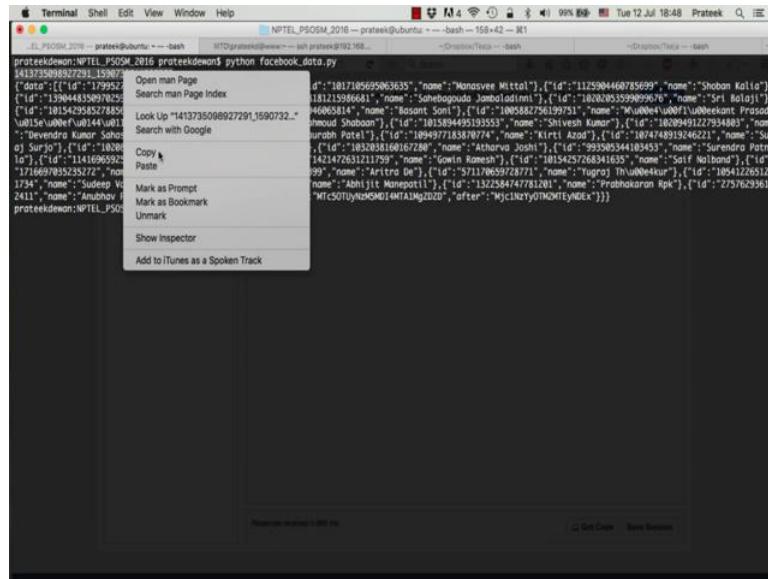
(Refer Slide Time: 25:15)

Save and exit the editor, clear the terminal.

(Refer Slide Time: 25:19)

And python space facebook underscore data dot p y enter. And you see the post id of the first post followed by the likes. To verify these results just copy this post id.
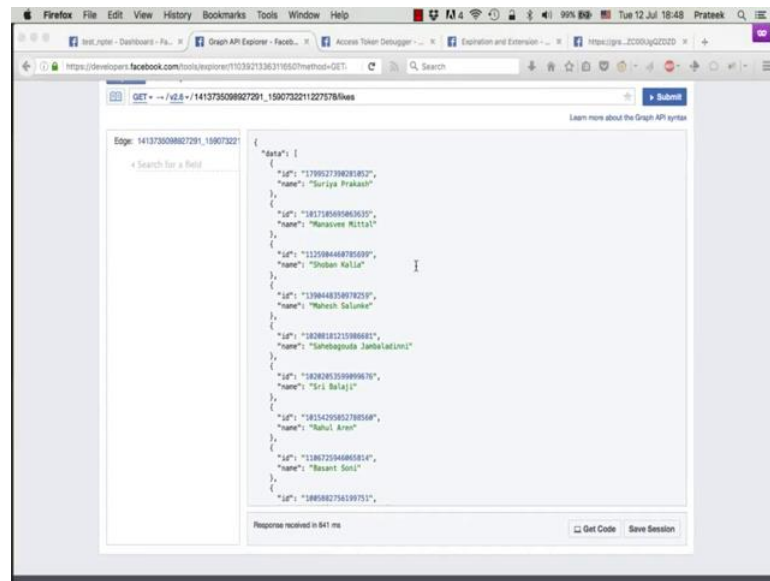
(Refer Slide Time: 25:37)

Go back to the graph API explorer and paste it in the query bar followed by slash likes, press enter.

(Refer Slide Time: 25:45)

The same results show up here as well. So, in this tutorial, we learnt the basics of the Facebook Graph API from scratch, and saw how to collect data from API programmatically using python. In the next tutorial, we will learn how to collect from Twitter and see how to store data in databases.

Thanks.