

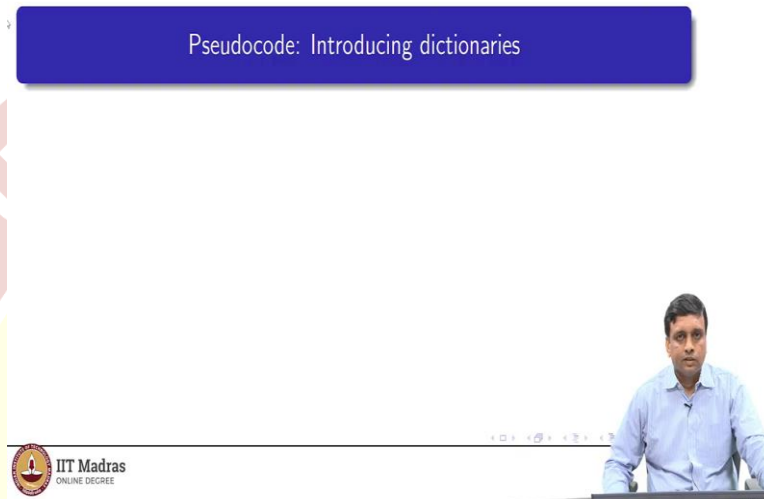


# IIT Madras

ONLINE DEGREE

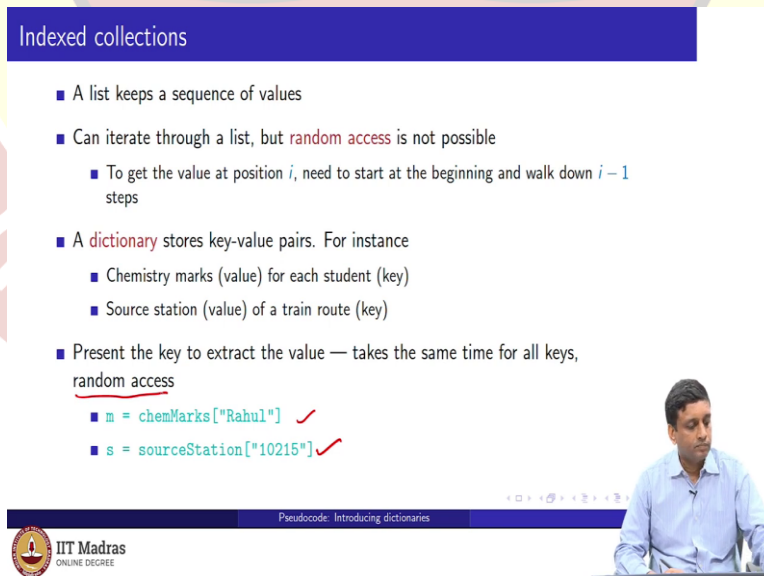
**Computational Thinking**  
**Professor Madhavan Mukund**  
**Department of Computer Science**  
**Chennai Mathematical Institute**  
**Professor G. Venkatesh**  
**Indian Institute of Technology, Madras**  
**Pseudocode for Dictionaries**

(Refer Slide Time: 00:15)



So, in this lecture we will introduce pseudocode for dictionaries.

(Refer Slide Time: 00:19)



So, we have already seen the form of indexed collections, that we can keep a list. So, a list has a sequence of values and we can iterate through those values by using `foreach`. So, we can say `foreach x in L do something`. But if we want to get to the middle of a list, supposing,

we want to get to the value at position  $i$ , then there is no way to get to this position except to walk through the list from the beginning to the end.

So, we have to keep a counter, initialize it to 0 and when we are doing `foreach x in L`, we have to keep incrementing the counter and after  $i$  minus 1 steps, when the counter becomes  $i$  then we will find that we have reached the  $i$ th position in the list. So, in contrast, we have mentioned in the lectures before, that a dictionary will allow us to access values by some index, which is typically called a key.

So, a dictionary stores what are called key value pairs. So, the key can be anything reasonable; for instance, we can keep values which are indexed by names. So, for instance, we could keep in our scores data set. We can keep the Chemistry marks of the students, indexed by the name of the student. So, in this case, the key would be the name of the student and the value would be the marks in Chemistry for that student. So, against each student's name, you would have the corresponding mark.

Similarly, if we look at a railway kind of situation, where we have a train list then we might want to know for each train where it starts. So, we could have now a dictionary, where the keys are the train numbers, and the value, that we associate with the train number, is the source station, the starting station. So, the advantage of doing this is, that once we have stored this key and value into the dictionary, we can extract the value associated with the key in a single step.

So, this is done for instance; although we will look at the notation, but we do this with the square bracket notation, which you see at the bottom. We take the name of the dictionary. We provide the key in square brackets and what the dictionary returns to us, is the value. So, we could store the Chemistry marks of Rahul in this dictionary, and extract it, and store it into a value  $m$  into a variable  $m$ . Similarly, if we wanted to know the source station of the train 10215, if we ask the dictionary, what is the value associated with the key 10215. It will return a name of a station and we will store that in  $s$ .

The important thing about this whole operation is, that it is, what is called random access. Now, random access essentially means, that regardless of which key you look up, it does not matter whether it was an early key, it does not matter it was a late key. All keys will return the value in roughly the same time. So, it does not take any more time to look up a key,

which was added to the beginning and at the end. In a truly random access structure, it will also not depend on the size of the dictionary.

So, we will assume that we have somehow an implementation of this data structure. This dictionary collection where we can provide a key, and get the value in a fixed amount of time very quickly. Unlike in a list, where if we want a specific value, we have to navigate through the list, we have to start in the beginning and we have to do this foreach and walk down the list. So, without doing this kind of explicit traversal of the collection, we can find on any value by just providing the key. So that is the advantage of a dictionary. So now, let us look at some notation that we will be using in our pseudocode for dictionaries.


(Refer Slide Time: 03:47)

Pseudocode for dictionaries

- At a "raw" level, sequence of **key:value** pairs within braces
  - {"Rahul":92, "Clarence":73, "Ritika":88}

Pseudocode: Introducing dictionaries

IIT Madras  
ONLINE DEGREE




Pseudocode for dictionaries

- At a "raw" level, sequence of **key:value** pairs within braces
  - {"Rahul":92, "Clarence":73, "Ritika":88}

( [ {

Pseudocode: Introducing dictionaries

IIT Madras  
ONLINE DEGREE



## Pseudocode for dictionaries

- At a “raw” level, sequence of **key:value** pairs within braces
  - `{"Rahul":92, "Clarence":73, "Ritika":89}`
- Empty dictionary, `{}`
- Access value by providing key within square brackets
  - `s = sourceStation["10215"]`
- Assigning a value — replace value or create new key-value pair
  - `chemMarks["Rahul"] = 92`
- Dictionary must exist to create new entry
  - Initialize as `d = {}`

Pseudocode: Introducing dictionaries



IIT Madras  
ONLINE DEGREE

## Pseudocode for dictionaries

- At a “raw” level, sequence of **key:value** pairs within braces
  - `{"Rahul":92, "Clarence":73, "Ritika":89}`
- Empty dictionary, `{}`
- Access value by providing key within square brackets
  - `s = sourceStation["10215"]`
- Assigning a value — replace value or create new key-value pair
  - `chemMarks["Rahul"] = 92`
- Dictionary must exist to create new entry
  - Initialize as `d = {}`

### Example

Collect Chemistry marks in a dictionary

```
chemMarks = {}  
while (Table 1 has more rows) {  
  Read the first row X in Table 1  
  name = X.Name  
  marks = X.ChemistryMarks  
  chemMarks[name] = marks  
}  
Move X to Table 2
```

Pseudocode: Introducing dictionaries



IIT Madras  
ONLINE DEGREE

So, at the raw level we will assume, that a dictionary is stored as a sequence; even though we will not navigate the sequence from beginning to end. We can create a dictionary by providing a sequence, each element in the sequence will have to have a key and a value. So, we write it as a key colon value. So here, for instance is a set of a dictionary, which has marks for 3 students. So, the keys in this case are Rahul, Clarence and Ritika. So, these are the 3 names and the corresponding marks are 92, 73 and 89.

So now, the thing about these keys and values is we have to connect them. So, we use this colon. So, each key and value is connected by a colon, and finally to distinguish this sequence from a list. So that, we do not get confused, that this is a list. We use this curly brace rather than a square bracket.

So, as we have seen so far, there are many different kinds of brackets. We have the round bracket, which we use in expressions in arithmetic expressions; for instance, to disambiguate how they are bracketed. Then we have the square bracket, which we have been using for lists. And we have using this curly brace so far in our pseudocode to indicate blocks of code. What happens inside a while or what happens inside an if. But without any confusion, because this is a context which is completely different, we are going to use these curly braces to indicate the sequence of values that go in a dictionary.

So, although, we can specify dictionaries like this, this is not the usual way that we create a dictionary. So usually what we do, is we create an empty dictionary. An empty dictionary is just signalled by a pair of opening and closing braces with nothing inside, and then we add values to it, as we will see. So, once you have a dictionary, as we saw in the previous page, you can get the value for a key; by just providing that key within square brackets. So, we can say, `s` is the source station of the key 10215.

Now, when we assign a value we just create a key and value, if it does not exist or we overwrite. So, supposing, we do not have an entry for Rahul in our dictionary, then if you say chem marks of Rahul is equal to 92. It will create a new key Rahul and assign the value 92. If there was already an entry for Rahul; say supposing we had Rahul's entry was 88. If we do this, it will overwrite that 88 by 92. So, at any given time along with the key we have 1 value and there is only 1 copy of that. Of course, we can keep complex values.

So, we can keep a list of values along a list, as a value along with the key. But if, we keep only 1 number and we write it and assign it again, it will overwrite. So, a dictionary must exist before we can create a key using this. So, if we do not have a dictionary, then in principle the notation will not know whether the variable, we are using is a dictionary or not. So, we have to write this `d` equal to empty dictionary to indicate, that the variable `d` is going to be used as a dictionary, before we create values for it.

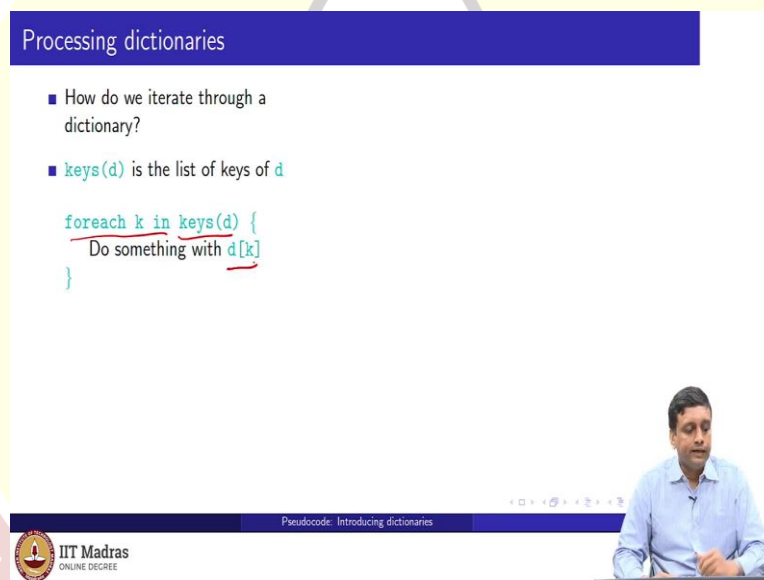
So, here is a simple example of how we could use a dictionary. So, we already talked about this dictionary to collect the Chemistry marks. So, supposing, we want to actually collect all the Chemistry marks in our scores table into a dictionary called `chemMarks`. So, first, we initialise these `chemMarks` and then we do the following. While the table has more rows, we read each row and we extract the 2 necessary items from that row, namely the name of the



student and the Chemistry marks of the student. And then we create a fresh entry, which says that the Chemistry marks for this name is the marks on this card.

So, this incrementally builds up the dictionary from beginning to end by creating first an empty dictionary, and then for each card creating a key for the name on that card, and the value as in Chemistry marks in that card. So, this is a typical way, in which we will construct a dictionary. We will not normally do this kind of an assignment, where we create a dictionary from scratch by just associating a sequence of key value pairs. But remember, that when you create a dictionary like incrementally, you must either start with something which has a value; or in the worst case, we must start with an empty dictionary, so that we have something to add onto.

(Refer Slide Time: 08:11)



The slide is titled "Processing dictionaries" and contains the following content:

- How do we iterate through a dictionary?
- `keys(d)` is the list of keys of `d`

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

At the bottom of the slide, there is a navigation bar with the text "Pseudocode: Introducing dictionaries" and the IIT Madras logo. A speaker is visible in the bottom right corner of the slide frame.

## Processing dictionaries

- How do we iterate through a dictionary?

- `keys(d)` is the list of keys of `d`

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

- Example

- Compute average marks in Chemistry

```
total = 0 ✓  
count = 0 ✓  
foreach k in keys(chemMarks) {  
    total = total + chemMarks[k] ✓  
    count = count + 1 ✓  
}  
chemavg = total/count
```



IIT Madras  
ONLINE DEGREE

Pseudocode: Introducing dictionaries

Navigation icons



So, like, we saw with a list, the most common thing, that we need to do, is to process the entire collection as a unit. For example, we might want to go through all the elements in the list and do something with it. So, this we called iteration and iteration list is very similar, to what we did the iteration with cards, iteration with tables. You just systematically go from beginning to end through each item in that list and do something with it, and that is what `foreach` gave us. So now, we said it in dictionary, we have this random access and there is no way to look in advance at the name of a dictionary and guess which keys are there.

So, how do we systematically process all the entries in a dictionary? So, we need a way of extracting the keys from a dictionary, and this is provided to us, we will assume by a function called `keys`. So, the function `keys` will take a dictionary as variable as an argument and give back a list. It gives back a list of all the keys in `d`. Once, we have a list, we know how to process it. So, this is how, we typically do it. We take `keys of d`. It gives us a list, and now we use our `foreach`.

So `foreach k in keys of d`, will go through all the keys in the dictionary `d` in some sequence. Now remember that we are not promising, in what sequence these keys are extracted. They may be extracted in some order, which is different even from the order in which we added them. So, in the previous example, when we looked up the scores database for instance. We added 1 student at a time, and we added it in some sequence, in which the scores table was presented to us.

Now, if I say what are the keys of the Chemistry marks table, it may or may not be in the same sequence. So, we should not assume anything, but it will give me all the keys. So, once



I have all the keys, I can go from beginning to end and do something with each key. So, with each key I can extract the value  $d$  of  $k$  and then do something with  $k$  and  $d$  of  $k$ . So, this is how we typically process the list.

So, for example, if we wanted to compute the average marks, after we have assembled all the Chemistry marks into that dictionary `chemMarks`. Now, I need to process that dictionary in order to get the average. So, I need to first add up all the Chemistry marks in that dictionary and then I need to divide by the total number. So, for instance, I could do the usual thing. I keep a variable to keep track of the total, and to keep track of the count the number of students, both of which are initialised to 0.

And now, I do my `foreach` over the keys. So, for every key, which I see in this Chemistry marks dictionary, I pick up the total and add it the Chemistry marks of that key, associated that key, added to the total and I increment the count. And finally, I can take the average as the total divided by the count. So, this is a typical example of how we process this dictionary.

(Refer Slide Time: 10:56)

**Checking for a key**

- Typical use of a dictionary is to accumulate values
  - `runs["Kohli"]`, runs scored by Virat Kohli
- Process a data set with runs from different matches
- Each time we see an entry for Kohli, update his score
  - `runs["Kohli"] = runs["Kohli"] + score`

Pseudocode: Introducing dictionaries

IIT Madras  
ONLINE DEGREE

## Checking for a key

- Typical use of a dictionary is to accumulate values
  - `runs["Kohli"]`, runs scored by Virat Kohli
- Process a data set with runs from different matches
- Each time we see an entry for Kohli, update his score
  - `runs["Kohli"] = runs["Kohli"] + score` ✓
- What about the first score for Kohli?
  - Create a new key and assign score
  - `runs["Kohli"] = score` ✓

Pseudocode: Introducing dictionaries



IIT Madras  
ONLINE DEGREE



## Checking for a key

- Typical use of a dictionary is to accumulate values
  - `runs["Kohli"]`, runs scored by Virat Kohli
- Process a data set with runs from different matches
- Each time we see an entry for Kohli, update his score
  - `runs["Kohli"] = runs["Kohli"] + score`
- What about the first score for Kohli?
  - Create a new key and assign score
  - `runs["Kohli"] = score` ✓
- How do we know whether to create a fresh key or update an existing key?
  - `isKey(d,k)` — returns `True` if `k` is a key in `d`, `False` otherwise
- Typical usage

```
if isKey(runs, "Kohli"){
    runs["Kohli"] = runs["Kohli"] + score
}
else{
    runs["Kohli"] = score
}
```

Pseudocode: Introducing dictionaries



IIT Madras  
ONLINE DEGREE



So, it is quite common, that we need to know, whether a key is there or not. So, suppose, we are trying to accumulate values. So, we are trying to accumulate values; for example, we have some dictionary, which contains the runs scored by different batsman in say, a one day series or a test series. And now, we want to accumulate these runs as we go along. So, in such a situation, if we look at a dictionary called runs. Then the keys will be the name of the batsman. So, if I look at runs of Kohli for example; it should give me the runs scored by Virat Kohli so far in the series.

Now, this is going to be accumulated. So, what we will assume is that, we have the data of different matches. So, in match 1 Kohli scored so much, in match 2 maybe he did not bat at all, in match 3 he scored so much more and so on. And I need to go through these matches one by one and accumulate the thing. So, each time, we see a score. So, assume that, the

current match provides us with Kohli's score in the current match in a variable score. I will take the dictionary, entry for Kohli and increment it, or append or add to it, the value score. I will augment it.

So, this is how we accumulate the runs of Kohli in a dictionary. But there is an interesting situation, which happens right at the beginning; because right at the beginning when there is no dictionary entry for first time, I see a score for Kohli. Kohli is not a key in my runs dictionary. I will create these dictionary keys only when a batsman appears. I cannot predict in advance, who are all the batsman in world, who are going to bat in this series.

So, as I see a batsman for first time, I need to create an entry. And when, I create an entry, I have to record the first score. I cannot augment, because there is nothing to augment. So, when I see Kohli's score for the first time, I need to create a key and assign the score. So, I need to use this kind of an assignment, rather than this kind of an update. I need to assign runs of Kohli to the value score and not append or augment runs of Kohli with the value score.

So, to do this, I need to check whether or not Kohli is a key. So how do we know? Whether we should create this key or we should update the value of the given key. So, let us assume that, we have a function, which tells us for a dictionary *d* whether *k* is a key or not. So, *isKey* takes a dictionary and a key, and returns true if *k* is currently a key in the dictionary, and false if *k* is not a key in this dictionary. So, if, we have *isKey* available to us, then what we need to do, every time we process a score of Kohli is to call *isKey*. So, we will check; does Kohli's entry exists; if Kohli's entry exists. So, if *isKey* reports true, then we augment Kohli's entry with the current score.

On the other hand, if Kohli's entry does not exist, then we have to create it. So, we do the second case, which we had at the bottom here. We do the second case, where we create a new entry in our dictionary with the key Kohli and assign it the current score. So, this is an important thing in a dictionary; that we need to know, when to update a value and when to create a value. Because very often, a dictionary is used to accumulate values and this is a very useful way to use a dictionary. So, the same thing, so we do not have to keep track, we do not have to make a slot in advance for every possible batsman. We create keys as the batsman come and as the batsman score gets augmented in the scorecard, we can augment it in the dictionary.

(Refer Slide Time: 14:25)

### Checking for a key

- Implementing `isKeys(d,k)`
  - Iterate through `keys(d)` searching for the key `k`

Procedure `isKey(D,k)`

```
found = False
foreach key in keys(D) {
    if (key == k) {
        found = True
        exitloop
    }
}
return(found)
End isKey
```

Pseudocode: Introducing dictionaries

IIT Madras  
ONLINE DEGREE

### Checking for a key

- Implementing `isKeys(d,k)`
  - Iterate through `keys(d)` searching for the key `k`
  - Takes time proportional to size of the dictionary
  - Instead, assume `isKeys(d,k)` is given to us, works in constant time
    - Random access

Procedure `isKey(D,k)`

```
found = False
foreach key in keys(D) {
    if (key == k) {
        found = True
        exitloop
    }
}
return(found)
End isKey
```

Pseudocode: Introducing dictionaries

IIT Madras  
ONLINE DEGREE

So, how do we implement this? Well, `isKey` can of course be implemented using our `foreach` with respect to the keys. So, here is a very naive implementation of `isKey`. So, we are looking for `k`, `k` in a dictionary `d`. So, we assume that we have not found it. And now, we go through all the keys of `d` and `foreach` variable, which we will call `key` again. `foreach` `key` that is there in the keys of `d`. If the value, that we see in the key, is the value that we are searching for, then we set `found` equal to `true`.

And once you have set `found` equal to `true`, we do not have to process anymore of that list, because we know that, we have found it. So, we can use the special thing, which we discussed, when we discussed list, we can have this `exitloop`, which quits this `foreach` and comes out here. So, this is an interesting loop for other reasons. So, one interesting thing

about this loop is, notice that if we do not find the key what happens, if we do not find the key, then for every key in keys of d, key is not equal to k. And therefore, this found is never reset to true.

So, we will start with found equal to false. And when, we come here found is still false. So, either we have processed the list, found the key and exited the loop with found equal to true. Or we have processed the list from beginning to end not found the key, and found has remained false. So, at the end, we know whether way we found it or not, because either found was reset, when we found it, or found was not reset because we did not find it. So finally, we just return the value of found.

So, except for this business about exitloop, and the way that found is initialised and returned. There is nothing very particularly clever about this loop. This is just a standard iteration through a list, except this list is the keys of the dictionary. So, we go through all the keys systematically, and check whether the key we want is there, and this gives us a correct implementation of isKey.

So, what is the problem with this? Well, the problem with this is that, it will of course take time proportional to the size of the dictionary. In particular, if the value is not there, first time we are trying to create a key, its value will not be there. And, the only way we will discover that, the value is not there, is by iterating through all the keys, and checking that all of them are not the key we want and then returning false. So, this is not consistent somehow with our expectation, that dictionaries are random access.

So, in a dictionary, we said that we would not be able to look up a key, and return the value in a constant amount of time without any reference to how many keys are there in the dictionary. So likewise, we would expect any operation involving the dictionary to be random access. So, although, this is a valid way of implementing isKey ourselves, we will assume that we have a better implementation given to us. So, we assume that isKey is given to us, so that it works in constant time. So, that we preserve this kind of random access property of dictionaries.



(Refer Slide Time: 17:25)

## Summary

- 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]`  
`}`
- `isKey(d,k)` reports whether `k` is a key in `d`  
`if isKey(d,k){`  
    `d[k] = d[k] + v`  
`}`  
`else{`  
    `d[k] = v`  
`}`



IIT Madras  
ONLINE DEGREE

Pseudocode: Introducing dictionaries



So, the summary of what we have discussed is that, in addition to lists which are collections of values stored in sequence, where we have to access the values from beginning to end iteratively, without being able to jump into the middle of a list. We need very often a mechanism to store, store an indexed set of values, where we can jump into a particular value by specifying its index.

So, this is what we call a dictionary. So, a dictionary stores keys and values. The keys are the indices. We specify a key, we get a value, we specify a key, we can update the value. So, this is a random access structure. So, it does not matter which key, we ask for or which key we want to look up, which key we want to update. It takes the same amount of time, it is a constant time operation. So, we can think of a dictionary as a sequence of key value pairs, where the keys are written as `k1 colon v1`. So, the pairs are `k1 colon v1`, `k2 colon v2` and so on.

We take each key, each values combine it with a colon, and then make a sequence. And we use this curly brace instead of this square brace in order to distinguish this from a normal list. So, we also said that the normal way of creating a dictionary is not to specify the sequence, but to actually create it entry by entry. So, we start off with an empty dictionary, and then we create keys. So, if we make an assignment to a key which does not exist, then it automatically creates the key.

So, this is a great convenience, we do not have to have a special operation to create a key, but we need to be careful about when the key exists, and when it does not exist. So, we can



iterate through the keys by using keys of d, and we can check whether a key exists, or whether we need to create a new key by using isKey.

So, the typical thing is that if the key exists then we are going to update something, if the key does not exist then we are going to create it with the value that we see. So, with this we are now well set to try and look at some of the examples that we have seen using dictionaries in terms of pseudocode.

