IIT Madras

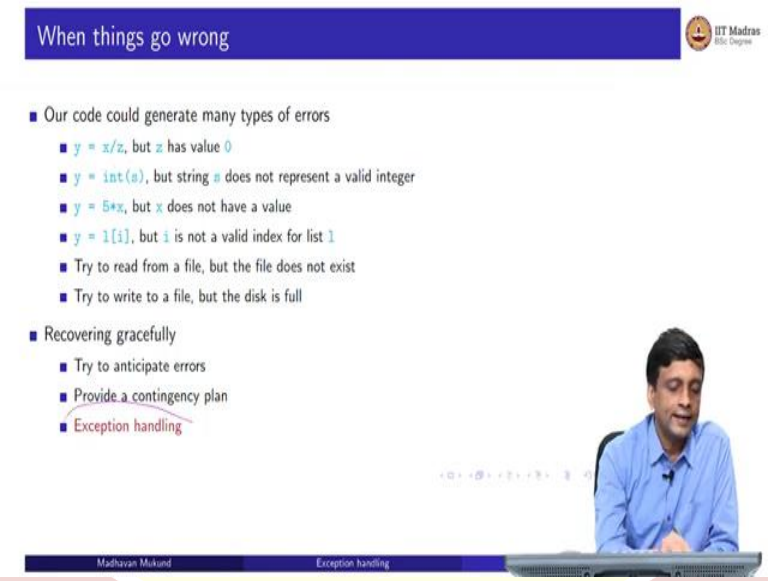
ONLINE DEGREE

Programming, Data Structures and Algorithms using Python
Professor Madhavan Mukund
Exception Handling

So, when we looked at the gcd problem and the primes examples, we covered some of the more standard aspects of python in terms of control flow, so we saw conditionals if, we saw loops for and while, we saw data structures like lists and dictionaries, we also looked at breaking out of a loop, we also considered the problem of initializing a variable properly so that python knows what type it is, so we saw all these standard things just to remind ourselves.

Now, let us look at something slightly more exotic which is we will encounter in code so it is important to understand it, so this is about how to handle errors in our code or what are called exceptions.

(Refer Slide Time: 0:54)



So, when we run code things go wrong, I mean if all code ran perfectly it would be a very simple world and sometimes they go wrong because there is something wrong with what we wrote the logic is wrong, sometimes they go wrong because the values are not what we expect. So, there are many different types of errors that a code, piece of code could generate.

So, the most fundamental ones are those to do with values. For instance, you could have an arithmetic expression, you could be dividing x by z but unfortunately at this moment when you

are dividing x by z you forgot to check and z actually happened to be 0 so this division is not defined or there are functions python supports type conversion.

So, in particular if you read input from the keyboard then it is always a string and now if you want to interpret it as a number you have to apply this function int in order to convert it to a number. But what if that string does not represent a number, supposing it has a decimal point or it has characters other than 0 to 9, then python will throw an error. So, in this case it is the problem is that the string is not a valid integer, it is a string but it does not represent something that python can understand as an integer.

Another type of error is that you actually use an uninitialized variable. So, you put an expression y equal to 5 times x but so far you have not assigned a value to x, so 5 times x has no meaning, so it cannot assign a value to y either or you have a complex structure like a list and you are trying to access a position in the list but the position you are trying to access is outside the bounds of the list.

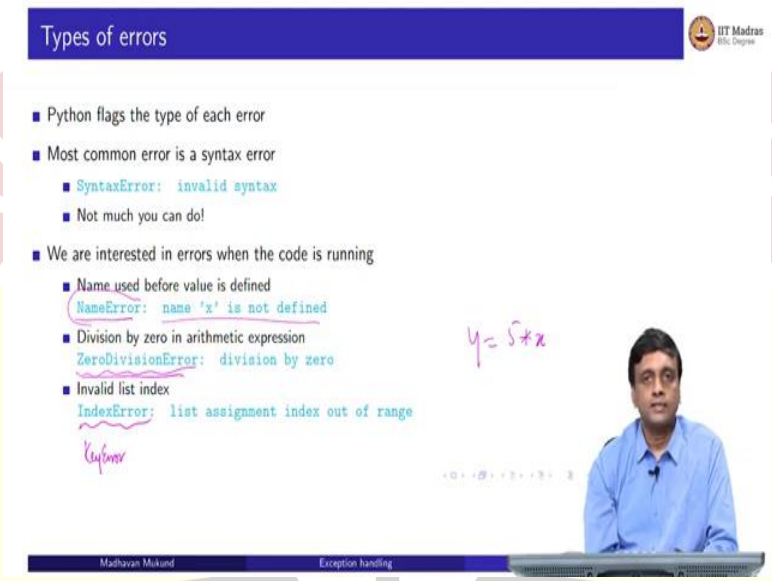
So, the list has positions remember from 0 to the length minus 1 and you are trying to access an i which is outside that. Similarly, you could be doing it for a dictionary also you are trying to access a key from a dictionary but that key does not exist. And finally, there are things which could be outside the scope of the program itself, the values are not wrong inside the program but you are interacting with something outside the program.

A typical example when you are trying to read a file so supposing you try to open a file but the file name that you are trying to use does not exist, somebody has renamed the file or deleted it, then your program will throw an error. Similarly, when you try to write a file, you might well find that the disk is out of space, you want to write a large file there is not enough space on the disk.

Now, these are things which you cannot do much about within the program, so the question is can you anticipate this? So, what the goal of exception handling is to recover gracefully from errors wherever it is possible. So, we want to try and anticipate the kind of mistakes that might occur when our code runs and provide these alternative paths, a contingency plan as it were saying if this happens, if it goes wrong in this way do this, if it goes wrong in that way do that or report something to the user so that they know.

Supposing, I asked the user to give me a file name and I was not able to read the file, instead of the program aborting and throwing some error it is better that I print out a message to the user saying the file name you provided does not exist try again, for example. So, this broadly is what is called exception handling.

(Refer Slide Time: 3:56)



So, the good thing for us is that python actually gives us information about each type of error and this confirmation comes in two parts. So, first there is a kind of a name, a type of error and the second is a description. So, the most common type of error is a syntax error, this is not valid python code.

Now, if it is not valid python code it cannot run and since it cannot run, we cannot take any corrective action about it. So, we will not bother about syntax there is an exception handling because nothing you can do, syntax errors come before the code runs, exception handling comes when the code is running. So, unless you have valid python code there is no question of exception handling, so there is not much we can do about this.

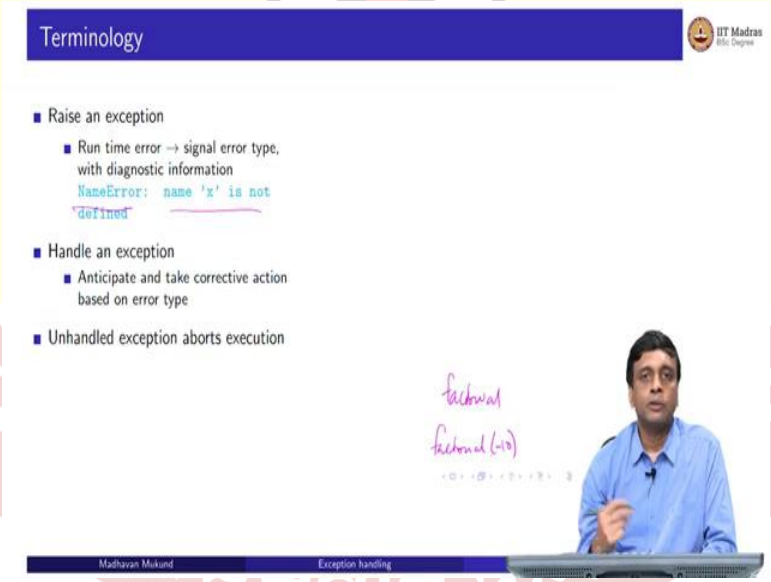
So, we are interested in errors when the code is running so if you have actually run python code and looked at the errors you will see some of these and recognize them, for instance, a name error, python calls a name error, an error which occurs when you try to use a value which is not defined, you have a variable x which is not defined.

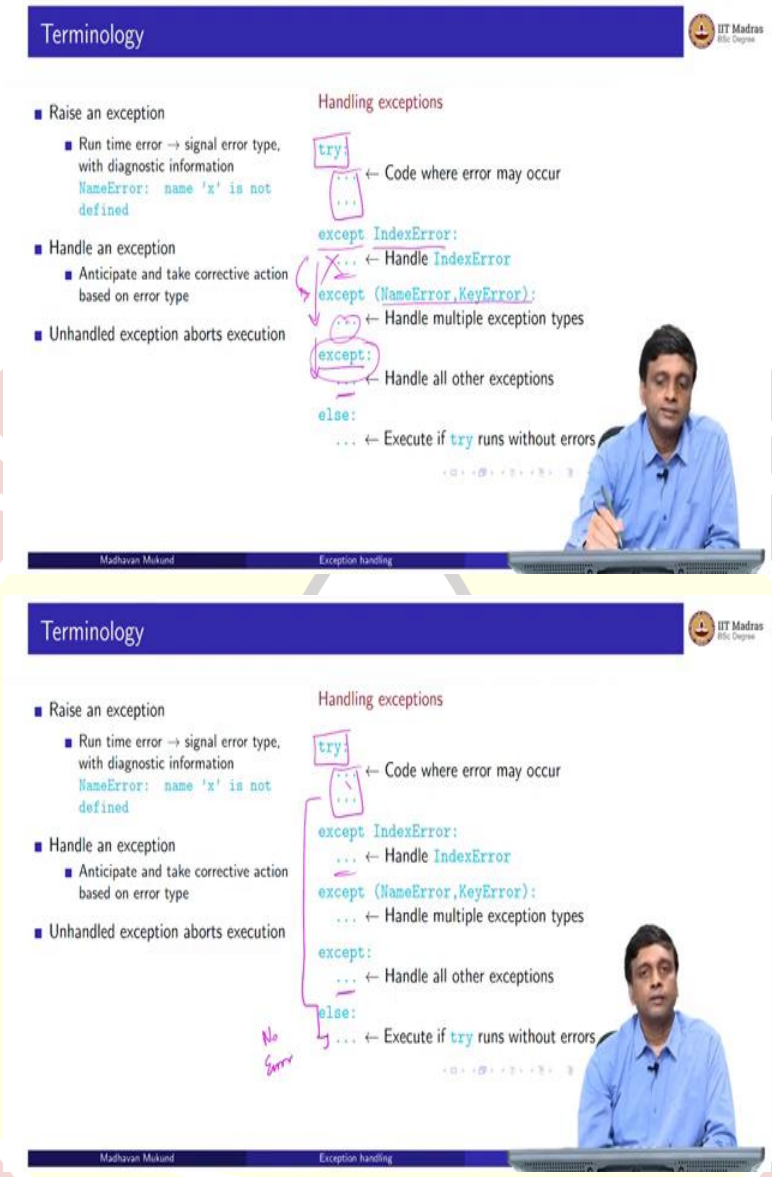
So, this is like our earlier thing where we said y is equal to 5 times x but x was not yet assigned a value then this would generate a name error. The other example we saw before where you try to evaluate an arithmetic expression where the numbers are defined but the denominator is 0 for a division will give you a 0 division error.

So, in each case there is a fixed kind of a terminology that python uses. So, every time you get a name error you will get this message name error, but there will be a diagnostic string, a kind of explanation which will give you a little bit more information which was the name that was wrong. So, that comes as a bonus for a human to read but as far as we are concerned for exception handling what we are really interested in is the first part because this will tell us what kind of error occurred in our code.

So, the final thing that we looked at was errors dealing with data structure, so index error is that a list is out of range and for example a key error would be that a dictionary key that we are trying to access does not exist. So, in all such cases we want to be able to take corrective action.

(Refer Slide Time: 6:05)





So, what happens when code generates an error is called raising an exception. So, when it raises an exception the code creates these two things, it creates the type of error and creates a message. Now, we will see that we can actually raise exceptions ourselves, so we can create an error if somebody uses our code in the wrong way.

For instance, supposing you define a function called factorial and somebody asks you to compute factorial of minus 10 so we know that factorial is not well defined for negative numbers because the factorial is 10 times 9 times 7 up to 1 but if I start with minus 10, I cannot keep going down because it will go infinitely far.

So, what you do when you see factorial of minus 10? Well, there are two things, one is you can return some default value or you can actually raise an exception saying do not call this factorial with function with a negative number. So, raising exception we can do but we are not going to focus too much on that, we will see a small example later on.

But the code that is running will in general raise exceptions and what we want to focus on is how we handle this exception. So, we know that the code that we call could generate an error so how do we deal with that? And if we do not deal with it what will happen is that our code will crash. So, we call a function that function generates an error, we do not have code to cope with that error, so our function will also crash because a function that we called crashed. So, it is in our interest to avoid that and that is why we want to handle the exceptions.

So, to handle an exception we use this kind of format so that we take the code that we are trying to run. So, this is the actual code that we had, we are trying to run this code but we know that potentially there are some things in that code which might generate errors, so we anticipate that by putting it inside this thing called a try.

So, the word try is self-explanatory, try to run this code. And now, if it generates errors take the following action, so that is what is except does. So, except suggest what to do when different types of errors happen, so with except there is an indicator about what exception or what error we are handling. So, if the code generates an index error then we will execute this code.

Now, it happens in sequence, so if it is an index error it will come here if it is not an index error then this code will not apply, so it will go to the next line and see is there any other except. The same except can handle more than one thing. So, this is saying if it is a index error the first one happens but if it is a name error or a key error then do this. So, we could have the same exception handling code, handling more than one type of error and these things are checked in sequence.

So, it will first check for index error. Remember the error will be only be of one type, an error comes to us its of one type, if it is an index error it will match this block, if it is not an index error it will not look at this at all, it will directly come to this. If it is not a name error or a key error, it will keep going down.

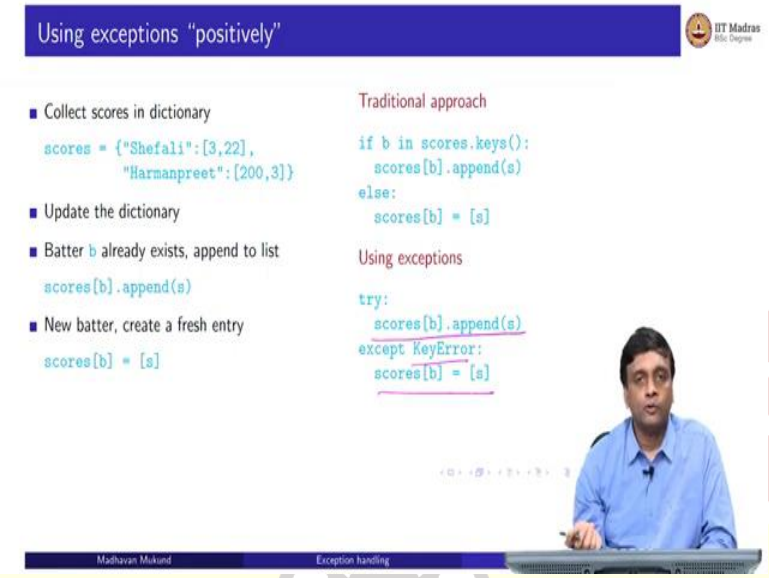
So, maybe there is some errors that we do not know about, but we still want to not have our code crash. So, then we can write an empty except, except with no side qualifier saying which exception it applies to and such an empty except will kind of trap everything. So, everything that reaches this point will execute this code.

So, in this case there are three types of errors which we explicitly deal with, index error, name error, key error, any error of any other type will come to this third except and stop here. So, therefore obviously this kind of a except with no qualifier should be at the end because we put it at the beginning it will not pass anything to any of the other things, this is the most general trap, it is like you are having some kind of a net to catch, a safety net supposing you have a football field, so you have a net and then you have another net and then you have another net, but the biggest net is on the outside if you put the biggest net on the inside nothing will reach the smaller nets, so that is the goal.

And finally this is something which happens even in for loops and while loops, sometimes you want to check whether the for loop or the while loop exited normally. I did not break out of it and I did not abort it. Similarly, the default case hopefully is that this code actually executes, the code at the top actually executes without error.

So, if I come without error then I will come here, so we can attach an else to this try and this else basically will indicate that I actually managed to finish that try block as it is called without encountering an error, so this allows us to distinguish in our code the situation where an error happened and also the, if there is something we want to do in case an error did not happen then we can do it here. So, this is the broad structure of how we use this exception handling in our code.

(Refer Slide Time: 11:08)



So, exception handling is not always used only to track errors, it can also be a style of programming in certain situations and most notably with dictionaries and keys. So, here for instance is a dictionary which stores the scores of different batters in cricket. So, the keys are names and the values that you store is a list of scores so far associated with that person.

So, when I want to update this list there are two possibilities as always with the dictionary that is the person whose name I am going to, whose score I am going to update already has an entry in the dictionary, in that case I just append the current score or this person does not have an entry in which case I want to create this, we saw the same thing when we were doing this counting the differences of primes.

We said if this difference has been seen before increment that difference for this key otherwise create a new entry in the dictionary for this difference because it is the first time we are seeing it. So, the way we did it there was a traditional way we check whether the new key that we are looking at already exists.

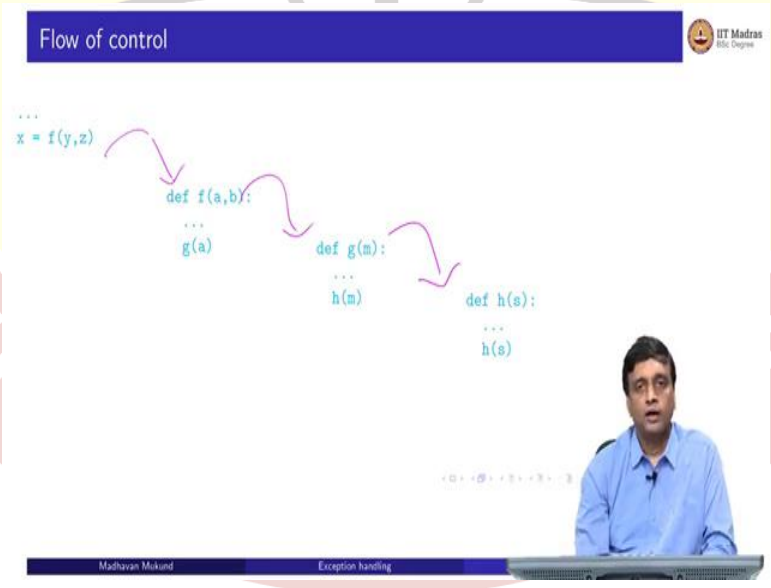
So, if the batter that we are looking at is already available in this dictionary as one of the keys then I append the score for this batter otherwise I create a new list with this one score and this is the first score that I have added for this batter, a new key and a new list, so this is the traditional way. So, you check whether it is there if it is there you append or you increment or whatever, update in some way otherwise you create a new entry.

So, how would we do this in an exception handling way? So, supposing we assume the key is there, so we assume the key is there and we try to update it, if it is there it will work, if it is not there then it will throw this key error, so here is how we do it, we try to update it that is our default, we assume that this key already exists and we try to append the score.

But if this is not the case then python will complain saying that there is no scores of b so this will give us a key error, so we say okay, if there is a key error then do this instead. So, this is not a case where we are flagging some kind of drastic error but rather we are using this exception to signal which of the cases, so it is a complicated way in one sense of doing if then else.

But if you read it in English this is much easier, try to do this and if you cannot do this do that instead which is more or less what you are saying before but you are saying it in a more complicated and convoluted way, instead of saying try to update and then create you are saying if this key already exist then update otherwise create, so it is a matter of taste and style which one is more easy for you to understand, but this is just to explain that exceptions can be used in a sense in a positive way, they do not have to only be used in a negative way to catch errors.

(Refer Slide Time: 14:07)



Flow of control

IIT Madras BSc Degree

```
...
x = f(y,z)
```

```
def f(a,b):
  ...
  g(a)
```

```
def g(m):
  ...
  h(m)
```

```
def h(s):
  ...
  h(s)
```

IndexError not handled in h()

Flow of control

IIT Madras BSc Degree

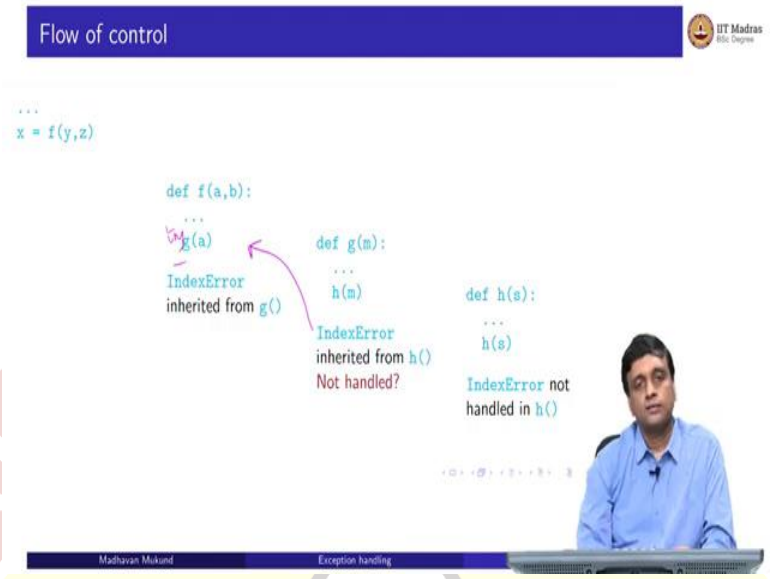
```
...
x = f(y,z)
```

```
def f(a,b):
  ...
  g(a)
```

```
def g(m):
  ...
  try:
    h(m)
  except:
    IndexError
    inherited from h()
```

```
def h(s):
  ...
  h(s)
```

IndexError not handled in h()

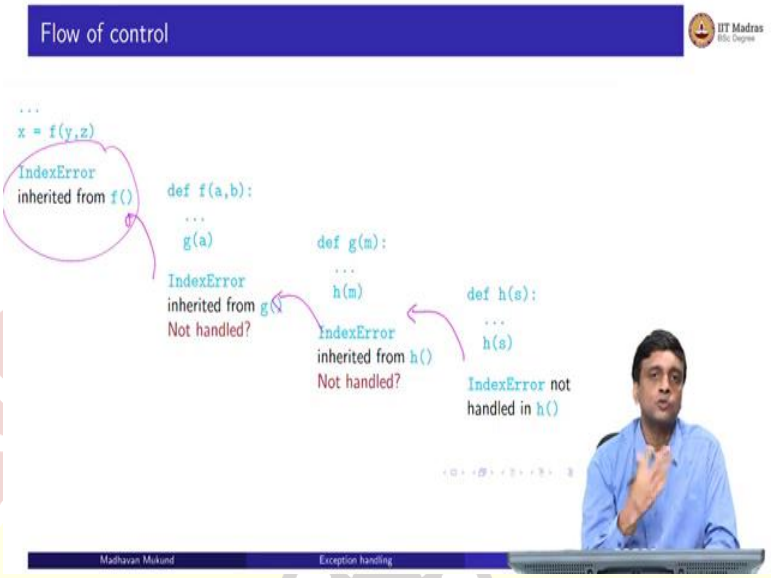


So, to conclude this discussion of exceptions let us see how this whole thing actually works. So, supposing I have this main code which calls a function f so this will transfer control to f which is defined somewhere else. Now, f in turn may call another function g, so this will again transfer control to g which is defined yet another place, and g in turn may call an h, so we have the sequence of calls so this called this, this called this, this called this.

So, we started with calling f and eventually unknown to us in some sequence h has got called. And now h generates an error and it does not internally handle it. So, what happens to this error? So, what happens to this error is that we said before that if an error is not handled then something will abort, but in this case it kind of is unfair to ask this to abort because an error happened deep down, so it will actually abort in stages.

So, this error will get passed back to g, so the call that g made to h will terminate in a kind of incomplete fashion and now it is up to g to figure it out. So, it inherits this index error. So, g will see with respect to h an index error, so supposing there had been a try here, except index error then g would have been able to handle this. But maybe g did not. So, if g did not handle it then the same index error goes back to whatever called g. So, in this case it goes back to f. So, again the possibility is that there was a try here which handles this.

(Refer Slide Time: 15:52)



But if there was not then it goes back to where it called. So, the sequence is that the index error or whatever error it is goes back up the calling stack as it were. So, you keep calling functions and the error goes back in the same sequence and wherever it is handled it will get handled, if it does not get handed it will go all the way back to the original code which started this whole sequence which is the code here.

So, at this point now if I did not anticipate it there is nothing more I can do because this was the beginning, this was the main function that was running. So, if I do not handle this then python has no option but to abort the execution of the program. But if at any intermediate stage there had been this try except thing which caught it, then we could have taken some corrective action and fix this error.