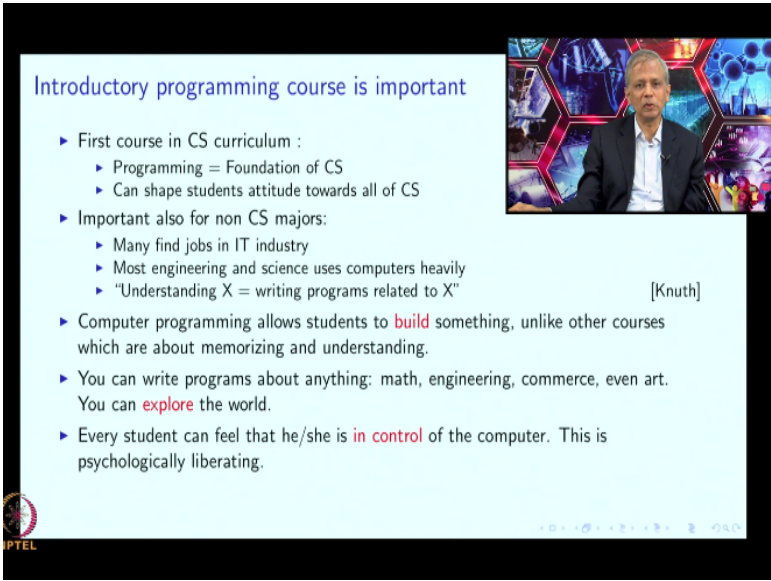


Design and Pedagogy of The Introductory Programming Course
Prof. Abhiram G. Ranade
Department of Computer Science and Engineering
Indian Institute of Technology – Bombay

Lecture – 01
Course Overview

Hello and welcome to the first lecture of the course design and pedagogy of the introductory programming course. I am Abhiram Ranade of IIT Bombay. Today I am going to give you an overview of the course.

(Refer Slide Time: 00:37)



The slide is titled "Introductory programming course is important" and contains the following text:

- ▶ First course in CS curriculum :
 - ▶ Programming = Foundation of CS
 - ▶ Can shape students attitude towards all of CS
- ▶ Important also for non CS majors:
 - ▶ Many find jobs in IT industry
 - ▶ Most engineering and science uses computers heavily
 - ▶ "Understanding X = writing programs related to X" [Knuth]
- ▶ Computer programming allows students to **build** something, unlike other courses which are about memorizing and understanding.
- ▶ You can write programs about anything: math, engineering, commerce, even art. You can **explore** the world.
- ▶ Every student can feel that he/she is **in control** of the computer. This is psychologically liberating.

The slide also features a small video inset of Prof. Abhiram Ranade in the top right corner and a PTEL logo in the bottom left corner.

Let me begin by observing that the introductory programming course is very important. It is typically the first course in the CS curriculum and programming forms the foundation of computer science. Because it is the first course, it can shape student's attitude towards all of CS, so if they like the first course, if they like programming then there is greater chance that they will like all of CS.

Programming is also important for non CS majors. This is because many of these students find jobs in IT industry. So it is good if they have a solid foundation in programming and even if they do not take up an IT job and do their engineering or science jobs, note that these fields use computers heavily. So even for that it is vital if they have a good knowledge of programming.

Finally, a well-known computer scientist one of the fathers of computer science you might say Donald Knuth has said that the ultimate test of understanding any topic is whether you can write programs related to that topic. This is because when you write programs you have to know the topic inside out, you have to know what happens in every case because in your program you will have to consider every case.

So you can write programs only if you know the topic very well and in that sense programming is really testing your knowledge. Now computer programming is a fairly unique course in that it allows students to build something. If you look at what students have learnt or what they have done until they do the first year course in college, you will see that they have done things like memorizing facts or even understanding facts and solving problems.

But they do not produce anything, they might do some projects here and there but those are small projects and nothing like a working model or anything that works comes out okay but computer science or computer programming enables them to build games maybe build things, so a program is almost like a live object and computer programming allows you to create something that interacts with you and that is very thrilling.

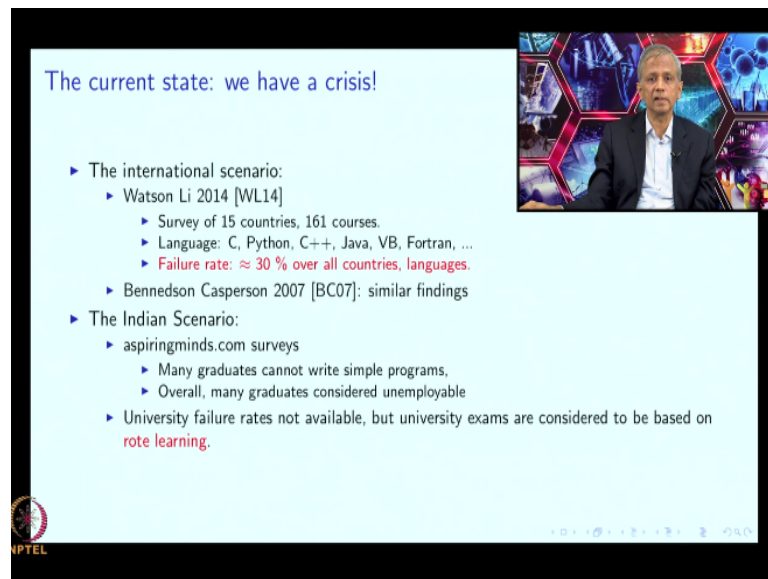
Now programming can be about anything, it can be about mathematics, it can be about engineering, it can be about commerce, it can even be about art, it can certainly be about games and it can in some sense you can explore the whole world, it can be about biology. Bioinformatics is a big field these days. So you may have likings and programming facilitates your tastes and your likings.

And last but not the least, there is a psychological aspect to all this as well. Throughout childhood our students have been obedient students or we like them to be obedient students. What that means is they are accustomed to following orders; however, when it comes to computers, they are actually in charge, they are actually commanding, they are in control of the computer.

So this psychological change I think can be extremely liberating. So what does all this add up to? It seems to me that there is potential in all of this for students to fall in love with

programming but is that what happens. If you look at the current state, we seem to be having the big crisis.

(Refer Slide Time: 04:51)



The current state: we have a crisis!

- ▶ The international scenario:
 - ▶ Watson Li 2014 [WL14]
 - ▶ Survey of 15 countries, 161 courses.
 - ▶ Language: C, Python, C++, Java, VB, Fortran, ...
 - ▶ Failure rate: $\approx 30\%$ over all countries, languages.
 - ▶ Bennedson Casperson 2007 [BC07]: similar findings
- ▶ The Indian Scenario:
 - ▶ aspiringminds.com surveys
 - ▶ Many graduates cannot write simple programs.
 - ▶ Overall, many graduates considered unemployable
 - ▶ University failure rates not available, but university exams are considered to be based on rote learning.

So let us look at the international scenario first. There are studies which indicate and Watson and Li are the two authors whose study I am going to talk to you about. They have surveyed introductory programming courses in 15 countries and 161 such courses and over several languages; C, Python, C++, Java, Visual Basic, Fortran and a few others as well and what did they find?

They found that over all countries over all languages the failure rate was about 30%. Actually, it was more than 30%, so less than 70% of the students who took the course passed it. That is really bad news in my opinion and this study is not the only one. There was a similar study sometime ago with similar findings. It is surprising that those numbers were also essentially the same.

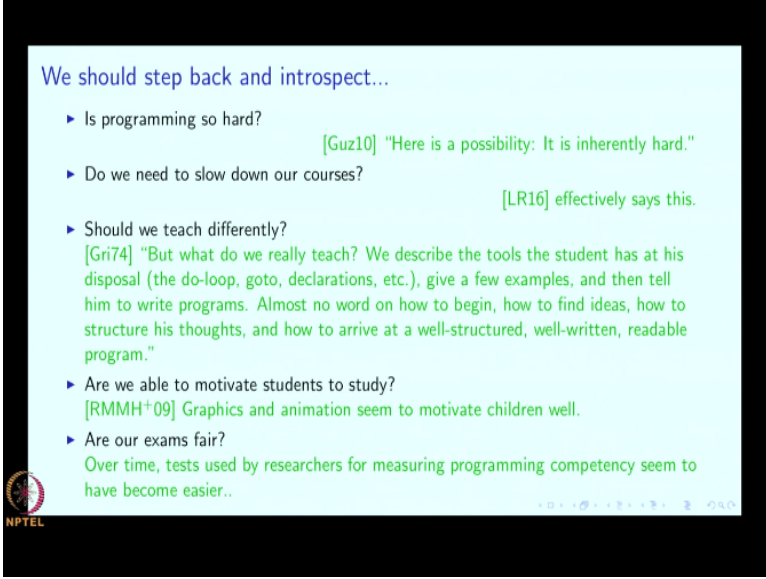
The Indian scenario is also similar. So there are several surveys but perhaps the most well-known survey is from aspiringminds.com and they say that many graduates cannot write simple programs. In fact, they say that even getting a program to compile is somewhat difficult for our Indian graduates and of course there are other studies which say that many of our graduates are really unemployable.

So things are not really much better over here. Now if you look at the university failure rates, they are not so easily available; however, university failure rates and difficulty of papers are

related. So I think the papers that are produced for exams in India often tend to be based on rote learning or memorization and therefore it is possible that the failure rate is not as high. On the other hand, the capabilities that we are testing for are certainly much lower.

So I think we should be worried about this situation as much as people outside India as well. So we really need to step back and think about what is going on.

(Refer Slide Time: 07:30)



The slide is titled "We should step back and introspect..." and contains a list of questions with corresponding references. The questions are:

- ▶ Is programming so hard? [Guz10] "Here is a possibility: It is inherently hard."
- ▶ Do we need to slow down our courses? [LR16] effectively says this.
- ▶ Should we teach differently? [Gri74] "But what do we really teach? We describe the tools the student has at his disposal (the do-loop, goto, declarations, etc.), give a few examples, and then tell him to write programs. Almost no word on how to begin, how to find ideas, how to structure his thoughts, and how to arrive at a well-structured, well-written, readable program."
- ▶ Are we able to motivate students to study? [RMMH+09] Graphics and animation seem to motivate children well.
- ▶ Are our exams fair? Over time, tests used by researchers for measuring programming competency seem to have become easier..

The slide also features the NPTEL logo in the bottom left corner and navigation icons in the bottom right corner.

The first question that arises perhaps is that is programming so hard? Is it so hard that 30% of the students have to fail? Well Mark Guzdial is an eminent educationist and here is a quote from his blog. He says here is a possibility. It that is programming is inherently hard. So what he is saying is that look there is nothing to be done here, this is a difficult subject and if 30% of the people are failing that is just an inherent property of the subject.

Now I do not subscribe to this and I would like it if this is false but that is what an eminent educationist say, so we had been to take notice. You could say that if 30% of the students are failing maybe we are going too fast. In fact, in the recent conference Luxton-Reilly said that there is nothing like a hard subject, if something is hard for students, you break it down, you slow it down and you slow it down until you can produce a fair question paper consistent with what you are teaching and you get reasonable failure rates.

Again that would be a very strong, very, very game changing decision if we decide it to slow down our courses. Furthermore, slowing down courses is easy to say but not easy to do because when you are talking about programming there is a certain body of knowledge that

you want to impart. It is very difficult to break it down, so we may consider that we should break it down but breaking down will not be completely easy.

We could also ask well are we teaching in the right manner or is there a different way to teach programming. Now there are many people who have said that actually if you look carefully at what we do in computer programming classes, we teach very little, well we teach the language but other than that we do not teach much. I am going to give you an old quote but there are newer quotes of this kind as well but this old quote is rather eloquent.

And this is due to David Gries who is another stalwart in the field who says but what do we really teach? We describe the tools the student has at his disposal that is the do-loop, goto, declarations, etc, give a few examples and then tell him to write programs. Almost no word on how to begin, how to find ideas, how to structure his thoughts and how to arrive at a well-structured, well-written readable program.

In fact, in the same paper Gries gives an even more detailed analogy. He says something like suppose you wanted to teach somebody to make wooden cabinets, suppose you did the following, suppose you showed him a few tools, maybe a saw, maybe some glue, maybe whatever tools are needed for making cabinets and you supplied the raw material and then you showed a few cabinets and then you say look now you know you have the tools, you know what is to be done, just go ahead and do it.

Gries says that in fact this is the standard of computer science education, computer programming education when it comes to programming. We teach the language but we really do not teach how to write programs. This is strong criticism and I am sure many of us will object but many of us will also agree that there is more than a grain of truth to this. Are we able to motivate students to study?

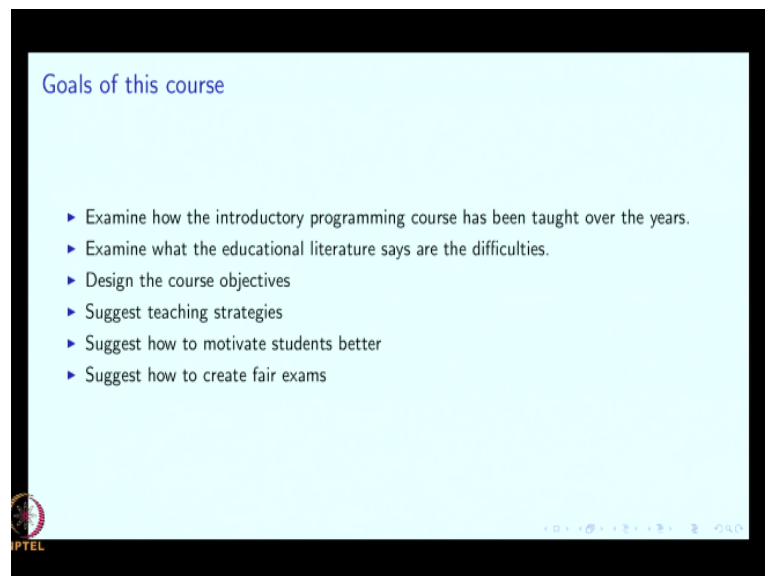
Now in this day of social media and cell phones, students are extremely distracted. So we need to do something to get their attention. There have been some studies and I am going to go over them a little bit more in detail during the course and they say that graphics and animation seem to motivate student, motivate children, even children can be motivated to write programs through the use of graphic set animation.

So perhaps if we feel that our students are not getting motivated and when I say our students I mean college going students, if they are not being motivated, maybe we should try things like this. Last but not the least is the question of are our exams fair? Now over time researchers have used tests for measuring programming competency of students, the kind of passing numbers, the number of students were passing the course is one kind of data but researchers have also conducted tests by asking students to write programs.

And if you look at the earlier such tests, they were hard and as time went on the tests that researchers have been using have become easier and easier and will see those tests as well. So this raises the issue were our exams fair, some time ago are they now becoming fair, even at this point are they fair, somebody might say that look the tests that we are using are so easy now that they are not really measuring anything.

So what is the fair test is a really important question and I think I would suggest that it is not an easy question to answer because experts seem to be divided, experts seem to be changing their minds about what is a good test okay. So all this leads us to this course, all this says that a course in which we discuss how to teach introductory programming is going to be very useful. So here are the goals of this course.

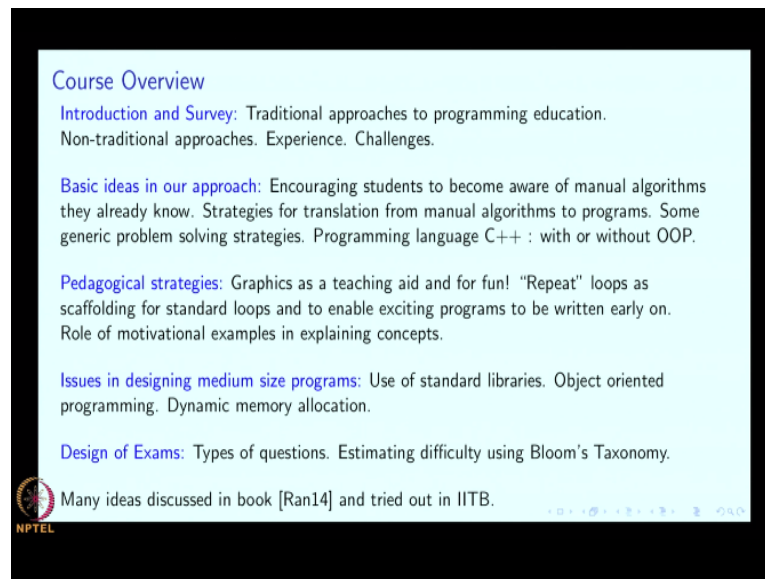
(Refer Slide Time: 14:19)



First, we will examine how introductory programming courses have been taught over the years okay. We will examine what the educational literature says are the difficulties. By this I mean specific aspect is a loop difficult is regression difficult, what is difficult? Then, we will

design the course objectives, we will suggest teaching strategies, we will suggest how to motivate students and we will suggest how to create fair exams.

(Refer Slide Time: 14:56)



Course Overview

Introduction and Survey: Traditional approaches to programming education. Non-traditional approaches. Experience. Challenges.

Basic ideas in our approach: Encouraging students to become aware of manual algorithms they already know. Strategies for translation from manual algorithms to programs. Some generic problem solving strategies. Programming language C++ : with or without OOP.

Pedagogical strategies: Graphics as a teaching aid and for fun! "Repeat" loops as scaffolding for standard loops and to enable exciting programs to be written early on. Role of motivational examples in explaining concepts.

Issues in designing medium size programs: Use of standard libraries. Object oriented programming. Dynamic memory allocation.

Design of Exams: Types of questions. Estimating difficulty using Bloom's Taxonomy.

Many ideas discussed in book [Ran14] and tried out in IITB.

NPTEL

So let me give you a quick overview of the topics in this course. So the first topic is going to be introduction and survey. Here we will first look at the traditional approaches to programming education. By this I mean what probably goes on in say 90% of the places in India and also abroad which is a course using a language like C, C++, Java something like that maybe Python.

Then, I will also talk about the non-traditional approaches and these might be based on functional programming for example and then I will talk about some experience with these. So what are people saying is difficult, what are they saying is easy and then I will try to extract what are the challenges from all the discussion that we are later will have. Then, I will go towards presenting and approach which I might call our approach.

And one of the big ideas in this approach is to encourage students to become aware of the manual algorithms that they already know. You might agree with me that our students that is first year college students already know a lot of algorithms and I can tell that by getting them to be aware, it will help us teach them to program and in fact what we are going to say is that we should be teaching them how they should translate from what they know about manual computation or they should translate their manual algorithms to computer programs.

And this translation is what we should be teaching them. In addition, we will also teach, we will also talk about some generic problem solving strategies which I believe we should be teaching them. In this discussion of the programming language that we will consider for examples is going to be C++. Most of it will be without object-oriented programming but some of it we will discuss to what extent we should use object-oriented programming.

Many ideas however will be applicable to all languages. So if you are going to use Java still many of these ideas will be applicable. Then, I will talk about pedagogical strategies. What I mean by this is that we already have in mind a certain curriculum that we want to teach which I will discuss in our second main topic that is what I have written down as basic ideas in our approach.

Once we have fixed our basic curriculum, we might need to teach things so as to help motivate students or we might have to teach things which help us in teaching them. So we are going to use, we are going to teach students some kind of graphics. So graphics is going to be a teaching aid and it will also serve as a fun element. It will serve as an element which will keep our students attracted.

However, you will note, you will be pleased to note that it is not going to be all fun. Graphics is a really, really powerful medium. It allows us to explain many things very nicely. It can be used to give very challenging project, challenging programs as assignments and also our projects. So we will have fun but we will have good learning as well. A second idea or second pedagogical strategy that we are going to be using is a so called repeat statement.

This is not a statement which is already in C++. This is a statement that we have designed and we have so to say inserted into the language through the use of a preprocessor macro. So as far as our students are concerned, it will look like a language statement. It will turn out that the repeat statement is very easy to understand and therefore students can start using repeat within 5 minutes of starting the course.

Because it is so easy which means that from the first day our students can start writing an interesting program because repetition is sort of key to anything interesting going on, otherwise if it is some kind of straight line code that they have to write then it cannot be too fine or too interesting or it cannot be too powerful. So with the repeat you will see that we

will be able to write quite interesting codes, quite interesting programs, programs which do interesting things from the very first day.

And we will see when we survey how programming has been taught that simple loops are actually not that straight forward to understand and what will happen is that because of the repeat statement we will get a foot in the door as far as understanding loops is concerned and therefore it will sort of help us actually or it will help the students in coming to grips with the standard looping statements like while and for and do while.

And finally as far as a pedagogical strategy is concerned, I am a big believer in motivational examples. If I have to teach something, the motivation has to be extremely clear. If you tell the student something like look you will not know I cannot tell you why you are going to be using this or why this is the useful thing.

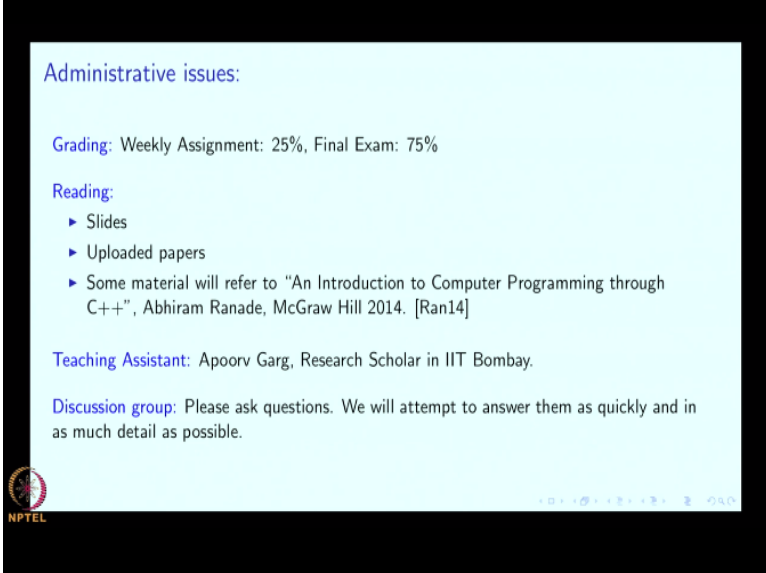
But believe me use it, believe me learn it, they are not going to feel very motivated to study whatever you are telling them. So as teachers, it is a really important responsibility for us to be finding motivational examples and that is going to be stressed a lot. The fourth topic is we will go and talk about designing medium sized programs. So until this point or until say 75% of our course the programs we write will be maybe 20 lines at most, 30 lines at most.

By medium sized programs I mean programs which are about 100 lines. So things do change at that point, you have to be a lot more careful. You have to be little bit more careful in how you spend your effort. So I will talk about standard libraries, I will talk about objective related programming, maybe I will talk about dynamic memory allocation. So these are the topics which are needed for designing medium sized programs.

But of course I realize that every course in every university in India may not talk about this because they might say that look we only have 25 lectures to devote to introductory programming and they may have a second course. So I will also discuss what parts of the course are going to be compulsory sort of the core elements and what you can consider as the optional elements and which you should do in a second course or which you should do if you have time and finally I will talk about designing exams.

What types of questions you should ask and how do you estimate the difficulty, what care you should be taking? As far as estimating difficulty is concerned, there is so called Bloom's Taxonomy which says how do you measure, how do you sort of talk about the hardness of a question. Many of the ideas that I will be discussing have been discussed in a book I wrote which is cited at the end and it has been tried out in IIT Bombay.

(Refer Slide Time: 23:26)



Administrative issues:

Grading: Weekly Assignment: 25%, Final Exam: 75%

Reading:

- ▶ Slides
- ▶ Uploaded papers
- ▶ Some material will refer to "An Introduction to Computer Programming through C++", Abhiram Ranade, McGraw Hill 2014. [Ran14]

Teaching Assistant: Apoorv Garg, Research Scholar in IIT Bombay.

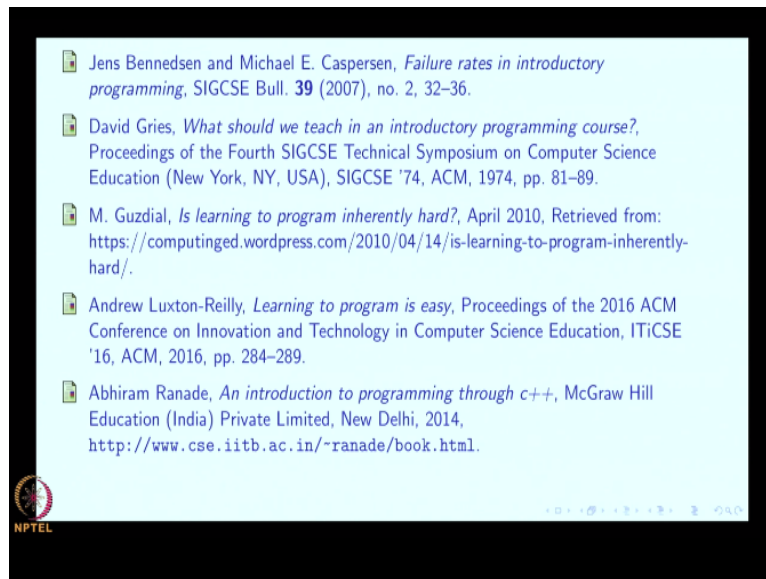
Discussion group: Please ask questions. We will attempt to answer them as quickly and in as much detail as possible.

NPTEL

So a few administrative issues, the grading will consist of a weekly assignment and that will contribute to 25% of your final score. There will be a final examination, which will contribute 75%. The reading will come from slides, from the uploaded papers and some material will refer to the book that I mentioned, the name of the book is an Introduction to Computer Programming through C++ and it was published by McGraw Hill 4 years ago.

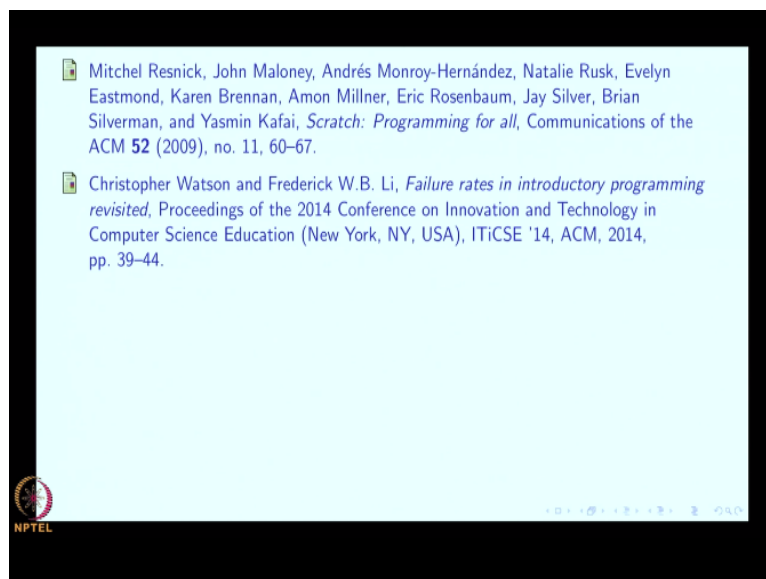
Apoorv Garg who is a research scholar or a Ph. D student in IIT, Bombay is going to be a teaching assistant for this course and like all NPTEL courses we will be having a discussion group. So please ask questions, we will attempt to answer them as quickly and in as much detail as possible.

(Refer Slide Time: 24:19)



So here are the references and you are welcome and you are encouraged to look at the references and find them on the net.

(Refer Slide Time: 24:27)



If you do not find the reference, I will be happy to upload it or send it to you. Thank you.
That is the end of the first lecture.