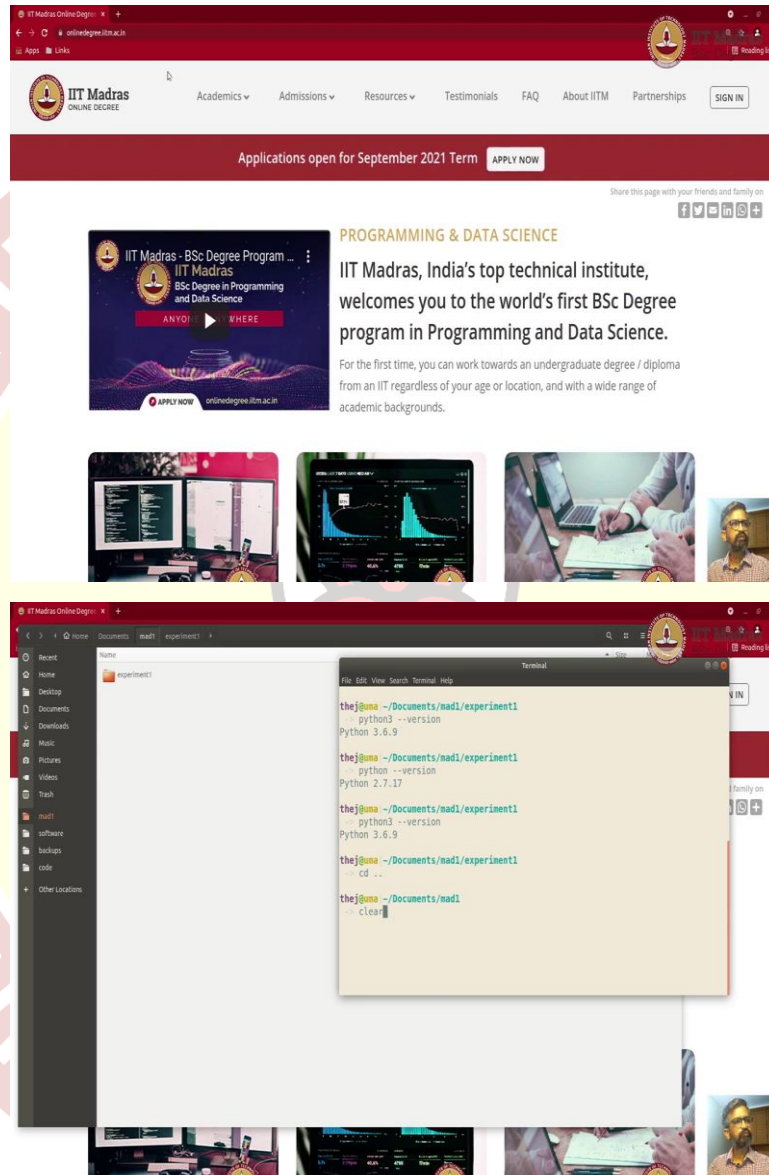


# **IIT Madras**

## **ONLINE DEGREE**

**Modern Application Development – 1**  
**Professor Thejesh G N**  
**Software Consultant**  
**Indian Institute of Technology, Madras**  
**Introduction to Jinja2 - I**

(Refer Slide Time: 0:14)



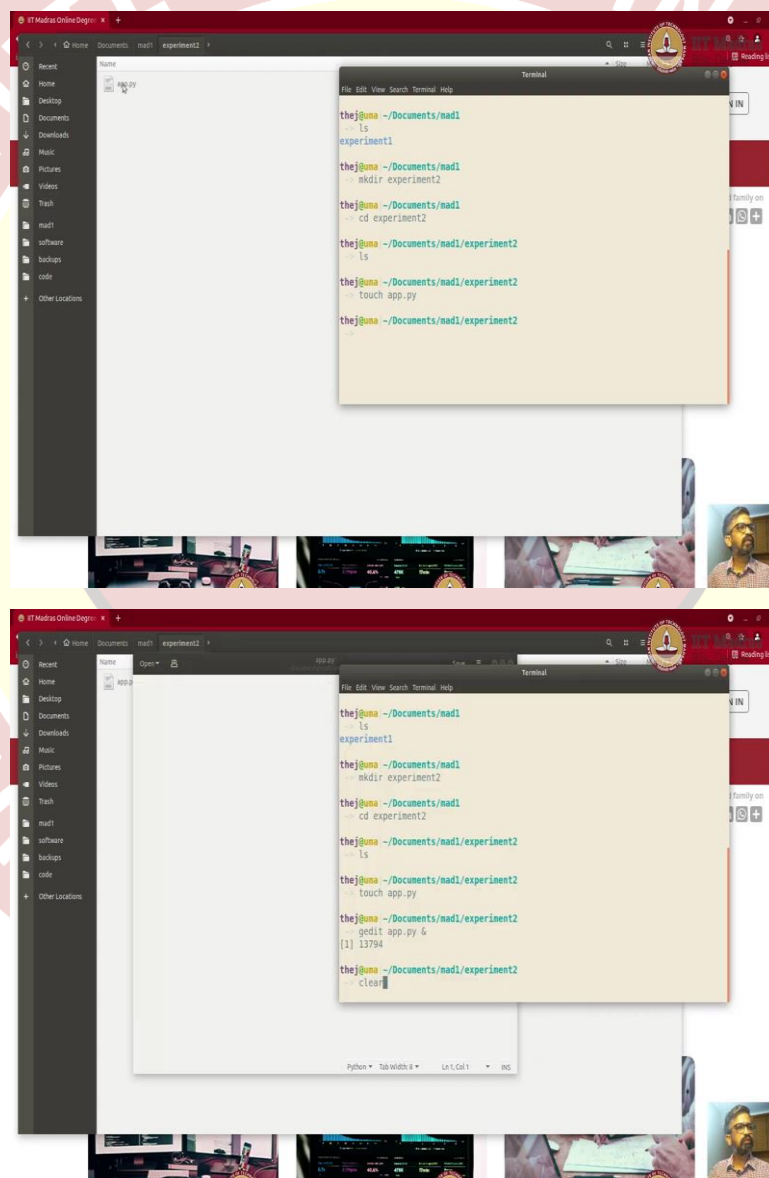
Hi, welcome to the modern application development screencast. In this short screencast, we will learn how to create templates, use these templates to create HTML document generate some of the documents dynamically. Before we start, open the following applications on your Ubuntu desktop so that you can work along with me.

As usual, you need a browser, I have chrome here, you need an editor, I am using a simple text editor called g edit. And then you need a terminal using a built in Terminal, and then you

need a file browser, I using just built-in file browser. And you also need to know you have installed Python 3, based on your settings, it could be command Python 3, or it could be Python. In my case, it is Python 3. So, you can see that if my default Python is python 2.7.

So, you will most probably see me using Python 3, just check whichever command is available in your system and use the one which gives it has the version Python 3 or any of the series in Python 3. Now, before we begin, let us create another folder called experiment 2. Let me just go to the folder structure, here I am.

(Refer Slide Time: 1:55)

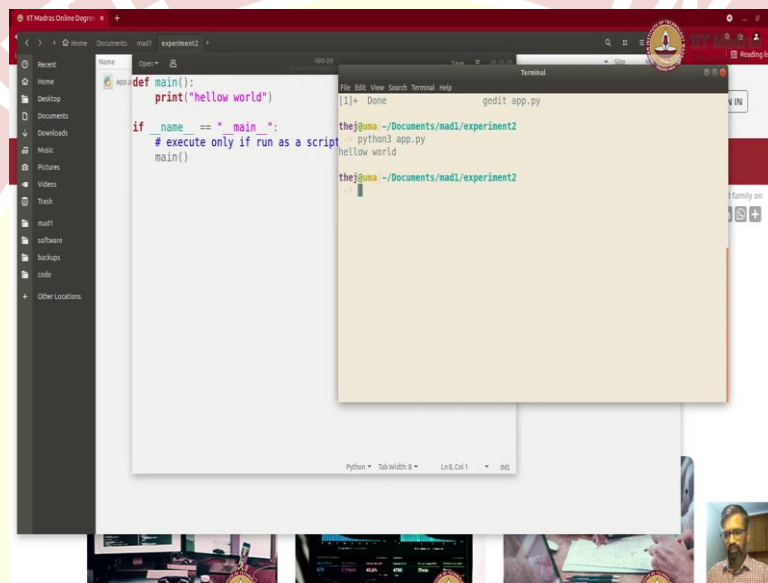


Let me create a folder called mkdir folder called experiment 2 where we will do all our experiment in this screencast. It is created, I will just go into the x folder, cd, we have nothing there. Now, let us create a file called app.py. This is going to be our application file. This is

where we are going to write some code. So, I am going to create this through command line, it should be straightforward.

If we do touch, it will create a new file. I will do app dot, let me just show you. Currently there is no file inside this. Now, if I do touch app.py, a file gets created here. Now, on to just open this file using g edit, you can right click and open with G edit. Or you can just open the g edit and file open. Or you can use the command prompt like, g edit app.py I should open file, clear the screen.

(Refer Slide Time: 3:17)

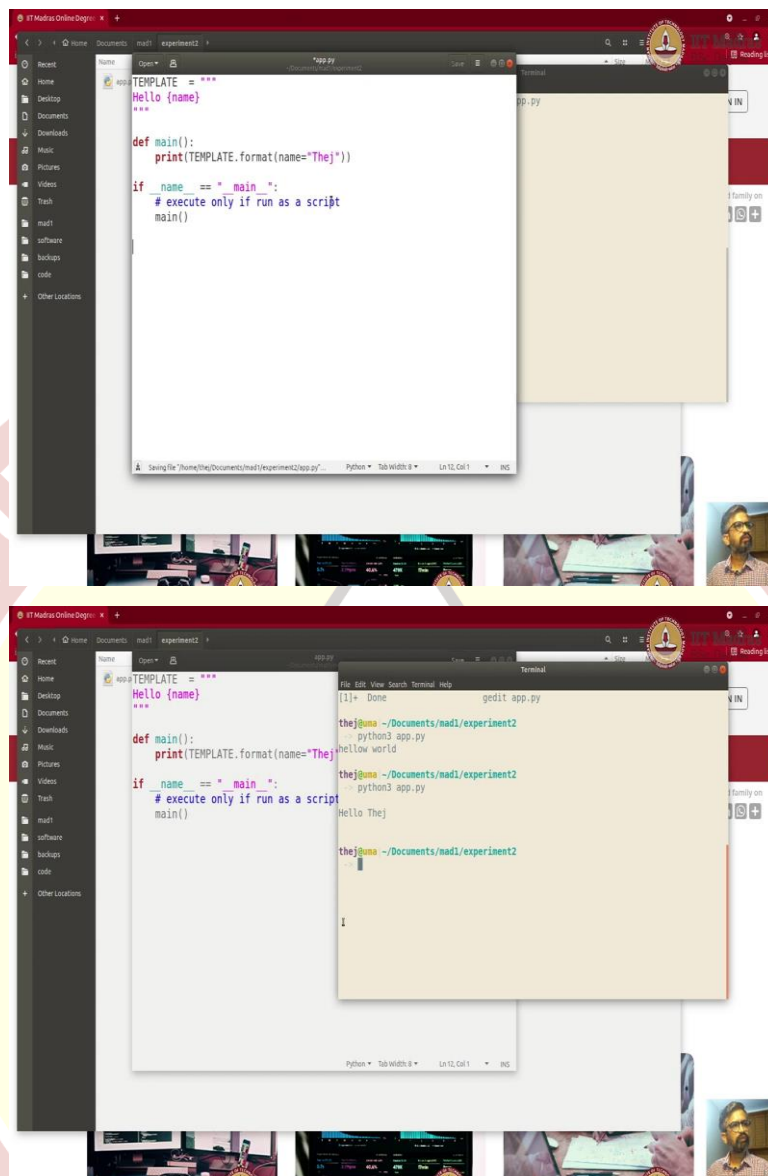


So, here is your app.py open. So, this is going to be your first Python file, we need to add a function called main. And also, we should run it as soon as it is called from the command prompt. Let me just do that. Let us define a function called main. Now, I want to run it only when it is invoked as a script.

So, this now when it is invoked, it will call this part. And then it will call function main the function main called print and types hello. So, let us try and run it, python 3, so it is working. Here, this part checks the current scope, name is underscore, underscore name underscore underscore is name. This current scope name is same as main. And then it then returns the function main.

This is just to make sure that script is called as Command prompt script and only run this otherwise, this would not run this otherwise. Let us continue, in the beginning, let us do a simple template or using Python for Mac. So, let me add a template.

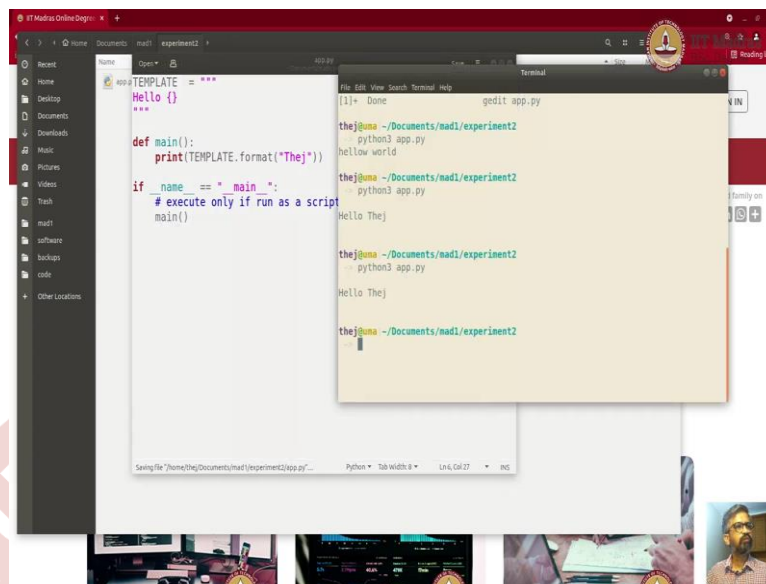
(Refer Slide Time: 5:24)



Instead of hello world, I want to add, let me just fix that typo. Instead of these I want actually to print the name or, format it as we go. So, I am going to add a template. I am presently adding it as a string here and say, hello name. And then instead of actually printing like this, I will just print template doc format, variable name is name, and the value is Thej. So, let me save this and run. As you can see, it is printing hello Thej. So, this is actually a simple string format, where it kind of replaces the variable with the value that we are passing. There are many ways to do it, we do not have to actually pass the name of the variable here.



(Refer Slide Time: 6:44)



The screenshot shows a Python IDE with a file named 'app.py'. The code in the file is as follows:

```
TEMPLATE = """
Hello {}

def main():
    print(TEMPLATE.format("Thej"))

if __name__ == "__main__":
    # execute only if run as a script
    main()
```

The terminal output shows the following commands and results:

```
thej@una: ~/Documents/nad1/experiment2
python3 app.py
Hello world

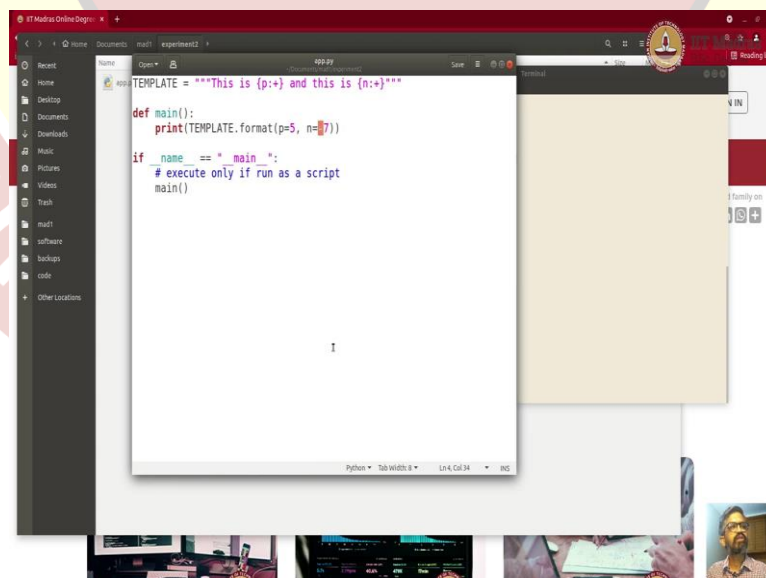
thej@una: ~/Documents/nad1/experiment2
python3 app.py
Hello Thej

thej@una: ~/Documents/nad1/experiment2
python3 app.py
Hello Thej

thej@una: ~/Documents/nad1/experiment2
python3 app.py
Hello Thej
```

It could be just empty, and we could just pass the value and that also should work. And if you want, for example, location of the thing, that also will work. There are many ways to do this. I think being named is much more easier so that you know what gets replaced, where. This is more intuitive. Now, it can do many other things. It is pretty advanced and can do complicated formatting's and stuff like that. Let us do another one where I want to print actually numbers.

(Refer Slide Time: 7:41)



The screenshot shows a Python IDE with a file named 'app.py'. The code in the file is as follows:

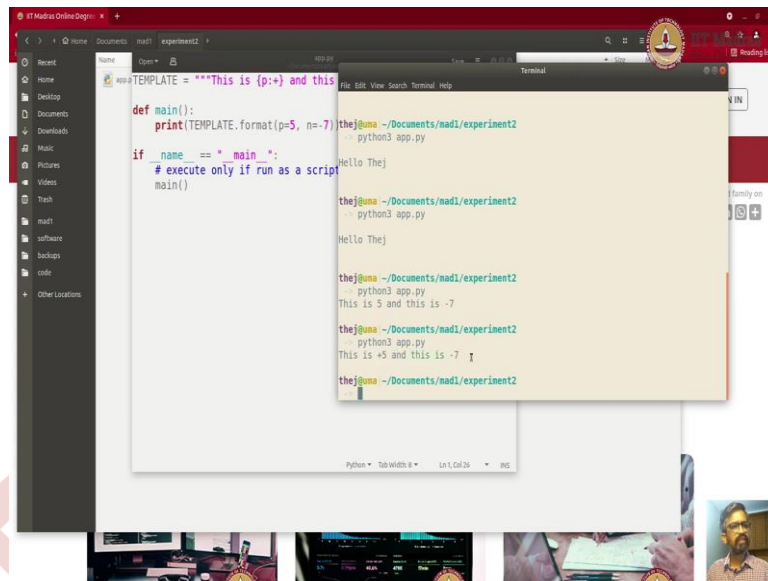
```
TEMPLATE = """This is {p:+} and this is {n:+}"""

def main():
    print(TEMPLATE.format(p=5, n=7))

if __name__ == "__main__":
    # execute only if run as a script
    main()
```

The terminal output shows the following command and result:

```
thej@una: ~/Documents/nad1/experiment2
python3 app.py
This is 5 and this is 7
```



```
def main():  
    print(TEMPLATE.format(p=5, n=-7))  
  
if __name__ == "__main__":  
    # execute only if run as a script  
    main()
```

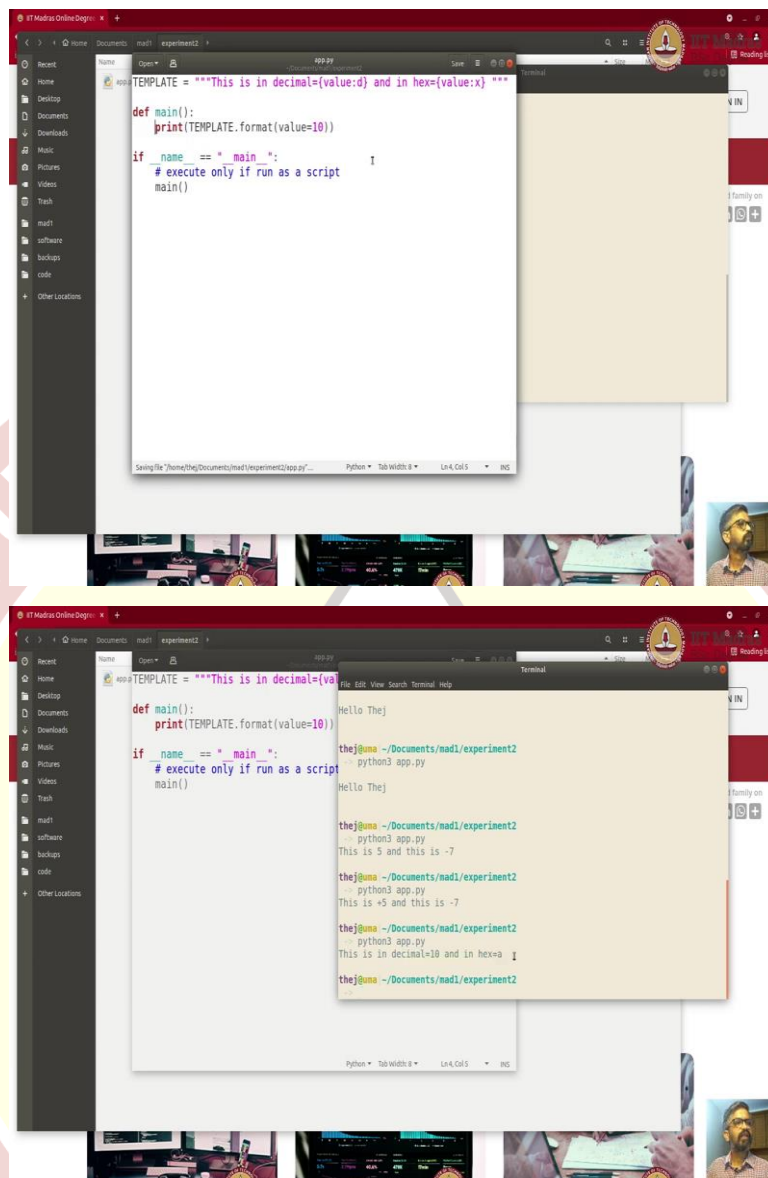
```
thej@uma ~/Documents/nad1/experiment2  
python3 app.py  
Hello Thej  
  
thej@uma ~/Documents/nad1/experiment2  
python3 app.py  
Hello Thej  
  
thej@uma ~/Documents/nad1/experiment2  
python3 app.py  
This is 5 and this is -7  
  
thej@uma ~/Documents/nad1/experiment2  
python3 app.py  
This is +5 and this is -7
```

Let us say if it was something like this. So here, I am actually printing two numbers. This is p, and then this is n, where p is plus and n is minus, a negative number. Let us run this. Actually, it does, please, this is 5. And this is minus 7. Let us say if you want actually print, this is as like a even for 5, which is positive, I wanted to show the sign. So, that can be done by adding specifiers you generally add it after the colon and name. So, it is usually done, let us add a specifier.

So, this actually tells it to add sign, regardless of the value, usually it will put sign only for negative numbers like before, it used to put a sign only for negative numbers. But when you say when you give a specifier plus, which is usually after given after the colon, if you keep plus, which is a specifier, which tells the format that please print sign of the number 2. Let us do now. So, you can see now it prints plus 5 and minus 7.

So, this is one of the things it can also print other format. For example, if you have a number in decimal, what do you want to print it in let us say in hexadecimal. You can use the same format for that. Let me just give an example. So, it is also done using the similar specifier.

(Refer Slide Time: 9:49)



```
TEMPLATE = """This is in decimal=(value:d) and in hex=(value:x) """  
  
def main():  
    print(TEMPLATE.format(value=10))  
  
if __name__ == "__main__":  
    # execute only if run as a script  
    main()
```

thej@nuna ~/Documents/nad1/experiment2  
python3 app.py  
Hello Thej  
This is 5 and this is -7  
This is +5 and this is -7  
This is in decimal=10 and in hex=a

So here, I have a template which says this is in decimal, whatever the value, and this is in hex, whatever the value; here colon format specifier after the colon d tells it print the decimal value. Here, x tells it to print the hexadecimal value. Let us print a number. Let us pass value 10 to it.

So, it is actually what is the hexadecimal value of 10? Let us find out it is a, as you would know from 9 until 9, it is nine and 10 becomes a in hexadecimal value is 10. So, you can use format specifiers to do such things. Well, lots of things that you could do, I will probably put a link in the notes, where you can go explore more about this.

There is also f strings, which is an advanced Python formatting, string literals. Also called f strings, that can be used to format string for printing or outputting to a file that is quite

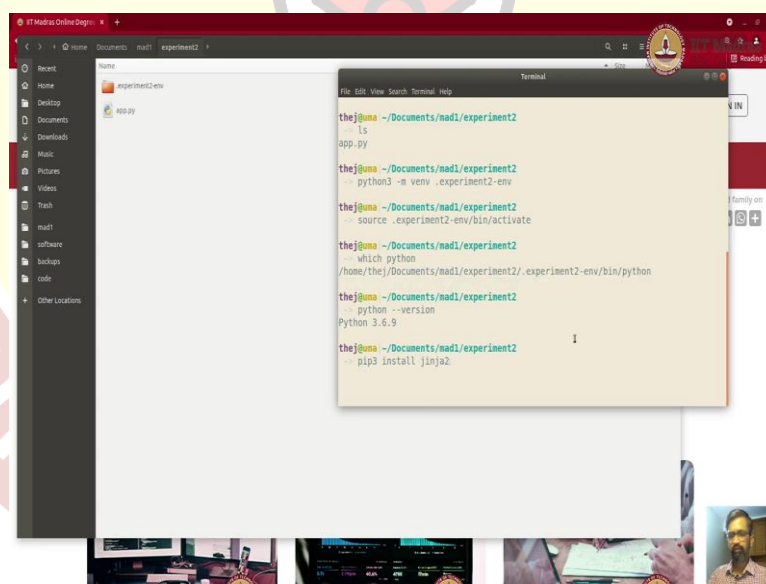


advanced, and also has lots of features. I will put a link, you can go explore it. Now, these are pretty good for, in line printing, logging, or giving message to us that kind of a smaller format of text.

If you want to do really big one, let us say you are generating a large document, for example, an HTML document from a template, then it is too complex to actually have it as a string for matter for which you would probably need an expressive, an extendable templating language. For example, Jinja2, there are quite a few of them, I am going to show Jinja2.

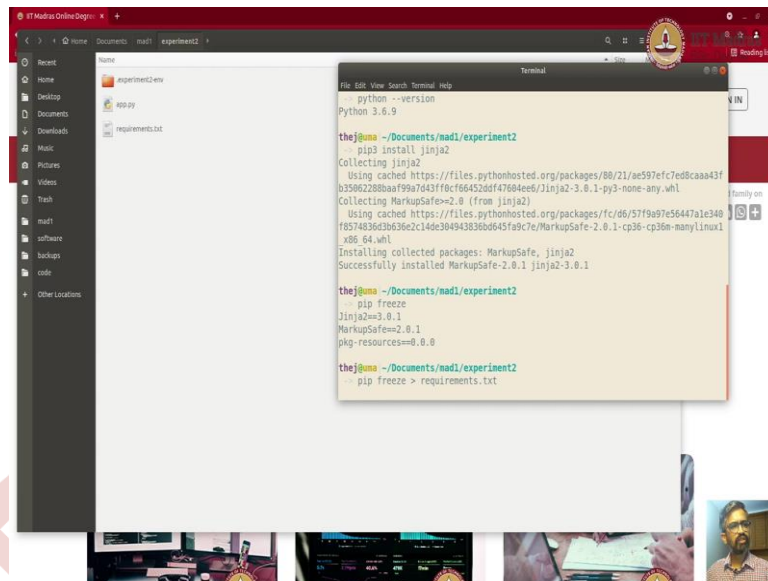
Jinja2 is a fast, expressive, extensible templating engine, special placeholders in the template allow writing code similar to Python syntax, and the template is passed the data to render the final document, that way, you are separating the view or the text or the content from the code, but also have a highly flexible and logic oriented templating language that can be used to produce no complex document. Jinja2 is an external library, it is not built into Python 3, so we need to install it. Before we begin installing, since we are installing an external library, I would actually like to do watch your environment before that.

(Refer Slide Time: 13:09)



The screenshot shows a terminal window with the following commands and output:

```
thej@uma ~/Documents/nad1/experiment2
ls
app.py
thej@uma ~/Documents/nad1/experiment2
python3 -m venv .experiment2-env
thej@uma ~/Documents/nad1/experiment2
source .experiment2-env/bin/activate
thej@uma ~/Documents/nad1/experiment2
which python
/home/thej/Documents/nad1/experiment2/.experiment2-env/bin/python
thej@uma ~/Documents/nad1/experiment2
python --version
Python 3.6.9
thej@uma ~/Documents/nad1/experiment2
pip3 install jinja2
```



So, let us create a virtual environment within the same folder that we have that is then by using this command. So, here I am telling use Python 3, create virtual environment command, use the virtual environment command or create virtual command, virtual environment command in a local folder called dot experiment 2. That is where all the virtual environment related files are stored. Let us do that. It might take a second. It is created.

Now, if you go see here, if you see all the hidden files because it starts with dot, you can see that there is experiment2-env. Now, we need to activate the virtual environment. So, that is done by sourcing the activate script, source dot go to our virtual environment directory, go to bin and activate, it is activated now.

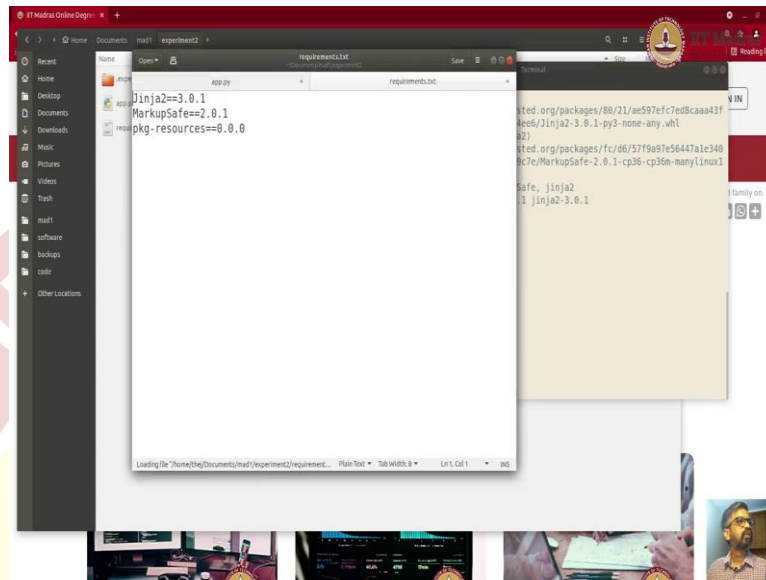
Now, just search which Python is available. Here you can see a local Python version is used. Since it is using the virtual environment. I can also check Python. So, it is 3.6 point. So, from now on we use of command Python, which is from the local. Now we actually want to install Jinja 3 so let us go ahead and install Jinja 3 using py py pip3. So, the command to install the package is pip3, install Jinja2, Jinja2 is the package name.

Now, this command will download and install the package within the local virtual environment that we have activated. Please note that this would not install in the whole system, only if the virtual environment is available, then Jinja2 will be available. So, now it is installed. I usually like to keep it installed package details in our requirements.txt. So, if you want to reinstall all the packages, all the dependencies, then we can do it very quickly.

You can read about how to use requirements.txt online, I will put a link but to generate requirements.txt automatically from installed packages, we can use pip freeze, which will

actually print all the installed package lists. So, you can see it has installed Jinja2 which indirectly has installed other things. You can directly create it or you can just output it to a requirements.txt. I am just going to output it to a requirements.txt.

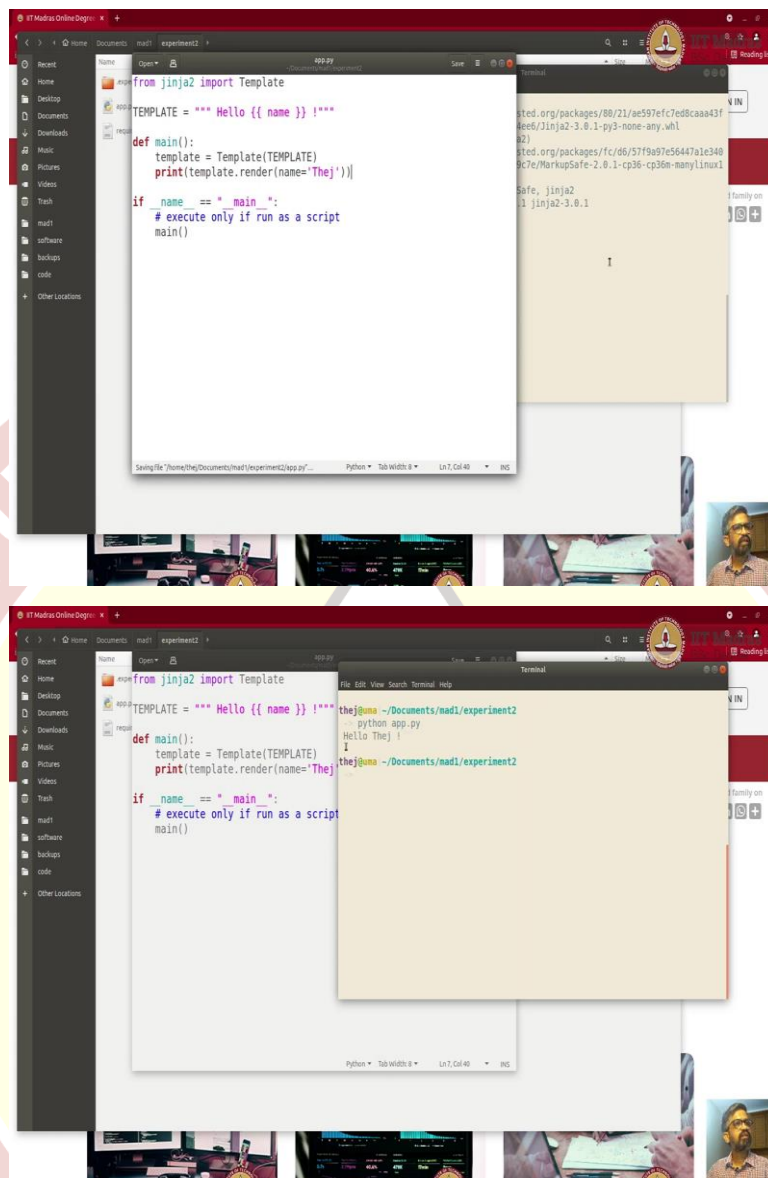
(Refer Slide Time: 16:37)



So, just create and then you have basically, if you open the requirements.txt, if you just open the requirements.txt, it just shows I have installed Jinja2 which is version 3.0.1 Mark safe, which is kind of installed by Jinja2 and the packages actually this is not required. So, I can remove it. Actually, even mark safe if is not required, you just put Jinja2, I think indirectly that will install mark safe, so should be fine. But I am just going to leave it at that.

Now, this will be very useful if you are going to share with your friends or you want to move to a production environment. What are the packages installed? To just to know that or you know, replicate the same thing, you can use the requirements.txt. Now, that we have installed the package, let us start using it. So, the simplest one will look like this.

(Refer Slide Time: 17:29)



So first, we have to import from our Jinja2 class called template. Let us installed input. So, that will be from Jinja2 import template. This is the template class that we are that we are going to use often I am probably going to use a lot here. So, we are importing that from Jinja. 2. Now, Jinja2 also uses somewhat similar templating system. But the way substitutions are done are a little different.

Let us say I want to do hello world here too, hello world, name. So, here in Jinja2, it is done hello, you can see this is a little bit different. Actually, where do you single flower brackets, here we are using double flower bracket. And then name, this is a simple template of Jinja2. Now, before we render it, what we need to do, we need to actually initialize the class template and pass on the string template to it.

Now, we are initializing this Jinja class template and passing the template string to it and we are getting a template object. Now, we can just render the template and pass on the variable that we want to print. So, here I am trying to render this template and passing on the variable name that gets substituted here, like you can see.

And then if we run it, now, we can run it here. Here, just be sure you are using Python, because the default type is here Python 3 within our virtual environment. So, I am going to run app.py. So, you can see it has done the substitution it has rendered and rendered gives you a final text and that gets printed. So, this is the most simplest Jinja template.

