

# **IIT Madras**

**ONLINE DEGREE**

**Modern Application Development – I**  
**Professor Nitin Chandrachoodan**  
**Department of Electrical Engineering**  
**Indian Institute of Technology, Madras**  
**Tools – Part II**

(Refer Slide Time: 00:16)



Programmatic HTML generation: PyHTML

- Composable functions - each function generates a specific output
  - Example:
    - to generate a h1 heading: function should return
- "<h1>text of heading</h1>"

```
def h1(s):  
    return "<h1>" + s + "</h1>"  
  
h1("Hello") → "<h1>Hello</h1>"
```



Hello, everyone, and welcome to this course on modern application development. Now, that takes care of the wire framing. In other words, it tells you how to construct the overall structure of the view. Then comes the question of how do I implement it? Because finally, I need to be able to generate html. Now, there are multiple ways of generating html. One of them, which I am, I am going to talk about a couple of them in a little bit more detail than others.

That does not in any way mean, that I am saying that these are the only ways of doing it. In fact, the thing that you need to keep in mind is, the simplest way for a program to generate html is just using print statements, you just keep print, print, print, print, and print out whatever should come as html. After all, if you can use it in order to output data into a file, it is perfectly legal and easy to do, to use it to generate the html, which is actually required for your application as well.

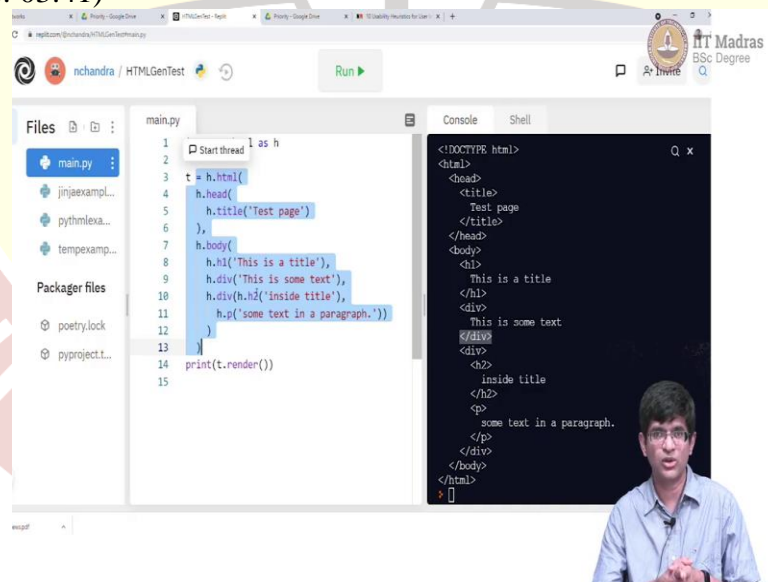
On the other hand, it is painful. Now, we are using Python for our application development. So, let us think about how we might go about solving this problem. And the first question that I would ask is, I want to generate a heading, so can I, for example, write a function that, given a string would basically generate the h1 tags corresponding to that heading, and return that to me? So, for example, I might have something which looks like this, def h1.

And it takes a string as input. And all that it does is, returns this plus the string plus. All that I have done is taken the tags h1 and slash h1, and put them on either side of the string and returned it. Why did I have to define a function for this? Because rather than having to type in the h1 slash h1, were there is a good chance that, just while typing, I will end up making a mistake, get one of the angle brackets wrong, forget the slash, and so on.

Instead, I can now just use h1 of Hello, and this will return to me, h1 Hello, slash h1. It takes care of the starting tag, the ending tag and everything else that is required very neatly over there. I can make this function a little bit more fancy, by adding on something which allows me to specify attributes for the h1, some CSS attributes. The CSS attribute might, for example, be used in order to say, use a different color for this or use a different font.

There could be locations where that is meaningful. But the point is, I, as long as I am sort of programmatically generating it using a function, I can do a lot more fancy things with it. So, I am going to take an example, which is this specific library called pyhtml. And what I am going to do is demonstrate this on the repl.it.com. playground.

(Refer Slide Time: 03:41)



```
1 from pyhtml import h as h
2
3 t = h.html(
4     h.head(
5         h.title("Test page")
6     ),
7     h.body(
8         h.h1("This is a title"),
9         h.div("This is some text"),
10        h.div(h.h2("inside title"),
11            h.p("some text in a paragraph."))
12    )
13 )
14 print(t.render())
15
```

```
<!DOCTYPE html>
<html>
  <head>
    <title>
      Test page
    </title>
  </head>
  <body>
    <h1>
      This is a title
    </h1>
    <div>
      This is some text
    </div>
    <div>
      inside title
    </div>
    <p>
      some text in a paragraph.
    </p>
  </body>
</html>
```

So, I have just created a example, repl.it, a repl, which can be used just to demonstrate what the pyhtml library is capable of. As you can see over here, let me zoom in a little bit. The main thing that comes across is that I am importing pyhtml as h, you will also find, another way of writing it which is basically from pyhtml import star. Now, I generally prefer to avoid doing that, there are locations where, it makes a lot of sense from pyhtml import star.

Just makes it easier to write your functions after that. But, there are also problems because, it now brings all of those functions, the exported names from the html module into your namespace, which means that now suddenly, there is a function called html, there is a function called head and so on, which might conflict with other functions that you had already defined. So, to avoid that, I prefer that you just import as h and then use h dot HTML h dot head and so on.

It gets a little painful, because you can clearly see I have h dot h dot h dot all over the place. But, by, choosing a single character out here, you are probably okay, it is not too bad. So, what does this code actually do? What it says is I am constructing a string. It basically says t is equal to h dot html. And as you can see, this html starts out here on line three, and ends out here on line thirteen. And inside the html, it essentially has two components.

It has actually only two components, one of them is the 'h.head', which starts on line 4 and ends on line 6, a comma, and 'h.body', which starts on line 7, and ends on line 12. So, what is actually happening over here? If you think about it, 'h.html' is taking two arguments. It could probably even take more arguments. Essentially, what it is going to do is take a list of arguments, and just output each of them one by one.

So, 'h.html' is pretty much what it is going to do is just put the html tag in front, and then whatever is inside it, will just be sort of sent out. And then we put a slash html at the end of the document. What is 'h.head' going to do? It is going to be, put a head and a slash head and send out whatever is inside that directly. What is 'h.title' going to do? Add a title slash title and send out whatever is inside its argument.

In this case, it is just text. So, what we can sort of imagine is that this 'h.title', is going to put out title test page slash title. So, in this way, I have constructed some complex set of, html structure out here. And what I do is, I basically say print 't.render()'. So, the render part of it basically says, this was a set of functions. Its t has been created as an 'h.html' output here. When I do 't.render()', what it is supposed to do is go and call all these functions.

So, let us try it. So, now we have the output that came from the print 't.render()'. What does it look like? Let us go take a look at it. We have this, it starts the first line is essentially this <!DOCTYPE html>. The next is html and if I go down to the bottom, I find that there is a slash

html. So, I can safely assume that the 'h.html' part of it, that function, its job was to print out this `<!DOCTYPE html>`, print the opening html tag and the closing html tag, those are the three things that it did.

Everything else inside it, what it did was it basically added two spaces of indentation. And then started calling the rest of the functions inside it. The first of those was 'h.head'. And here we have a head slash head. Inside that I have a title slash title. And inside that I have the actual text corresponding to the title. The nice thing out here, as you can see is that all of it has been nicely indented by two spaces.

I believe I can, that can also be configured, change the amount of indentation that you want, that is easy enough to do. And this is perfect, I will never forget a closing tag, if I have a title, I have a slash title guaranteed. If I have a div, I have a slash div. So, this programmatic approach of generating html has this nice property that basically what, we are doing over here is actually generating the data in such a way that as long as this function, this 't.render()' the this t was constructed properly, it will always generate valid html.

There is never any question of going wrong in terms of, forgetting to close a tag or using the wrong tag or anything else. All of that is done automatically by the code. Now, this looks very nice. There is still a question about whether it is scalable. Let us say that I need to generate an overall page, let us say Amazon, is trying to generate its output page in this fashion. And I then wanted to go into one part of it, and modify what happens in one particular block.

Very likely, if this is the style of code that I am using in order to generate the entire text, I will find that I have a problem. Because, I generate all of those html very nicely. But, then I want to go in and change one particular aspect of it. It is going to be quite difficult to figure out where exactly I need to change something. So, this is tremendously useful, but may not be the ideal sort of solution for what we wanted, what we have in mind.

(Refer Slide Time: 10:04)


IIT Madras  
BSc Degree

### Programmatic HTML generation: PyHTML

```
main.py
1 import pyhtml as h
2
3 t = h.html(
4     h.head(
5         h.title('Test page')
6     ),
7     h.body(
8         h.h1('This is a title'),
9         h.div('This is some text'),
10        h.div(h.h2('inside title'),
11              h.p('some text in a paragraph.'))
12    )
13 )
14 print(t.render())
```

Console Shell

```
<!DOCTYPE html>
<html>
<head>
<title>
  Test page
</title>
</head>
<body>
<h1>
  This is a title
</h1>
<div>
  This is some text
</div>
<div>
  inside title
  <p>
    some text in a paragraph.
  </p>
</div>
</body>
</html>
```



So, as you can see over here, with this example, I basically have all of this text that we had, in fact, whatever we saw in the example, and it generates this output.

(Refer Slide Time: 10:17)


IIT Madras  
BSc Degree

### More complex HTML

```
def f_table(ctx):
    return (tr(
        td(cell) for cell in row
    ) for row in ctx['table'])
```

*Handwritten annotations:*

- element* (pointing to `tr`)
- array/list* (pointing to `cell in row`)
- array* (pointing to `row`)
- array of arrays* (pointing to `ctx['table']`)



Now, the nice thing about programmatic html generation is that you could do something more fancy. This is an example from the website of pyhtml. You can define a function that is basically going to return table rows with table data, by iterating, so this `for cell in row` is essentially Python code. It is basically, a `'td(cell)'` for cell and row is what is called a list comprehension in Python.



So, essentially, row is an array or a list. This is an element. And that is an taken and placed within td tags. And how do I get row? This is an array, but the 'ctx[table]' is now going to be an array of arrays. So, as long as I have an array of arrays, and each one of those elements comes out as a row, I can then pass that in, and each of those elements of that row can then be encapsulated within td tags placed within this tr.

And the whole thing then comes out as one nice displayed tabled. And all of that was encapsulated into this one function that I wrote. So, it is possible to generate something fancy like this by using very minimal amounts of code by using a library such as pyhtml.

(Refer Slide Time: 11:58)

### Templates

- Standard template text
- Placeholders / Variables
- Basic (very limited) programmability
- Examples:
  - Python inbuilt String Templates - good for simple tasks
  - Jinja2 used by Flask
  - Genshi
  - Mako
  - ... - just pick one and go with it

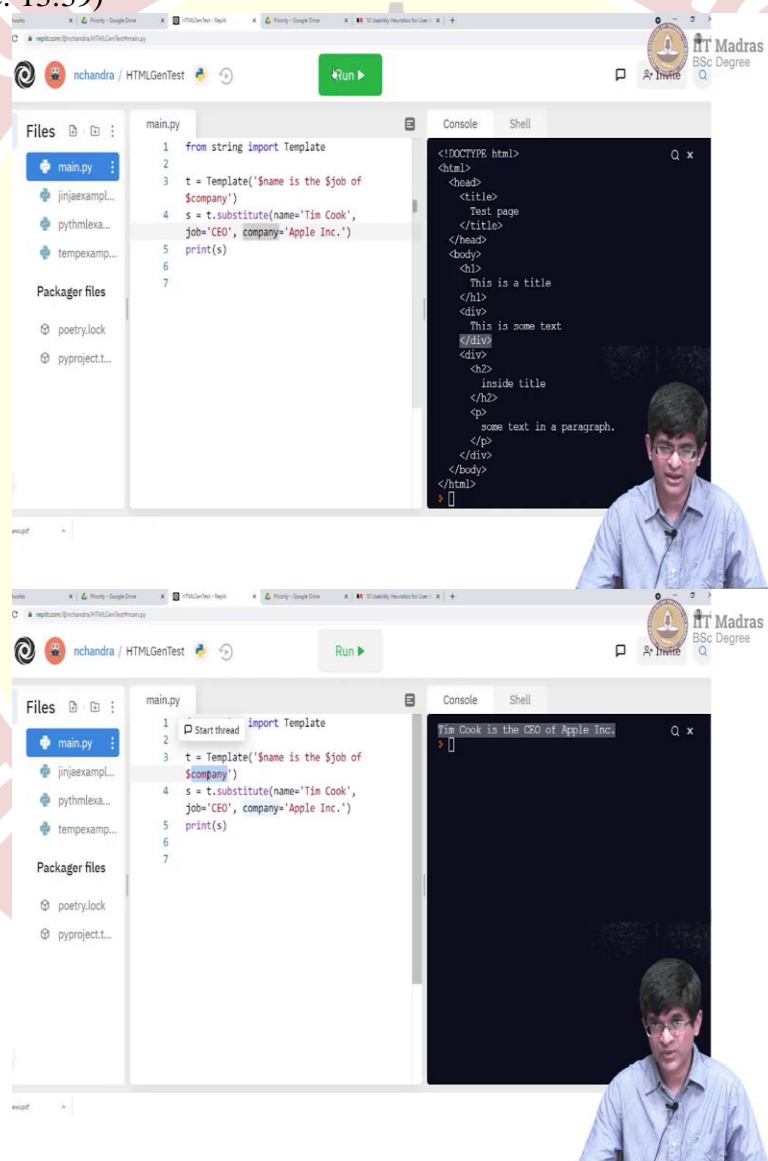


The problem, of course, as I said, is that let us say you need to go in and make a change in one part of the design. You will probably need to wade through a whole lot of pyhtml code and try to figure out where it is without damaging anything else. An alternative, which is preferable in general, is something called templates. And templates over here, there are, essentially they consist of some kind of a standard template text.

And some way, by which variable information can be brought into that text, it can be interpolated into the text. They also have some very limited means of programmability, usually to sort of iterate over loops. And there are many different examples of how you can implement templates. That is one of the drawbacks, which is that there are so many different ways of doing it that you do not know which one to choose.

So, ultimately, you probably just need to pick one and go with it. For our case, we are, since we are going to be using the flask framework. Again, the question might come, why are we using flask, it is because it is simple, and relatively straightforward. It is by no means the only one or many people would argue that it is not even the best one to use. But it is a good enough framework for illustrating all the concepts that we need, which is why we are choosing to go with flask. And flask works well with this templating engine called *jinja two*. And that is the one that we are going to be using for our applications over here.

(Refer Slide Time: 13:39)



So, using the same repl, we are now going to look at another example where we use the Python string template functionality. So, what we can see here is I am just from string importing

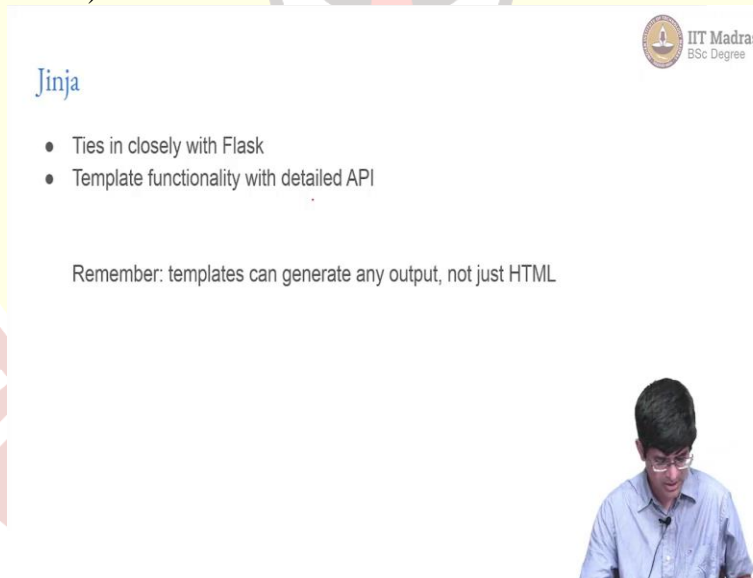


template. This is part of the standard Python library, I do not even actually need a separate module out here.

What I do is I create a template, where I say \$name is the \$job of \$company, anything with \$ is now a variable that needs to be replaced with some other information. How do I get the variable information? I say 't.substitute', and I give name, job, company and run it. So, what happens if I run this? It actually without any further delays, because after all template was part of the Python string library.

It straight away printed out Tim Cook is the CEO of Apple Inc. So, what has it done? It took the value for name from here, and put it in place of \$name. It took the value of job and put it in place of \$job. And it took the value of company and put it in place of \$company. So, this is once again, in some ways is slightly better than the programmatic html. Because you could have one chunk of text and then decide that, you need to make modifications in places. But, the programmatic html has its own benefits in other contexts.

(Refer Slide Time: 15:19)



Jinja

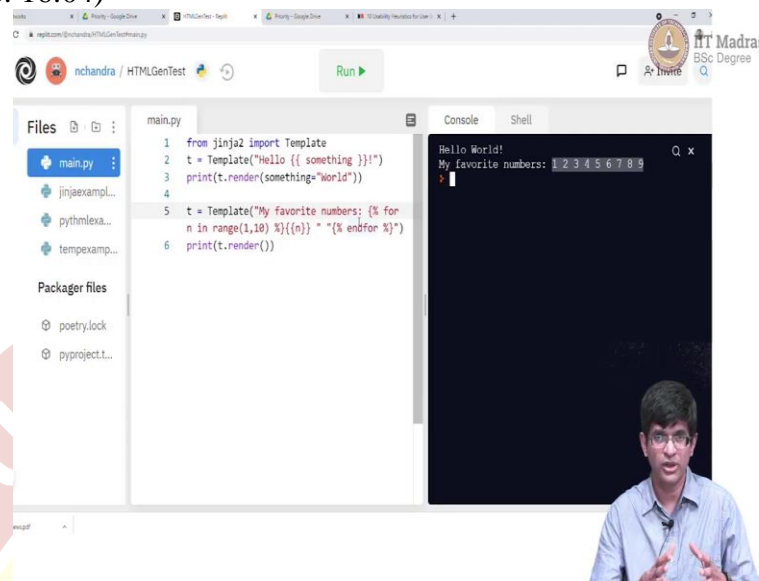
- Ties in closely with Flask
- Template functionality with detailed API

Remember: templates can generate any output, not just HTML

IIT Madras  
BSc Degree

Now, coming to the third way of generating output, which, as I said, is using the actual standard as templates. What we are going to do over here is use the jinja framework. And jinja is something that ties in closely with flask. And it has complicated template functionality with a detailed application programming interface. One thing to keep in mind with something like jinja is that it can generate any kind of output. It is not restricted to only html. So, let us once again look at the repl try and see what could be an example of using jinja.

(Refer Slide Time: 16:04)



```
1 from jinja2 import Template
2 t = Template("Hello {{ something }}!")
3 print(t.render(something="World"))
4
5 t = Template("My favorite numbers: {% for
6 n in range(1,10) %}{{n}} " "{% endfor %}")
7 print(t.render())
```

Console output:

```
Hello World!
My favorite numbers: 1 2 3 4 5 6 7 8 9
```

## Jinja

- Ties in closely with Flask
- Template functionality with detailed API

Remember: templates can generate any output, not just HTML

So, let us first understand what the code exactly had. So, from `jinja2` I imported `template`. And then I have created my first template, which was to say `t` is template of `Hello`, and you can see these double curly brackets. That is something that `jinja` uses in order to indicate that this, whatever is within the curly brackets, close curly brackets is something that needs to be evaluated and replaced. So, this something is now the name of a variable.

So, when I say `print 't.render()'` with something equal to the value `world`, it takes that and puts it in over here. I see that `Hello world` is my output. Because it was basically `Hello something` but something has now been replaced by `world`. Now, simple, straight forward, easy enough. Where

it gets interesting is when we, for example, have a loop. So, if I look at this new, the second template on line 5, I see that it is something which says that my favorite numbers.

And now rather than having two curly brackets, I have curly bracket and percentage. And inside that I have a for, which looks very much like a for loop in Python, or C or whatever it is, but a more like Python actually. Because it says for n in range(1,10). But, it is not exactly Python, because Python would then have had a colon sign over here. And, it would have like indentation and various other things to think about.

You do not need that over here. But the interesting thing, it, it allows you to run a for loop over this range. And it just basically prints out the n. And finally has a end for again, something which is not there in Python. What happened when we ran that print 't.render()' on that? It prints out the my favorite numbers part? And then it just prints the numbers 123456789. Why did it stop at 9, because the range(1,10) stops at 9.

Exactly like in Python. So, this is one way where it just sort of iterated, you could have used this in order to iterate through a, other, some other kind of variable, something which is an array or a list, or a dictionary, all of those could have been passed in as the argument. And it would have been able to handle those and iterate through them, as needed. So, that is where the power of jinja basically comes in.

The nice thing about it is, it also allows a number of other ways of using it, you cannot just construct a template like this. You can also embed templates one inside another. You can create a template for some small block, that needs to appear in one corner of your page. And then reuse that block across different pages. That kind of thing is generally harder to do with programmatic html.

And becomes relatively easy to do when you use templates, especially a good templating engine like jinja. One thing to keep in mind over here, is that these templates can generate any output. Think about what happened or in the examples that we saw on the replit, they just generated text. It was not really html, it could have been html. But it could also have been anything else. I could even in principle, have used a jinja template to actually generate a program by constructing some kind of a macro structure around it.

And maybe printing out what looks like a program after all, or I could have generated a PDF file or even an image, by directly putting data into the appropriate bitmap parts of a file and outputting. So, jinja output in other words can be used for much more than just html generation. As far as we are concerned, we will primarily be using it for html.

