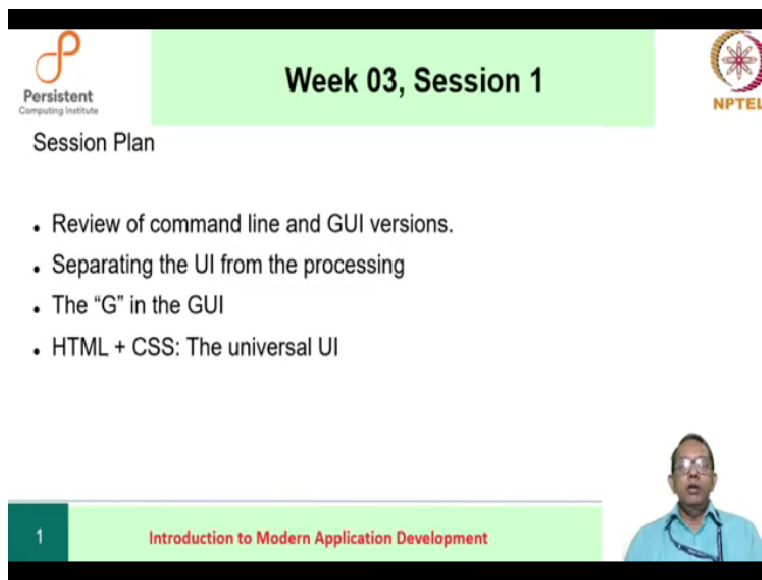


Introduction to Modern Application Development
Prof. Aamod Sane
FLAME University and Persistent Computing Institute
Abhijat Vichare
Persistent Computing Institute
Madhavan Mukund
Chennai Mathematical Institute

Lecture-11
Producing HTML + CSS output - Part 1

(Refer Slide Time: 00:11)

The image shows a presentation slide from NPTEL. At the top, there is a green header bar with the text "Week 03, Session 1". To the left of this bar is the Persistent Computing Institute logo, and to the right is the NPTEL logo. Below the header, the text "Session Plan" is displayed. Underneath, there is a bulleted list of topics: "Review of command line and GUI versions.", "Separating the UI from the processing", "The 'G' in the GUI", and "HTML + CSS: The universal UI". At the bottom of the slide, there is a small video feed of a man in a blue shirt. Below the video feed, there is a green bar with the text "Introduction to Modern Application Development" and a small number "1" in a green box on the left.

Hello, everyone, welcome to the first session of the third week of the course of introduction to modern applications development. Last week, we saw the command line version of the fair share application. We also saw the graphical version of the same application. The command line version was a simple code that actually implemented the basic algorithm that we saw in the very first week of this course.

As written, the command line version illustrated a procedural style of programming in a Java language. Despite Java being an object-oriented language, we used it to program in a procedural style. However, the object-oriented facilities support that Java provides was used during the graphical version of this code.

In the command-line version, we distinguished between interactive and non interactive style of interacting with the user. Our command line approach was very non interactive. In other words, all the relevant information that was required for operation had to be given right from the command line before our program started running.

Our program had **three** different modes or user commands which were as follows:

1. The registration mode, which began an interaction of the registered roommates.
2. This was followed by one or more expenses mode, where the application recorded various events for which the amount of money and roommate who paid were recorded.
3. The third mode was the reporting more. This mode was used by one roommate to ask the system what is the money that he or she must either receive or pay.

We have introduced some shortcomings in the program that we had presented during the command line, we hope you have found what they are, if any, and you have improved upon them.

The command line version as presented also illustrated some aspects about how to design a user interaction. One of the restrictions that it gave was the command could be used by one and only one roommate at a time. The graphical version improved a little bit on the command line rigidity; *it provided interactivity*. And more than interactivity, it allowed the user to not worry about the exact syntax, but focus on the information that is to be manipulated.

In this case the only information that really is required by the roommates is the amount of money and events for which expenses were made, and by whom. The command line forced us to remember the exact syntax of the command for each mode we want the program to run in. For example, for recoding an expense, we have to remember that the keyword **exp** has to come after the command but before the amount; it was not possible to switch the order of the two. So, for example, you could not run the fair share command with arguments in which the amount is followed by the keyword **exp**.

You could do that only if you change the architecture of the programming inside the fair share program, but the graphical version allowed us to be free from such kind of issue. Nevertheless,

both the graphical as well as the command line work on a single computer, they could be used by one and only one roommate at a time. At this point, it is possible that some of you may object that well, most operating systems nowadays that we use on our computers are multi user, so all we need to do is to have multiple users on that machine. Sure enough, that is possible. But at any given point of time your machine would be used by one and only one single roommate. If another roommate uses it, then he or she logs in into his or her account on that same machine. But at any given point of time one and only one user is using it. Second, we have not designed our command line program such that it works with multiple users simultaneously.

For example, it uses a single database of file as a single database, and that file is same whether the program is being run by user 1 or user 2. This gives rise to a lot of problems. For example, if both users try to change the file at the same time, how do we handle that? We have not programmed our system for such a thing. So, even for program can run on a multi user system, it is not really strong enough to deal with the complexities of a multi user system; *it is by design a single user program.*

At some point we would like to generalize this to be able to handle multiple users. So, that multiple users can run the same program. Okay, these were some of the things that we saw in the past week, where we developed a command line as well as a graphical version of our fair share application. We now take the next step in this week.

Although graphical version of our application was a bit better than the command line version, both of them worked on a single computer and a single user. To put it in a different language, if our fair share program is considered as offering a service to have the roommates, then our computer becomes the access point for that service, and one way to describe the situation is this service has a single access point. So, a consumer of the service, which is any of the roommates, will have all of those consumers will have just one single access point.

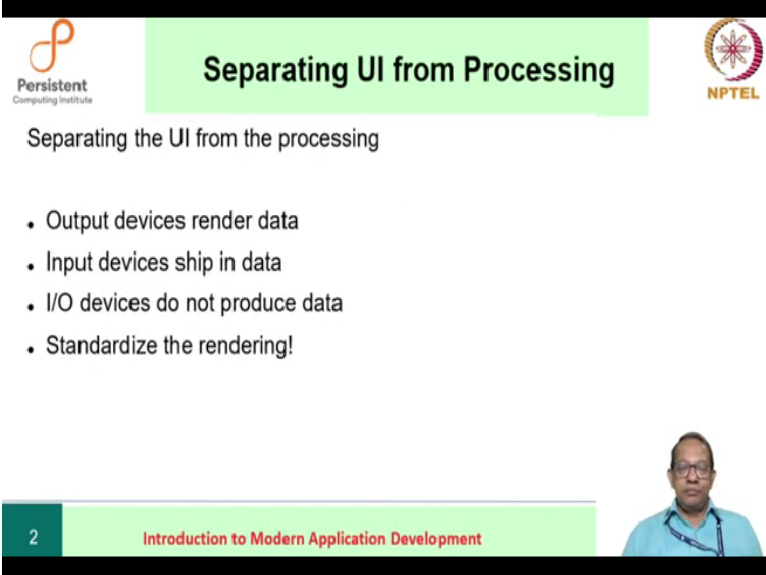
Our next step therefore, is to see how we can change it from a single access point to multiple ones. This is where we will introduce networking. Computer networks allow us to actually increase the number of access points for that service. In our case, we will again proceed step by step. First, we

will only focus on presenting the output the results over the net. In other words, we will be separating the user interface from the processing.

So, the results of the processing are presented out to the user, and it is this part that is going to be separated. On our command line application presents the output on the terminal, but that terminal is on the same place as the machine is. Similarly, for a graphical system, what we would like to do is to take the output user interface from the local machine to a remote machine across the network.

If we can allow the input and output both to be networked, then this will allow us to have multiple access point each connected via the network to the single processing system. So, multiple access points will be possible and all our roommates will be able to access the fair share application from different points, and they will not be required to necessarily use the same computer.

(Refer Slide Time: 11:14)



The slide features a green header with the title "Separating UI from Processing". On the left is the Persistent Computing Institute logo, and on the right is the NPTEL logo. Below the title, the subtitle "Separating the UI from the processing" is displayed. A bulleted list contains four items: "Output devices render data", "Input devices ship in data", "I/O devices do not produce data", and "Standardize the rendering!". In the bottom right corner, there is a small video feed of a man in a blue shirt. The bottom of the slide has a green bar with the number "2" and the text "Introduction to Modern Application Development".

- Output devices render data
- Input devices ship in data
- I/O devices do not produce data
- Standardize the rendering!

We will look at this aspect in this session. The key idea of separating the UI from the processing is in appreciating a few basic points output devices render data, input devices ship in data. Neither of them really produces new data. They do not generate anything. If we could just standardize the rendering, we are focusing on standardizing the rendering because we right now are interested in just generating the output.

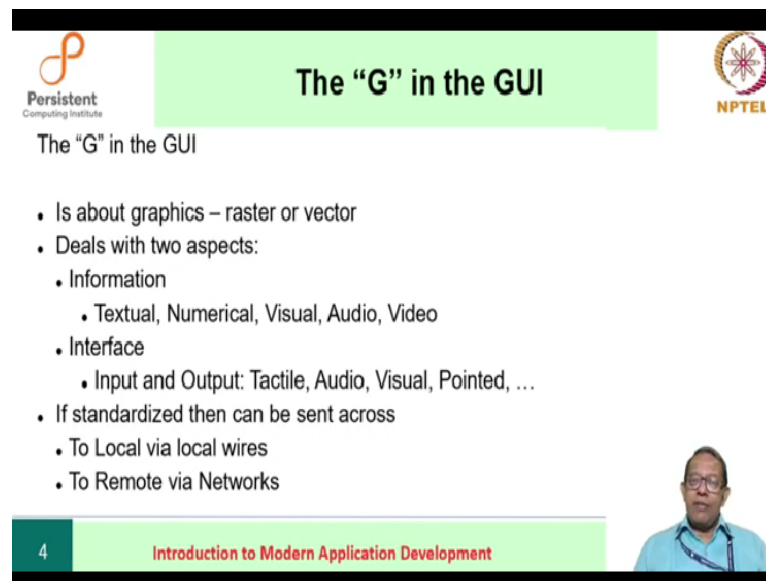
Why do we standardize? the graphical application as well as the completely command line version of our application, both of them presented some very interesting concepts to us. The command line version argued that there are a variety of output devices, and instead of dealing with all of them, let us create an abstraction called as a **standard output device**. We wrote our programs to just print on the standard output device, and we left it to the operating system to decide what that device will be.

In contrast, a graphical application could not do that, and this will get us stuck. Depending on the particular graphics system that we have used to program, or the code that we wrote to present output, would require us to present output in different format. It won't be the case that we can simply output the data to STDOUT like we did in command line version.

This problem is very similar to the scenario of having a variety of output devices. And just like on the command line application, we created a standard output device, we need to do something similar for graphics. And that is what *standardizing the rendering means*.

How do we standardize? Well, we use HTML and CSS. In fact, HTML and CSS are the universal user interface. That is what we would like to really draw your attention to.

(Refer Slide Time: 13:55)



Persistent Computing Institute

The "G" in the GUI

The "G" in the GUI

- Is about graphics – raster or vector
- Deals with two aspects:
 - Information
 - Textual, Numerical, Visual, Audio, Video
 - Interface
 - Input and Output: Tactile, Audio, Visual, Pointed, ...
- If standardized then can be sent across
 - To Local via local wires
 - To Remote via Networks

4 Introduction to Modern Application Development

Before we go to HTML and CSS combinations let us take a quick look at what the graphics – what do we mean by graphics. At its very core, as we might have learned it in computer graphics course, graphics is either raster or vector graphics. And it deals with two main aspects: information and interface. Information is of various kind, it could be:

- Textual
- Numerical
- Audio
- Visual (like video, images...)

On the interface side, we have a variety of input-output mechanisms ranging from touch tactile to pointing devices (typically a mouse), audio output (speaker), video output (monitor)... so on and so forth. Recall that on a command line application our output hardware could be varied, and we standardized the output by creating the standard output device, which allowed our programs to be written very simply; they simply print on the standard output device and rest is taken care of by the us. Similarly, the same way if we standardize the graphics system while developing GUI applications, then we could take the output display from local machine to the local machine's output device or from local machine across the network to a remote machine. This possibility opens up. Not only there are multiple ways of presenting the output, there are multiple output devices we can present the output to.

(Refer Slide Time: 15:51)

HTML + CSS

- HTML – Hyper Text Markup Language
- CSS – Cascading Style Sheets
- DOM – Document Object Model

HTML

The "What" to display:
Information marked with
Logical structure.

CSS **DOM**

6 Introduction to Modern Application Development

This is what gives us our standard web tools, the HTML, the CSS and what we call as the document object model DOM. There are times when we feel that it is better to view the web as a system of 3 entities, the HTML, the CSS, and the DOM. The HTML is about what to display, the content. HTML as it stands for hypertext markup language, what it means is that the content to be displayed is marked for the kind of presentation it should be.

It is a logical marking. So, as an author I might say that I want to, in my content, start emphasizing my content from point A to point B. I just say that I want to emphasize it and mark the emphasis from point A to point B. Now it is left up to the rendering system to actually draw. Not only does it draw it first decides how to render. So, it might say that an emphasis is to be done in a maybe with an underline or it might be possible to even imagine that an emphasis will be done by just changing the color.

The separation from content and presentation is one of the key factors that makes the web so interesting.

(Refer Slide Time: 17:55)

The slide features a green header with the title "HTML + CSS". On the left is the Persistent Computing Institute logo, and on the right is the NPTEL logo. Below the header, a bulleted list defines the terms: HTML – Hyper Text Markup Language, CSS – Cascading Style Sheets, and DOM – Document Object Model. To the right of the list is a blue triangle with "HTML" at the top vertex, "CSS" at the bottom-left vertex, and "DOM" at the bottom-right vertex. Each of the three sides of the triangle is labeled with a white "W". A yellow callout box points to the DOM vertex and contains the text: "DOM The Logical Structure of the 'How' to display; and their parameters." At the bottom left, a green bar contains the number "7" and the text "Introduction to Modern Application Development". A small video inset of a man in a blue shirt is visible in the bottom right corner.

Apart from the HTML, our next entity in the web is the DOM, the document object model. The document object model specifies the logical structure of the components of HTML, as well as CSS, in fact the document. So, it says that a document is made up of a head, a body; a body would be made up of a header followed by a paragraph, maybe one or more paragraphs. It could have a sub header, which could again have a paragraph.

It is this tree like structure starting from the document: the body, the first header, the paragraph of the header and so on, so forth. This tree structure, is what the document object model really captures. This framework that the DOM offers is specified, how to use it, is specified by HTML and how to display it, how to present it is specified by cascading style sheets.

(Refer Slide Time: 19:02)

HTML + CSS

- HTML – Hyper Text Markup Language
- CSS – Cascading Style Sheets
- DOM – Document Object Model

CSS

The "How" to display using values for parameters.

HTML

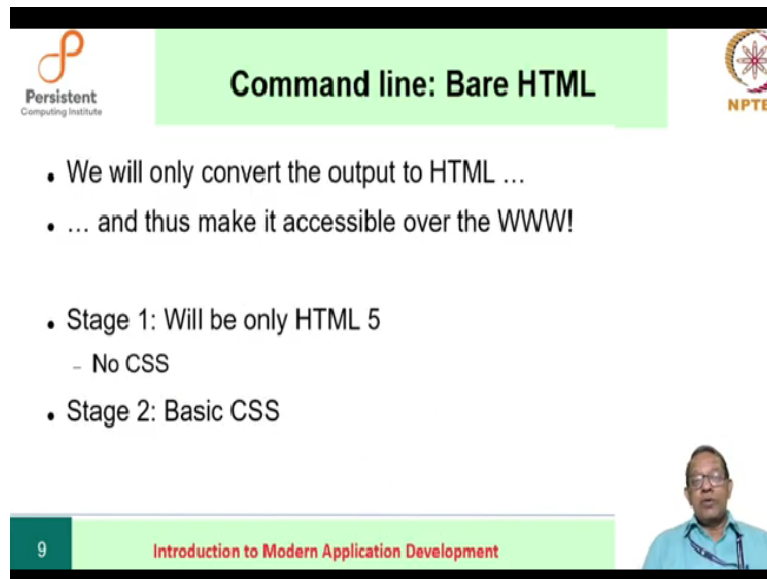
CSS **DOM**

8 Introduction to Modern Application Development

At their simplest cascading style sheets simply specify how a certain markup element of HTML is to be rendered on to the output system. Note that this allows us to specify or adapt to different output systems by simply changing the style sheet. So, for example, if our device were a video device, then we could mark up by using visual cues. So, for example, we could mark up by changing the color used for emphasis.

On the other hand, if our output is a simple sound speaker, audio channel, then an emphasis could be in terms of changing the volume by simply making it louder, or more stressed while speaking. So, the actual presentation details for a given device are handled by the CSS. So, the CSS really deals with how to display using the content that HTML supplies. We will gradually build up more as we go along. But we suggest that you really look at the main source of information about these standards, HTML, DOM and CSS on the world wide web. You will find this information at www.w3.org.

(Refer Slide Time: 21:02)



The slide features a green header with the title "Command line: Bare HTML". On the left is the Persistent Computing Institute logo, and on the right is the NPTEL logo. The main content area contains a bulleted list:

- We will only convert the output to HTML ...
- ... and thus make it accessible over the WWW!

- Stage 1: Will be only HTML 5
 - No CSS
- Stage 2: Basic CSS

At the bottom left, a green bar displays the number "9" and the text "Introduction to Modern Application Development". On the bottom right, there is a small video inset showing a man in a blue shirt speaking.

Let us now go ahead and look at the actual application. We are first going to create, from our purely command line application, a bare HTML version of our application. By bare HTML, I mean, our application will only output completely only in terms of HTML. Having done that, we will then go to the next step where we will add styling information via style sheets.

We are going to use style sheets in a very simple manner to illustrate how the changes, how the styling change occurs just by using style sheets. So again, we will be going by first changing from a pure command line application to a bare HTML application. And then we will add, as a next step, style sheets. The final comment here is I would want you to note that our fair share command line application is a procedural style of using Java.

And we will be changing that same piece of code to create an HTML or bare HTML, whatever case it may be. We are going to change the same command line application. You might want to actually change the object-oriented version of the application such that it generates HTML and HTML with style sheets.

This could be a good point to take a pause.

(Video Starts: 23:13)

Now we will open the command prompt. We already have written the Java programs that we need. And here is the FairShare application. We have exactly the same program that we had seen last time, when we actually did in detail the command line version of this program. As we can see that the program will be started from 'main', and then went on to do setup and process command lines, arguments and so on, so forth.

Let us compile this program and create a class file. The Java compiler 'javac.exe' that we have used has been installed at the point that we have shown on our screen. Please use your path name for your Java compiler, this changes because it depends on where you have installed your Java JDK system. So, wherever you have installed your Java JDK system within that folder, go into the bin folder and then you will see your Java compiler.

So, here we go. Voila, it has compiled successfully! Let us now execute this. Again, the exact path will vary for each one of you, because it depends on where you would have installed your Java JVM. The way we use the fair share application is by using any one of the 3 user commands but quite naturally we will start by registering our friends f1, f2, f3 and f4. Let us just say 4 of them are there. Voila, it has registered 4 friends!

Let us have some sample expenses. Let us say if we spend some 200 rupees. We write the proper commands in the prompt to add the expenses.

So, we add following events:

- f3 spent 200 rupees for the entire set of roommates is recorded by our application
- f1 spends 100
- f4 spends 250

So, at this point we had 3 events and paid for by the roommates that we just saw. The roommate f3, f1 and f4 have paid something for some event. f2 hasn't paid for any event.

So, when we ask for the reports for various roommates, we get following:

- We see that f2 is required to pay 137 rupees and 50 paisa to others.

- f1 is required to pay 37 rupees 50 paisa to others.
- f3 is to receive 62 rupees 55 paisa from others
- f4 is to receive 112 rupees 50 paisa from others.

I do not know whether this particular in this particular case the invariant is satisfied or not. I would like you to be sure that whatever is the amounts calculated, they are correct; we leave it to you to verify if they are correct or not.

In the meanwhile, let us proceed with the bare HTML program. So, the fair share, bare HTML Java, is the program that we are now going to compile and run. After we have compiled it successfully, let us of course run it now. Let us start afresh by registering the friends again. When we run the new program, unlike the command line version of this program, our command gave out a lot of output, it is actually HTML. And as the name stands, it is a markup language.

So, for example, `<h2>` marks the beginning of the header level 2, and `</h2>` marks the end of that header. So, whatever is the information in between is marked by HTML to be header level 2. Now when this HTML reaches the browser, the browser will then decide how to render header level 2. In other words, it could render it by say for example, making it a bold font, or underlining it, or maybe changing the color, whatever we the user of the browser desires.

Our program generated this output under standard output. Let us capture this into an HTML file. And let us open this file in a browser. The output of our program is now in HTML form, and the browser that can show it. Note that this was just registration, therefore, it said that it is registering roommates, there are 4 of them. It says that it has found 4 roommates, and that registration is done. It was not asked to do any expenses nor was it asked to do any report.

As far as the database information is concerned, it says yes, there is a file there of that kind but it is a fresh start and so there is no data in it. Let us go back to our command line and add some data which means the data about expenses. So, let us say that friend f1 spent some 250 rupees. It again gave HTML output and what we should have done is redirected that output to an HTML file.

Let us for clarity sake, keep the 2 HTML files separate. So, the file that was generated by our program, when it did registrations, let us keep that separate from the file that it generates whenever we do expenses. We will also keep the third file whenever we do reports. Let us take up a new tab and open up the file when we did expenses in this new tab. Compare it to things over here it says “roommate registrations”. Now it says “registered roommates are”, followed by whatever they are.

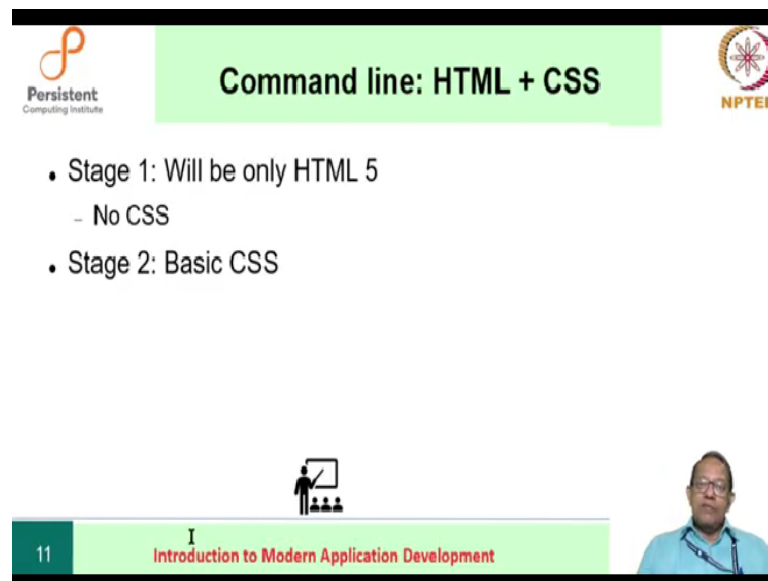
And now there is an expenses information which were added as events. At this position, it was written “no report requested” in the previous version. And if we scroll down, we see now that there is a database. It has found 2 events. There are 4 roommates, and here are the events.

Note that because we ran the program twice, it shows two events by f1. Okay, you have 3 events, number of entries in the database is 3, over here f3 has spent 450 rupees. This event was added and other display is the same as before. At this point let us ask for reports and, as I said before, collect the results in a separate file. We have been leaving f2, so let us ask a report for f2. Let us look for the latest file that we saved and then open it.

As before it informs us about the registered roommates. It says that no new expenses were recorded. The amount f2 owes is 237 rupees 50 paisa. If we go down on the database, the database has no extra events than before. So, the number of events is 3. And a number of entries in the database is still 3.

Our friend f2, since he or she has not contributed so far is now supposed to pay the entire contribution. So, this tells us, this shows us how the bare HTML file program works. Let us now go ahead and do the same with the final version.

(Refer Slide Time: 39:35)



The slide features a green header with the title "Command line: HTML + CSS". On the left is the Persistent Computing Institute logo, and on the right is the NPTEL logo. Below the title, a bulleted list outlines the stages of the project:

- Stage 1: Will be only HTML 5
 - No CSS
- Stage 2: Basic CSS

At the bottom of the slide, there is a video inset of a man speaking, a small icon of a person at a whiteboard, and a footer bar containing the slide number "11" and the text "I Introduction to Modern Application Development".

This is a point where you can stop, review the material, and then come back.

Let us now go ahead with the CSS version. In this we are going to add CSS code to our HTML part. As before, let us simply examine it. Okay, we have our HTML and CSS version over here. Let us as before compile it using our Java compiler, which is typically in some part of your installation.

We will need to register roommates; this again gives output. And as usual, we should really capture the output in some file. We will preserve the earlier files for our own reference.

And so, we will take it in instead of a b c, we will say x y z, that is x.html y.html and z.html. So x.html will contain the registration version. Let us go ahead with some expenses as before, we will simply be adding some sample expenses.

And, again as before, let us take a report for a f3. We had ignored f3, so let us have a report for f3. What I have done during this time is I have first collected all the HTML output for the entire program, and then now we will open it in a browser and see how it looks. First of all, I should open x.html because that represents the registration, but this time with CSS. Let us see how it looks. After pasting the CSS file in the same directory as the HTML file, if we reload, we should get the

styled version of our previously bare HTML. Thus, it is the same bare HTML file, but this time it has been touched by CSS and the presentation has been changed.

The registration part is highlighted in blue. All the details are separated out: expenses, reports, database information, and the fact that it is a fresh start, that is a new database. So, let us look at the expenses record which was in y.html. There you go. But notice the difference. The roommate part is no longer in blue, and the registration – expenses record is in blue. But the record – report part is not in blue.

In other words, the blue band is used to draw attention of the end user to whatever user command the fair share application is responding to. Let us check that by looking at the final report version of the same command. We are going to use the fair share application, we had to use the fair share application to generate report for one of the roommates. We should now see the report area in a blue band, and there you are.

So, we have used style sheets in a very different way, what was purely just a piece of information that was given out by HTML has now, just by using the styling system, provides us a better presentation and helps us focus on what exactly is being responded to.

If our application program, FairShare, is responding to a registration request, then that part is given a blue background. If our program is responding to expense request, then the expenses block is given a blue background. If our program is responding to a report request, then that is given a blue background. At every point, the actual database information is also displayed out.

If you observe, this database information looks similar to the way the spreadsheet looks. But there are some things that are remaining, for example, the vertical lines here. But that separate the columns of the table and the horizontal lines that separate the rows of the table have not been shown.

One of the practice exercises that we suggest you do is to try to write CSS such that these columns and rows, the lines, vertical separators and horizontal separators are also seen. That would be a good way to really get going on learning CSS and HTML.

(Video Ends: 47:07)

With that we come to the end of this session. We have seen just how an HTML + CSS version will sort of look. In the next session, we will actually take the code and see how it differs from the simple command line application.

When we see the differences, we hope that you can see how the command line code was changed incrementally to obtain the HTML version, or HTML + CSS version of the code. The incremental step is simple, not difficult, but still it is tedious.

See you in the next session.