ProgrammingParadigms-Lecture01

Instructor (Jerry Cain):For 240, now there won't be.

Student:I can sit in [inaudible].

Instructor (Jerry Cain):Oh, that's fine. Hey, everyone, welcome, we are live on television. So this is cs107. My name is Jerry Cain. I have two handouts for you today. They're the type of handouts that you expect to get on the first day of a course. As far as pertinent information about myself; my life synopsis is on the board right here. There's my email address, those are my office hours, and that is my office upstairs, just more or less, directly above this room. By far, the best way to get in touch with me is email via that address. Certainly if you have a question that you need to ask me specifically about then email is great, I'm very good about checking email. I'm the type of person that just clicks on send and receive every three seconds, so I know when your email comes in so I'm always very good about responding. The office hours are early in the mornings on Mondays and Wednesday, I know that's not the best time in the world, but just for various reasons, that's the only time I can really have them, but in general, if I'm in my office you are more than welcome to stop by and you're always welcome to call my office to see if I'm in during some other time in the week. Okay. I'll try to make it a point to just casually be around the office as much as possible, but Monday and Wednesday, 9:00 to 10:30, is really the official time when I'm around. Okay. I have a good number of staff members; I can have them just wave, you don't need to stand up, just wave, you can wave a little bit more enthusiastically, thank you. Okay. That actually only represents about a third of what the staff will be. Some people have a conflict today so they're not here. And I actually only figured out last night how large this class is. We expected a 130 - 140 students and I just checked at 10:55 that we have 241 signed up for the course, which is the largest that cs107 has ever been. It's really weird having this many people come and take and interest and listen when you talk.

I feel a little bit like Barack Obama. I'm delighted that this many people are taking it and so I certainly expect to have, on the order of 15 to 18 TAs and undergraduate section leaders working and they're very pertinent to the course. I almost require – in fact, it is the case this quarter, that every single person I'm gonna hire has taken 107 either as a undergraduate or as a first year graduate student, so they are great resources for questions and their office hours, with very little exception, they've all done all the various assignments that you'll be doing this quarter, so they already know what your questions are gonna be. Okay. As far as their office hours are concerned, they will have office hours and they'll rotate through a grid of evening hours at the main computer cluster where you'll be expected, or at least invited, to do all of your online programming work. Okay. Those will start next week. You're not gonna have an assignment until this Friday, and I make that one as easy as possible as far as the coding assignment is concerned because you have to get used to Unix and this new development environment. A lot of you have probably never been there before, so I go soft on your on assignment one so you can actually not be intimidated by the whole Unix development process. We do have a discussion section. We had scheduled one for tomorrow – I'm not even gonna say the time because I don't want to confuse people – but it turned out that it was terrible for three of my four head TAs so I'm in the process of rescheduling that. It's not gonna be on Thursday, it's probably gonna be on Tuesdays. That's what I'm shooting for now. At the time I had to take these handouts to press I just didn't have a time yet, but I will announce, certainly in lecture on Friday and probably prior to that via the emailing list. You're not required to go to discussion session, it is gonna be televised just like the lectures are – there will always be section handouts and section solutions, so even if you just don't want to bother watching it, not that I'm recommending that, but you certainly have the resources to figure out what they went over during that section because I'll be very good about putting up section handouts and section solutions so you know exactly what topics they were discussing. Okay. Whatever the time is next week, it'll probably be Tuesday sometime in the afternoon.

The first section, if you go to one section off quarter and go to one live section off quarter, next weeks is probably the one to go to if you've never really dealt with Unix before. They'll invest some energy coming up with some slides and some examples to show you what it's like to open an [inaudible] in Emax and to actually code there and then to use a command line to compile the program and see how you actually compile and execute and debug programs in a Unix world. Okay. And it's a little weird at first, you kind of hate it for three or four days then you start to like it because it's really light weight and very fast. So that's another reason why I make the first assignment less about coding and more about just getting used to Unix. Okay. Not next week, but the week after when we start having discussion sessions again, and having more of them, that'll adopt a more traditional format where we bring section problems and we talk and everybody raises their hand and asks questions and things like that. Okay. As far as reading material is concerned, in the handout I specify three books that I think are really great books. Two of them are on C++; they're actually quite advanced books. There's also one book on a language called Scheme, which I'll talk about in a second. These are by no means required and I don't even want to say that I recommend them in a sense that those who are really fastidious about going and buying every book they think is gonna help them, I'm not really even saying that you should go buy it, I just want them to be on your radar so that if after you get through the C and C+ segment of the course, you can see yourself doing more serious C++ programming later on, then you might want to go and get these book. But don't spend money on books yet because this class has more or less been taught with the same type of structure over the past 15 years so we've compiled pages and pages of handouts. This is just the tip of the iceberg. Anything you're gonna be responsible for is covered either in lecture of in the handouts. I'm not gonna have any external reading from text books so just worry about the handouts. The handouts are very polished. I think they're in really good shape. You'll average one to two handouts every single lecture and probably Friday and Monday, you're gonna get four or five each day. As far as the exams are concerned – let me talk a little bit about that. Midterm, there is one, and it is Wednesday, May 7. Okay. I schedule it for 7:00 to 10:00 P.M. People roll their eyes when they see a three-hour window set aside for a midterm.

I'm not gonna write the midterm with the intent of you needing every single minute of those three hours. I genuinely make an effort to make the exam so that it can be completed, patiently, in 60 to 90 minutes, okay, and occasionally, I'm off. When that happens, you have this buffer of an extra hour and a half to really work through it. Okay. I just give people three hours with the intent of giving the illusion of an infinite amount of time so that they don't feel that they're pressured to get through it. That may sound like I'm being really nice, but I'm actually being slight cruel by giving you three hours because if you do badly on it, then you have one less reason to blame me. You actually have to work through the problems and you really have to know the material, and you can't blame time pressure as being a factor as to why the exam didn't go well. So that's why I want to give people as much time as I can reasonably give them without going into 2:00 A.M., 3:00 A.M. Okay. I recognize that because this is scheduled outside of lecture that many of you may have classes, orchestra meets at night, people have various activities they have to do at night, I want to displace you from those because I decide to have my exam outside of lecture so if you can't make that time, that's fine, just let me know soon and plan on taking the exam during some three hour window that fits between 10:00 A.M. and 6:00 P.M. Okay. So most people that works with. If that's not gonna work out, then let me know now. I'll make sure you take the exam sometime, but I'll be around more or less or TAs will be more or less all day Wednesday to make sure that you can take an exam when you need to. I'll accommodate any reasonable requests. As far as the final is concerned, I offer the final exam twice. Our dedicated slot is Monday, June 9 and the official time is 8:30 A.M. Now, because we're on TV and because Stanford students – there's just not that many times to offer classes, a lot of people have conflicts, a lot of people are taking two classes, sometimes even three classes Monday, Wednesday, Friday at 11:00, and if that's the case, then you have two or three finals on Monday, June 9th at 8:30. If you want to take all three then, you're more than welcome to, but I assume you're not gonna want to do that so if you can't make this time, it is fine, provided you can make an alternate slot of 3:30 P.M. that day. If you cannot make either of those times, please let me know now so I can take care of that.

As far as the breakdown of assignments versus exams, this is how it all pieces together in the end. You have assignments, you have the midterm, you have the final, these contribute 40 percent of your grade. The midterm is the least of the three and the rest is the final. Okay. That's a pretty sizable chunk is dedicated to the assignments. People, in general, do very well in the assignments. They might struggle to figure out what kind of things we're expecting on assignment one, but eventually, people figure out what I'm expecting and they get the programs to work and they write nice clean code, so we see a lot of A's and A-s on the programs. We just do. And I imagine 80 percent of you will be getting A's and A-s on some of the later programs because you'll just start figuring things out. And that's not to say that grades will be much lower prior to that, but just toward the end everybody seems to be doing very well on the assignments. There's probably gonna be six or seven assignments. One of them will not be handed in, it'll be a written problem set where the deadline will, more or less, be at the time you take the midterm because it'll be a series of written problems that serve as practice for the midterm. You're certainly responsible for that, but all the other assignments take a programming assignment approach where you actually code up stuff and you electronically submit it and one of the section leaders or the TAs grade your work and they electronically or they email you back feedback. I don't need any paper copies at all. Everything is dealt with electronically. We're gonna really put a lot of effort into trying to turn the assignments around very, very quickly. In order to do that, I do allow you to turn work in late. I have the same type of late day policy that 106b does.

I actually give you the ability to grant yourself little extensions in 24-hour units. I give you five of those little days. And if you want, you can use one or two or all five later days on any particular assignment, but after five days, whether using free late days or extra ones that actually cost you a little bit, I need all the assignments within five days of the deadline so that I know that the TAs can start grading them and crank out grades and crank out feedback and get them back to you. Does that all make sense to everybody? Okay. As far as the midterm or final will go I actually try to make this – I don't want to say that I make it easy, I think, of the two exams, the final is a little bit more difficult because I think the midterm actually deals with the more difficult material so I go soft on you here knowing that I'm gonna revisit some of that material on the final. Okay. Now, when I say, "Soft," I don't mean easy, I just meant softer, comparative. Okay. Then I would normally go, but nonetheless, that's why I have the breakdown of 25 to 35 – really, all the material is equally represented on the final exam. I'm trying to think what else is going on. Oh, of course, as far as resources for keeping in touch with us, if you have a question specifically for me, then by all means email that address right there, but if you're asking a question about an assignment or lecture material that you know or suspect any section leader or TA would be able to answer, then you're gonna benefit by sending mail to this address. Cs107@cs.stanford.edu. That's precisely the address that you would assume would be attached to a staff emailing list for this class. When you do that, you're sending mail to an account that all 15 or 18 or 19 of us whole on a fairly regular basis and so you're that much more likely to get a quick response. Maybe not so much on a Saturday morning, but certainly on a weeknight when everybody's coding and deadlines are approaching. We're very good about pulling that and getting responses out as quickly as possible. Okay. If you don't hear back from this within, like say, 12 or 24 hours – let's say 12 hours in a weekday and 24 hours on the weekend, then it's okay to email me directly and say, "Hey, I'm not getting my answers, can you answer this question for me?" But be patient with that. We're actually very good – very often, we get back to people within minutes. A lot of times the TAs are sitting in front of the computer right next to you in the term or cluster where you're working, you just don't know what they look like so they respond to your question very, very quickly. I'm also gonna try to experiment – I have no idea what I'm planning to do with this, but I have a Facebook group. The URL is for cs107 – the URL is in the handout. You do have to be a member of Facebook.

I'm not trying to evangelist Facebook just because I work there as well, but if you have a Facebook account already and want to go join this Facebook group, just visit the URL that's posted in the handout or just do a search for cs107 and I'm sure it'll come up. I also have a Twitter account for 107. If you want to follow cs107 – I think this is an interesting idea because I could send little announcements, these very lightweight, noninvasive announcements to everybody who's paying attention to cs107 on Twitter saying, "Webpage has been update or exam cancelled," or something like that. Okay. And you can all follow what's going on there. I don't want you to think that you have to pay attention to these. Anything that's truly important about assignment deadlines or things that need clarification, I will actually use the mailing list and post to it. That will be the official forum for things that really matter but these are intended to be lighthearted. As far as that mailing list is concerned, I have 241 people signed up for the course already. That means that most of you, if not all of you, are automatically enrolled in the mailing list for the course, so when I send mail to cs107-spr0708-students@list.stanford.edu, you automatically get that email provided you're registered for the class. If you haven't registered for the class yet, let me pester you a little bit to go and sign up for the class even if you're not sure if you're gonna take it because, probably, I send more announcements in the first week of the course than I do anytime later on because just a lot has to happen and we have to up ramp a lot in the first few days so I'm certainly gonna send an email out as soon as I found when section time is. A lot of you have signed up for the course may not realize this, but sometimes you can inadvertently set some privacy settings so that you're email is not put on that list – do you know what I'm talking about? Okay. So go to Stanfordu.com or u.stanford.com, I forget what it is or wdu where you just go make sure that your email is public just to the Stanford community so that you can get on that emailing list. If you have a problem with that, I understand that you don't want your email address advertised, but then just make it a point to email me and say that you're not that so I can put your email address on some sub list that I always include with the announcements. Does that make sense to everybody? Let me just talk a little bit about the syllabus. Let's make sure there's nothing in here. Is there anyone in the room who has not taken 106b or 106x, a couple people I'm assuming, okay. I'm just taking a pulse on this. I don't necessarily require 106x or 106b, specifically, but I do kind of expect that you've done some C++ programming, that the notion of a linked list or a hash table is familiar to you, a binary search tree, a function point or all of those things that are covered in our 106 courses are familiar to you. If that isn't the case or you don't know C++ all that well, then I'm a little worried because the first half of 107 is all about advanced C and C++, so if you don't have the basics down or you haven't been exposed to that stuff yet, it makes the first two weeks of the course, which is normally actually quite fun because you're learning all about the inside, under the hood machinery of the C language, it can make it pretty miserable unless you actually anticipate spending a little bit more time getting up and running. So if you don't know C or C++ and you've never written an assignment for a course in C or C++ then talk to me after lecture so I can kind of gauge as to whether 106b or 107 is better for you. Okay.

Here is, in a nutshell, a reduced version of what's presented in handout two. This is the syllabus. I have several languages that I'm gonna put on the board and one concept. C, Assembly, C++, Concurrent programming – that's not a language, that's just a paradigm, Scheme and then it is official, I'm actually gonna cover Python from now on in 107. People look at this and they go, "Wow, I'm gonna be able to legitimately put all of these languages on my resume," and it's from a separate list at the bottom and it feels really good. It isn't so much about that. Certainly we want to give you some mileage in some very relevant languages that are very good for both research and for industry, but the real intellectual value in learning all of these languages is to really study the paradigms that they represent, and I'll explain what that means in a second. I think a lot of C++ programmers really program in C and just incidentally use the objects within the classes that are available to them, okay, which is a perfectly reasonable way to program. Most people learned how to program in C, at least the people I know in the industry, know C very, very well, and in spite of the fact that there's 50 million newer languages that are better in many regards, they still stick with what they know and that's why C and C++ are still such popular languages. There's nothing wrong with programming in C if you know it very well and you write clean, readable code, it's just more difficult. I don't really care so much about teaching you how to program an assembly, I use it as a vehicle for showing you how C and C++ programs compile to dot 0 and to object files and to binaries and that become executables and show you how a line like I = seven or J++ or FU of X and Y is a function call, how that all translates to assembly code. Okay. Does that make sense? You know when you write C++ code that when you execute the program, it's not C++ anymore, it's assembly code. It's all zeros and ones eventually. I want to give you some insight as to how C is translated to assembly code, how all these variables and your functions and your objects and all of that, eventually, get boiled down and hashed down to a collection of zeros and ones and I want to do a little bit of the same thing for C++. It turns out that – well, C++ and C represent different paradigms, that they really compile the zeros and one, and after you get enough experience with this assembly and the manual compilation process that we're gonna learn about and how to look at C code and figure out what the assembly code would look like, you're really gonna see that C++ and C almost look like the same language as far as the zeros and ones are concerned. I'm gonna be able to do something like **&**P arrow ***=7 and you're gonna know exactly what it means. Okay.

So it takes a little bit of work and it's almost laughable how arbitrary you can be with formulas, but if it compiles, it means something so when it runs it actually does something. It's probably not good if you have a lot of asterisks and ampersands, but nonetheless, you can have some idea as to why it's crushing, not just that it is crushing. Okay. I spend a good amount of energy talking about concurrent programming. We actually – at the moment, do that type of programming in C, but all the programs you've written in the past two quarters, if you've just taken the 106A and 106b courses here or 106x, all the programs you've written at Stanford, prior to 107, have been sequential programs. That means that whether it's object oriented or procedurally oriented, you have this outline of steps that happen one after another. Nothing is done in parallel or pipelined or done side-by-side, everything happens in one clean stream of instructions. Okay. Well, what concurrent programming is about is within a single program trying to get two functions to seemingly run simultaneously. If you can get two functions to seemingly run simultaneously then you can extend that and get 10 functions to run simultaneously or 20 functions to run simultaneously or seemingly simultaneously. I say it that because, technically, they don't run at the same time. When I go over assembly code, and I think you can intuit enough about what assembly code is like, but if you have this one function, okay, my hand is a function, this hand is another function, okay and you concern yourself with the execution of one of them and then when I do this you can just think about it reading the code and executing it for side effects. Does that make sense to people? When you deal with concurrent programming you have two or more functions, just two right here because I only have two hands, to do this and they both seemingly run at the same time, but what really happens is it's like watching two movies at the same time where –

because there’s only one processor on those machines, it doesn’t really run like this, it runs like this and switches back and forth between the two functions, but it happens so fast that you just can’t see the difference. Okay. It’s more than 24 transfer seconds, it’s, like, a million. There are a lot of situations where concurrent programming is not very useful, but there are several situations, particularly networking, whenever that’s involved, where concurrent programming is actually very useful.

There are some problems that come up when you deal with concurrent programming that you might not think about. The example I always go over the first day of class is just that it uses two Wells Fargo ATM machines. Okay. Think about you have a Wells Fargo checking account, you may not have to think about it because you probably do, so just imagine your checking account is in danger because two people are using ATM machines and you have a $100 in it and you share your PIN with your best friend and you go up to a neighboring ATM machines and you make as much progress as possible to withdraw that $100 and then you both, on the count of three, press okay to try and get $200 collected. Does that make sense? That is not a sense sensible example because both of those machines are basically very simple computers, okay, that ultimately need to access the same master account balance to confirm that $100 is available and in this transactional – transactional makes sense both in terms of money and also in the sense of concurrent programming – you have to make sure that the $100, being identified as the account balance, is maintained with some atomic flavor so that if you have two people trying to withdraw $100 at the same time that only one person gets away with it. That $100 account balance is the shared resource that two different processes have access to. Does that make sense to people? Okay. So there has to be directives that are put in place to make sure that the account balance check and the withdrawal are basically done, either not at all, or in full so that it really does function, truly, as a transaction, both in the finance and the concurrent programming sense. Okay. As far as this Scheme and Python are concerned, once we get through concurrent programming we really switch gears and we start looking at this language called Scheme. You may not have heard of this. If you haven’t heard of this, you may have heard of a language called LISP, which it’s certainly related to. This is a representative of what is called the functional paradigm. Okay. There’s two things about Scheme and functional languages – purely functional languages that are interesting in contrast to C and C++. When you program using the functional paradigm you always rely on the return value of a function to move forward. Okay. And you program without side effects. Now, that’s a very weird thing to hear as an expression when you’ve only coded for a year, but when you code in C and C++, it’s very often all about side effects.

The return value doesn’t always tell you very much. It’s a number or it’s a bouillon, but when you pass in a data structure by reference to a function and you update it so that when the function returns, the original data structure has been changed, right, does that make sense? That’s programming by side effects. Well, the idea with Scheme and particularly the functional paradigm is that you don’t program with side effects. You don’t have that at all. You always synthesize the results or partial results that become larger partial results that eventually become the result that you’re interested in and only then are you allowed to print it to the screen to see what the answer to the problem was.

Okay. It’s very difficult to explain Scheme if you’ve never seen it before in a five minute segment of a full introduction, but when you get there, we have tons of examples of the paradigm. It’s a very fun, neat little language to work in. This language called Python – I’m suspecting that most people have heard of it even if they’ve never seen it. It seems to be the rage language at a lot of significant companies in the Bay Area. They’re very smart people with these companies so when they use a language and they like it, there’s usually a very good reason for them liking it. You’ve probably heard of a language called Pearl. Okay. It’s not a very pretty language. You can just think about, in some sense, Python being a more modern, object oriented version of Pearl. Okay. Now, if you don’t know Pearl and you don’t know Python it doesn’t mean anything to you, but just understand that this sexy little language that’s been around for probably 16, 17 years that’s really established itself as a popular language since year 2000, 2001. I know a lot of people who work at Google that program in Python on a daily basis. There’s a subset of us at Facebook that program in Python every day. It actually has a lot of good libraries for dealing with web programming. Web programming can seem really boring to a lot of people because it just seems like it’s just HTML and web pages and things like that. Real web programming is more sophisticated than that. You dynamically generate web pages based on content and databases and things like that and Python, being a scripting language, which means its interpretive and you can type in stuff as you go, it recognizes and reads and executes the stuff as you type, it’s very good for that type of thing, and if all goes well, meaning I have time to develop this assignment idea I have, you’re gonna write a little miniature dynamic web server in Python for your final project. It won’t be that sophisticated. You’re not gonna write all of apache, but you are gonna probably write some little thing where you really do have a web server behind the scenes making decisions about how to construct an HTML page and serve that over to a client. So it’ll be an opportunity to learn Python, to learn with libraries, to see that as a language because it’s fairly young, it has the advantage of not bothering to include C and C++’s and Java’s mistakes. It says, “No, I’ll leave that part out and I’ll go with this more interesting, well formed core,” it has great libraries, it has object orientation, you can program procedurally if you want to and just program like you’re in C, but using Python syntax.

There are even functional programming aspects in the language so even though the syntax is different from Scheme, conceptually, you can use Scheme like ideas in Python coding if you want to. Okay. I’ll be able to illustrate the client server paradigm and how it’s different from traditional programming. That’s not so much a Python thing, but Python is a good vehicle for learning that stuff. There are a few other paradigms that aren’t represented here, but I think I really cover all the ones that you’re likely to see for the next 15 years and you’re a coder. Okay. There are a couple of the languages that I may briefly mention the last night that are just fun, but they all have some overlap with some languages represented right here. Okay. You guys are good? Okay. So I don’t like starting in on any real material when I only have 10 minutes left, so I’m actually gonna let you go, but recognize that Friday I’m gonna have tons of handouts for you, I’m gonna have an assignment, okay, we’re gonna dive right into the low level pointer stuff of C and C++. Okay. So have a good week.

[End of Audio]

Duration: 37 minutes