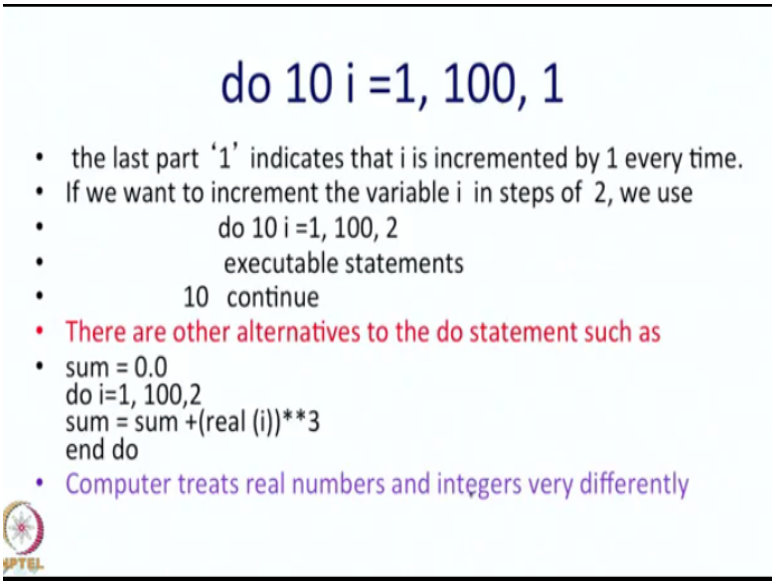
Computational Chemistry & Classical Molecular Dynamics

Prof. B. L. Tembe
Department of Chemistry
Indian Institute of Technology- Bombay

Lecture – 04
Programming Techniques 2: Do Loops and If Statements

Hello again, so last time we discussed a few programs and one of the objectives of the course, so this lecture, I will consider some more information about do statement, if statements and finally declaration statements. Remember, last time we said declaration statement also constitute one of the ingredients of the program, so declaration statement really determines what variables are going to do in the program.

(Refer Slide Time: 00:51)



So, coming back to the do statement, remember that we had statement like to do 10 i going from 1, 100, 1, so earlier we did not have this last comma 1 because it was understood that in this do statement, I will start with the value 1, next time it will be 2, next time it will be 3 and so incrementing by 1 was understood, so this is the default value. The default value is that which it considers without our specifying anything.

But suppose I want to increment something not by 1 but by 2 that is I want to consider all odd numbers between 1 and 100, how would I do that? 1, 3, 5, 7 and so on, so to do that instead of incrementing the do loop variable i/1 every time, I will incremented by 2 every time, so how will

I write that statement? Look at this now, do 10 i going from 1, 100, 2, so every time the loop comes back, so what is this 10; 10 is again a line number.

A do statement executes everything from the starting do statement up to the last statement which is line number 10 and it will do it until the value of i is matching this or becomes greater than this, so in this case first time it is i = 1, execute all the statements, come back, second time it is not 2 but 3 because it is incrementing i/2, so second time it is 3, execute everything up to line number 10, come back.

Now, i becomes 5 the third time all the way, then go back, you will go on repeating 5, 7, 9 all the way, i will become 99, go back do everything, when it comes back, so when; it is already 99, next time it will be 101, so it will not execute 101 because this statement says nothing more than 100 is permitted, so what this new type of do loop will do; it will execute all those statements for only odd numbers.

Now, my next question is; what will you do if you want to do all even numbers. Suppose, you want 2, 4, 6, 8 give a thought to that the way to do would be; do 10 i going from 2, 100, 2 so instead of starting with 1, I start with 2, so when I start with 2, first time I do the calculation for 2, next time, that 2 is incremented by 2, so I will get 4, so in this way by changing the initial value of i, I am able to get even numbers instead of odd numbers.

So, one more point; in many of the programs we have studied so far, I am always saying do 10, 10 continue, what is this 10? 10 is not some object worth 10, 10 is just a line number, instead of do 100, i going from 1 to 2, I could have say do 103, i going from 1 to 2, so since this is 103, then this line number will be 103, instead of 103, I can say do 999, i going from 1, 100, 2, 999 continue, after all this is a line number.

And in programs, there is no requirement that all line numbers should be in an ascending order, the first line label could be 10, the next line label could be 3, the third line could be 22, so these line numbers are just for our reference but computer does not bother, we can give whatever line number we want but still it is a good practice that you keep all the line numbers increasing, so

that you know that once you come to a line number 100, may be you are somewhere in the middle of the program.

So, it is a good idea to call the last line as 9999, it is just a suggestion, so whenever you come to the; such a large number, you may think that it is the end of the program, okay. Now, we have consider do statements with a line number, there is another way to do this without line numbers because line numbers are good in many ways and sometimes, they are not good as well, so there is a way to do this set of operations without using line numbers.

So, what is the way; let us look at the next program, this program is really it calculate the sum of some numbers, it calculates the sum of the cubes of real numbers between 1 and 100 incremented by 2, so look at this line; sum = 0, do i= 1, 100, 2, there is no line number, so it is a do statement. The next line is sum = sum + real i raised to third power, so first time, i = 1; third power of 1.

Next time, this i has become 3, so sum is 1 cubed + 3 to the third power, next time i is 5, so the sum would be sum + 3 cubed + 5 cubed, so each time increment i/2, take the cube and add to the value of sum, the sum keeps on incrementing, so instead of this 10 continue, now I am saying end do; do something for a given sequence of numbers and end that do. So finally, when i = 99, next time, i will be 101.

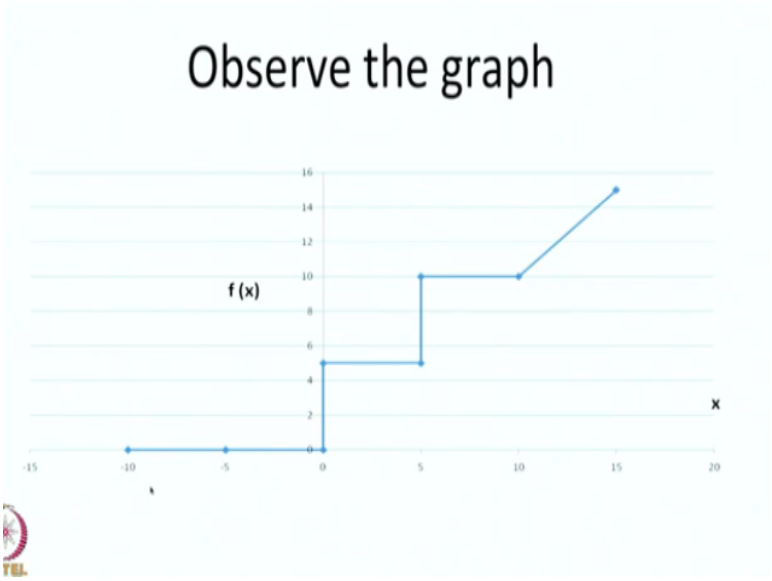
So, it knows that I will exceed the upper limit, so the entire do loop will end, the do loop will end when the variable i is > the second value in my loop statement. So, as we have seen already now, there are many, many ways of writing your do statements, so in something like a C program, they do not have a do, they have what is called a for loop. So, whatever is a do loop in fortran is a for loop in C.

So, different programming languages will have different labels for these operations but the spirit is the same, what is the spirit; you want to have a repeated set of operations, start that operation with the given number, keep on doing increment that number by something, keep on doing when

that number exceeds an upper limit, you end that do loop, okay and the other point computers treats real numbers and integers very, very differently.

Integers require a lot of space, real number do not require the same amount of space because in a real number, I can represent the decimal part by 8 digits and the exponent part I can keep it separately, suppose I have .1, if the exponent is 5 that is.1 reach to 10 to the 5, okay, so whereas integers are different, they need a lot more space, so integers are treated very, very differently, so I have discuss this do loop now.

(Refer Slide Time: 08:06)



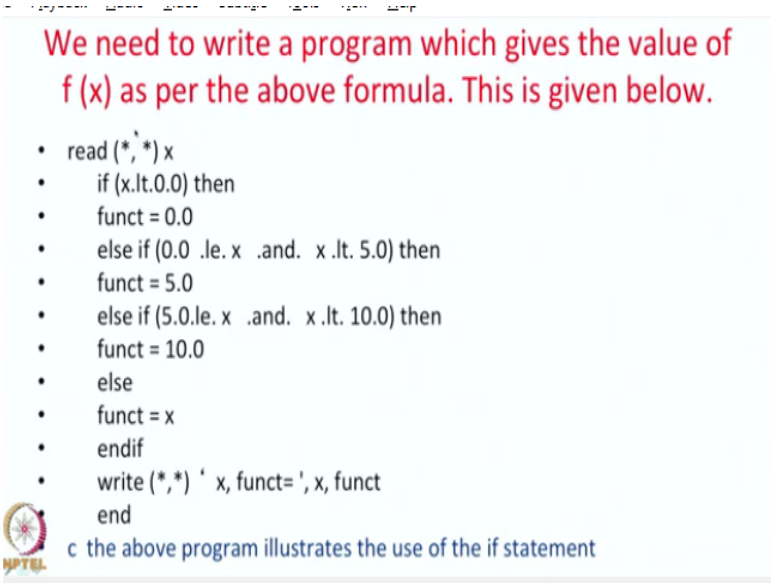
Next, what I will discuss is how to branch in a program, branching is a very crucial thing, okay, so we want to show the branching by plotting this function or by seeing the plot of this function, what is this function? This function has certain values for all negative values of x, the function value is 0, between 0 and 5, the function has a value of 5, between 5 and 10 the function has a value of 10.

Beyond 10, so 10 to 15, the function takes the same as the value of x, so if x is 13, function will be 13, if x is 15, function is 15 and beyond 15, I have not defined anything, so this function is defined from -10 up to 15, so -10 to 15 are all the values that the variable x takes and the value of the function goes from 0 up to 15, so the range is 0 to 15 and the x takes the values from -10 to +15.

So, how will I evaluate this function in a program; the program will read the value of x and if the value of x is let us say, -5. I know from the graph that the function of x will be 0. So, now if the value of the function, if the value of x is say 3, the function has a value of 5, if the value of x is a 7, the function value is 10 and beyond 10 to 15, it is a linear function that is whatever is the value of x, so is the value of fx.

So, my program should do that so, what will the program do; it will read the value of x and depending on what is the value of x, it will give you the value of fx, so naturally now there will be many conditions in the program, the moment I read x, I asked the question whether x is between 0 and 5, whether x is between 5 and 10, so I have to have many, many if statements to find out what the value of the function is.

(Refer Slide Time: 10:16)



So, this entire thing is illustration of the use of the if statement, so let us say now, this is my program, so what is this now? Program; we are writing a program which gives the value of function x as per the above formula, so now that is what we are giving here, so since the program starts again this read should start in the seventh column, it should not start in the first sixth column because then it will give an error.

All fortran statements will start from the seventh column, so what is this program doing; read *, * x, so what is this *, *? That means it reads from the screen, from the screen the computer value will read x, then you are asking the question if x is < 0, so this actually what it means, this is the condition now, the conditions start with this left bracket x. lt. 0. 0, bracket complete that means if x is < 0.0, then function = 0 that is exactly what the graph said.

If x is < 0, then function is 0, the next statement else if; now see if x is < 0, then function is 0 then there is nothing like else if. All else if will be wide, then it will go write x and function, this is in quotation mark, what is the meaning of quotation mark? This x and function x; function = is written on the screen. Again, write *, * means it will write on the screen, it will on the screen, it will write x, function = quotation x and function.

So, if x was; let us say -5, this function will be 0, so it will; if x is < 0 only these 2 will be evaluated and you go back and complete. Now, suppose x is > 0, then this else if comes into play, else if, okay, what is this statement? 0 < = x and x < = 5, so now this statement, it is a little more complicated than this, the first one had only one comparison, x < 0, this < 0 is got a relational operation.

What this lt does; it compare is whether the left side is < the right side, if true then it will go forward, if not then it will skip and go to the next line, so now this if statement is a combination of 2 statements, what are those 2 statements; 0 < = x and x < = 5; < 5, see there is also a difference < =; means, it could be less, it could be equal to, less than means it is actually, less than, this is an inequality, where the upper range can be equal.

So, when you come to this else if certainly, x is not < 0, since f is; x is not < 0, it will be > 0, if it is >0, it could be between 0 and 5 or it could be >5, so what this is doing; if x is > 0 and x is < 5, then the function takes the value 5, remember go back to this, what was this function, when x is between 0 and 5, it took the value 5, when x is between 5 and 10, it took the value of 10, okay, so I am executing this part of the program; x is > 0 and x is < 5, then x will be = 5.

So, this is the else if part, the second part of that statement, the third else if will be between 5 and 10, the fourth will be beyond 10, okay. So, now going back, so if x is between 0 and 5, it will give a value 5. Now, suppose this is also; this is satisfied, once this is satisfied, it goes to the end and writes the value of the function, if it is not satisfied at this level, then again it goes to the next line, else if 5 < = x and x < 10.

Again, there are 2 statements and is a combination and means the left has to be satisfied and the right has to be satisfied, so again it is .and., this is an operation, .lt. is an operation, .le. is an operation, there are 3 operations here. the first operation says that 5 is < = x that means, x is = or > 5, x is > 5 and the second one says, x < 10, so x is > = 5 and < 10, so it is in the third range, then the value of the function is 10.0.

So, once these 3 conditions, if this condition is satisfied, again it will go to end if and write the value of the function. Now, suppose x is > 10; if x is > 10, first condition is not satisfied, second is not satisfied, third is not satisfied, then I will go, so when I come to else, all the conditions are not satisfied because x is > 10, when x is > 10, we said function is = exactly x, it was a linear thing between x = 10 and x = 15.

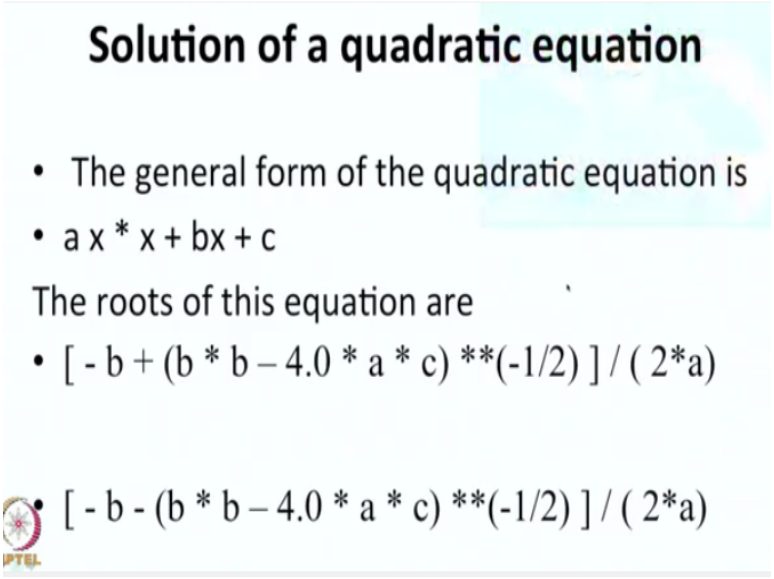
So, if x is 12, it will write the value of 12 and 12, so once all these if and else if are satisfied, then I am saying end if, so end if says the work of that if function is completed, okay, the if function is completed, when all those 4 are tested and it comes out, so now there are 2 problems with this program; one problem is what if x is -40 because I never told in my graph if you see, in my graph the value of x was between -10 and +15, okay.

Whereas in the program, there is nothing for it to recognise if x is say, -40, now suppose x was -40, what will it do; certainly x is < 0, if it is -40, then it will make function = 0 and end it, write and end. So, if x is < 0, it will give the value 0 for all values of x which are < 0. Similarly, all values of x, which are > 15, the function will be 15, so this program actually is actually calculates the function where this horizontal line is extended to all values of negative.

And this straight line is extended to all positive values beyond x = 10, so we wrote the program, it does quite alright for the range that we are considering but it also gives values for values of x which are outside the range because I did not give any condition. So, now I will leave as an exercise, suppose x is -40, then your program should say that function is not defined at -40 because my graph did not have any values for x < -10.

So, these additional lines you can add to make sure that x takes on only those values where you have defined on your particular graph.

(Refer Slide Time: 18:09)



So, the next task we want to do is now we will go to a slightly more complex problem now, we want to solve a quadratic equation, so you will say that look, this is something not very complicated I have already solved it but we want to solve a quadratic equation for all values of a, b and c, a quadrate; so general formula is ax square + bx + c, now if you write a program to solve, there are 2 roots of this equation.

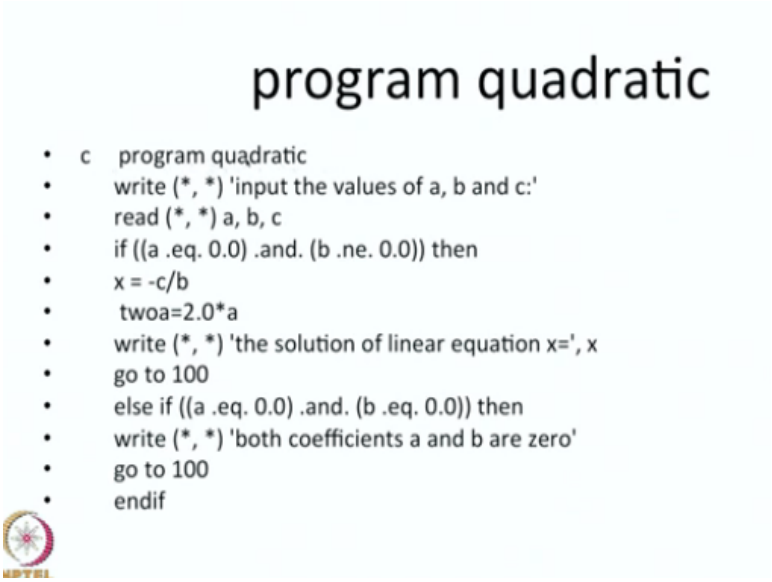
Once you write a program to solve this, you do not have to do any further calculations, you just input have the values of a, b and c, what you will get is a result of this calculation, which are the roots of the quadratic equation. So, what are the formulas for the roots? You know very well that the formula are -b + b square -4 ac - b + square root of b square -4 ac divided by 2a that is the first root.

Second root is -b - square root of b square -4 ac divided by 2a, so these are the 2 roots, now you will see that when I say square root, so this is b square -480; - 4 ac * * to the -1/2, there is again a problem here, this should not be minus because I just want to take the square root, if I do minus the whole thing will go in the denominator okay, so therefore this minus should not be minus, so it is –b + or - b square -4 ac divided by 2a.

Remember these bracket, the square bracket is here; the starting square bracket is here, all those things in this are part of these square brackets, this 2a in the denominator, the other thing you will notice, I have called it 2 * a both are put in brackets. Now, what will happen if these brackets are not there; if the brackets are not there, it will calculate this numerator, it will divide by 2 and it does not know that a is in the denominator.

It will multiply the numerator by a, so in a computer, you have to be very, very careful, it goes from left to right, so unless you put the brackets, it will think that a is not in the denominator, so make sure that you are divided by 2a, I am putting this bracket, so this is my formula for the quadratic solutions. So, now let us see how to write the program.

(Refer Slide Time: 20:52)



So let us begin that program, first line is a comment card program quadratic, okay, so these okay, these are starting line, it just gives you information that you are trying to solve a quadratic

equation, then on the screen you will write this, going back we did mention about compilation and execution. How will you compile? Suppose, the file name is prog.f, if you recollect the compilation is f77 prog.f or gfortran prog.f, this will give you an executable file a.out, the way you have executed; ./a.out.

Tomorrow, we will actually evaluate this, so once the program starts, you want to see on the screen, input the values of a, b and c, so the first line, what it is doing; *, * means write on the screen, whatever is written between these quotation marks, this is written on the screen, it says input the values of a, b and c. So, once you write this, you will see on the screen, input the values of a, b and c.

So, the third line of the program, it says read *, * a, b, c, this really is the input to your program, why do I say these the input? Because your program calculations will start only, when you give values of a, b and c. So, once you read a, b and c, so what I am doing here now, I am checking whether a = 0, okay. So, if a = 0, go back to this, if a = 0 okay, the first term is not there at all, so it is just bx = c and if b is != 0, the solution is x is = -c/b.

So, when I solve this, remember in this particular line, I am dividing by 2a and in a computer, you cannot divide by 0, if you divide by 0, it will be a big error, computer does not know what to do therefore, you have to make sure a is not 0 and that is why you have that statement, a is not 0 and if a is 0, solution is x = -c/b, if b is not 0 that is my first statement. If a != 0, then you can proceed to the next 2 lines.

So, this if statement again, it is the joint if statement, it does 2 things; if a is 0, so that means, it is not a quadratic equation, okay. So, the first part a. eq. 0 .0, if a = 0 and b != 0, okay, if both these are satisfied, it is not a quadratic equation then it is only a linear equation with a solution x = -c/b, okay, then 2a is 2a, okay, write the solution of a linear equation is x, correct, so if a = 0, okay, it calculates 2a, it writes the solution of a linear equation is x and go to 100, okay.

So, there is a slight problem in this statement, we will see tomorrow when we actually execute it. Now, I have consider the situation, where a = 0, now next I come to that statement, if a = 0, it

will do all these things, it is okay, I thought that there could be some error here, there is no problem, if a = 0 and b0 =0, it will do all these statements up to this one, go to 100. So, if a = 0, it will come up to this point and go to 100, it will come here and end if.

So, the last line should be 100 and some statement with respect to that so now, the next part is if a! = 0, okay, no, no so, let us see here, a = 0 and b = 0, both are = 0, then what happens when both are = 0; there is nothing to solve because go back, you will see that if a is 0 and b is 0, there is nothing to solve because then c has to be 0 because quadratic equation is ax square + bx + c = 0. So, if a and b = 0, there is no solution, okay, so go to 100, so all these lines will consider a = 0 and a and b = 0, so that if ends here, okay.

So, if either a and b was 0, it will go to the line 100, which is really the end of the program.

(Refer Slide Time: 25:47)

```
      ww = b * b - 4.0 * a * c
     if (ww .lt. 0.0) then
     go to 50
     else
     rtofww = sqrt (ww)
     root1 =( -b + rtofww) / twoa
     root2 =( -b - rtofww) / twoa
     write (*, *) 'real roots 1 and 2 are', root1, root2
     go to 100
     endif
 50  continue
c    the roots are complex b**2 - 4 * a * c is -ve
     ww = 4.0 * a * c - b * b
     rtofww = sqrt (ww)
     realpt = -b / twoa
     ximpt = rtofww / twoa
     write (*, *) 'complex roots'
     write (*, *) 'root1', 'real part=', realpt, 'imaginary part=', ximpt
      ximpt2 = -ximpt
      write (*, *) 'root2', 'real part=', realpt, 'imaginary part=', ximpt2
100  continue
     end
```

So you see this is continuing from the last screen, so 100 continue end, so the earlier things are executed, if a = 0. Now, if a != 0, then I come to this next line, so when I come here now, ww is your b square -4 ac, so I know that when I come to this ww b square -4 ac, a is not 0, so this is what is the discriminant, they call it whatever in the bracket, so now it is possible that this ww is = 0, if ww = 0, go to 50, so go to 50, then you will continue this and if ww is not 0, then you do this, okay.

So, if ww is < 0, okay then go to 50, so 50, it will do all these things, so what this is doing is that if ww is negative, you cannot take the square root of a negative number, since you cannot take the square root of a negative number, because they are imaginary, you will have a real part and an imaginary part, so this part will consider in the next class. Today, what we will consider is that ww is not < 0, then I have 2 real roots; 2 real roots I will take root of ww; rtofww, this is just a variable.

I have the habit of giving variable names, which tell you what it is doing, root of ww because I have a ww is a number, it is positive, so now I consider root of ww is the square root of ww because I need it to solve the quadratic equation, so first root would be – b + square root of b square – 4 ac, after all, what is www; ww; b square -4 ac, so I calculate b square -4 ac - b divided by 2a, this is my first root.

Now, what is the second root? - b - root of ww divided by 2a, this is the second root, so then what do I do? Write *, *, okay, real roots; real roots 1 and 2 are, so this will write on the screen, real roots 1 and 2 are root 1 and root 2, then go to 100, when you go to 100 that is the end of the program. See the last line although, I told you that you need not label lines in a sequence, giving large numbers to the end is a useful thing.

So, let me summarise what we have done, we started with the quadratic equation; ax square + bx + c, we consider the case where a is 0, in which case, it is a linear equation and the second case we consider is; when b square -4 ac is positive because if b square -4 ac is positive, I will have a real roots, I printed the real roots here. If b square -4 ac is < 0, I will have imaginary roots, so that I shall consider in the next class.

So to summarise what we have done, we consider different parts of the program especially, those which involve iterations, so different types of do statements we consider, do 10 i going from 1 to 100 or do i going from 1 to 100, we also considered how to increment by 2 units your do loop variable, then we consider the elementary quadratic equation, how to solve it, so far we have considered to solve it for only real roots.

For imaginary roots, we have to change a little bit that I will start in the next class, okay, thank you. Revise all these things and from next class onwards, we will start actually doing the computer work, we will actually calculate all these things on the computer, so that you can participate in the process, okay, thank you.