Programming Methodology-Lecture12

Instructor (Mehran Sahami): So welcome back to yet another fun filled, exciting day of CS106A. A couple quick announcements before we start. There's actually no handouts for today, and you're like if there's no handouts for today, why are there two handouts in the back? If you already picked up the two handouts from last class, you might want to double check to make sure you don't already have them, but if you don't have them, feel free to pick them up. We just don't want to cut down any more trees than we need to, so if you accidentally picked them up, you can put them back at the end of the class, pass them to a friend, whatever you'd like to do.

As per sort of the general consensus last time, everyone wanted to have class slides on the web, so now all the class slides are posted on the web and after every class when we have some slides in class, they'll be posted on the web as well. There's a little link called, strangely enough, class slides that is now on sort of the left hand navigation bar on the 106A website, and so you can get all the class slides form there. In some cases, what I actually did was if we covered some slides over multiple days but they were all about the same topic, I might have aggregated them all into one set of slides or one deck, so you'll see them in there as just one topical kind of thing.

Just wondering – how many people have started break out? Good. How many people are done with break out? You folks get to see good times. You're well ahead of the curve. I won't ask how many people have not started. I assume it's just the compliment of that. It is a fairly beefy assignment. It would be in your best interest to start soon. It's due on Wednesday. With that said, it's time to launch into an entirely new topic, and the entirely new topic is something that we refer to as enumeration. Enumeration is a pretty basic idea, and it comes from the word enumerate, as you can kind of imagine.

When you enumerate something, you basically just have some way of referring to something through numbers. So if we wanted to enumerate, for example, the year that someone is in college, we might have freshman and sophomores and juniors and seniors – that's the enumeration of the year you might be. And so the basic idea is we just want to have some set of things that we enumerate or give a set of numbers to, essentially, and you don't necessarily want to think of it as having to be a set of numbers, but it's basically just some set of items that go together.

We generally give them some numbers or some listing as a way of keeping them all – a way we might keep track of them. One way we might do this in java, for example, is just have a series of constants that are integers and so just to save myself a little bit of time in writing, constants. Yeah, a beautiful thing. So we might have some constant public static final [inaudible], and so if we're going to do enumeration, oftentimes we just use integers to refer to each of the individual items and we just count them up. So frosh would be one, sophomores two, juniors three, seniors four, and grad is five.

That's just what year you might be in school. Oftentimes, computer scientists actually start counting from zero, but sometimes it actually makes sense to have these things be

numbers that start at one. For example, if you want to know which year someone is in school or if you're doing months of the year, January is generally number one as opposed to number zero, so just to keep with common [inaudible], we might actually number it this way.

Now, there's something that was introduced in one of the later versions of java, java 5.0, which is kind of the second to latest version, which is something called enumerated types, and the book talks about them briefly. I'm not going to talk about them here sort of for the same reasons the book doesn't talk about them. The book actually talks about them and then says the advantages versus disadvantages of doing enumerations using this thing called enumerated type versus just listing them out as integers. This way, at least for the time being in sort of the development of java's world, seems to win out.

We're just going to teach you this way. As a matter of fact, in the old school, anything before java 5.0 had to do it this way, so it's just best that you see it this way, because most code these days is written this way and it probably will continue to be until at some point this new enumerated type thing takes off. As you see in the book, it talks about enum type. Don't worry about it. We're just gonna do it this way.

The only problem that comes up with doing something like this, though, is that you want to figure out, well, how do I read in these things and display them? Well, these are all just integers, so if I actually want to ask someone their year in school, I would have to keep track of that with some ints that I might call year, and so I would read in an int, and I might ask the person for their year, for example. And when I read that in, that's all good and well. The only problem is this thing is just an integer. The user gives me some number, hopefully between one and five.

I might want to actually have some logic in here that checks to make sure they gave me a number between one and five, 'cause if they gave me a six, I don't know what that corresponds to. Maybe that's the dreaded other student category. I need to do something to guarantee that it's in this list one through five. The other problem is if I actually want to print out someone's year, there's no way for the computer to know that year one should print out the word frosh, so if I do something like print lin here and I just write out year, that's gonna write out an integer.

It's gonna write out a value, whatever the value the user gave me, presumably between one and five. So if I actually want to have some nicety in there and say, oh, if the year is one, I actually want to write out frosh as opposed to writing out a one, I need to do that manually. So somewhere in my program, I need to have some function that I do a switch statement or I can do cascaded ifs and I switch, for example, on year.

I might say, well, in the case where year happens to be frosh, then what I'm going to do is actually print out print lin frosh in quotes, because that's the only way the computer knows about it, and then I would have a break, the funky syntax from our fun, the switch statement, and then I might have some case over here for sophomore. I need to do this

manually, and that's just the way life is in the city. These things are just integers. We have some enumeration.

So in the program, the program can read a little bit more text because we can refer to frosh, sophomore, junior, senior and grads as constants in the program, but when it comes to displaying them on the screen, we need to actually write it out manually because the compute has no other way of knowing that a one means frosh.

Well, case one is the same thing as case frosh, 'cause if these are in the same program, frosh is just the constant one, and so in fact that's why we want to refer to it that way because it's more clear for someone reading it. They see oh, what do you do in the case where you're a frosh? Well, I'm gonna write out frosh. It's fairly straightforward enough. But I'm just using these constants. If someone else wants to come along and say you know what? I love frosh and they're all great, but I'm a computer scientist. I start counting from zero.

It'll just change everywhere in your program as long as you've used the constants if you're referring to zero. They might just say, well, actually frosh are the coolest. They're a six. That's fine. You can do whatever you want. The computer really doesn't care. Most people probably won't care, either, so we just start counting from zero or one most of the time. Any questions about the general idea of enumeration?

Well, that's the thing. In your program, you want to set up the expectation that they're entering a number. If they were to enter the string frosh, because read in does error checking, it's going to say that's not the right format. So one thing you could actually do is rather than reading an int, you can read in a line, which would read in a string, and then you'd need to have some logic to convert that over to one. So you'd sort of do this process but backwards. That's why enumerations are something that are useful to have when you're writing real programs, but they can get a little bit bulky to use because you have to do the conversions.

Right now, I'm just making the constants public because I might want to refer to them in some other class. If I have some other class, they can also refer to these constants. If I was only going to refer to these constants within this class, I'd make them private. With that said, it's time for something completely different. I know the sun isn't out much right now, but I sort of had this thrill where I wanted to barbecue one last time, and I figure if I can't barbecue outside, I'm going to barbecue inside. Now I wanted to light a fire in here, but as you can imagine for various reasons, that was frowned upon by the university.

So I can't actually light the fire, but I can basically do everything else that would be involved with grilling, which doesn't really turn out to be that exciting when you don't have a fire. But the basic idea is if I leave some things on the grill for a while, they'll get a little hot. Just pretend they were on the fire for a while, okay? If you leave something on the grill too long, what happens to it? It gets kind of burned. It accumulates something. It accumulates something we refer to as char. It turns out as a computer

scientist, you get a different kind of char than other people get on their burgers when they're grilling them, or, as happens to be the case, on their ding dongs.

Turns out they don't make ding dongs. How many people know what a ding dong is? Good times. If you don't, come and look at the box. It turns out they don't actually make ding dongs. I'm sure this is copyright Hostess 2007. I went so many places last night trying to find ding dongs, but you can eventually find them. The basic idea, though, is we want to think about something that we want to refer to in a program that isn't always just a number. So far, we've been dealing with a whole bunch of numbers. We had doubles. We had integers. Life was good.

I told you about this thing called string, but we didn't really explore it, and now it's time to explore it a little bit further to figure out what these little puppies, this little char or little characters are all about. So what we're gonna do is we're gonna explore the world of char. There's a type called CHAR, which actually is a character, and so the way we refer to this type, even though I just called it char, is we don't call it a char. We call it either a char like the first syllable in character. Some people call it a car, even though it's not called a character because they just look at it and they're like oh, if it was one syllable, it would just be car.

And some people look at it and say no, Mehran, if it was just one syllable, it would be char. Don't call it char. I will beat you. That was a joke, by the way. I'll be trying to explain that to the judge when I'm in jail and he's like, well, the video I saw, it didn't appear to be a joke. CHAR is just a character. We can declare variables of this type by just saying CHAR and I'll call it CH. I have these little CHAR CH, and what do I assign to these puppies? What I assign to them is something that's inside single quotes. That's the key.

I can say something like CH equals little a, and I put the a inside single quotes. Not to be confused with double quotes. If I put it in double quotes, that's a string. That's not the same thing as a CHAR. So I put it inside single quotes. That indicates that it's a character. The other thing that differentiates a character from a string is that a character always is just a single character. You can't put more than one character there. It can't be like CH equals AB. That's not allowed. It's just a single character. The funky thing, and you might say, so Mehran, why are you telling me about this thing called characters right after you talk about enumeration?

Because it turns out in the computer, the computer really doesn't know about characters in some deep way that you and I know about characters. All characters inside the computer are actually just numbers. At the end of the day, we can refer to them as these characters in their niceties for us, but to the computer, they're actually an enumeration. A happens to be some number and B happens to be some number and C happens to be some number, and so now it's time for you to know what those numbers are. So if we just look at the slides real briefly, there's this thing called ASCII.

Anyone know what ASCII stands for? It's American standard code for information interchange, and we'll just do it as a social. The basic idea for this is that somewhere, some time ago, someone came along and said we're gonna create a standard enumeration of all the characters that exist, and so here's the first 127, as it turns out, character enumeration, which is the part that's mostly been standardized. All the rest of the stuff, there's still some debate about, but this one, everyone's standardized on. Now this little diagram that's up here is actually in octal numbers, and you're like octal numbers? It's base eight.

Computer sciences think that way. We think in base two. We think in base eight and we think in base 16. Most people think in base ten. Yeah, that's why most people aren't computer scientists. Here is base eight. I had this roommate at Stanford who thought everyone should count in base 12, because base 12 was divisible not only by two but by also three and four, and ten wasn't, and he would just try to convince people of this all the time. I was like that is so wrong. Now he works for the Senate, which I wouldn't be surprised if we have some resolution someday. The United States will now count in base 12.

But anyway, the basic idea here, as you can see in this, is that first of all, the character A is not the number one, and there's actually a distinction between the uppercase A and the lowercase a. Another thing that's also kind of funky is you might notice the numbers up here, like zero, one, two, three – the digits. The number zero is not the same thing as the character zero. The character zero just has some enumeration that's some funky value. Who knows what that is. It would actually be 60 in octal notation, which turns out to be 48.

That's not the same thing as the number zero, so it's important to distinguish that the number zero that we think of as an integer is not the same thing as the character zero which we would put in single quotes, and that's true for all digits. You don't need to care about what the actual numbers are. If you really do care about memorizing it, the way you can read the chart is the row on the chart is like the first two digits, and then the third digit is given by the column.

So A is actually 101 in octal notation, if you really care about reading this, which makes it the value 65, but we don't really care. We just think of it as A inside single quotes. There's a couple things that that gives you, though, that are useful, and the couple things that it gives you that are actually useful is it guarantees that the letters little a through little z are sequential. It guarantees that uppercase A through uppercase Z are also sequential, and it guarantees that the digit zero through the digits nine are also sequential. That's all you really need to care about.

Other than the fact that these guarantees are given to you, you don't care about what the actual numbers are up there. There's a couple other things that look a little weird up here that I'll just kind of point out. There are some characters that are like these backslash characters that are \c and \n. The backslashes are just special characters that are treated by the computer as a single character, so \n is new line. It's like a return. \c is the tab

character, and if you're really interested in all these things, you can look them all up in the book, but those are the only ones you need to know.

You might wonder hey, Mehran, how do I get a single quote, because if I'm trying to put characters inside quotes – if I want little ch over here to be a single quote, do I write it like three quotes in a row? No. That will confuse the computer to no end. The way you actually do it is you put in a backslash and then a single quote and then another quote, and this is what's referred to as an escape character. When it sees the backslash, it says treat the next character as a little character and not as any special symbol that, for example, closes the definition of a character or something.

So this whole thing is just the single character apostrophe or single quote, and it's inside single quotes, so that's how it knows that this is supposed to be the character single quote as opposed to the closing of the character. A couple other things that are sort of funky. What happens if you want to actually do a backslash? Backslash is actually just backslash backslash. If we put that inside quotes, that would be the single character backslash. There's a couple others that are worthwhile. Double quote – same thing. We would put a backslash and then a double quote if we wanted to have the single double quote character.

Not a huge deal, but you should just know that if you want to put apostrophes inside some text that you're writing or something like that. How do we actually get these characters? Rather than getting single characters, so before we talked about over here our friend, read int, which reads in a single integer, you might say hey, do we have a read char? Can I read a single character at a time? Not really. What I end up doing is I read a whole line at a time from the user and then I can break up that line.

I'm going to have some string, and that string S, I would read a line from the user, and that's going to be a whole bunch of characters that are stored inside that string S, and I can pull out individual characters using something called char at. So I can say CH equals S dot, and so the string class or the string objects have a method called char at, and you give it a number and it gives you the character at that position. So I could say char at and as computer scientists, we always start counting from zero, so I could say char at zero, and that's the very first character in the line the user actually entered.

I can do a print lin on that character directly, and it'll just write out that first character. The way to think about this – let's say it read a line, S, and the user gave me a line and they typed in hello. Then they hit enter. Hello gets broken up into a series of characters where this is character zero, one, two, three, four, and that return that the user types is thrown away. It's cleaned up for me so it gets rid of it automatically. If the length of the string that the user typed in is five, it's actually indexed from character zero to four. So char at zero would give me the H character out of here. That's a critical thing to remember.

We start counting from zero as computer scientists. Any questions about that? We have our friend over here. Let's see if I can actually get this all the way out of the way. We have our friend over here that tells us, hey, the letters are guaranteed to be sequential in

the lowercase alphabet, in the uppercase alphabet and the characters of the digits zero through nine, so how can I use that? It turns out you can actually do math on characters, and you're like oh, Mehran, the whole point of having characters was that I wouldn't have to do math on them. Well, you can actually do math on characters. It's kind of fun.

Let's say we want to convert a character to a lowercase character. We might have some method. It's going to return a character, which is a lowercase version. I'll call it two lower of whatever character is passed into it. So it's passed in some character CH, and what it's going to return is the lowercase version of that character. So the first thing I need to check is hey, did you give me an uppercase character? If you give me an exclamation point, what's the lowercase version of an exclamation point? A period. No. We can't lowercase punctuation. It just doesn't work.

It's like what's the lowercase version of a period? A comma. What's the lowercase version of a comma? A space. What's the lowercase version of a space? Yeah, somewhere, it stops, and it stops here. How do we check that this thing is uppercase? Well, these are really numbers, and we're guaranteed that they're sequential. So we can treat them just like numbers. We can do operations on them like they were integers. We can say if CH is greater than or equal to uppercase A and CH is less than or equal to uppercase Z, if it falls into that case, then we know that it's an uppercase character 'cause they're guaranteed to be sequential.

If it doesn't fall into that range, then we know that it's not an uppercase character because it's outside of the sequential range of uppercase characters. So what do we do? We're going to return something, which is a lowercase version of that character if it's an uppercase character. How do we convert it to lowercase? Anyone want to venture a guess? We could. How do we know how much to add? It's on the chart, but we don't want to use the chart, because we don't want to have to remember what's in the chart. Yes. For that, A. It's the difference between the uppercase and the lowercase character.

So think about it this way. First thing we want to do is figure out – I'm going to explain this a slightly different way, which is first we want to figure out which character of the uppercase alphabet you typed. So we take the CH you gave us and subtract from an uppercase A. If we do that, if CH was uppercase A, we get zero, which is the first letter of the alphabet. If it's B, we get one. If it's C, we get two. This will give us in some sense the number of the letter in the uppercase alphabet. Once we get that number, we need to figure out what the corresponding letter is in the lowercase alphabet.

So translate according to that chart into the lowercase alphabet. I don't want to memorize that chart. How do I know the starting position of the lowercase alphabet? It's lowercase a, so I just add lowercase a, which is the same thing, basically, as taking the difference between the lowercase alphabet and the uppercase alphabet. But if I do that, basically this portion tells me figure out which letter, in terms of the index of that letter, and then offset it by the appropriate amount to get the corresponding letter in the lowercase alphabet, and that's what I return.

Otherwise, what happens if I'm not in the uppercase alphabet? I just say hey, I'm going to give you back – you wanted lowercase version of exclamation point. Not happening. You get exclamation point back. I have to still give back a character. I can't say I'm gonna give you back this big giant goose egg, and it's like sorry, thanks for playing. I can't do that because goose egg is not a character. I just return CH unchanged. We can do a little bit of math.

It doesn't matter. I just get the offset, but I'll still give you a B for that. The other thing we also want to think about is we can not only do this kind of math on characters, we can even count through characters. You remember in the days of Yore when you learned your little alphabet, like the little alphabet song? I thought for five years of my life L M N O P was one letter. Totally screwed me up. That's just the American educational system. If we have some character like CH, I can actually have a four loop counting through the characters.

I can say start at uppercase A as long as the character is less than or equal to uppercase Z CH++. I treat it just like an integer, and now what I have is a four loop that's index is a letter that counts from uppercase A through uppercase Z. I can do this lowercase. I can do it with the digits from zero to nine, but I can treat these things basically just the same way I treated integers, because underneath the hood, it's an enumeration. They really are integers. Other things to keep in mind is characters are a primitive type. This type CHAR is like an integer or like a double. It's referred to as a primitive type. It's not a class.

You don't get objects of type CHAR. You actually get these low level things called [inaudible] the same way you got integers, which means when you pass characters, you get a copy of the character. It also means that a character variable like CH is not an object, so it doesn't receive messages. It can't get messages. But it turns out there's a bunch of operations we'd actually like to be able to do on characters, and so java gives us this funky class called character, and character actually has a bunch of static methods. They're methods that you can apply to characters but you don't call them in the traditional sense of sending a message.

This is how they work. If I have CHAR CH, I can say CH equals – I give it the name of the class instead of the name of the object. So I say character got and there is, for example, something called two uppercase that gets passed in some character and returns to me the uppercase version of that character. Just like we wrote two lower here, you can imagine you could do a very similar kind of thing with slightly different math to create an uppercase version. It does that for you. This method is part of a class called character, but it is what we refer to as a static method. It's not a method that every object in that class has.

It's a method that just the class has, so the way we refer to it is we give it the class name and then the method name, because this CHAR thing is not an object. It turns out there actually is a class called character that you ca create objects of, but we're not gonna get into that. We're just gonna use these little things called CHARs, which are our friend. There's a bunch of useful methods in this character class, and I'll just go over a few of

them real briefly. Real quickly, you can check to see if a character is a digit, is a letter or is letter or digit.

That's good for validating some inputs if you want the user to type in something in particular. These taken characters are Booleans. Question? We're not gonna worry about letters from different alphabets for now, but in fact, they're all supported. The numbers just get bigger from what they could be. Though I only showed you the first 127 letters, it turns out that the standard that java uses actually supports over one million characters, and so you can have all kinds of stuff like Chinese and Arabic and all that. In terms of other things you can do with characters, you can check to see if a character is lowercase or uppercase, and all these at this point are trivial.

I know how to write these myself. In fact, you do, and you could, but they're so easy to write that they're just kind of given to you for free as well. A couple others like [inaudible] white space. That was actually convenient. It checks to see if a character is either a tab or a space or a new line. Question? It returns a Boolean. It just says the thing that you gave me is a letter or a digit. For example, here is the digit 2. So it would return true for that, except it might hit someone else along the way. Yeah, if it's either a letter or a digit. It doesn't let you know which one. It's just if it was a letter or a digit.

It's not punctuation is kind of the idea. And then finally, two lowercase and two uppercase, you actually just wrote two lower yourself, but you also get those. These are all in the book, so you don't need to worry about copying them down, and the slides will be posted on the website. So characters are kind of fun because we can do math on characters, and it's kind of like oh, that's sort of interesting, but it gets more fun when you can put a whole sequence of characters together into our friend, the string. It's time to bring the string back out.

Time to polish off the string. I actually wanted to see if I can do this just for laughs. I want to see how far it will go. All right. Strings – I'm gonna try that again by the end of class. Our friend, the string class. Strings, in fact, are a class and there are objects associated with strings, as you sort of saw before. So we could have, for example, a string that we declare called STR, and we might read that in from the user using our friend read line that you just saw an example of previously, and we pulled out individual characters.

Here, we're going to read a whole line. It turns out there's a bunch of things that we would like to be able to do on strings. The key concept with strings is a string is what we refer to as immutable, which means that a string cannot be changed in place. If the user types in hello and I say, yeah, hello is kind of fun, but I really like Jell-O, and so I want to get rid of the H and replace it by a J, I cannot do that in a string. So if you worked with other languages where you can directly change the context of the string, not allowed in java.

They are immutable, which means if I want to go from hello to Jell-O, what I need to do is somehow create a new string that contains Jell-O. I might take some portion of this string and add some new character to it, and I'll show you some examples of how we

might do that, but the key concept is strings are immutable. They cannot be changed in place. When we do operations on strings, what we're actually gonna do is create new strings that are modifications of the previous strings we had, but we're still creating new strings.

I'll show you some examples of methods for strings in just a second, but I want to contrast between strings and characters just real briefly before we jettison our friend the character and deal all with strings. So CHAR as we talked about is a primitive type. It's not a class versus string is a class, so we have objects of the string type. If I were to have CHAR CH and I were to have string STR and I want to do something like converting to uppercase, for CH I have to call character dot two uppercase CH. I don't actually pass this message to CH, 'cause CH is not a class. It's a primitive type.

I need to call this funky class and say hey, class character, let me use your two uppercase method to make this character. In string, there actually is a string operation two uppercase, and the way that works is I could say string equals STR, so the receiver of the message is actually an object. It's this string thing. I'm not writing out the whole word string. I'm saying STR dot two uppercase, and I pass it in STR. Now here, things might look a little bit funky.

The first thing that looks funky is you say hey, Mehran, if you're telling the string to convert itself to uppercase, why do you need to assign it back to itself? Why can't you just say hey, string, convert yourself to uppercase? Why wouldn't that make sense in java's model? 'Cause strings are immutable. That's just beautiful. The basic idea is strings are immutable, so I can't tell a string to change itself to an uppercase version. I can say hey, string, create an uppercase version of yourself. You haven't changed yourself and give me that uppercase version. So it says oh, okay. I say string, create an uppercase version of yourself.

It says here you go. Here's the uppercase version, and it's all excited. It's like oh, here's an uppercase version of me. It's gonna be like me and my uppercase version, and what do you do? You just say yeah, it's not you anymore. I'm just slamming it over you with your uppercase version. So I've replaced the old string with it's uppercase version, but for a brief, gleaming moment in time, I had this thing was a separate string until I signed it over here. I wasn't actually overwriting or changing STR. If I assigned it to some other string like string two, I would actually have the original, unchanged from the string two, which would be a different kind of thing.

A bunch of things you can also do on strings – [inaudible] you've actually seen before. I'll show you one more example of [inaudible], but you've been doing it this whole time. I have string S1, which will be CS106. I have string two. It's okay for a string to be a single character A. It is distinguished from the character A by having double quotes around it. So a character always one character. A string can be one or more characters. As a matter of fact, it can be zero characters. It can have double quote, double quote, which is the empty string.

So I can create a new string, string S3 by just [inaudible] using the plus operation. So I can say I got an plus string two plus N plus string one plus string two. What is that gonna give me? I got an A in CS106 A, and it just [inaudible] them all together. Be happy it's not like CS106 F. I don't know what's going on in CS106 X. We're just dealing with the A here. It's amazing how small the difference is between an A and an F. Just kidding. It's actually a huge, wide gulf. I'm sure all of you will get As. [Inaudible].

Another thing you might want to do with strings is say hey, the user's giving me some particular string like I do a read line over here. Can I actually check to see if that string is equal to something? So one thing I might be inclined to do is say hey, is that string STR equal equal to some other string, like maybe I had some other string up here, S2. And I might say is that equal to S2? Turns out, this looks like a perfectly reasonable thing to say. Bad times. This is not how we check for equality with strings. It's actually valid syntax. You will not get a complier error when you do this.

What this is actually doing is saying are STR and S2 the same object? Not do they contain the same characters but are they the same actual object, which most of the time when you're comparing two strings, they're not the same actual object. They are two different objects that may contain the same characters. So how do we actually test for equality? Well, there is a little method we use that is a method of the string class called equals, and the way we write it is kind of funky. We take one of the strings and we say S1 dot equals and then we give it as a parameter the other string, STR.

So what it says is it basically sends the equals message to string one and says hey, string one? You know what your own contents are. Are they happen to be equal to the contents of this other string I'm passing you as a parameter? And this returns true or false if these two strings contain the same characters. It is case sensitive. There's another version of it that's called equals ignore case that is case insensitive, but this version is case sensitive. The other one you can also see in the book. It's not a big deal. So these are some very useful methods. I'll show you some more useful methods of strings very briefly.

We'll talk about some of these in more detail next time, but I want you to see them briefly right now. The three made its way back up here. The string class has some methods like the length of a string. How many characters does that string contain? So for a particular string like STR, you take STR dot length and it would give you back as an integer the number of characters in the string. CHAR at you already saw. You give it some index starting at zero and it gives you back the character at that particular index. There's a couple things you can do – substring, where you can pull out a portion of the string.

Remember our friend hello? At some point, we want to say oh, just slide this over. Where we have some string that I'll call STR, which may be a set hello. What the substring method actually does is it says give me back a piece of yourself and the piece of yourself is determined by some indices P1 and P2. P1 is the beginning index of the substring you want to get. P2 is the end of the substring you want to get but not counting P2. So it's

exclusive of P2. What does that mean? So if I say string dot substring where I start at position one, that's gonna start at the E, 'cause that's position one.

Then I say go up to three. What it actually gives me back is L as a string, and so I can assign that somewhere else. Maybe I have some other string S. It does not change the string STR, 'cause it's immutable, but what it gives me back is starting at this position up to but not including this position. It's kind of funky. That's just the way it is. There's a version of it that only has one parameter, and if you specify just one parameter, it's [inaudible] start at a particular location. Give me everything to the end because sometimes you want to do that. You just want to say give me everything from this position on to the end of the string.

A couple other things – equals you just saw. This lets you know if two strings are equal to each other. You might say hey, Mehran, I don't want to just check to see if they're equal. Can I do greater than or less than? Well, you can't do greater than or less than using the greater than or less than signs. Those will not work properly. What you do is you use a function called compare to, and the way compare to works is it actually gives you back an integer. That integer is either negative, zero or positive. If it's negative, that means one string is less than the other string lexigraphically.

If it's zero, it means they're equal, and if it's positive, it means one string is greater than the other string in terms of the ordering that you actually – what you send the message to and the parameter that you actually pass. It allows you to essentially check for not only equal to but greater than or less than as well in lexigraphic order. A couple other things very quickly – index of allows you to search a string for a particular character or some other substring, so you tell a string, hey, string, I want the index of the character E in your string, and if I asked hello, the string STR for the index of E, the character E, it would give me back a one.

So if it finds it, it gives you back the index of the first instance that it found it. So if I ask for the L, it'll give me aback a two. It won't give me back the three. Or I can pass in some substring like EL and say hey, where's EL located and it'll say it's located starting at position one. If it doesn't find it, it gives me back a negative one to say it's not found. So just a few things that you should see, and we can kind of put these together in some interesting ways. Let's actually put them together in some interesting ways.

One of the things that's common to do is you want to iterate through a whole string to do something on every character in the string. So you might have some four loop, and it's I equals zero. You're gonna count up to the length of the string. So if our string is called STR, we'd say as long as I is less than STR's length, and then we would add our little friend, the I++ out here at the end.

This is going to go through essentially indexing the string from zero up to the number of characters that it has, and then inside here we might say CHAR CH equals STR dot CHAR at sub I, and what that means is one by one, you're gonna get each character in the string in CH, and then potentially you do something with those characters and then

you're done. So something you also sometimes do along the way is you do some work and you say hey, I want to build up some resulting string. Like, maybe I want to take a string and create a lowercase version of it.

So if I had some string STR and I wanted to build up some lowercase version of it, I would say, well, I need to keep track of what my result is. So I'll start off with some string result that I'll set to be the empty string. So it's two quotes in a row. That's the empty string. It means there's no characters in that string. It's still a valid string, but there's no characters. There's not even a space there. Then for every character that I'm gonna get in this loop, what I'm gonna do is I'm gonna tack on some version of it to result. So I might say result plus equals, which means [inaudible] onto the end of result, essentially what this is really doing.

Yeah, what it's really doing is creating a new string with an extra something added to the end and reassigning that back to result. So I might say result plus equals character dot two uppercase of CH. Let me erase this over here so we don't get anymore confusion. What this is actually gonna do is it goes through this string, character by character, pulls out the individual character, converts it to uppercase and pins it onto the result that I'm building up. So at the end, what the result will be is an uppercased version of the string that I'm originally processing.

That's an extremely common thing to do with strings – not necessarily converting them to uppercase, but pulling out characters one at a time, doing something on each character and building up some result as a result. Any questions about that? So I'm not checking the length of the result. I'm checking the length of string, but the length of result, if I were to actually compute it, is zero. Yeah, I need to put the double quotes here to say that results starts off empty. Otherwise I don't know what result starts as.

Last but not least, two uppercase and two lowercase I mentioned returns to you a new string that is the upper or lowercase version of that string. So basically, we've just written over there the equivalent of two uppercase if we returned whatever the result was. So let me show you one final example before you go, and that's reversing a string. You might say how might we reverse a string? Here's a little program that reverses a string.

It's gonna read in some line from the user and it's going to call a method reverse string that as you can notice over here, reverse string can't change the string in place because it's immutable, so it's gonna return to us a new string that we're gonna pass or we're gonna store in this thing called reverse. So we call – we ask the [inaudible] string. We're going to enter stressed, because I think by week four of the quarter, by the end of it, most people are feeling a little stressed, and so we're going to call reverse string, and reverse string comes over here doing this funkiness that we just talked about.

It starts off with some result, which is the empty string, and it's gonna count through every character in our original string, but it's gonna do it in a way so that by adding the characters one by one from the beginning and pinning them on, it's going to actually append it to the front instead of the back and that way, it's gonna reverse the string. So

that's the difference with this. Here, we're saying take whatever you have in result before and the thing you're gonna add is a character at the beginning instead of at the end, which ends up reversing our order.

So let's just go through this real quickly. I starts off at zero. It's less than length. I get the character at zero, which is gonna be an S, and I add that to result, so nothing exciting going on there. Result was empty before. Now it's just the string S. Add one to I. Still less than length, and what I now get in the character at one is the T. I'm gonna add the T to the beginning, right, so I'm gonna say T plus whatever's in result, so now I have TS in my result. The T doesn't go on at the end. I'm pre-pending it, basically, and then I add one.

I'm gonna get the R now added to the beginning and I keep doing this, adding the characters one by one at the beginning and at the end, I now have the value that's greater than the length. Even though the length is eight, I have the value that's eight, and so it's not less than the length, and so I return desserts, and back over here, stressed still remains stressed. People are still stressed out. Now they're just eating a little dessert along with it. You can actually multiply and divide characters, it's just sort of meaningless if you do. If you do, you sort of get – yeah. Sorry.

I'm sure you won't get an F, but you won't be multiplying characters, either. One final thing – we've still got a few minutes. Remember, we talked about how education is one of those things that if you get less of it, you're happy? If you bought a car and the car had three doors, you'd be really upset if it was supposed to be a four door car because you'd be like I paid $10,000.00 for this car and it's only got three doors. It's missing one of the doors. But if I let you out of here five minutes early, you'd be like rock on! Less education! Same money.

I've never been able to understand – well, actually, I did understand that about 15 years ago when I was sitting there. But now I don't understand it anymore. I want to give you one more quick example, which is computing a palindrome. So what we're gonna do is write a function that computes whether or not something is a palindrome. Do you know what a palindrome is? How many people know what a palindrome is? It's a word or phrase that is the same forward or backwards. Most people know what it is.

Now, I will ask you the more difficult question – who created the world's longest palindrome? No, it was not me. It was, actually, however, my old boss, interestingly enough, a guy named Peter Norvig, who claims to have created the world's largest palindrome on November 23, 2003, and it's a palindrome that is 17,259 words long. Yeah. He did it with a computer. He's actually a wonderful guy, but created the world's longest palindrome, and you could probably create one that's longer using a computer. But how do you actually determine that something is a palindrome?

We're going to just do a little bit more math on strings than we might have otherwise done. Here's a function that computes is something a palindrome when we pass it a string. Now, to figure out if something's a palindrome, it has to be the same going

forward and backwards, which means to do that, one simple way of doing that is to say hey, something like the word racecar is a palindrome. One way I can figure out if it's a palindrome is the first letter equal to the last letter? If it's not, I stop immediately and if it is, then I check the next letter with the next to last letter, and I keep doing this.

Now, the question is how long do I keep doing this? How long do I need to keep doing this for? The length divided by two. I don't need to do it the whole length because eventually, I'll just cross over and I'll just redundantly check the other side. But if I check halfway this way, it means that if I was comparing the characters pair wise, I also checked halfway that way. The other thing that's fun about this is if I happen to have something that has a length three, like the word bib, the middle character is always equal to itself. I don't need to check it.

As a matter of fact, if I check the length three and divided it by two, I get the integer one, which means you really only need to do one check of the first letter and the last letter, and if those match up, don't even bother checking the middle. So it works nicely for words that have an odd number of letters by just optimizing out that middle character. If I actually have four letters like noon, you actually have four divided by two. You need to do two checks, and that will actually check all the characters you care about. So that's what we do here. We have a four loop up to the length divided by two.

It's doing integer division, and what we're gonna do is we say is the character at the beginning or at index I equal to the character at the end? Because we start counting from zero, the character at the end, if it has a length of nine, is actually at index eight, which is why we have this extra – when we do the subtraction, we add an extra one. It means subtract off an extra one 'cause if your length is nine, you really want the eighth character. So basically, as I increases, we check increasing characters from the front and decreasing characters 'cause we're subtracting off I from the back.

As soon as we find the pair that doesn't match, so as soon as these two things are not equal, we say hey, I don't need to check anymore. I returned false in the middle of the function, which means the rest of the function doesn't need to worry about. All the local variables get cleaned up and I return false, because as soon as I find it, I don't even say hey, let me just check the rest just for laughs. Oh, you were just one character away. No, we just put you out of your misery immediately and we return false.

If you manage to make it through this four loop, it means you never hit a case where the two characters you were checking were not equal to each other, because if you did, you would have returned false. So if you never return false, you're good to go. You completed the whole loop, and so you'll return true at the end. Any questions about that? I will see you. Have a good weekend. I'll see you on Monday.

[End of Audio]

Duration: 51 minutes