INDIAN INSTITUTE OF TECHNOLOGY MADRAS

सिद्धिर्भवति कर्मजा

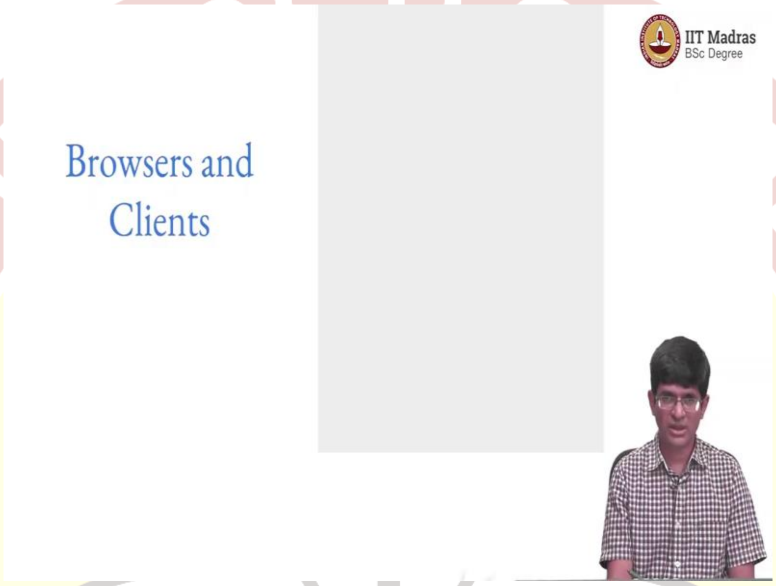
IIT Madras

ONLINE DEGREE

Modern Application Development – I
Professor Nitin Chandrachoodan
Department of Electrical Engineering
Indian Institute of Technology, Madras
Browser/Client Operations

Hello, everyone, and welcome to this course on Modern Application Development.

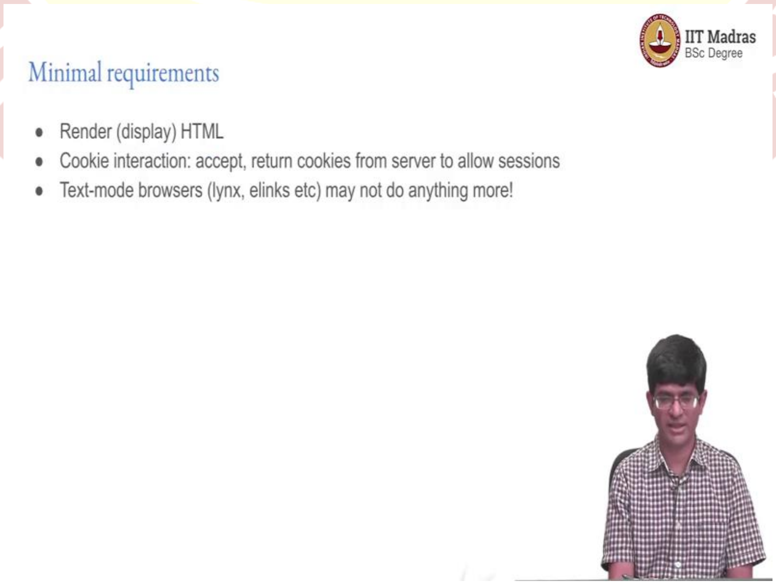
(Refer Slide Time: 0:17)



Browsers and Clients

So, now let us look a little bit deeper into what exactly a client looks like in the context of an overall web application. And in particular, we need to understand a bit more about what browsers are capable of doing and how they are used in general.

(Refer Slide Time: 0:29)



Minimal requirements

- Render (display) HTML
- Cookie interaction: accept, return cookies from server to allow sessions
- Text-mode browsers (lynx, elinks etc) may not do anything more!

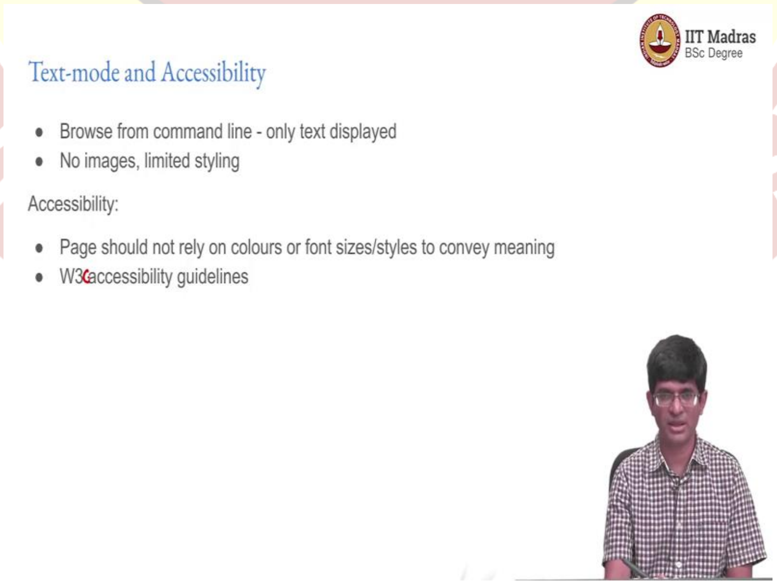
So, the minimal requirement from a browser client is that it should be able to render or display the HTML in some way. Now, keep that in mind when I say that, it should be able to display the HTML or render it in some way, it does not necessarily mean that it has to obey all the requirements of the HTML. In other words, typically, we are used to seeing h1 tags being bigger than the surrounding text, and so on.

That is not compulsory, that is not part of what HTML specifies. It just says this is a heading. So, it has to be rendered, the client has to be able to render it in some way that the end user will understand what it is that it is trying to show, that is all. There is also some amount of interaction with the browser, especially this thing called cookies, which we had mentioned at some point earlier in the context of authentication, but not really discussed in too much other detail.

But any browser to be useful beyond the bare minimum of just browsing different web pages should also be capable of accepting cookies from the server and returning them on demand, primarily to allow this concept of sessions to exist. Now, the interesting thing is there are browsers that operate in completely text mode.

They run from a command line, there is no GUI to speak off and what they can do is pretty much just show you the HTML output in a certain way, accept and respond to cookies, allow form interactions to some extent, but that is it, nothing more.

(Refer Slide Time: 2:09)



Text-mode and Accessibility

- Browse from command line - only text displayed
- No images, limited styling

Accessibility:

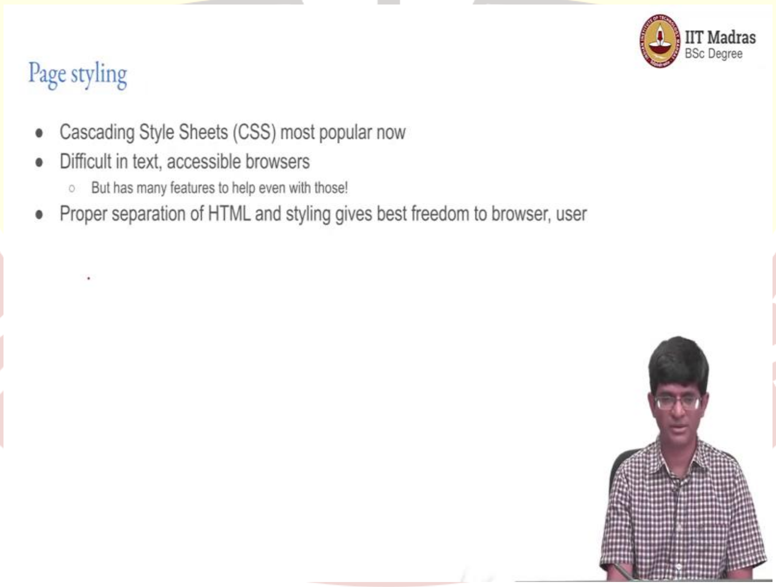
- Page should not rely on colours or font sizes/styles to convey meaning
- W3C accessibility guidelines

So, one thing which needs to be kept in mind is that it is potentially possible that somebody is browsing your website completely from the command line, and only the HTML text that is being written by the server can be displayed by such a browser, it cannot do too much JavaScript, it cannot do images, well, except maybe in some very specific cases, very limited amounts of styling, some amount of form interaction is possible, but most of them will not be able to handle JavaScript, for example.

Remember the accessibility guidelines we had spoken about in an earlier lecture, that is also something to keep in mind at the front end. Generally speaking, I mean main pages should not rely on colors or font sizes in order to convey meaning. The W3, the World Wide Web Consortium, W3C has accessibility guidelines, which basically talks about how this is to be handled and this should be W3C actually. All of those we have discussed earlier, all these also need to be kept in mind when you are thinking about developing a front end.

Having said that, you may consciously choose to say that, my target is some other set of browsers, and even accessibility guidelines sort of account for this, they say that, you cannot really assume that everyone is going to use a text mode browser, because that cuts down functionality too much. But if you are using other kinds of things, at least make it so that, even browsers that may not be seen by the user will be able to deliver useful content.

(Refer Slide Time: 3:45)


Page styling

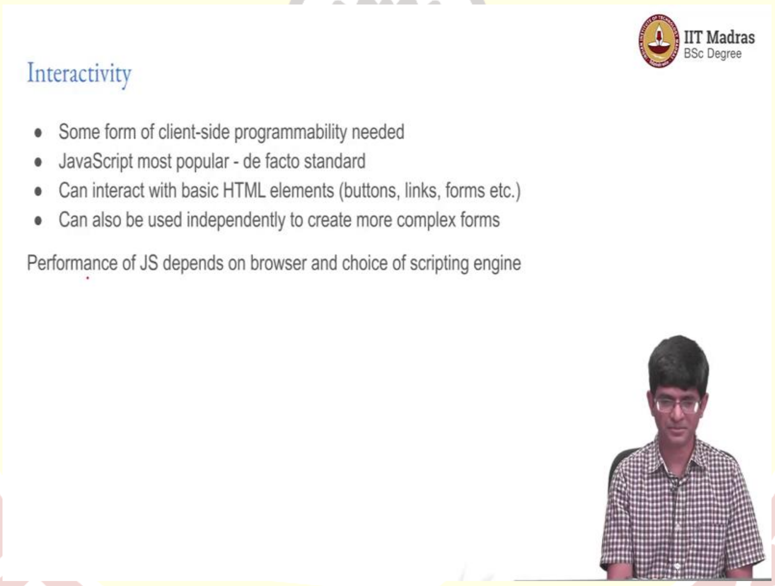
- Cascading Style Sheets (CSS) most popular now
- Difficult in text, accessible browsers
  - But has many features to help even with those!
- Proper separation of HTML and styling gives best freedom to browser, user

Now, the one of the things that browsers do is so called page styling, nowadays, of course we are, we just automatically assume that every all kinds of page styling is done using CSS Cascading Style Sheets. Once again, CSS is difficult in accessible browsers. But the thing about CSS is that it actually can even help with that. So even though you can misuse CSS in some ways, and try to use different font colors and font sizes, and so on.

The fact that you have done a proper separation of the HTML and styling, means that as long as you use CSS and HTML properly, you are actually able to give the best combination of freedom to the browser as well as to the user. And depending on what kind of browser is being used, it may be able to either ignore the CSS or use some hints from the CSS and make it even more accessible to the user than would normally have been possible otherwise.

So, all of these I mean, even the split between HTML and CSS came about because of this notion of separation, encapsulation at some level, all of these things come back to core ideas, such as each part of your design should do one thing and do it well. So, the CSS just takes care of styling and does a good job of it. HTML just takes care of giving you structured data or semantically informative data, and does a good job of that. So that is a good principle to keep in mind, generally speaking, when designing any app or a webpage.

(Refer Slide Time: 5:28)


Interactivity

- Some form of client-side programmability needed
- JavaScript most popular - de facto standard
- Can interact with basic HTML elements (buttons, links, forms etc.)
- Can also be used independently to create more complex forms

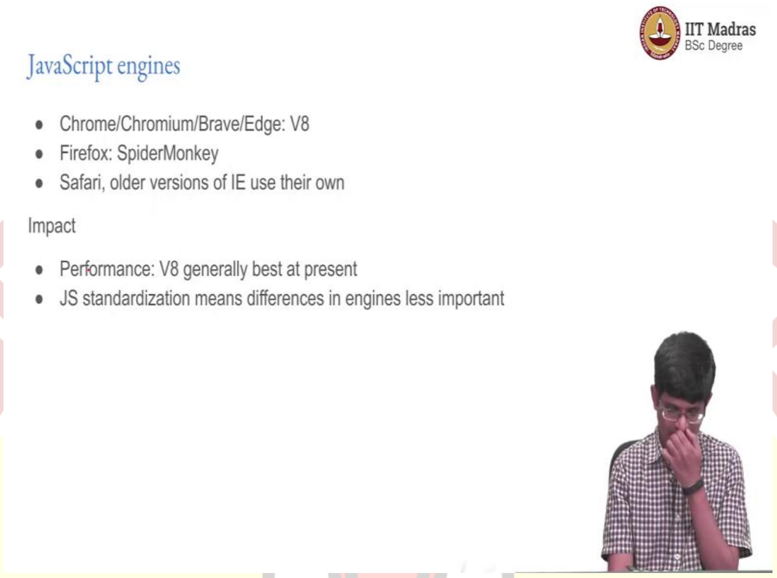
Performance of JS depends on browser and choice of scripting engine

Now, with all of this, you need some kind of interactivity. And some form of client-side programmability is needed. And as I had mentioned earlier, JavaScript became pretty much the de facto standard, meaning that even though nobody actually came up with an idea of standardizing JavaScript, it just was there, it was sufficiently good, it was sufficiently popular that nobody has been able to replace it with another language.

Now, JavaScript has very interesting, the original JavaScript was very limited in scope in terms of what it could do. But now a days, it can interact with lots of I mean pretty much the entire DOMs, so all basic HTML elements, and can also be used independently of the underlying HTML in order to create more complex structures that can, were not originally present on the system.

Now, of course, the performance of this JavaScript depends on the browser, and also on the choice of the scripting engine. So, there are many different interpreters that are available for JavaScript.

(Refer Slide Time: 6:31)



JavaScript engines

- Chrome/Chromium/Brave/Edge: V8
- Firefox: SpiderMonkey
- Safari, older versions of IE use their own

Impact

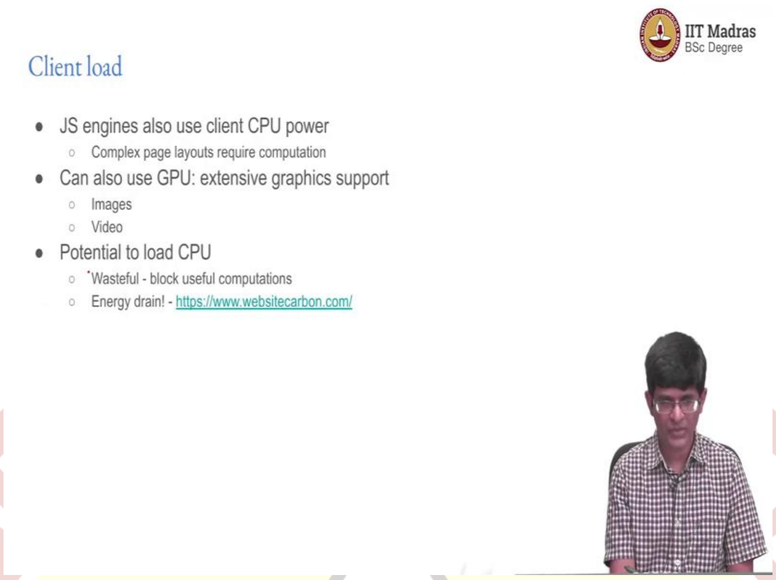
- Performance: V8 generally best at present
- JS standardization means differences in engines less important

And the most common ones are what is called V8, which is part of the Chrome browser, which I suspect that the majority of you are using. Those of you who are using Firefox, using actually a different JavaScript engine. But the thing is, both of them are, at this point, mostly API compatible, meaning that they have access to the same set of resources and the same kind of functionality.

So, what is different, why even have two engines? Well, developed independently by two different groups of people, one of them is faster at certain things. One is faster at another; V8 is generally considered one of the most performant engines at this point. But there might be reasons why you would want to choose, let us say Firefox. The Safari browser from Apple uses another variant, which is their own JavaScript engine.

The important point over here is all of these differ primarily in the kind of performance or the kind of memory resources or other things that they might use. But the fact that JavaScript has been reasonably well standardized at this point means that the differences between the engines are actually quite less important at this point in time, most of them can be expected to behave more or less the same on any JavaScript that you throw at them.

(Refer Slide Time: 7:50)


Client load

- JS engines also use client CPU power
  - Complex page layouts require computation
- Can also use GPU: extensive graphics support
  - Images
  - Video
- Potential to load CPU
  - Wasteful - block useful computations
  - Energy drain! - https://www.websitecarbon.com/

Now, what happens when you actually use JavaScript or a JavaScript engine, remember that this JavaScript is now going to run on the client side in the browser, which means that it is using the CPU power of the client. And this means that it is quite possible that you could have complex computations that are happening as part of the page layout, maybe, which all are going to load the CPU of the client.

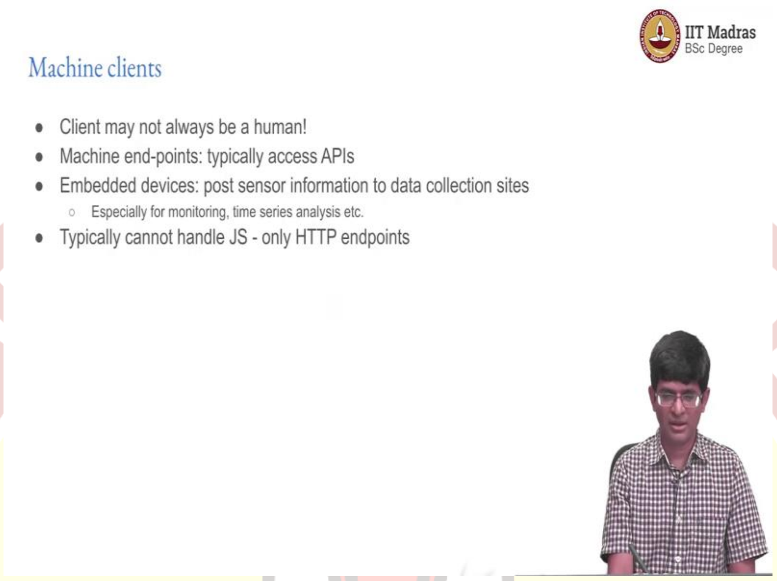
Many JavaScript engines also allow you to make use of the graphics processing unit, they have extensive graphics support, especially when you want to render images or let us say play video directly inside webpage. All this in turn means that you are actually loading the client-side CPU and in the worst case, it is even possible to get it to the point where you are blocking useful computation.

In fact, there are certain sites that can cause your browser and in fact, your entire system to hang just by running JavaScript code. Increasingly, the browsers try to prevent it from getting to a point where your entire system hangs. So, most of Chrome, for example, has this ability to compartmentalize each page so that even if one page sort of tries to use too much CPU, it can be isolated from the rest and maybe killed off without harming the rest of your system.

There is an interesting website in this relation, which talks about the energy drain related to any webpage, you can just go to this and they basically access whichever URL you give. And based on the number of requests, the size of the requests, where it is coming from, and so on, they do some kind of calculation and tell you that, this is the approximate amount of impact that you have on the carbon dioxide emissions worldwide.

Now, it is more of a sort of toy example than anything else. But it is kind of interesting, I mean, they are trying to refine their computation so that they are reasonably at least keeping up with most of the main important things.

(Refer Slide Time: 9:57)


IIT Madras
BSc Degree

Machine clients

- Client may not always be a human!
- Machine end-points: typically access APIs
- Embedded devices: post sensor information to data collection sites
    - Especially for monitoring, time series analysis etc.
- Typically cannot handle JS - only HTTP endpoints

Now, apart, so far, we have been talking about JavaScript and the various forms of clients that we have. But one important thing to keep in mind is that a client may not always be a human being. So, when I say, the end user, rather, may not always be a human being, because the client is actually speaking the program that is making the requests, and is obviously not a human. But usually the client is making the request on behalf of a human being.

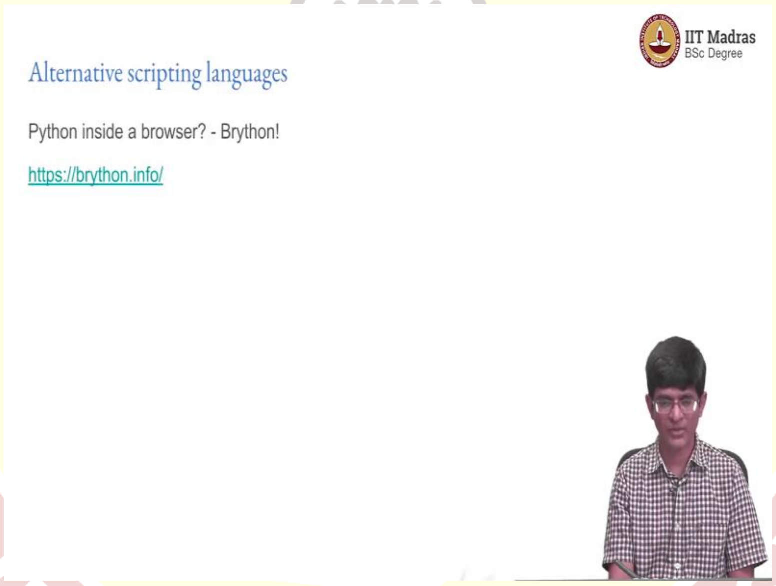
But there are machine endpoints, especially for things like embedded devices, let us say that you have got temperature sensors being spread out all around the city. And they all need to sort of pick up the local temperature and post the information onto a central database. One of the simplest ways to do this is to actually define an API on the central database. And each of these things basically just runs a small web client, which connects to this server and post the information.

Why? Because this actually turns out to be easier to construct than having your own network protocol. Because when you construct your own network protocol, and the clients need to connect on a given port and send in a certain way, it means that you are losing out on a lot of things that are naturally available at the server end, which would be possible if you used a web framework.

So, for example, if I tried building the server as a web application, it means that interacting with the database, having something else which can then read out the information, providing an API for others to use it, all of that becomes very easy. And as far as the end clients, the temperature sensors are concerned, it is no different, whether you have to connect to a socket and send structured information, or you just do a web request, it is pretty much the same amount of code as far as the endpoint is concerned.

But the important thing to keep in mind is there can be cases where your end client is not a human being, the end user is not a human being it is a machine. And of course, in such cases, they cannot handle things like JavaScript, typically, of course, they would be handling only specific endpoints, an API or something like that. But still, it is something to keep in mind while designing the system.

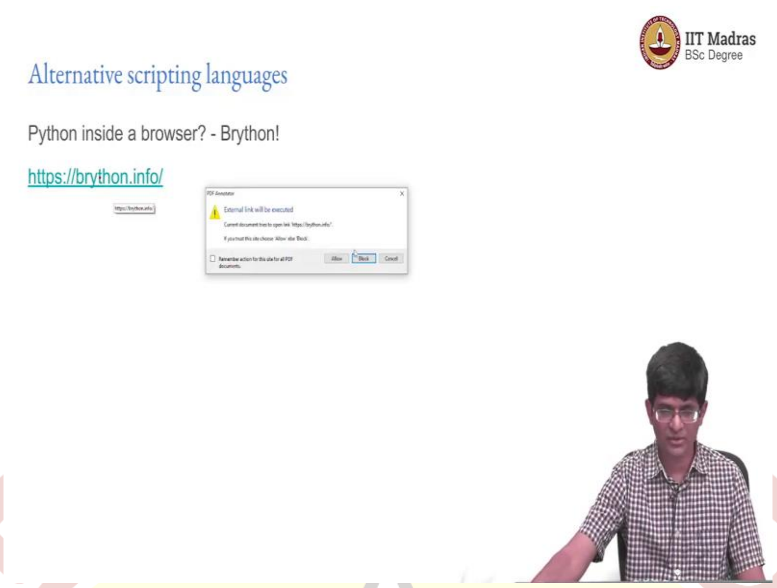
(Refer Slide Time: 12:15)



Alternative scripting languages

Python inside a browser? - Brython!

https://brython.info/

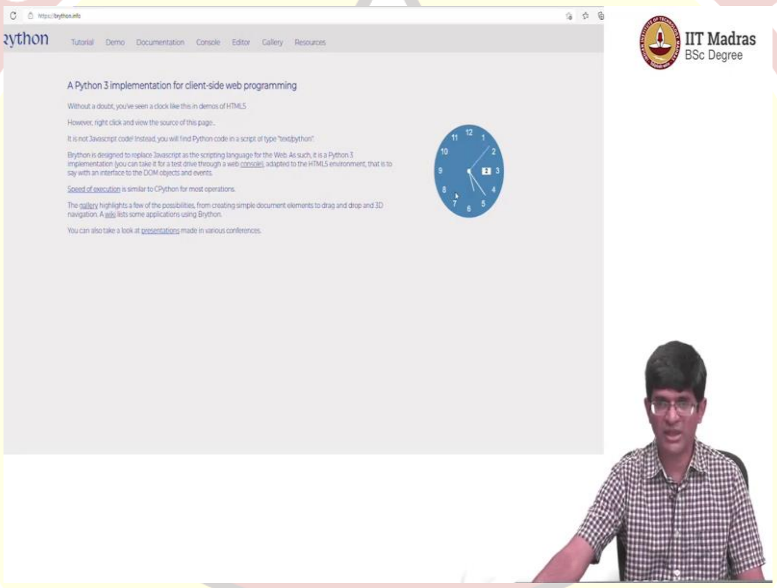


Alternative scripting languages

Python inside a browser? - Brython!

https://brython.info/

https://brython.info/
PDF Annotator
External link will be executed
Current document tries to open link 'https://brython.info/'.
If you trust this site choose 'Allow' else 'Block'.
Remember action for this site for all PDF documents.
Allow
Block
Cancel
IIT Madras
BSc Degree



A Python 3 implementation for client-side web programming

Without a doubt, you've seen a clock like this in demos of HTML5

However, right click and view the source of this page...

It is not Javascript code! Instead, you will find Python code in a script of type "text/python".

Brython is designed to replace Javascript as the scripting language for the Web. As such, it is a Python 3 implementation (you can take it for a test drive through a web console), adapted to the HTML5 environment, that is to say with an interface to the DOM objects and events.

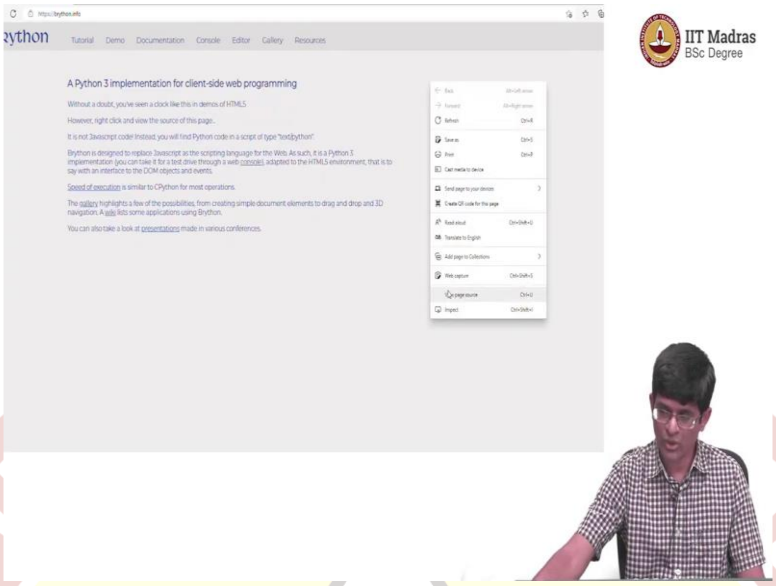
Speed of execution is similar to CPython for most operations.

The gallery highlights a few of the possibilities, from creating simple document elements to drag and drop and 3D navigation. A wiki lists some applications using Brython.

You can also take a look at presentations made in various conferences.

Now, one other interesting thing that can come up is so far, we have been talking about JavaScript as being sort of, as I said, the de facto standard on the web. There are other things in particular, there is this one thing called Brython, which if you go to this webpage, write brython.info, you will see that it essentially talks about a Python 3 implementation for client-side web programming. And you can see on the right-hand side over here, there is this clock. And you look at it, it looks no different from a typical JavaScript clock.

(Refer Slide Time: 12:55)



https://brython.info

ɛython Tutorial Demo Documentation Console Editor Gallery Resources

A Python 3 implementation for client-side web programming

Without a doubt, you've seen a clock like this in demos of HTML5.

However, right click and view the source of this page..

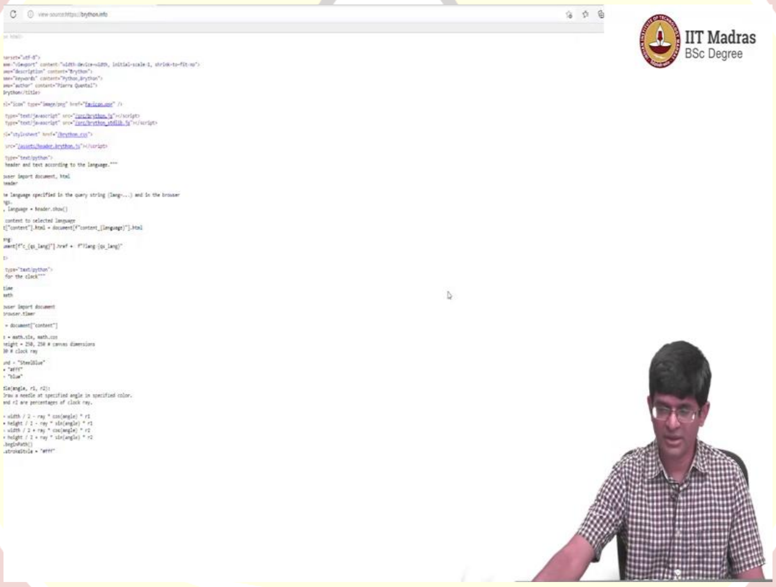
It is not Javascript code! Instead, you will find Python code in a script of type "text/python".

Brython is designed to replace Javascript as the scripting language for the Web. As such, it is a Python 3 implementation (you can take it for a test drive through a web console), adapted to the HTML5 environment, that is to say with an interface to the DOM objects and events.

Speed of execution is similar to CPython for most operations.

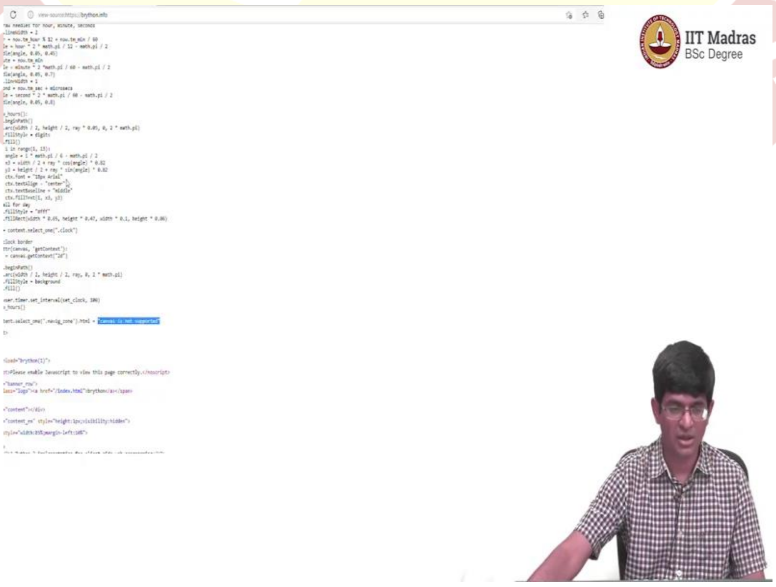
The gallery highlights a few of the possibilities, from creating simple document elements to drag and drop and 3D navigation. A wiki lists some applications using Brython.

You can also take a look at presentations made in various conferences.



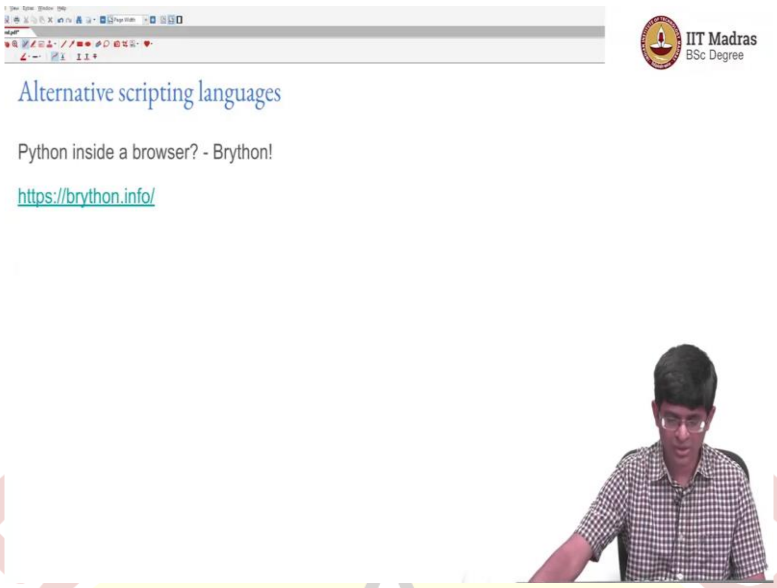
view-source:https://brython.info

```
:e html>

harset="utf-8">
ame="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">
ame="description" content="Brython">
ame="keywords" content="Python,Brython">
ame="author" content="Pierre Quentel">
Brython</title>
el="icon" type="image/png" href="favicon.png" />
 type="text/javascript" src="/src/brython.js"></script>
 type="text/javascript" src="/src/brython_stdlib.js"></script>
el="stylesheet" href="/brython.css">
 src="/assets/header.brython.js"></script>
 type="text/python">
 header and text according to the language."""
owser import document, html
header
he language specified in the query string (lang=...) and in the browser
ngs.
, language = header.show()
 content to selected language
t["content"].html = document[f"content_{language}"].html
ng:
ment[f"c_{qs_lang}"].href += f"?lang={qs_lang}"
t>
 type="text/python">
 for the clock"""
time
math
owser import document
browser.timer
 = document["content"]
s = math.sin, math.cos
height = 250, 250 # canvas dimensions
00 # clock ray
und = "SteelBlue"
= "#fff"
= "blue"
dle(angle, r1, r2):
Draw a needle at specified angle in specified color.
and r2 are percentages of clock ray.
= width / 2 - ray * cos(angle) * r1
= height / 2 - ray * sin(angle) * r1
= width / 2 + ray * cos(angle) * r2
= height / 2 + ray * sin(angle) * r2
.beginPath()
.strokeStyle = "#fff"
```



view-source:https://brython.info

```
raw needles for hour, minute, seconds
.lineWidth = 3
r = now.tm_hour % 12 + now.tm_min / 60
le = hour * 2 * math.pi / 12 - math.pi / 2
dle(angle, 0.05, 0.45)
ute = now.tm_min
le = minute * 2 *math.pi / 60 - math.pi / 2
dle(angle, 0.05, 0.7)
.lineWidth = 1
ond = now.tm_sec + microsecs
le = second * 2 * math.pi / 60 - math.pi / 2
dle(angle, 0.05, 0.8)
_hours():
.beginPath()
.arc(width / 2, height / 2, ray * 0.05, 0, 2 * math.pi)
.fillStyle = digits
.fill()
 i in range(1, 13):
 angle = i * math.pi / 6 - math.pi / 2
 x3 = width / 2 + ray * cos(angle) * 0.82
 y3 = height / 2 + ray * sin(angle) * 0.82
 ctx.font = "18px Arial"
 ctx.textAlign = "center"
 ctx.textBaseline = "middle"
 ctx.fillText(i, x3, y3)
ll for day
.fillStyle = "#fff"
.fillRect(width * 0.65, height * 0.47, width * 0.1, height * 0.06)
= content.select_one(".clock")
clock border
ttr(canvas, 'getContext'):
 = canvas.getContext("2d")
.beginPath()
.arc(width / 2, height / 2, ray, 0, 2 * math.pi)
.fillStyle = background
.fill()
wser.timer.set_interval(set_clock, 100)
_hours()
tent.select_one(".navig_zone").html = "canvas is not supported"
t>

load="brython(1)">
t>Please enable Javascript to view this page correctly.</noscript>
="banner_row">
lass="logo"><a href="/index.html">brython</a></span>
="content"></div>
="content_en" style="height:1px;visibility:hidden">
style="width:85%;margin-left:10%">
```


Alternative scripting languages

Python inside a browser? - Brython!

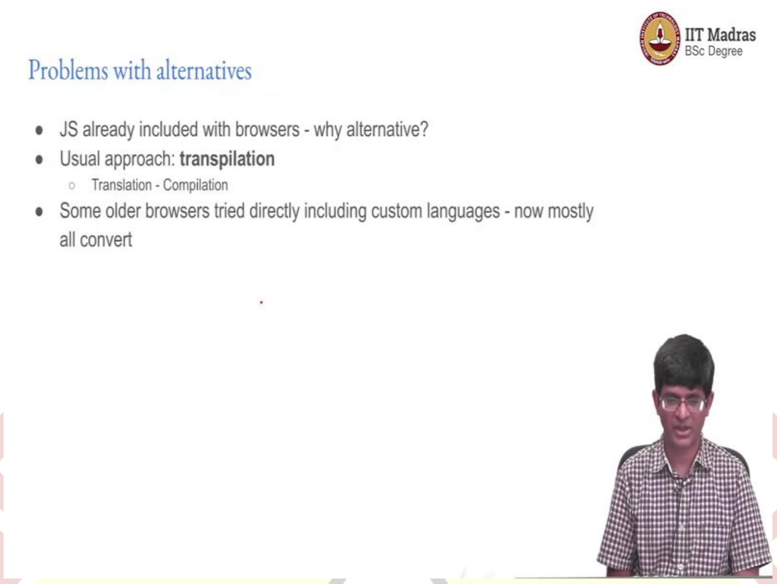
https://brython.info/

The interesting thing is when you go and view the page source click on it, and view page source. And as you go down over here, you will see that there is a script type equal to Python. and all of this is Python code. It defines a needle, it basically says set clock, it raises the clock to start with, it basically shows the hours, it creates a canvas. And finally, it basically says, show the entire thing out here.

And if none of this was possible, in other words, the canvas was not present, in other words, your browser does not support this, it will finally say canvas is not supported and give you an error message. All Python code put into the browser. Now, what is the catch?

Even though this looks very nice, and basically gives the impression that you now have Python code running inside the browser, in practice, what is actually happening is there is a JavaScript engine, which is taking that Python and interpreting it, it is doing it sufficiently fast that it sort of gives the impression of actual Python code running in the browser. But the bottom line is that it is still JavaScript, which is running at the inside the engine.

(Refer Slide Time: 14:06)

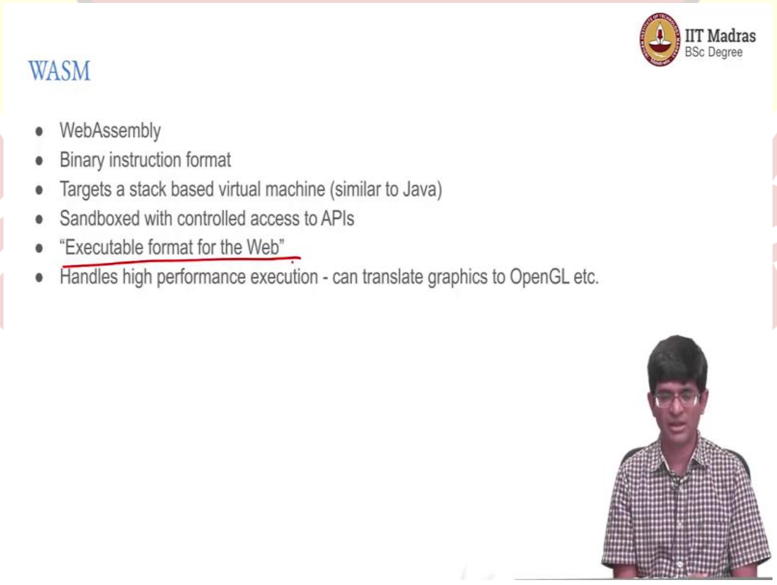

IIT Madras
BSc Degree

Problems with alternatives

- JS already included with browsers - why alternative?
- Usual approach: **transpilation**
  - Translation - Compilation
- Some older browsers tried directly including custom languages - now mostly all convert

So, JavaScript is already included with browsers. So, then anybody who wants to propose an alternative to that has to give a very good reason. So, what is usually done is this approach called transpilation, a translation plus compilation step, which is done, I mean, so you can actually write code in many different languages and get it converted into JavaScript. Some older browsers tried to directly include custom languages. But now, most of the time, the more popular approach is to convert into JavaScript and run it.

(Refer Slide Time: 14:41)


IIT Madras
BSc Degree

WASM

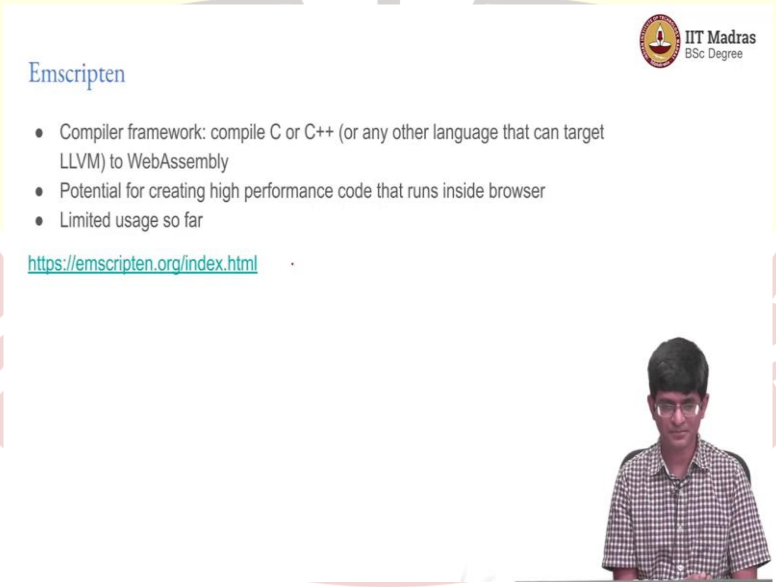
- WebAssembly
- Binary instruction format
- Targets a stack based virtual machine (similar to Java)
- Sandboxed with controlled access to APIs
- "Executable format for the Web"
- Handles high performance execution - can translate graphics to OpenGL etc.

Having said that, nowadays, there is increasingly another sort of front-end language, which is called WebAssembly. It is not so much a language as a binary instruction format, which basically claims to be a so-called executable format for the web. It literally allows you to create binary executables that can run inside a web browser.

And it sort of uses the idea of a virtual machine similar to what Java does, and potentially can handle fairly high-performance execution, including access to graphics cards, and so on. It is still not very popular. Part of the reason is that, I mean, it becomes too specialized when you are actually creating a compiled executable, which is going to run on a browser. And, most of the time, at least the basic interactivity that people are used to, can be done with something much simpler than that.

So, is it really necessary? Where could it help? Especially in things like online office suites, Microsoft Office on the desktop is actually usually more quicker to respond than on the web. If it is compiled into WASM, will that actually help? Will it become faster? Potentially, yes, you could actually have like full blown applications, you could even have things like, 3D editing and so on happening inside a browser when it is powered by something like this. So, all this is sort of, front end moving into the future. I mean, it is not really all that popular right now, at least.

(Refer Slide Time: 16:10)



Emscripten

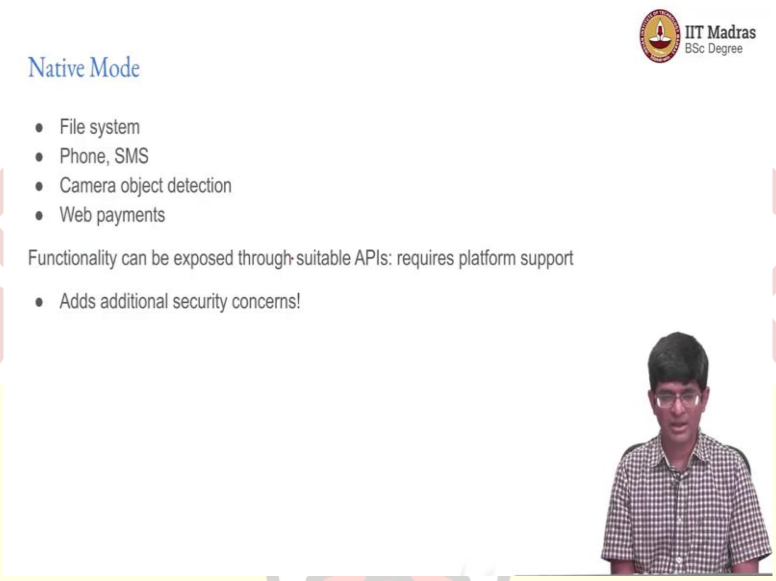
- Compiler framework: compile C or C++ (or any other language that can target LLVM) to WebAssembly
- Potential for creating high performance code that runs inside browser
- Limited usage so far

https://emscripten.org/index.html

There is a framework called Emscripten, which actually can compile code. So, you can write C or C++ code, or actually many other languages as well, and convert them directly into WebAssembly. Once again, it is sort of seen limited usage so far, but has a lot of potential in terms of what it can do to the whole front-end development moving forward.

So, all of this at the end of the day is mostly information that is useful for you. It is not like; we are expecting you to use this at least as part of this course. But I think as app developers, you need to know what are the possibilities and what are the likely things you are going to see moving forward.

(Refer Slide Time: 16:50)



Native Mode

- File system
- Phone, SMS
- Camera object detection
- Web payments

Functionality can be exposed through suitable APIs: requires platform support

- Adds additional security concerns!

One last thing in the context of the browsers that we interact with, there is also this notion of native mode. And what we mean by native mode is I would like the web application to behave like a regular application and what is the main difference? A regular application that is running on my laptop, let us say, allows me to access and create files, it allows me to save documents.

If it is on a phone, it will allow me to sort of send and receive messages, make phone calls, use the camera APIs or the web payment APIs in order to interact with various other things. Now, the reason why most of the time JavaScript does not have direct access to these things is because of security, which we will get to later. We do not want too much sort of functionality being exposed directly to the browser.

But it can be done through APIs and if you can, sort of, as a user can somehow confirm that, yes, this app is going to be used in a specific way after installing it locally, and so on. There are many cases where it sort of opens up the ability to use many of these APIs. And in that way, it is actually possible to create an app, which directly will, which is run using web technologies, but runs almost like a native client on your machine.

Now, sort of an example of that is something like the Visual Studio Code editor, VS Code. VS Code is basically a web app, it is a chromium browser, which is running on your machine. And the entire thing, including, the rendering of the pages, the syntax highlighting all of that is ultimately HTML plus CSS, which is why you can also sort of customize it, all the themes are ultimately about writing some form of CSS and JavaScript. So, there are things that can be done in native mode which enhance the functionality of the front end even further.