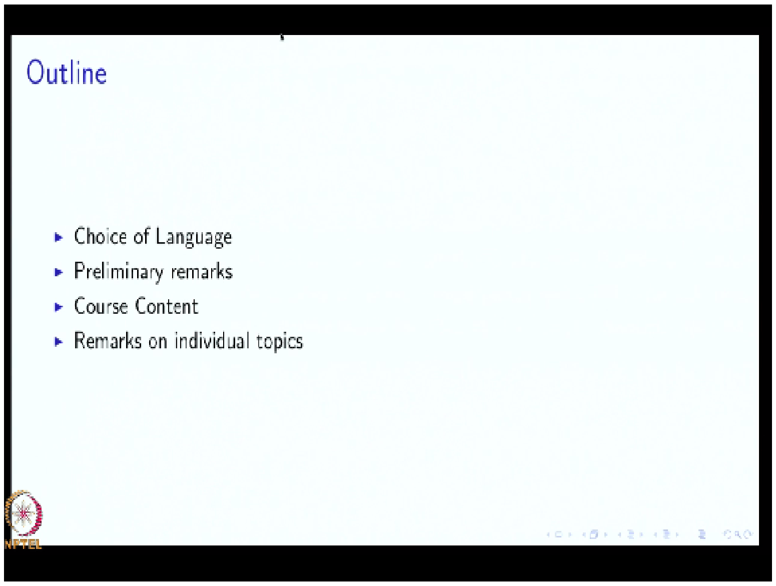
Design and Pedagogy of The introductory Programming Course

Prof. Abhiram G. Ranade
Department of Compute Science and Engineering
Indian Institute of Technology - Bombay

Lecture – 09
Basic Ideas in Our Approach. 4: The design of the Course

Hello and welcome back. In this lecture, I am going to talk about the design of the course that we want to design. So some preliminary thoughts and then on to the actual design. So here is going to be the outline.

(Refer Slide Time: 00:35)



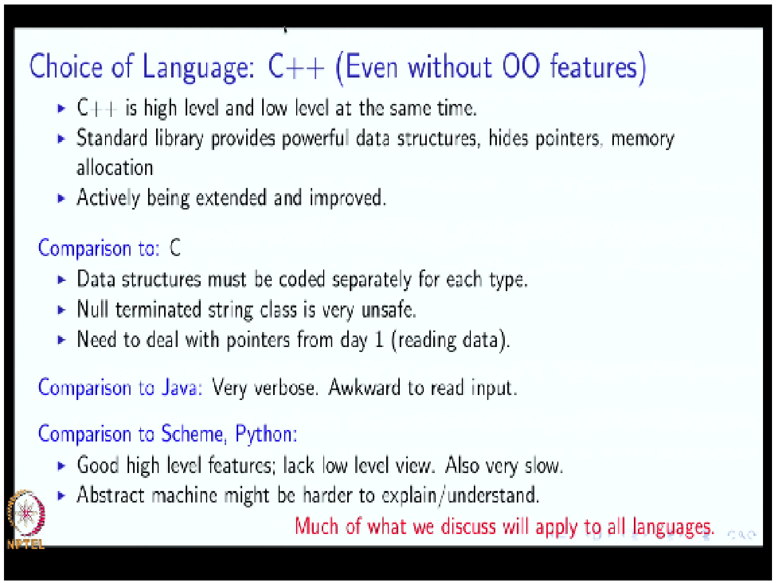
Outline

- Choice of Language
- Preliminary remarks
- Course Content
- Remarks on individual topics

First, we will talk about choosing the language. Then I will make some preliminary remarks. Then we will talk about the course content. By this, I mean what we are going to cover in the course, what outcomes we expect, what skills we expect to impart. So we are going to take a long look and write these things down and then we will end by making remarks on the individual topics because just mentioning a few topics does not really convey to what extent or to what depth we want to cover them.

So we will make some detail remarks on the individual topics. So regarding the choice of the language. So as far as this course is concerned, I am going to assume that the language of the programming course that you are teaching is C++. I have a personal preference for C++ and so I shall explain that to you. I believe that C++ is a high level and a low level language at the same time, okay. So maybe, maybe I should explain what I mean by that.

(Refer Slide Time: 01:46)



Choice of Language: C++ (Even without OO features)

- C++ is high level and low level at the same time.
- Standard library provides powerful data structures, hides pointers, memory allocation
- Actively being extended and improved.

Comparison to: C

- Data structures must be coded separately for each type.
- Null terminated string class is very unsafe.
- Need to deal with pointers from day 1 (reading data).

Comparison to Java: Very verbose. Awkward to read input.

Comparison to Scheme, Python:

- Good high level features; lack low level view. Also very slow.
- Abstract machine might be harder to explain/understand.

Much of what we discuss will apply to all languages.

It is high level in the sense that it has a library which allows you to write code which includes even data structure and your code can be talking about abstract mathematical entities, very high, very powerful abstract mathematical entities. On the other hand, you can write code which talks about pointers and pointers are in some sense the lowest level concept in programming. So C++ is one of the few languages which allows you to do this.

Go from the lowest level of programming to the highest level of programming. So C++ has a standard library which provides powerful data structures which hides pointers, hides memory allocation and it is, it is a very live language. It is actively being extended and improved. So as better and better programming strategies get discovered, okay, these are being incorporated into C++.

So let us compare the C language which is also very common, a very, perhaps very popular especially in India, okay. So I should say perhaps at the beginning that C is essentially a low level language. So everything that we have to do, we have to do at a low level. You can build a functional interface on top of it. You can build functions which do more complex things but you really have to worry about low level things all the time.

For example, if you want to build any data structure, you have to build the data structure separately for each type. So if I want a data structure or if I want to have a cue to store integers, that cue has to be written differently from a cue to store some structures. So these low level considerations are, are going to bother you all the time if you are going to program in C. Then the null terminated string class of C which is extremely heavily used, I would say it is really a minefield.

It is waiting, it is, it is, if you program that way, you are just waiting for all kinds of errors to crop up, okay. So first of all, you have to deal with pointers and then you can overrun arrays and you can do all kinds of bad things, okay. The language does not warn you. Sometimes you have to do strange looking things but, and, and the language sort of forces it on you. Another difficulty as far as novices are concerned or the introductory programming course is concerned, is that in C you have to deal with pointers essentially from the first day.

Even to read data, if you write scanf, you have to pass it the pointer to the variable which is being read. So you have to explain pointers to your students practically on day 1 which is a pretty, pretty tricky undertaking. Java is another popular language. Java turns out to be very verbose, okay and it is somewhat high level in the sense that pointers are sort of hard to get to but in addition to its verbosity, it is somewhat awkward for somethings like, even reading input.

So for reading input, we have to construct all kinds of objects and worry about parsing and things like that before you can just read something. Scheme and python are 2 languages which have been popular or are very popular. I believe that these are very high level languages, okay. So they lack a low level view of what is going on. So they do not have pointers. So you do not really see the machine.

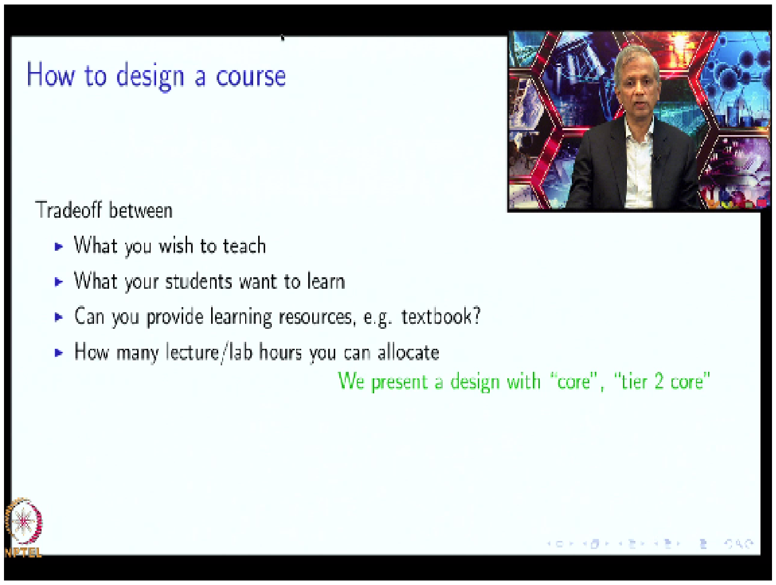
You are not really, you are not really knowing of what is exactly going on. So if you ask for a variable, what exactly happens on the machine. Now you might say that I do not need to know all of that. On the other hand, it depends on your agenda. So if you feel that your introductory student should really get a good view of what happens in a computer, then these languages probably hide it.

They hide this low level view and then these languages are also very slow. So if you want at some point to do a very fast programming, then these languages are probably not the right way to go. These machines have very high level functions and sometimes, what exactly those data structures or what exactly those functions do, is pretty tricky to explain, okay. So if they create an object, when does that object go away, okay, so all those considerations are somewhat hard to explain to a beginner, okay, explain.

So they are hard to explain and because they are hard to understand. They are somewhat complicated, okay. On the other hand, the language is not really the important one part of the course, we are talking about C++ because we need a vehicle and most of what we are talking about is going to apply to all languages. So certainly this idea that manual computations is very important.

You should encourage students to do manual computations first. To design algorithms in a manual setting first is extremely important. So in fact everything that I have talked about so far is basically applies to all languages. Everything that I have talked about so far applies to all languages and in fact, you will see that most of what we teach will apply to all languages.

(Refer Slide Time: 07:57)



How to design a course

Tradeoff between

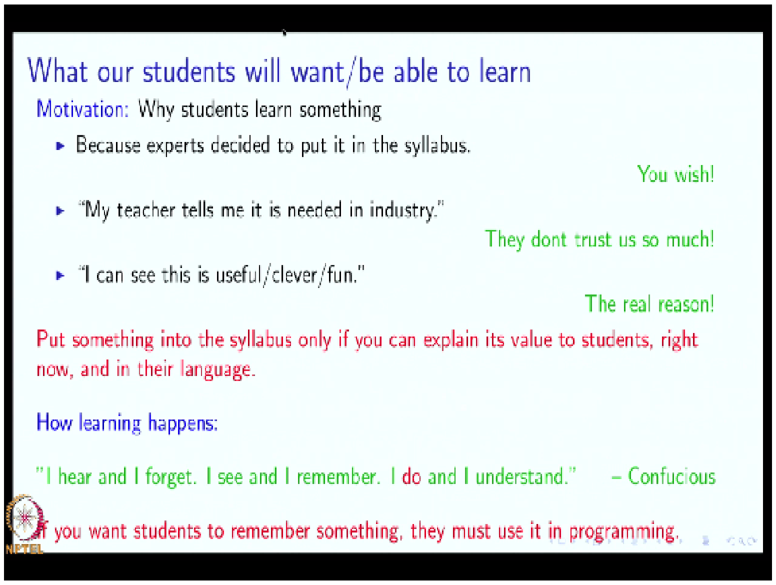
- What you wish to teach
- What your students want to learn
- Can you provide learning resources, e.g. textbook?
- How many lecture/lab hours you can allocate

We present a design with "core", "tier 2 core"

Okay, now let us consider abstractly how would you design a course? So first of all we should remember that designing the course is a tradeoff. So it is a tradeoff between what you wish to teach, what your students want to or perhaps I should have even said can learn, okay. It also depends on things like whether we can provide good learning resources. Is there a good textbook to teach what you want to taught, okay?

And how much time do you have? How many lab hours can you allocate? So this number can vary quite widely. So depending upon what you have, you may be able to teach a smaller course or a larger course and we are going to cater to that to some extent, okay. So we will present a design with a core and also another called a Tier 2 core. So we can design which of those topics we should take depending upon how much time you have.

(Refer Slide Time: 09:02)



What our students will want/be able to learn

Motivation: Why students learn something

- Because experts decided to put it in the syllabus.

You wish!

- "My teacher tells me it is needed in industry."

They dont trust us so much!

- "I can see this is useful/clever/fun."

The real reason!

Put something into the syllabus only if you can explain its value to students, right now, and in their language.

How learning happens:

"I hear and I forget. I see and I remember. I do and I understand." – Confucious

If you want students to remember something, they must use it in programming.

Okay, so let us look at in more detail or let us ask ourselves what do our students want to learn or what will they be able to learn, okay. So first of all, we should ask why do students learn something, okay? Do they want to learn? What happens, what exactly happens in the classroom? So one answer to this why students learn something could be that experts decided to put it in the syllabus.

Therefore, the teacher teaches it and therefore, the students learn it. You wish that were true but unfortunately, things are a lot more complicated. Just because you think somebody must learn something, does not mean they will learn it. Certainly, it does not mean they will learn it well. So we have better, we have better things through this a lot more. Another idea is to say that not experts but my teacher who perhaps I trust more says it is needed in the industry.

Well frankly, I have not met students who trust me that much, okay. So they want, they want to know for themselves. So if you can persuade them that whatever you are teaching is useful, it is clever, students like cleverness. So if you can persuade them that it is clever or you can persuade them that it is fun, then they will learn. So to me, the real reason for learning something is just this last one.

So you need to persuade your students that there is some payoff, okay. Then they will see something exciting, they will see something useful but they have to feel that themselves. The new generation is not willing to take any ones word for it. If you force them to learn because I say so, then they will go through the motions. They will not really learn. Okay, so what does this tell us.

Well it says that put something into the syllabus only if you can explain its value to students right now and in their language. That is in a way that they can understand. Do not use big words. Do not tell them experts say that this is needed, okay. Do not say that industry needs to maintain program for tens of months or tens of years and therefore, you should code in a certain style. That is something that they have to learn through experience.

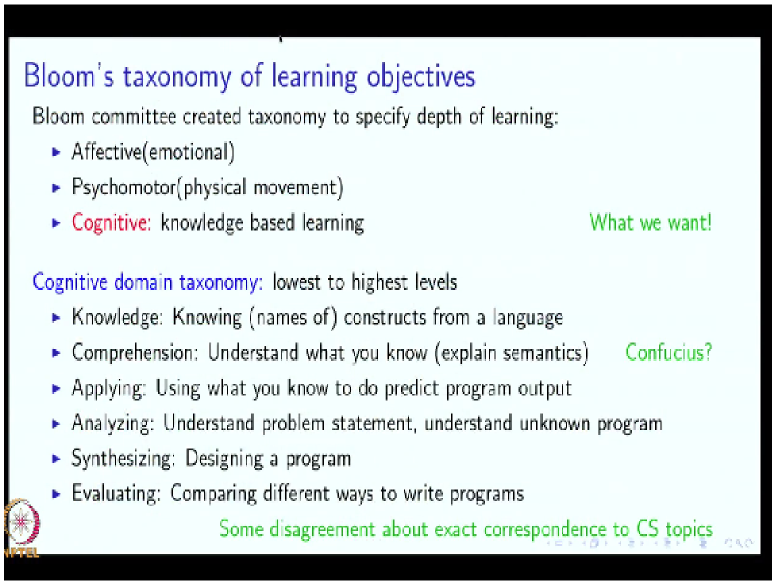
So my suggestion would be do not even bother to talk about it. If you talk about it, you will lose your credibility, okay. Another point is, about how does learning happens and here, I would like to quote what Confucius says. I think it is a beautiful quote. Confucius says, "I hear and I forget. I see and I remember. I do and I understand." I think all of us have probably experienced this, perhaps with our students, perhaps with our own children, our friends, whatever.

So I think we should keep it in mind. What does it means? If we are teaching something and if you want students to remember that, we should make them use it. Do, do is the key word there, not hear. They must do something with it and in our case, doing something with it is using in a program. So if you can make the students use whatever you are teaching them in a program that they write, much better chance of them remembering it.

So in this regard, I would like to tell you about this taxonomy that a committee chaired by Benjamin Bloom talked about some time ago. So he made taxonomy or perhaps you could even call it a hierarchy of learning objectives. Now Blooms committee created the taxonomy to specify the depth of learning and this was for many different domains. So what am I might be teaching, might be effective or it might be based on emotional ideas, perhaps your teaching literature.

So how do we specify depth when you are talking something about literature or you might be talking about sports. So what is it mean to say that somebody learns and somebody knows something deeply. So the committee created a taxonomy and the third domain was the cognitive domain. So this is what you might call knowledge based learning which is what we are interested in. So our, our subject can be said to be cognitive or knowledge based, it does not deal with muscles or emotions or but it deals with knowledge and thought.

(Refer Slide Time: 14:22)



Bloom's taxonomy of learning objectives

Bloom committee created taxonomy to specify depth of learning:

- Affective(emotional)
- Psychomotor(physical movement)
- Cognitive: knowledge based learning — What we want!

Cognitive domain taxonomy: lowest to highest levels

- Knowledge: Knowing (names of) constructs from a language
- Comprehension: Understand what you know (explain semantics) — Confucius?
- Applying: Using what you know to do predict program output
- Analyzing: Understand problem statement, understand unknown program
- Synthesizing: Designing a program
- Evaluating: Comparing different ways to write programs

Some disagreement about exact correspondence to CS topics

So this is what we want and what did Bloom say about it? Well, Bloom said that the learning objective is or the depth of learning from the lowest to highest levels could be something like this. At the lowest levels, you might expect your, expect knowledge from your students, what does it mean? You might expect students to know the names of the constructs from a language.

So maybe because of this, the students know that there is end data type, there is a double data type and so on or maybe they know that there is a While Loop or For Loop whatever.

The slightly higher level is comprehension. So in this the student is not just knowing the names but the student also understands. Slightly more deeply. In our case, it might mean that the student can explain the semantics, how exactly does the loop work, okay. So that is what the student can perhaps explain. Now does this relate to Confucius? Well, what did Confucius say? He says I hear, I forget.

So I think the point over here is that if you are only telling students and giving them information, they are likely to forget. So what do you want them to do? You want them to work with it. You want them to apply and this exactly is the third level of Bloom's taxonomy. So the third level, if you teach, if, if a student is at the third level, then the student knows how to apply the knowledge that he or she has learnt.

So in this case, application would mean perhaps that I give you a program, you know language constructs, so can you apply your knowledge to decide how exactly the program executes? What exactly does it print? You could go to a higher level of understanding which is analysis level in which you would be analysing, you would be capable of analysing. So this could perhaps mean that you are able to understand the problem statement, okay.

So understanding the problem set of statement, understanding the requirements nicely is a pretty difficult task, okay. Or you might be able to understand what exactly not, not just what a program prints but you might be able to understand what exactly is it doing. Is it calculating? Is it maximising something even if it is not written as comments in the code? Can you, can you analyse, analyse the program if, if it is a tricky program and say what it is doing.

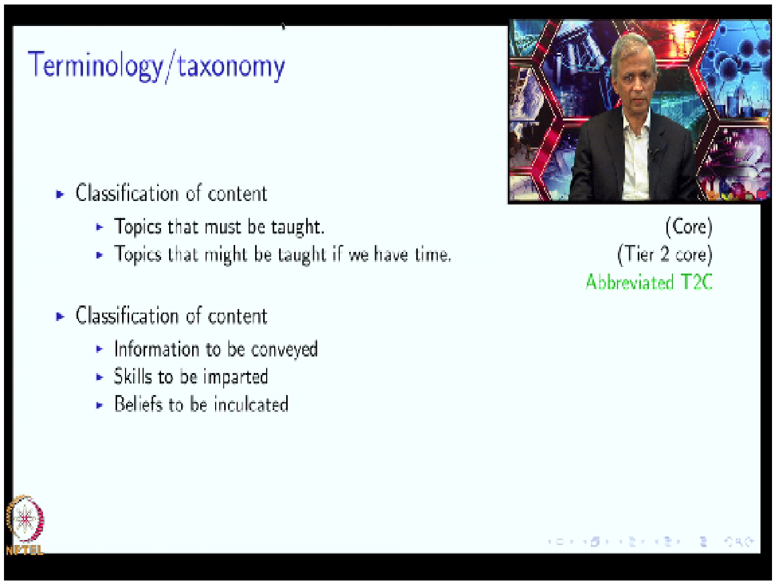
In applying the program is fairly straightforward and you just have to execute it by hand so that is application. Analysis might mean, you might actually have to figure out what is the, the human level function that the program is accomplishing and the next level is synthesizing. Perhaps we can think of it as designing the programs. Synthesis means build something new. So this might be designing a program, okay.

Write a new program. Evaluating is the higher, the highest level where you might be saying that look this is a good way of writing a program. This is not a good way of writing a program. These are the objectives that you are expected to satisfy. These are satisfied over here. These are not satisfied over here. The objectives could be that look this program is easy to read. This program is, is running fast.

It requires less memory and you are able to evaluate all these things. Now Bloom did not really talk in terms of computer science and the match that I have made between computer science and the taxonomy levels is not a universal match. Some people might match things differently but anyway, the reason I brought this up is that we, we want at least to go to the application level, okay.

So, so in high school perhaps it is okay to expect students to learn something by heart, okay but at, at this level, just because you know something does not mean much, okay. So it does not mean much to the students either because the student is going to forget it. So we want the student to exercise their knowledge. So whatever topics we are going to discuss, we should keep in mind that we should try to link them with some practice, something related to program. Okay, now we are going to use some terminology, okay.

(Refer Slide Time: 19:21)



Terminology/taxonomy

- Classification of content
  - Topics that must be taught. (Core)
  - Topics that might be taught if we have time. (Tier 2 core)
    Abbreviated T2C
- Classification of content
  - Information to be conveyed
  - Skills to be imparted
  - Beliefs to be inculcated

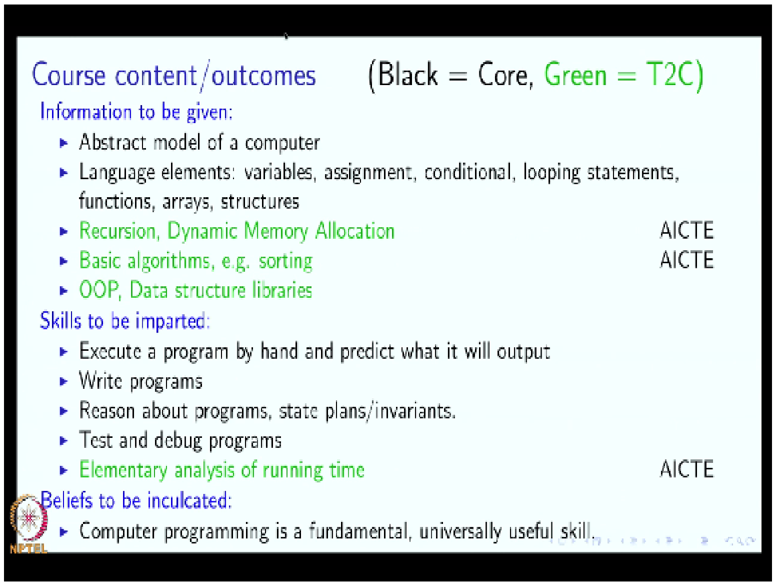
And we are going to, so we are going to classify the content, okay, so we are, in the following 2 ways. The first way is, so we are going to talk about the topics which must be taught, okay. So by that I will talk about core topic. So this, the topics that must be taught I will call core topics. This is as per the ECM guidelines, the association for computing machinery guidelines and they classify topics as core and a Tier 2 core.

So these are also important topics but they recognize that you may not have time and so in some sense they are less important or less basic than the topics which we are going to put in the core. Some of these topics could be elective, some of these topics might be very advanced and here I am not making a distinction between Tier 2 core and elective. But I am just stating right now that yes, it is possible that you may not have time to teach all the Tier 2 core and you may decide that yes this topic does not need to be taught right now.

It may be taught later on, things like that. But I would say you should try to make an effort to teach everything that is considered core that will be, that will be described as core. And Tier 2 core, we will abbreviate as T2C in what follows. So whenever you see T2C, remember that by that I mean Tier 2 core. Now we are going to classify content in another way so it could be information to be conveyed.

It is sort of like the first level, the first 2 levels of Bloom's taxonomy but we will match it with practice. So it will be applied as well, okay. So there will be skills to be imparted and the skills will use the information but the skills are going to be sort of the central part of the course. And while you do this, some beliefs have to be inculcated. So a student must come out of the course with sort of a changed mental attitude as well, okay. So what, in what way should the mental attitude be changed, this is also something that we should have in mind when we design the course.

(Refer Slide Time: 21:48)



Course content/outcomes (Black = Core, Green = T2C)

Information to be given:

- Abstract model of a computer
- Language elements: variables, assignment, conditional, looping statements, functions, arrays, structures
- Recursion, Dynamic Memory Allocation AICTE
- Basic algorithms, e.g. sorting AICTE
- OOP, Data structure libraries

Skills to be imparted:

- Execute a program by hand and predict what it will output
- Write programs
- Reason about programs, state plans/invariants.
- Test and debug programs
- Elementary analysis of running time AICTE

Beliefs to be inculcated:

- Computer programming is a fundamental, universally useful skill.

Okay, so now I am going to tell you about the course content or the course outcomes as they are sometimes called. I am going to use the black colour to, to write down the core topics, the green to write down the Tier 2 core. Sometimes as you will, as I will show, this might be a little bit fuzzy, okay. So let us begin. So first is the information to be given. So we should talk about an abstract model of a computer.

Everybody needs to know that. We should talk about the language elements, variables, assignments, conditionals, statements, looping, functions, arrays, structures. So everybody should know these things as well. So they are black and therefore core. Recursion and Dynamic Memory Allocation, AICTE thinks of as core but other people might think of both of these as Tier 2 core, slightly less important, okay.

So you have to make a call on this if you are pressed for time, okay. But amongst the Tier 2 core, these are probably the most important Tier 2 core since AICTE is saying that they are core. Basic algorithms like sorting also AICTE thinks of as core, okay but I would think of it as Tier 2 core, so less important. Objective Oriented Programming, Data structure libraries, okay are certainly not that important.

Object Oriented Programming is certainly a useful, there are useful ideas in it but they are not central. So you can learn a lot of programming. You can accomplish a lot in programming even without knowing Object Oriented Programming. Skills to be imparted, okay. Execute a program by hand and predict what it will output. So this is definitely needed. Write programs, this is sort of the major skill.

This is sort of the focal point of the course I might say. Reason about programs, state plans/invariants. So this last skill and the first skill are feeder skills. They feed into how you write programs. And of course, testing and debugging programs is important. Testing and debugging is very frustrating and of course, they are core skills. So in your, in your teaching, do take the time to sit with students, take a program which is not working and show them step by step how it gets, how will, how it will be debugged.

Elementary analysis of running time is in my opinion a Tier 2 core topic. AICTE thinks that it is a core topic. Now lets us come to the beliefs to be inculcated. So I believe that at the end of the course, the student should feel that computer programming is a fundamental, universal useful skill, okay.

So they really should feel the power of computing, okay. So they should understand that this is, that they have learnt something powerful. They should be able to do powerful things and it should sort of change the way they look at the world. So we are going to stop here and we will discuss the individual topics and detail out what should be taught in them and all that will happen in the next lecture. Thank you.