IIT Madras

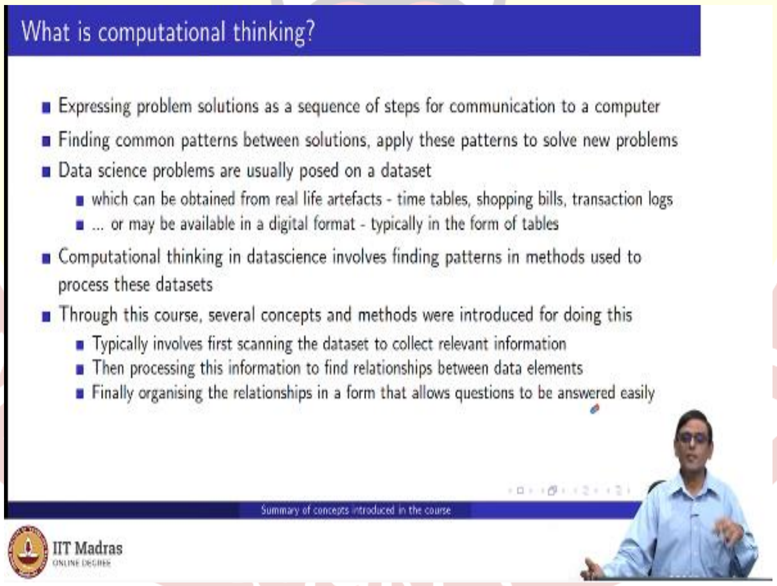
ONLINE DEGREE

Computational Thinking
Professor. G. Venkatesh
Indian Institute of Technology, Madras
Summary of Concepts Introduced in the Course

Welcome to this last and final lecture of the Computational Thinking course, what I want to do is quickly summarize all the concepts that we have learned so far that have been introduced in the course. So, as you go through the slides, and I hope to finish the slides reasonably quickly, you will see certain words highlighted, it means shown in dark, bold those was the keywords, those are the glossary, think about it like the glossary of the books, those are the words, those are concepts that we have introduced in the course.

So, we want a quick summary of just the words, by words you want remember the course, and the ones that are marked in bold are the ones that you should keep watch out for.

(Refer Slide Time: 0:57)



What is computational thinking?

- Expressing problem solutions as a sequence of steps for communication to a computer
- Finding common patterns between solutions, apply these patterns to solve new problems
- Data science problems are usually posed on a dataset
  - which can be obtained from real life artefacts - time tables, shopping bills, transaction logs
  - ... or may be available in a digital format - typically in the form of tables
- Computational thinking in datascience involves finding patterns in methods used to process these datasets
- Through this course, several concepts and methods were introduced for doing this
  - Typically involves first scanning the dataset to collect relevant information
  - Then processing this information to find relationships between data elements
  - Finally organising the relationships in a form that allows questions to be answered easily

Summary of concepts introduced in the course

IIT Madras
ONLINE DEGREE

So, we started this course by asking what is computational thinking? We said computational thinking basically is trying to express the solution to a problem as a sequence of steps. So, that we can communicate these steps to a computer, that basically the objective and then we said, let us try and see whether you can write these steps in a way that basically allows us to identify, write it in a correct way, you can identify patterns between different solutions.

And then, looking at these patterns, you can create codes, which basically can then be used for across multiple application that is basically if I have a new problem, completely new problem then I can say, this problem looks very much like that other problem that I saw, so, I can actually take the code from there and use it. And that basically, is enabled by computation.

Now, specifically in this program, we are looking at data science, we are not just thinking generally about the computer thinking we are thinking about computational, computation thinking in the context of data science. So, in data science, problems are usually post on data sets that are available, if you have a data set then you have some question about it. Now, these data set can come from real life artifacts, so first we say in this course we have basically a number of real life artifacts like shopping bills, we had, we had the train timetable, we had classroom data, we had words and so on.

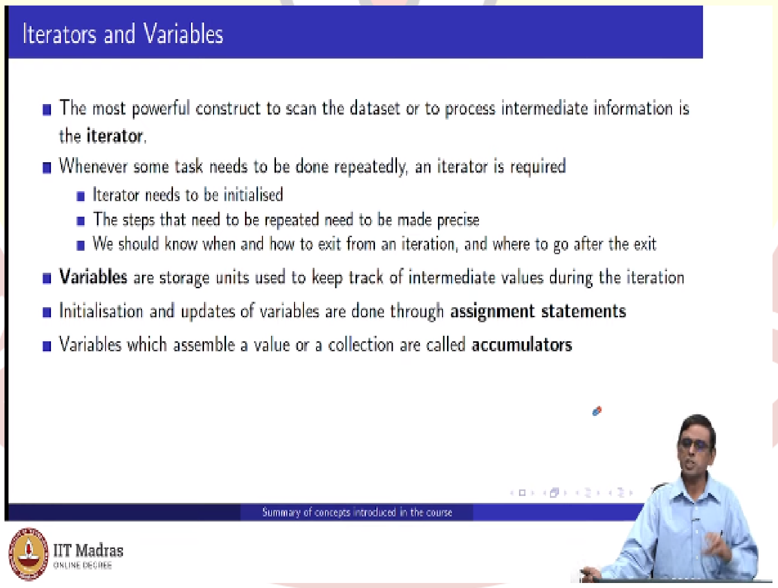
In principle you could also have transaction logs, could have many things like that, you could have, emails, you could have tax, all these kind of things these are all basically various kinds of real life artifacts, you can analyze you can like do lot of things or your data might be actually be processed and available in digital format, like in the form of table or maybe stored in a excel sheet and you may have it in the form of table, so you can use the table data also, so we have tried in this course to accommodate both by using cards of how you do things with real life data and drew tables and rows on the table and so on, you get an idea of how to do things with digital data.

Now, in datasets computational thinking involves finding patterns in methods used to process dataset, there are generic pattern, in fact actually there are many, many interesting computational pattern but we discuss many of them in this course. So, do not think this is a course on all of computational, it is not, it is just a peek into the world of computational thinking wearing the lens of data science, so you are basically entering data science and looking at it from a lens, from looking at computational aspects of data science.

So we were, we have introduced a number of concepts and methods that it helps, will help you, hopefully, when you finish your 3 years of whatever it is that you do in this program, you will probably come out with a good idea on how to process datasets that come out from various places. And when doing that, what kind of thinking you need to have in computationally processing them. So, we saw that basically there are some elementary steps because that basically is to take the dataset scan it and we saw that iterator is the way to scan we will come to it.

So, you scan the dataset and collect relevant information. The second is processing, which means that you will make a list additionally something like that and or a graph. Finally, you want to put all this together in some neat way. For example, you will write some object, make it concurrent all those kinds of things. So, first scan for relevant information, next is process to make relationships, which are basically into organize that we learned in the course as various things.

(Refer Slide Time: 4:25)



Iterators and Variables

- The most powerful construct to scan the dataset or to process intermediate information is the **iterator**.
- Whenever some task needs to be done repeatedly, an iterator is required
  - Iterator needs to be initialised
  - The steps that need to be repeated need to be made precise
  - We should know when and how to exit from an iteration, and where to go after the exit
- **Variables** are storage units used to keep track of intermediate values during the iteration
- Initialisation and updates of variables are done through **assignment statements**
- Variables which assemble a value or a collection are called **accumulators**

Summary of concepts introduced in the course

IIT Madras
ONLINE DEGREE

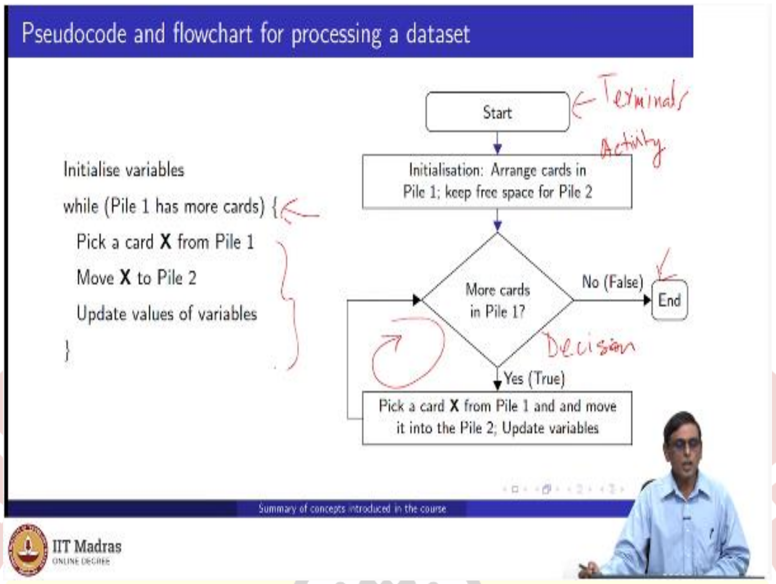
Now, the most powerful construct that we saw, that can be used to scan or process intermediate information is iterator. Iterator is basically something that needs to be initialized and you have to say what is it that is iterating over, what is the repeat step, what is it repeating again and again, and most importantly you know how to exit we saw that, it is not that easy to figure out how to exit, but you can write the exit by writing the while condition carefully, by adding to the condition and exit.

Sometimes in the middle of the iterator you can put an exit loop when you count, that is also possible, so you should know how to exit from a iterator and not only that, when you exit from the iteration where should you go next, you should know that, so that is another thing that is important to know.

Now, while iterating you may have to hold values and these are basically variables are the storage units that are used to hold values that are used, computer, produced with iteration, you are iterating over the data set like bunch of cards, rows of the table or a list, elements of list or something like that and while you are doing that you are keeping track of something and that keeping track is done using variables.

Now, we have to initialize the variables and then you have to update the variables both initialization, update, operations are done by using what are called assignments statements, and when you have certain kinds of variables, which are just accumulating something, like it is accumulating the sum of something, total of something, or it is basically accumulating a list, making a list, these variables are called accumulator and accumulation operations look very similar. which is why you want to separate them out as a pattern, accumulator pattern.

(Refer Slide Time: 5:58)



Pseudocode and flowchart for processing a dataset

```
Initialise variables
while (Pile 1 has more cards) {
  Pick a card X from Pile 1
  Move X to Pile 2
  Update values of variables
}
```

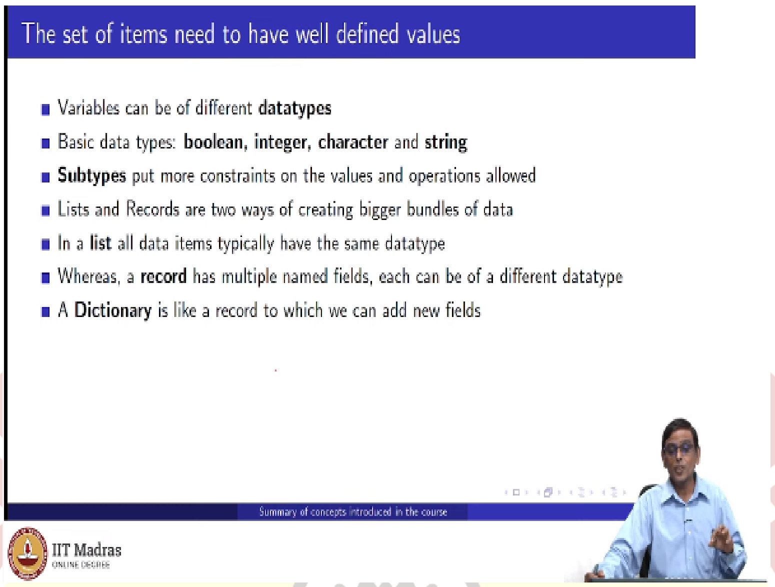
Start
Terminals
Activity
Initialisation: Arrange cards in Pile 1; keep free space for Pile 2
More cards in Pile 1?
No (False)
End
Decision
Yes (True)
Pick a card X from Pile 1 and and move it into the Pile 2; Update variables
Summary of concepts introduced in the course
IIT Madras
ONLINE DEGREE

These are two kinds of ways to write this, one is a flowchart we did not talk much about flowchart after some time because after some time drawing these kind of diagrams is very hard, but we saw that a flowchart was as useful skill to depict as a pictorial, when it is a small thing, it is always good to write a flowchart, easier to see. So, we saw basically, there is this symbols which are called the terminal symbol, Start and End are terminal symbols.

Then we saw the square boxes, which are the processes or activities, activity or process. And then there is this decision box, this diamond. So, you can have diamonds, which are basically decision boxes and then you can connect the boxes using these arrows and that is how you make flowchart. And the same thing can be also written in pseudocode. So, the pseudocode, the most important thing we saw was a while loop, while loop basically allows you to do iteration.

So, while Pile 1 has more cards, do something, something. So, this is an iterator, this pattern over here which is called a flowchart. So a group in the flowchart basically is an iterate, this is an iterator pattern in the flowchart and this is an iterator pattern in the pseudocode.

(Refer Slide Time: 7:11)



The set of items need to have well defined values

- Variables can be of different **datatypes**
- Basic data types: **boolean, integer, character** and **string**
- **Subtypes** put more constraints on the values and operations allowed
- Lists and Records are two ways of creating bigger bundles of data
- In a **list** all data items typically have the same datatype
- Whereas, a **record** has multiple named fields, each can be of a different datatype
- A **Dictionary** is like a record to which we can add new fields

Summary of concepts introduced in the course

Now, we also saw basically that the items that we are collecting, the variable, the data that we are processing and collecting the variables can have different data types. So, basic type, basically I talk about the Boolean, integer, character and then we had this interesting type called string, it is like a list of characters but then we wanted to, then we said that we can make subtypes to put constraints on the range of values. So, we do not want to allow negative values, we do not want to allow we can put constraints and operations zone, we can say this operation allow, does not allow,

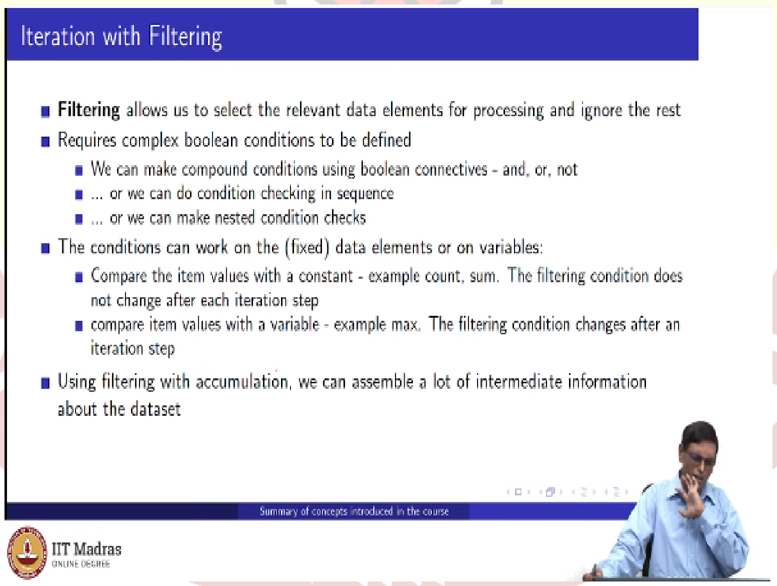
Then we said we can make compound types like this, when it lists out of things, we can make record out of things, the list is typically a sequence of things with the same data type and record is a set of things with different data type. But because they are all different data types, you need to name each thing, at least you can identify by its position whereas, in a record you cannot identify by its position, you have to identify by a name, so you have names in record.

Now, later on, we introduced this new crazy thing called a dictionary, list we saw, list is fine, list is okay, introduced later on thing called a dictionary, some of you may be wondering what is record, what is this dictionary, are they the same or are they different, it is not, the difference really is that in a record is fixed all the things that are given to you, when the data type is defined, the record is defined, so all the fields are defined, so you cannot do anything.

Whereas a dictionary, you can keep adding fields, you can delete the fields you can do all the things. So, dictionary is like a record in the sense that you can do x dot a, in a record you can do x dot a, in a dictionary also you can do x along with the and put in square bracket and everything, dictionary is like a record but the difference between dictionary and record is that in dictionary you keep adding new fields, any number of fields you can add to the dictionary, so that is basically dictionary.

And it also looks very much like a list in some sense, random access, it looks like a matrix and all that so that is why we separated out and we had special treatment given for dictionary, so it is a very important thing that we would keep using again and again throughout the entire programming language you will.

(Refer Slide Time: 9:12)



Iteration with Filtering

- **Filtering** allows us to select the relevant data elements for processing and ignore the rest
- Requires complex boolean conditions to be defined
  - We can make compound conditions using boolean connectives - and, or, not
  - ... or we can do condition checking in sequence
  - ... or we can make nested condition checks
- The conditions can work on the (fixed) data elements or on variables:
  - Compare the item values with a constant - example count, sum. The filtering condition does not change after each iteration step
  - compare item values with a variable - example max. The filtering condition changes after an iteration step
- Using filtering with accumulation, we can assemble a lot of intermediate information about the dataset

Summary of concepts introduced in the course

IIT Madras
ONLINE DEGREE

When we saw that there is this notion of filtering, filtering basically means writing a condition which basically can select data elements of the things that we want to process, we want to ignore long, long data elements only selecting that are interesting to us, you can put a filter and that filter speaker is written by using conditions. Now these conditions can be simple conditions, Boolean condition, they can be compound conditions and that can be made by using and, or, not and all simple conditions.

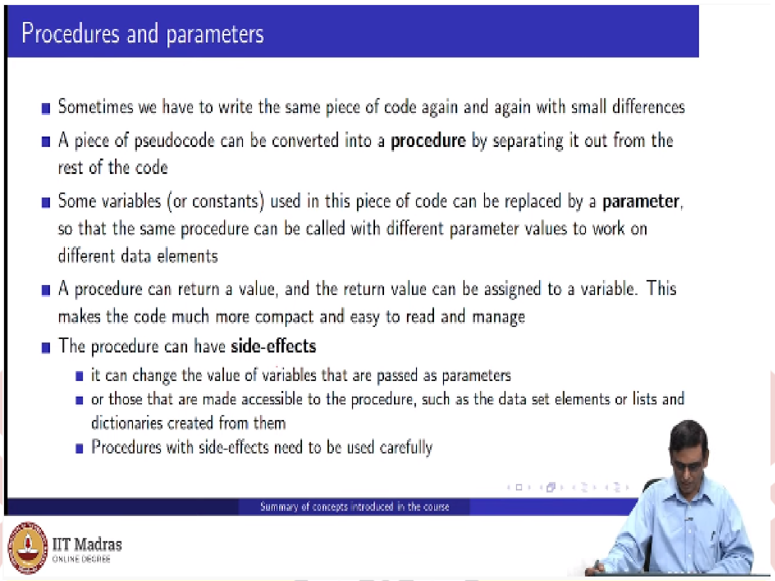
Or you can write conditions one after the other sequentially, you write one condition and do something, write another condition and do something else, so if something do, then again follow it with another if something, do something or even you can nest it so you can say if A, then open bracket if B open bracket then do something, so you can nest the condition, you can sequence the conditions they are not the same, they are sometimes the same, sometimes not the same, you have seen all these examples.

Hopefully by now, you figured out all these little tricks that we ask in terms of questions for exams. But you know, you have to understand that these conditions can be written in many way, some are equivalent, some are not equivalent and the programmers may choose to use one of these based on whatever they like it. Usually you can do it in one way, you can do it in another way also, but people have a preference for one way over the other.

Now, the filtering condition can be working on fixed data constants or it can work on variable. If it is constant, then the filtering condition is not changing with every iteration step. So, we saw for example, accumulation of count and sum and so on, filtering condition typically does not change, typically in accumulation the filtering condition does not change, whereas, if you are doing something like max, finding the max or something your filtering condition is comparing the value with the current variable value back. So, that is comparing with the variable so, in this case, the basically your iteration, filtering conditions keep changing as you proceed.

Accumulation with filtering give you plenty of power. Typically, with just iteration and filtering, you can do 90 percent of the task, most of the things you can do, so accumulation with filtering powerful construct that allows you to collect all kinds of interesting data, information from your dataset.

(Refer Slide Time: 11:30)



Procedures and parameters

- Sometimes we have to write the same piece of code again and again with small differences
- A piece of pseudocode can be converted into a **procedure** by separating it out from the rest of the code
- Some variables (or constants) used in this piece of code can be replaced by a **parameter**, so that the same procedure can be called with different parameter values to work on different data elements
- A procedure can return a value, and the return value can be assigned to a variable. This makes the code much more compact and easy to read and manage
- The procedure can have **side-effects**
  - it can change the value of variables that are passed as parameters
  - or those that are made accessible to the procedure, such as the data set elements or lists and dictionaries created from them
  - Procedures with side-effects need to be used carefully

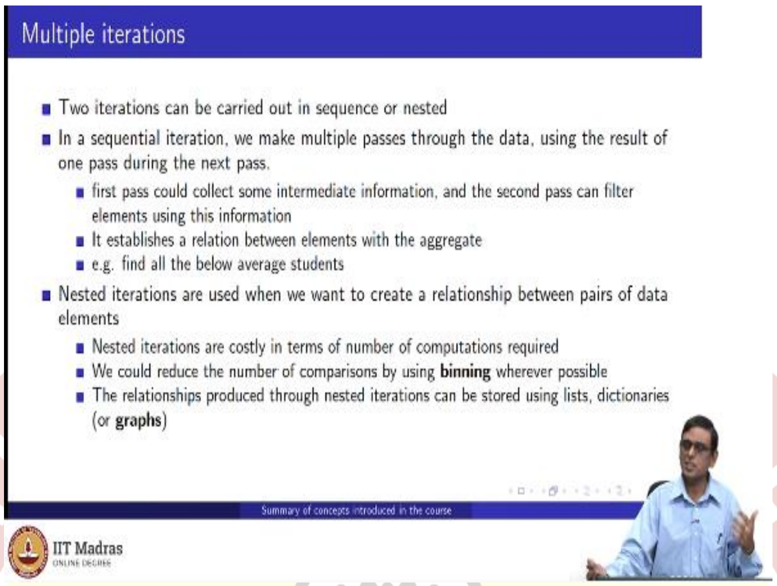
Summary of concepts introduced in the course

IIT Madras
ONLINE DEGREE

Now, once you have collected, done all this many things, you may want to encapsulate whatever code you have developed in terms procedure, parameters so that it can be reuse somewhere else, we have seen basically sometimes it is necessary for us to find kind of pattern of code. so you will write it off, you will separate out the code that you have written in the form of a procedure and give it the right parameters, then, that facilitates reuse of the same code many places.

So, we saw examples of it in average, for physics, added for maths, added for chemistry they are all the same, so we can make a procedure out of it. So, the procedure also return a value and the return value can be assigned to a variable this is a very, very compact cute way of writing code, writing X is equal to P of A, then A to P get the result and show it in a, all written in one line, beautiful compact way of writing code. Then we saw that procedures can have side effects, it can change the values of the parameters, it can change other variables and so on, so the side effects you need to manage these procedure extremely carefully because if you call a procedure and it does something to some other variable which is not even passed to procedure. you do not know what is going on.

(Refer Slide Time: 12:42)



Multiple iterations

- Two iterations can be carried out in sequence or nested
- In a sequential iteration, we make multiple passes through the data, using the result of one pass during the next pass.
  - first pass could collect some intermediate information, and the second pass can filter elements using this information
  - It establishes a relation between elements with the aggregate
  - e.g. find all the below average students
- Nested iterations are used when we want to create a relationship between pairs of data elements
  - Nested iterations are costly in terms of number of computations required
  - We could reduce the number of comparisons by using **binning** wherever possible
  - The relationships produced through nested iterations can be stored using lists, dictionaries (or **graphs**)

Summary of concepts introduced in the course

IIT Madras
ONLINE DEGREE

Now, iterations can be not single they can be multiple, how can they be, how can you create multiple iterations? Two iterations, for example, can be carried out one after the other, or one within another, and they are different you know that they are different, so you have studied by now, you have done enough problems to learn that these are very different things.

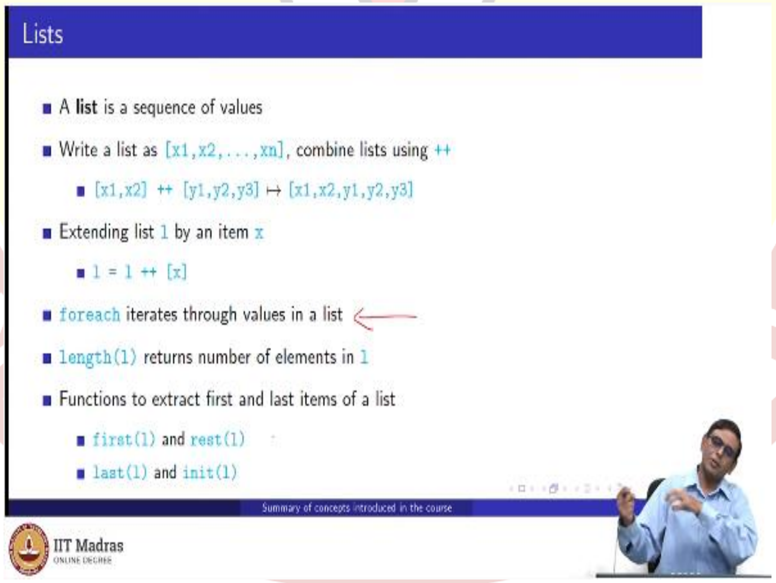
So, when you do sequence integration, that is one after the other, you are processing one pass is processing the data to create an intermediate information. The second pass is using this intermediate information to do something. So, usually you are finding the relationship between an element and the aggregate. For example, you can first pass find an average and the second pass you can find to collect all the students who are above average.

So, the first pass is basically accumulating some finding average, second pass is basically using that average to accumulate a list of students above average, this is what you doing. So, both filtering accumulation only but you are doing one after the other, and you are using the result of the first and the second. So, this is an example of sequential iteration. Whereas in nested iteration, you are trying to find the relationship between pair of data elements.

So, whenever you try to find a pair of data elements you have to nest one iteration between another iteration, and you are doing therefore n square, there are n people, n elements when you are comparing n element with n elements, so being n square operations is very costly. So, you could try and say that I will not compare everybody with everybody, I will try and compare only with those who are relevant to compare, that is called binning, and whenever possible, you should do that because otherwise n square can turn out to be very bad,

Specially with the kind of data set we learn to deal with, they are going to be large like a 100,000 data elements if you try to do n square 100,000 into 100,000 it is very large. So, we have to be careful about. Then we said basically that we could produce the relationship and store those relationships, how do you store the relationship? Typically in the form of list, maybe some time dictionaries, then we learn this beautiful thing called graph.

(Refer Slide Time: 14:44)



Lists

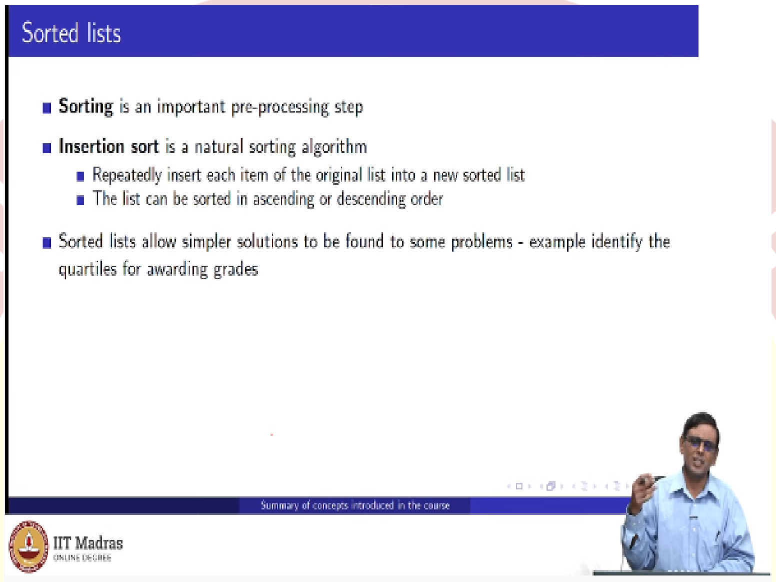
- A **list** is a sequence of values
- Write a list as `[x1,x2,...,xn]`, combine lists using `++`
  - `[x1,x2] ++ [y1,y2,y3]` ↦ `[x1,x2,y1,y2,y3]`
- Extending list `l` by an item `x`
  - `l = l ++ [x]`
- `foreach` iterates through values in a list
- `length(l)` returns number of elements in `l`
- Functions to extract first and last items of a list
  - `first(l)` and `rest(l)`
  - `last(l)` and `init(l)`

Summary of concepts introduced in the course

What is a list? List is just a sequence of values, you can append lists, you can add an element to a list, you can iterate on it. So, this is most interesting. So, we saw that we create it not only did we create this object called list, we created an iterator for the list, for each which will basically go through the list and we have ways of disassembling the list, you can take the list and break it up, how do you break it up?

You can find out the first element from the rest of the list, or you can take out the last element and the beginning of the list, we can take out the beginning of the list in the last the last element or first of the of the list in the list in the last element, we can break the list up and do things, so this is another interesting thing we have.

(Refer Slide Time: 15:23)



Sorted lists

- **Sorting** is an important pre-processing step
- **Insertion sort** is a natural sorting algorithm
  - Repeatedly insert each item of the original list into a new sorted list
  - The list can be sorted in ascending or descending order
- Sorted lists allow simpler solutions to be found to some problems - example identify the quartiles for awarding grades

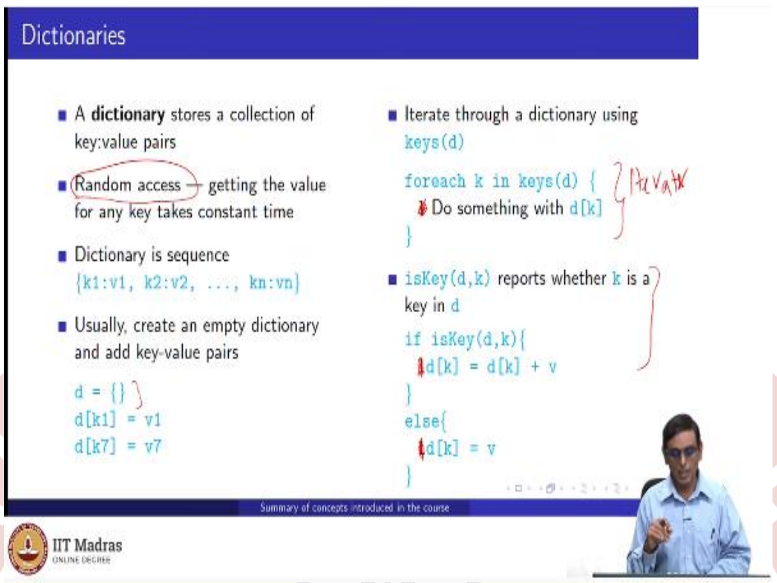
Summary of concepts introduced in the course

IIT Madras
ONLINE DEGREE

Then we said that basically list which are sorted in ascending or descending order are extremely useful to solve many problems. How do you sort a list? You can, simplest way is insertion sort, there are other ways of sort but we are not going to discuss in this course, there are algorithm course but this one just introduced one is called insertion sort, which is simply repeatedly inserting an element at the right place.

Now, when you sort the list, then you can do many things. Like for example, if you want award grades, just have to take the list divided into 4 parts and give it away very, very simple, okay, if it is sorted, if not sorted it is not possible. So, it is easy to do things in a sorted.

(Refer Slide Time: 16:02)



Dictionaries

- A **dictionary** stores a collection of key:value pairs
- Random access → getting the value for any key takes constant time
- Dictionary is sequence `{k1:v1, k2:v2, ..., kn:vn}`
- Usually, create an empty dictionary and add key-value pairs

```
d = {}
d[k1] = v1
d[k7] = v7
```

- Iterate through a dictionary using `keys(d)`

```
foreach k in keys(d) {
  Do something with d[k]
}
```

[illegible]

- `isKey(d,k)` reports whether `k` is a key in `d`

```
if isKey(d,k){
  d[k] = d[k] + v
}
else{
  d[k] = v
}
```

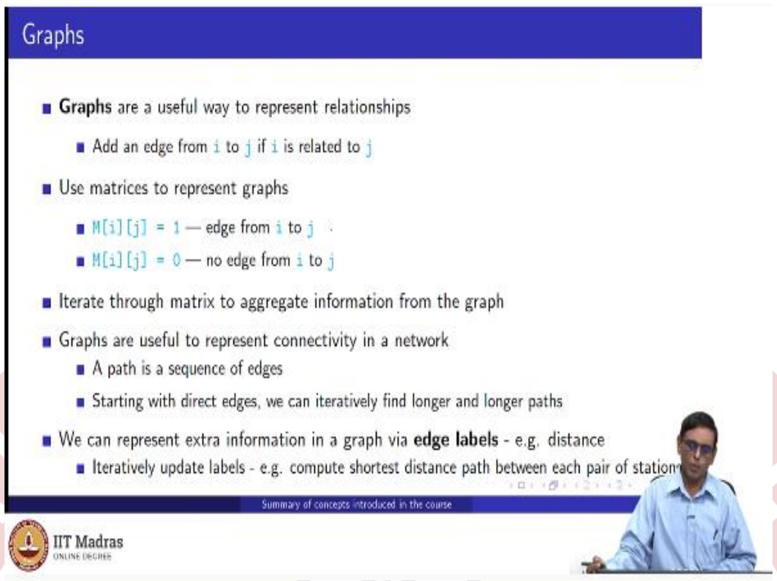
Summary of concepts introduced in the course

IIT Madras
ONLINE DEGREE

Then we saw this thing called a dictionary, which is basically a set of key value pairs, you can insert a key into dictionary, you can iterate over a dictionary, so for each key, this is an iterate, this is an iterator for dictionary is one got item, I made a mistake yesterday but, so this is an iterator, I can check whether the dictionary has a key by using this key and the basic template basically is to check whether the key is in the dictionary or not. If it is the dictionary, then you can do something, not in the dictionary, you can do something like for example, you can create a key, put the key in the dictionary and so on.

This is a simple for empty dictionary, so dictionaries are very, very useful thing because you random access it, is much faster than accessing an element in a list, finding an element in dictionary is much, much faster than finding an element in a list, which is why you should use dictionaries wherever it make sense to you, sometimes it makes sense to use list.

(Refer Slide Time: 17:07)



Graphs

- **Graphs** are a useful way to represent relationships
  - Add an edge from i to j if i is related to j
- Use matrices to represent graphs
  - M[i][j] = 1 — edge from i to j
  - M[i][j] = 0 — no edge from i to j
- Iterate through matrix to aggregate information from the graph
- Graphs are useful to represent connectivity in a network
  - A path is a sequence of edges
  - Starting with direct edges, we can iteratively find longer and longer paths
- We can represent extra information in a graph via **edge labels** - e.g. distance
  - Iteratively update labels - e.g. compute shortest distance path between each pair of station

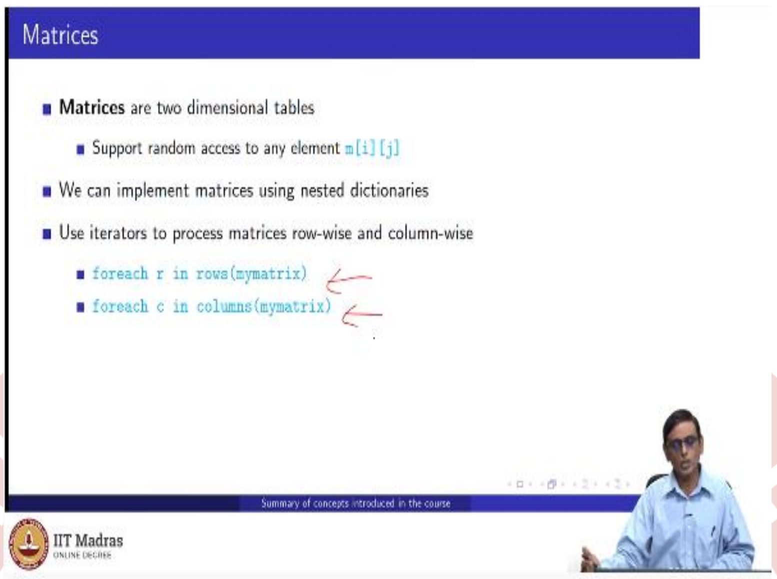
Summary of concepts introduced in the course

IIT Madras
ONLINE DEGREE

Now, we saw another type of interesting thing called a graph, whenever you have a relationship where you want to show that i is related to j, j is related to k, k is not related to i you draw a edges, because if i and j are related, j to k you draw an edge from j to k, k is not related i, so no edge will be there. So, the matrix, you can write a matrix because adjacency matrix M i, j you set it to 1, if there is an edge, you set it to 0 if there is no edge.

So, when you draw the matrix, full matrix, you look at it, the entire graph is captured inside this matrix nicely, it is a compact otherwise, you have to draw a graph, it takes a long time and matrix is much easier to see and do things with so, so sometime you do things matrix. So, you can iterate through a matrix to get aggregate information from a graph. For example, you can find out the path in the graph which is sequence of edges, you can basically find longer and longer paths. And if you label the edges, you can use it for example, to find the shorter distance, things like this you can do in.

(Refer Slide Time: 18:09)



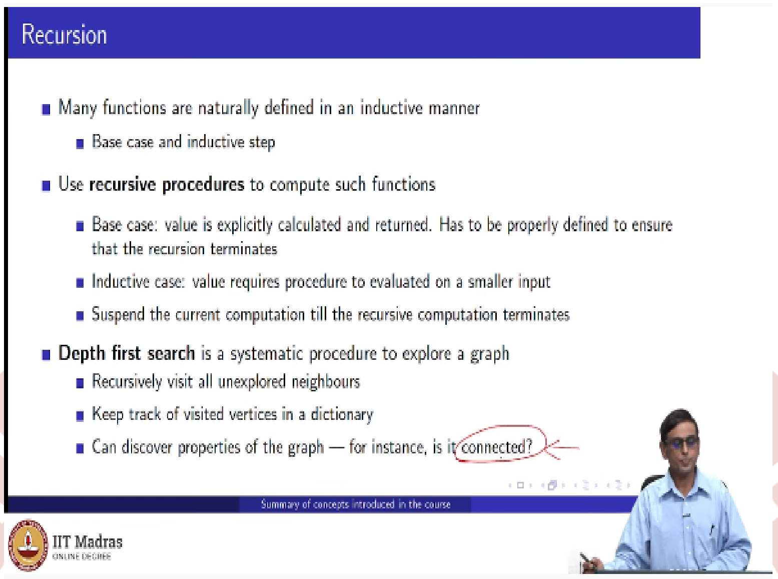
Matrices

- **Matrices** are two dimensional tables
  - Support random access to any element `m[i][j]`
- We can implement matrices using nested dictionaries
- Use iterators to process matrices row-wise and column-wise
  - `foreach r in rows(mymatrix)`
  - `foreach c in columns(mymatrix)`

Then as a sub product, as a kind of a byproduct of doing graph, we basically brought out the idea of matrices which allows you to access an element i comma j randomly. So, we can find i, j quickly, m i, j and there are direct implementations of matrices in many languages, like most languages that you have seen there is very direct implementation of matrices, you would have seen it.

But they have deliberately chosen to introduced matrix only from the perspective of data science in this way, after we have talked about lists and dictionaries, because otherwise, we get into the lazy habit of using matrix only, so we do not want to do it. So, we can implement matrices using nested dictionaries. So, that allows us to use with iterators for each row in the matrix, for each column in the matrix. So, we can nest for each row in a matrix followed by for each column in a matrix or each column in a matrix followed by an each row in matrix, you can do this nested iteration and process the entire matrix.

(Refer Slide Time: 19:09)



Recursion

- Many functions are naturally defined in an inductive manner
  - Base case and inductive step
- Use **recursive procedures** to compute such functions
  - Base case: value is explicitly calculated and returned. Has to be properly defined to ensure that the recursion terminates
  - Inductive case: value requires procedure to evaluated on a smaller input
  - Suspend the current computation till the recursive computation terminates
- **Depth first search** is a systematic procedure to explore a graph
  - Recursively visit all unexplored neighbours
  - Keep track of visited vertices in a dictionary
  - Can discover properties of the graph — for instance, is it connected?

Then we move on to the more advanced concepts in computational thinking, the first most advanced concept we discuss was recursion. Now, why do we need recursion? You can say I do not need recursion, I am happy with iteration, iteration is very powerful, problem is iteration works on sub sets which are kind of linear, for a list it make sense, for a deck of cards it makes sense because it is a deck, I can go one after the another, some structure are not linear, so in that case it becomes harder, like a graph, how do I iterate over a graph? Very difficult, how do I iterate over a tree, extremely difficult.

So, for these kinds of structures, the more natural thing to do is use recursion. Now you will find basically as many of these structures which are graphs and trees and also defined inductively which means, you say that you make a tree by starting with a leaf node, taking two leaf nodes and connecting them to another node like that you make a tree, similarly a graph is a set of nodes which are connected by edges we can talk about that.

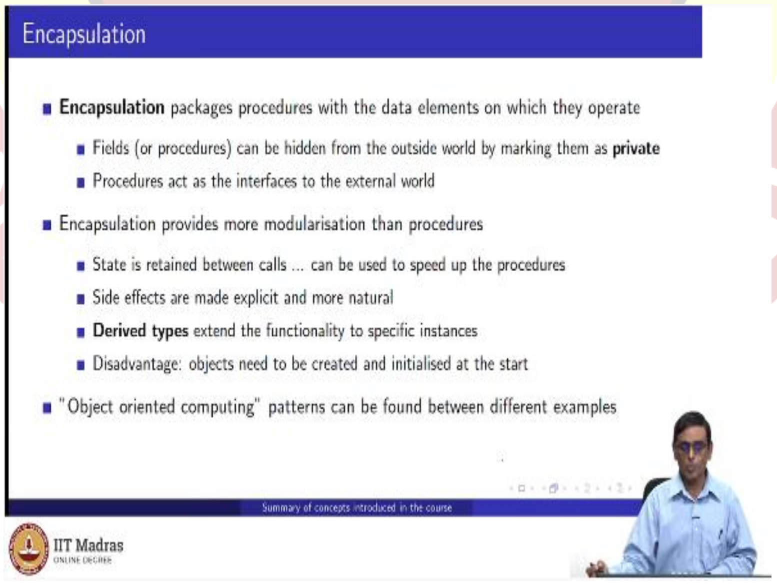
So, recursive procedure basically take functions which are inductive or structure with inductive go through them systematically, how does it go through it systematically? Firstly, it identifies what is the base case, in the case of our tree, for example, the leaf would be a base case, in the case of integers, 0 might be the base case, for a list an empty list might be base case, if you have to define exactly what you should do in the base case.

If you do not take care of base cases, all the bases, your recursion may not terminate, this is very, very important so keep in mind, do not be lazy, when you do recursion, think about your recursion, think about the base cases, think about what are the possible base cases that will come up with your recursion and handle all the base cases correctly, otherwise the recursion may not terminate.

Then comes the inductive case where basically you take the input and reduce it to a smaller input, the smaller input it eventually becomes a base case and then you will address it there, and then when you find that you have broken it down into smaller inputs, you call the smaller, you call the recursion, call the same function with small input and you wait for it to complete.

A simple example that we saw for recursion is that depth first search, it is a systematic way to explore a graph, you keep track of all the visited vertices in dictionary, so a C in dictionary or something like that and you can discover all the properties of a graph, for example, you can check whether the graph is connected in beautiful ways using depth first search.

(Refer Slide Time: 21:31)



Encapsulation

- **Encapsulation** packages procedures with the data elements on which they operate
  - Fields (or procedures) can be hidden from the outside world by marking them as **private**
  - Procedures act as the interfaces to the external world
- Encapsulation provides more modularisation than procedures
  - State is retained between calls ... can be used to speed up the procedures
  - Side effects are made explicit and more natural
  - **Derived types** extend the functionality to specific instances
  - Disadvantage: objects need to be created and initialised at the start
- "Object oriented computing" patterns can be found between different examples

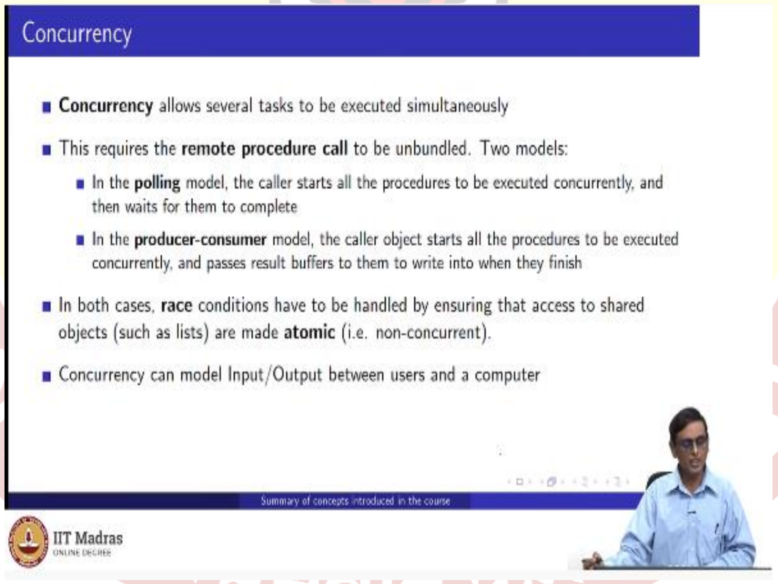
Summary of concepts introduced in the course

IIT Madras
ONLINE DEGREE

We then move on to the second advanced concept which is encapsulation, there we said make sense sometimes to package both the fields and the procedures into one packet which is called object. And what are the advantages? The advantages basically are that the state retains between two procedure calls. So, you can use that to speed up the procedures, side effects are made explicit, in fact, more natural sometimes good to have side effect into object oriented programming, instead of making it bad in procedures, bad in object orient programming side effects are good.

Then we said we can have derived types which will extend the functionality of an object for more specific example we saw in the case of shirts and in the case of. The disadvantage with object oriented programming is that you have to make a lot of objects and initialized, so that takes time and also efforts and also you can try to find patterns which not only involve code, but also it was objects and code together, we saw that of course, that is a interesting thing that we can do in object orient computing, find object oriented pattern.

(Refer Slide Time: 22:32)



Concurrency

- **Concurrency** allows several tasks to be executed simultaneously
- This requires the **remote procedure call** to be unbundled. Two models:
  - In the **polling** model, the caller starts all the procedures to be executed concurrently, and then waits for them to complete
  - In the **producer-consumer** model, the caller object starts all the procedures to be executed concurrently, and passes result buffers to them to write into when they finish
- In both cases, **race** conditions have to be handled by ensuring that access to shared objects (such as lists) are made **atomic** (i.e. non-concurrent).
- Concurrency can model Input/Output between users and a computer

Summary of concepts introduced in the course

IIT Madras
ONLINE DEGREE

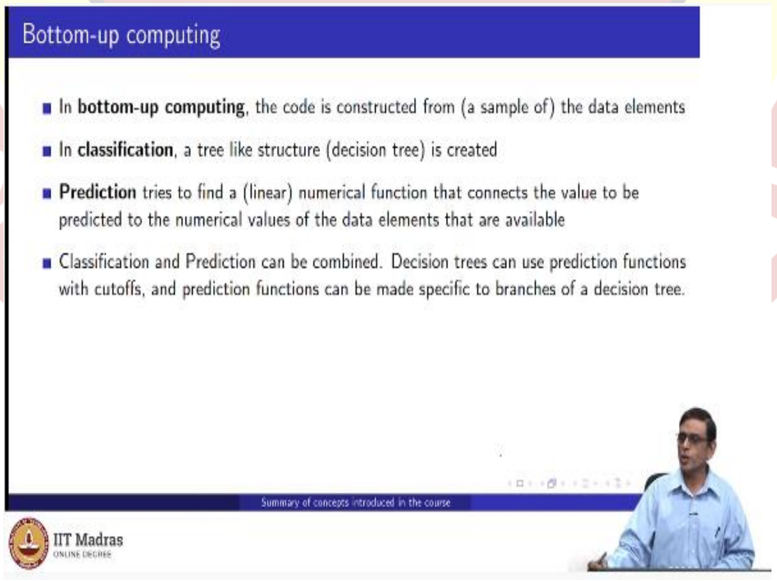
Then we move on to concurrency, very hard topic to understand. Hopefully, you will keep on coming across more and more and as you absorb more and more of this concurrency thing you will get to, tips with how to deal with this, basically it means you are running many things together and when you do that lot of can go wrong, can go right. But basically what we are doing is we are taking a remote procedure call and unbundling it.

Two ways of unbundling it we did, one is polling, and other one is producer consumer, in polling the caller starts the procedure and waits for the procedure to complete, the procedure then basically tell it the object which is called basically tell it is complete and it tells it, the caller the result.

In the producer consumer model, the producer sends a buffer to the consumer or the consumer sends the buffer to the producer, producer when it finishes it writes the result in the buffer meanwhile the consumer can be doing other things, when the consumer is done it can delete the value from the buffer and move on. This is how producer consumer model works. In both cases there are race conditions, these race conditions can cause corruption of various trade objects like list and sort, so we have to ensure basically that shared object are made atomic or non concurrent.

So, there is no concurrency inside, concurrency outside that is so we have to make it work. Finally, input output between a user and a computer for example, actually, you concurrent network, you have to understand concurrency if you have to understand input output.

(Refer Slide Time: 23:52)



Bottom-up computing

- In **bottom-up computing**, the code is constructed from (a sample of) the data elements
- In **classification**, a tree like structure (decision tree) is created
- **Prediction** tries to find a (linear) numerical function that connects the value to be predicted to the numerical values of the data elements that are available
- Classification and Prediction can be combined. Decision trees can use prediction functions with cutoffs, and prediction functions can be made specific to branches of a decision tree.

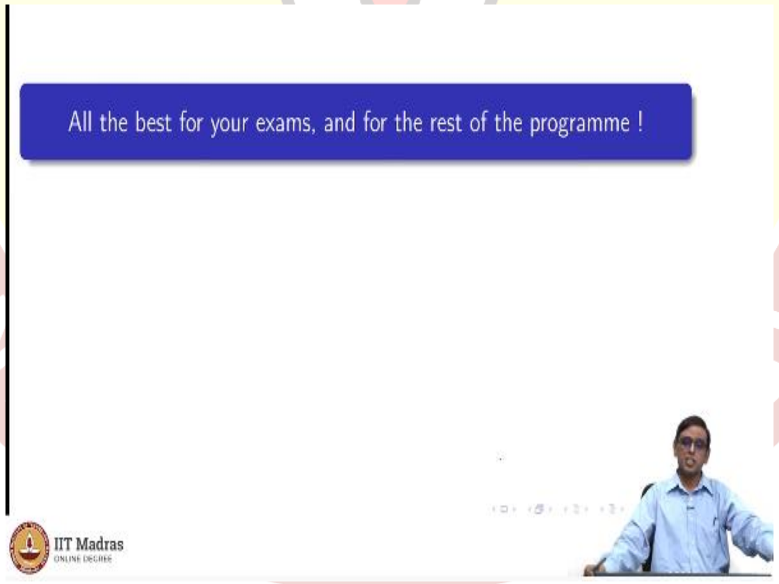
Summary of concepts introduced in the course

IIT Madras
ONLINE DEGREE

Finally, we looked at bottom up computing, which is a completely different paradigm. The reason why we wanted to touch on bottom up computing in a computational thinking course is all of it in this program. As you go forward, you will see more courses which are going to come machine learning and this and that and so on and all of them are bottom up they are top down, and you will of course, see there are two screens that you have in this program, one is this programming screen which you will have all top down and you have this machine learning and kind of thing which are all bottom up. So you see both screen are in parallel.

So, we wanted to give flavor about the bottom up computing is, we saw that there are two kinds of problems, the classification problem where you try to label prediction every element by using a decision tree, and a prediction problem where you try to make a numerical function which can predict the value of unknown thing, unknown values from a known value. Both of them can be combined, you can make a decision tree where branches are taken based on prediction functions, you can make a decision tree and then leaves of it can be prediction function so that is all about it, both can be combined.

(Refer Slide Time: 24:56)


All the best for your exams, and for the rest of the programme !

With that we come to the end of the computational thinking course. I hope you enjoyed all these lectures and material that we put out for you. Madhavan and I really, we can share with you that we really enjoyed putting all these. seeing all your comments and interactions and so on we have enjoyed it and I hope you enjoyed it as much as we did, wishing you all the very best for your exam and more importantly for the rest of this program which is going to be I can bet much more exciting (())(25:28) it will get more and more exciting, all the best. Thank you.