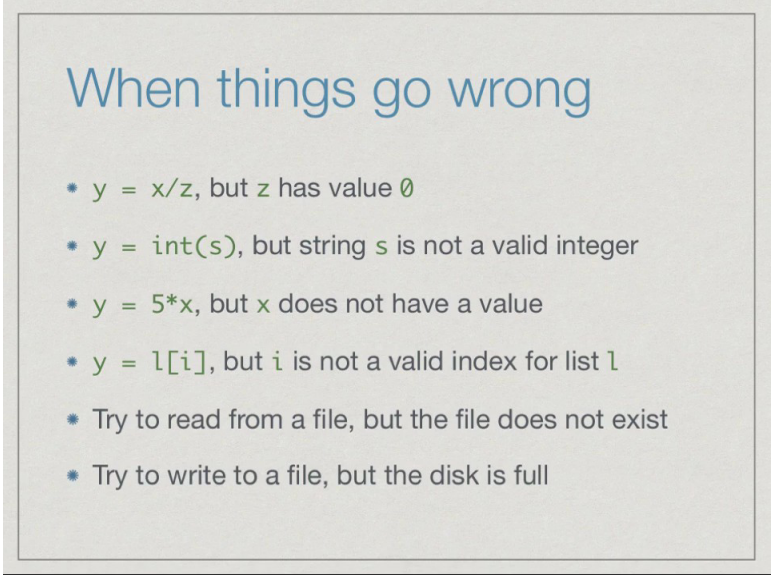
Programming Data Structures and Algorithms in Python
Prof. Madhavan Mukund
Department of Computer Science and Engineering
Indian Institute of Technology, Madras

Module - 05
Lecture - 01
Expecting Handling

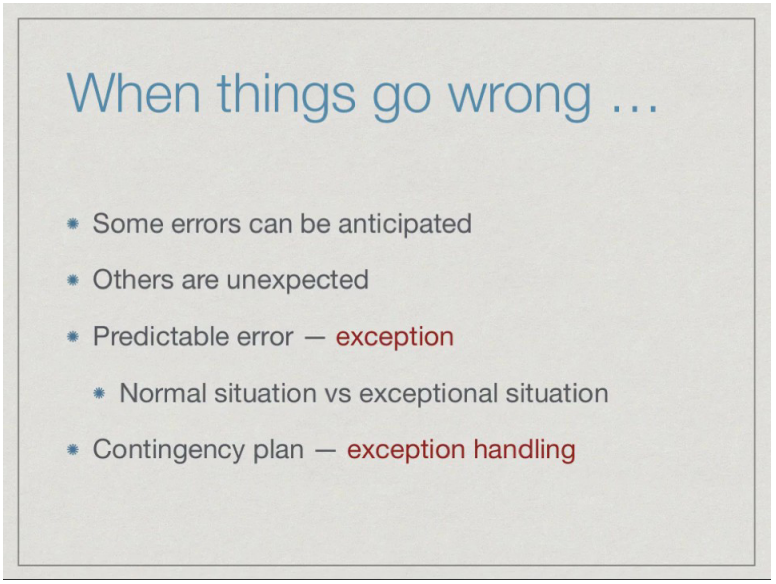
(Refer Slide Time: 00:03)



Let us see what to do when things go wrong with our programs. Now there are many different kinds of things that can go wrong. For instance we might have an expression like x divided by z, and z has a value zero. So, this expression value cannot be computed, or we might be trying to convert something from a string to an integer where the string s is not a valid representation of an integer.

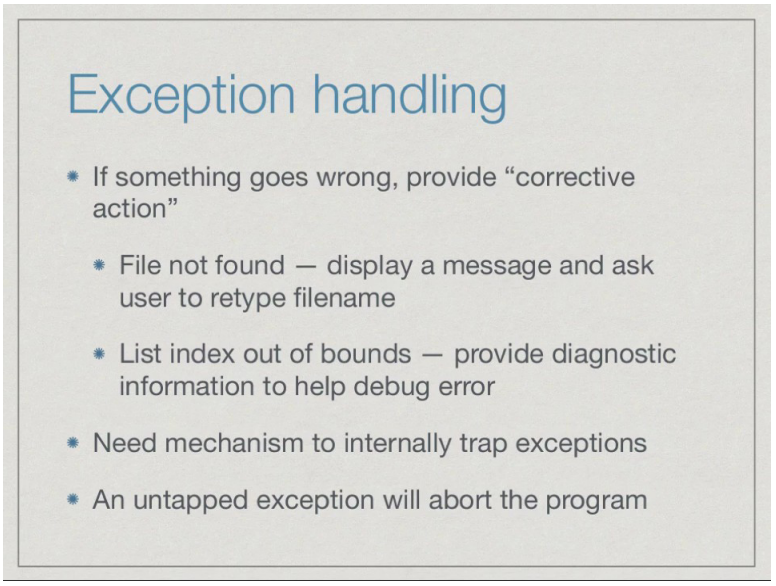
We could also be trying to compute an expression, using a name whose value has not been defined, or we could try to index a position in a list which does not exist. As we go forward we will be looking at how to read and write from files on the disc. So, we may be trying to read from a file, but perhaps there is no such file or we may be trying to write to a file, but the disc is actually full. So, there are many situations in which while our program is running we might encounter an error.

(Refer Slide Time: 00:59)



Some of these errors can be anticipated whereas, others are unexpected. If we can anticipate an error we would prefer to think of it not as an error, but as an exception. So, think of the word exceptional. We encounter a normal situation, the way we would like our program to run and then occasionally we might encounter an exceptional situation, where something wrong happens and what we would like to do is provide a plan, on how to deal with this exceptional situation and this is called exception handling.

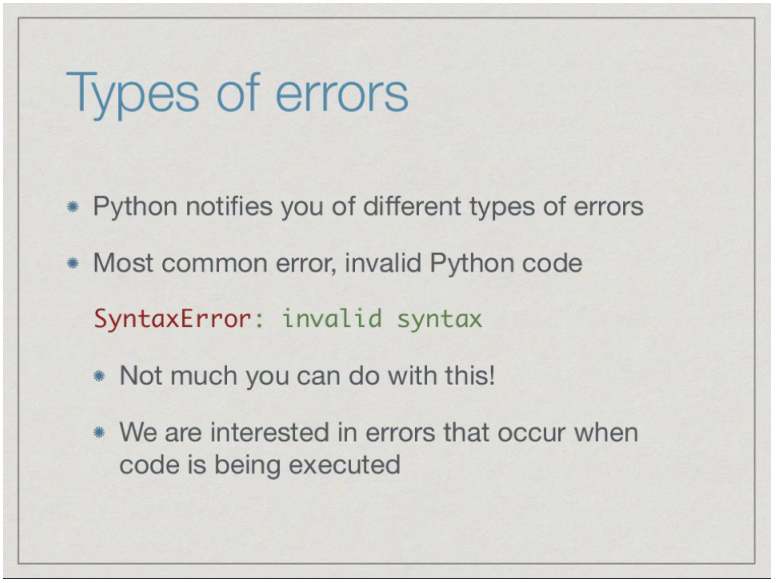
(Refer Slide Time: 01:36)



So, exception handling may ask, when something goes wrong how do we provide corrective action. Now the type of corrective action could depend on what type of error it is. If for instance we are trying to read a file and the file does not exist perhaps we had asked the user to type a file name. So, you could display a message and ask the user to retype the file name, saying the file asked for does not exist.

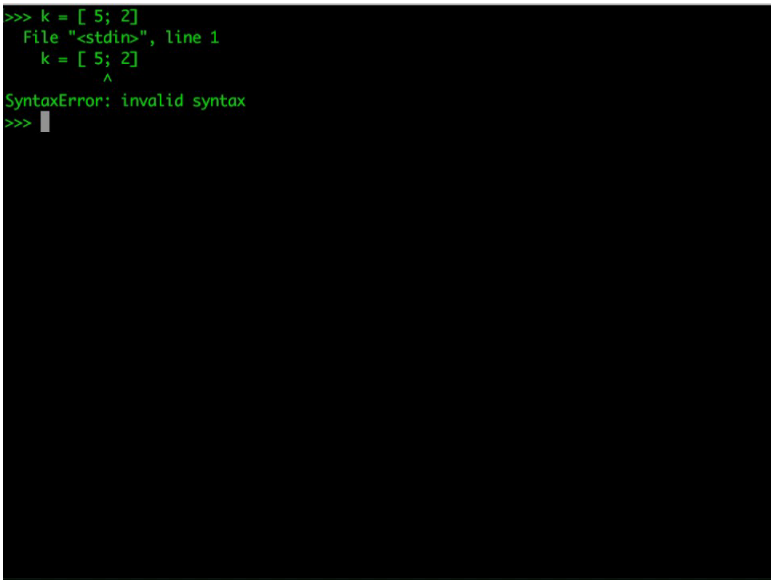
On the other hand if a list is being indexed out of bounds there is probably an error in our program, and we might want to print out this value the value of the index to try and diagnose what is going wrong with our program. Sometimes the error handling might just be debugging our error prone program. For all this what we require is a way of capturing these errors within the program as it is running without killing the program. So, as we have seen when we have spotted errors while we have been using the interpreter if an error does happen and we do not trap it in this way then the program will actually abort and exit. So, we want a way to catch the error and deal with it without aborting the program.

(Refer Slide Time: 02:53)



Now, there are many different types of errors and some of these we have seen, but we may not have noticed the subtelty of these for example, when we run python and we type something which is wrong, then we get something called a syntax error and the message that python gives us is syntax error invalid syntax.

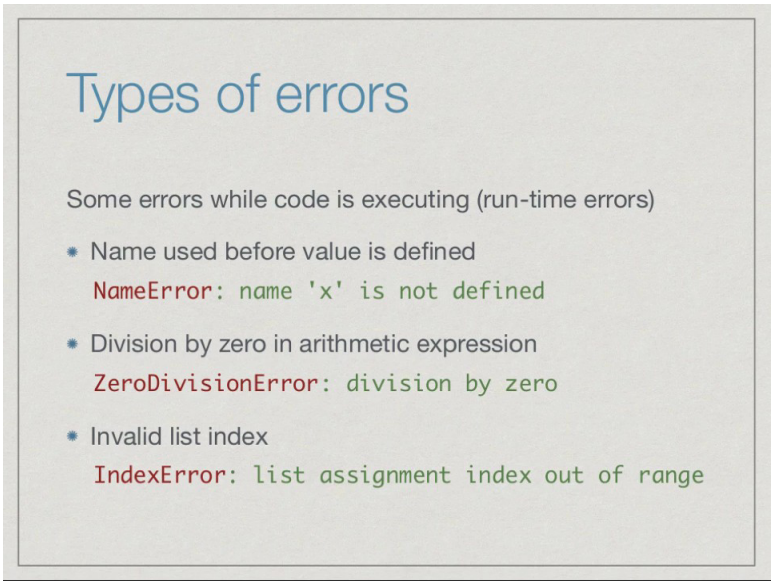
(Refer Slide Time: 03:19)



For example, supposing we try to create a list and by mistake we use a semicolon instead of a comma.Then immediately python points to that semicolon and says this is a syntax error it is invalid python syntax.

Of course, if we have invalid syntax; that means, the program is not going to run at all and there is not much we can do. So, what we are really interested in is errors that happen in valid programs. The program is syntactically correct it is something that the python interpreter can execute, but while the code is being executed some error happens.

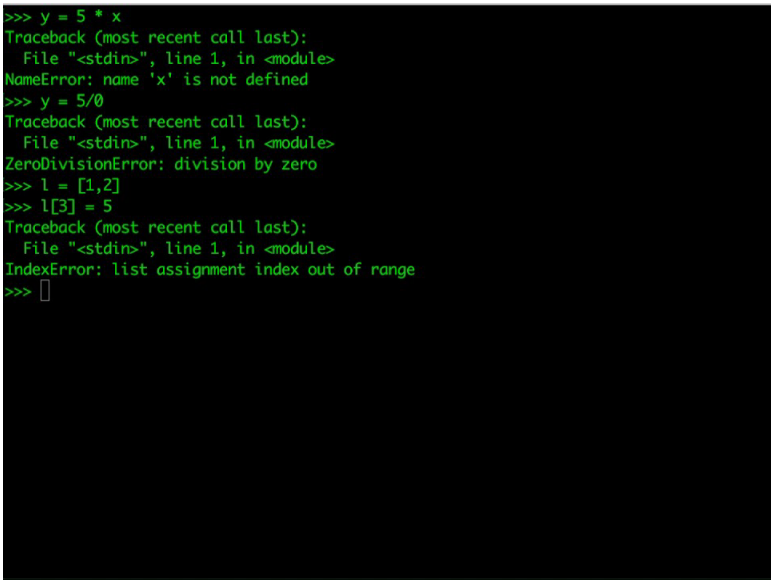
(Refer Slide Time: 04:01)



These are what are called run time errors these are errors that happen while the program is running and here again we have seen these errors and they come with some diagnostic information. For instance, if we use a name whose value is undefined then we get a message from python that the name is not defined and we also get a code at the beginning of the line saying this is a name error.

This is python's way of telling us what type of error it is similarly, if we have an arithmetic expression where we end up dividing by a value 0 then, we will get something called a zero division error and finally, if you try to index a list outside its range then we get something called an index error.

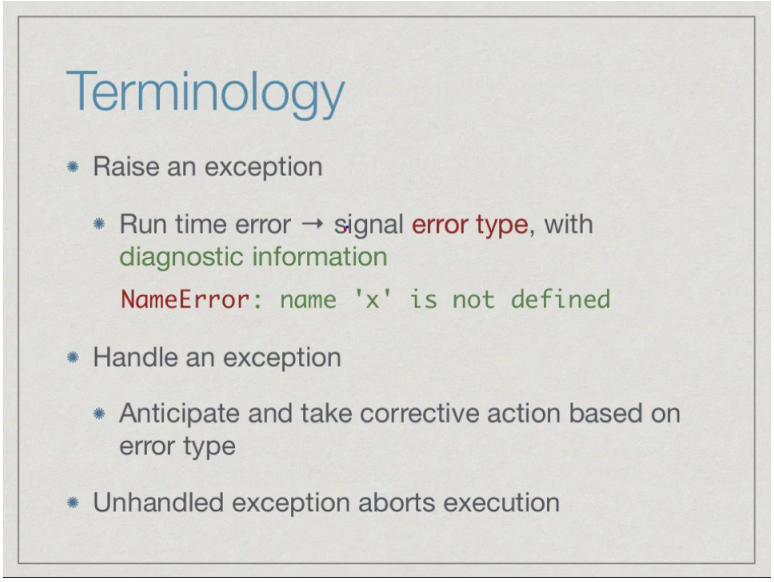
(Refer Slide Time: 04:54)



Let us look at all this error. Just be sure that we understand. Supposing we say y is equal to 5 times x and we have not define anything for x then, it gives us an index error a name error and it says clearly that, the name x is not defined.

On the other hand, if we say is equal to 5 divided by 0 then we get a 0 division error and along with the message division by 0 and finally, if we have a list say 1, 2 and then we ask for the position three then it will say that there is no position three in this list. So, this is an index error. So, these are three examples of the types of error that the python interpreter tells us and notice that there is an error name at the beginning index error name error zero division error plus a diagnostic explanation after that.

(Refer Slide Time: 05:45)

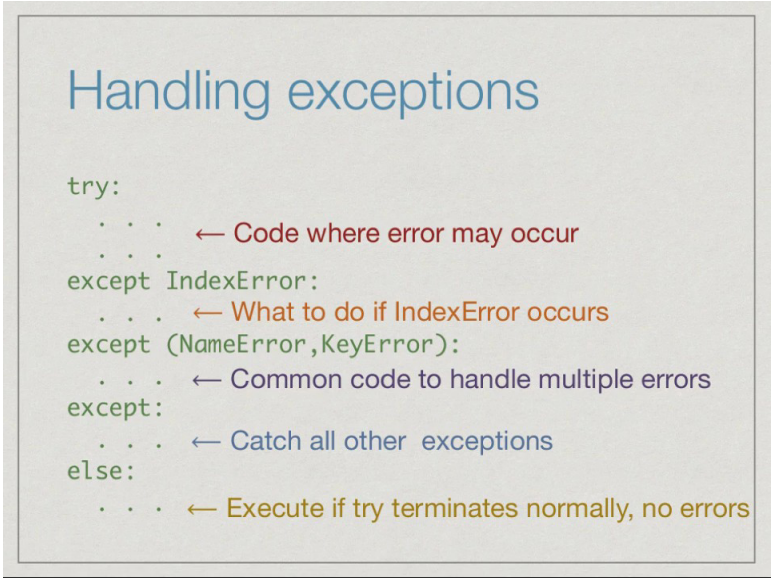


Let us first quickly settle on some terminology. So, usually the act of signalling an error is called raising an exception. So, when the python interpreter detects an error it gives us information about this error and as we saw it comes in two parts, there is the type of the error give what kind of error it is. So, it is name error or an index error or a zero division error and. Secondly, there is some diagnostic information telling us where this error occurs. So, it is not enough to just tell us oh some value was not defined it tells us specifically the name x is not defined.

This gives us some hint as to where the error might be now when, such an error is signaled by python what we would like to do is from within our program handle it right. So, we would like to anticipate and take corrective action based on the error type. So, we may not want to take the same type of action for every error type. That is why it is important to know whether it is a name error or an index error or something else.

And depending on what the error is, we might take appropriate action for that type of error and finally, if we do get an error or an exception which we have not explicitly handled then the python interpreter has no option, but to abort the program. So, if we do not handle an exception if an exception is unhandled then aborts the execution aborts.

(Refer Slide Time: 07:15)



This is done using a new type of block which we have not seen before called try. So, what we have is try block. So, when we have code, here in which we anticipate that there may be some error we put it inside a try. This is our usual block of code where we anticipate that something may go wrong and now we provide contingencies for all the things that could go wrong depending on the type of error and this is provided using this except statement. It says try this and if something goes wrong, then go to the appropriate except one after the other.

The first one says what happens if an index error occurs. So, this is the code that happens if an index error occurs on the other hand maybe I could get a name error or a key error for both of which I do the same thing, so this is the next except block. So, you could have as many except blocks as you have types of errors which you anticipate errors for it is not obligatory to handle every kind of error, only those which you anticipate and of course, now you might want to do something in general for all errors that you do not anticipate. So, you can have a pure except block.

So, kind of a naked except block in which you do not specify the type of error and by default such an except statement would catch all other exceptions. The important thing to remember is that this happens in sequence. If I have three errors for example, if I have an index error and a name error and a zero division error then, what will happen - is that, it will first go here and find that there is an index error. This code will execute. The name

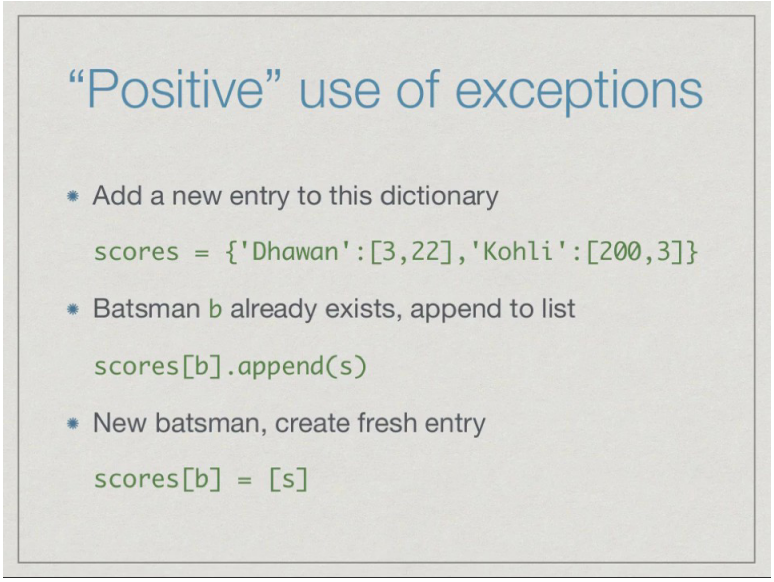
error code will not execute on the other hand, if I had only a name error and if I had a zero division error for example, then because there is a name error first it will first it will come here and will find that there is no index error then will come here and say there is a name error and will execute this code.

The zero division error will not be explicitly handled, the program will not abort, but there will be no code executed for the zero division error; it is not that it tries each one of these in turn it will try whichever except matches the error and it will skip the rest. So, finally, if I had only a zero division error in this particular example then, since it is not an index error and it is not a name error, it would try to go through these in turns that would come here find this is not a type of error. It is not a type of error and it will go to the default except statement and catch all other exceptions.

Finally, python offers us a very useful alternative clause called else. So, this else is in the same spirit as the else associated with a 'for' or a 'while' remember that a for or a while that does not break that terminates normally then executes the else if there is a break the else is skipped in the same way, if the try executes normally that is there is no error which is found then the else will execute otherwise the else is skipped right.

So, we have an else block which will execute if the try terminates normally with no errors. This is the overall structure of how we handle exceptions we put the code that we want inside a try block then we have a sequence of except blocks which catch different types of exceptions we can catch more than one type of exception by putting a sequence in a tuple of exceptions, we can have a default except with no name associated with it to catch all un other exceptions which are not handled and finally, we have an else which will execute if the try terminates normally.

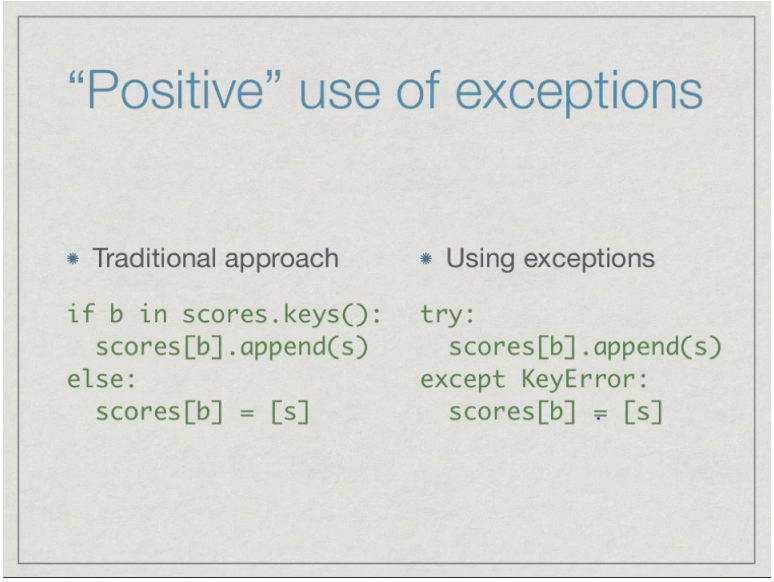
(Refer Slide Time: 10:58)



Now, while we normally use exception handling to deal with errors which we do not anticipate. We can actually use it to change our style of programming. So, let us look at a typical example. We saw recently that we can use dictionaries in python. So, dictionaries associate values with keys here we have two keys Dhawan and Kohli and with each key which is a name we have a list of scores. So, this score is a dictionary whose keys are strings and whose values are lists of numbers. Now suppose we want to add a score to this.

The score is associated with a particular batsman b. So, we have a score s for a batsman b and we want to update this dictionary. Now there are two situations one is that we already have an entry for b in the dictionary in which case we want to append s to the existing list scores of b the other situation is that this is a new batsman, there is no key for b in which case we have to create a key by setting scores of b equal to the list containing s right.We have two alternative modes of operation it is an error to try an append to a non existent key, but if there is an existing key we do not want to lose it by reassigning s. So, we want to append it. So, we want to distinguish these two cases.

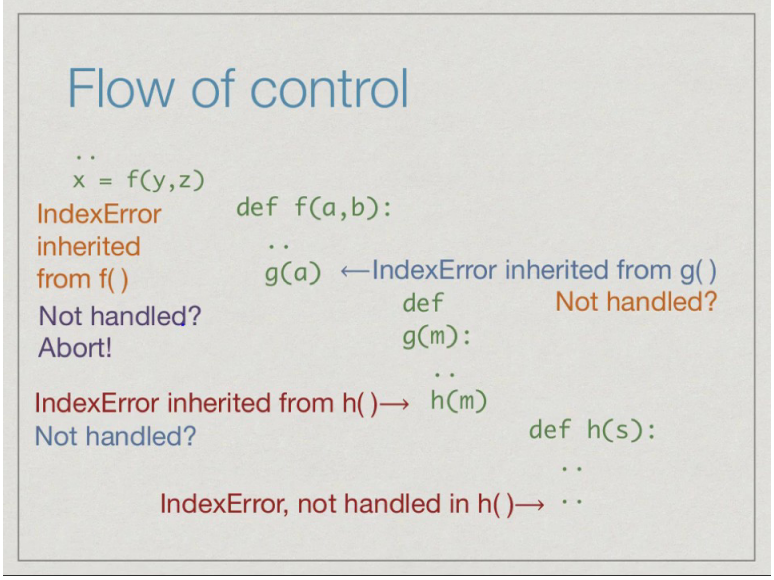
(Refer Slide Time: 12:21)



A standard way to do this in using what we have already seen, is to use the command the the statement in to check whether the value b already occurs as a key in scores. So, we say if b is in the scores, dot keys if we have b as an existing key then we append the score otherwise we create a new entry. So, this is fine, now we can actually do this using exception handling as follows; we try to append it right we assume by default that the b batsman b already exists as a key in this dictionary scores and we try scores b dot append s. What would happen if b is not there? Well python will signal an error saying that this is an invalid key and that is called a key error.

So, we can then revert to this except statement then say oh if there is key error when I try to append s to scores of b then create an entry scores of b is equal to s. So, this is just a different style, it is not saying that one is better than the other, but it is just emphasizing that once we have exception handling under our control we may be able to do things differently from what we are used to and sometimes these may be more clear it is a matter of style you might be further left or the right, but both are valid pieces of python code.

(Refer Slide Time: 13:48)



Let us examine what actually happens when we hit an error. So, suppose we start executing something and we have a function call to a function f, with parameters y and z this will go and look up a definition for the function and inside the definition perhaps. So, this call results in executing this code and this definition might have yet another function called in it call g. This will in turn transfer us to a new definition sorry this should be on the same line g and this might in turn have another function h and finally, when we go to h perhaps this where the problem happens.

Somewhere inside h perhaps there is an index error and where we used this list for example, in h we did not put it on a try block and so, the error is not handled. So, what happens we said is when an error is not handled the program aborts, but the program does not directly abort; this function will abort and it will transfer the error back to whatever called it. So, what will happen here is that this index error will go back to the point where, h was invoked in g.

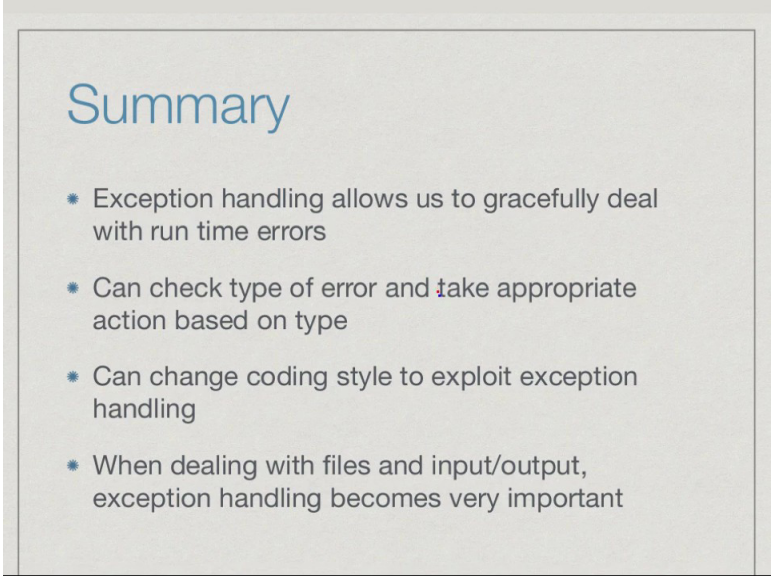
Now, it is as though g has generated an index error calling h has generated an index error. So, an index error is actually now within g because h did not do anything with that error we just passed it back with the error. Now, we have two options either g has a try block, but if g does not have a try block then this error will cause g to abort.

So, what will happen next is that if g does not handle it then this will go back to where g was called in f and likewise if now f does not handle it then it will go back to where f

was called in the main program. So, we keep back going back across the sequence of function calls passing back the error.

If we do not handle it in the function where we are right now, the error goes back this function aborts it goes back and finally, when it reaches the main thread of control the first function of the first python code, that we are executing there if we do not handle it then definitely the overall python program aborts. So, it is not as though the very first time we find an error which is not handled it will abort it will merely pass control back to where it was called from and across the sequence of calls hierarchically we can catch it at any point. So, we do not have to catch the error at the point where its handled we can catch it higher up from the point that is calling us.

(Refer Slide Time: 16:23)



To summarize exception handling allows us to gracefully deal with run time errors. So, python when it flags an error tells us the type of error and some diagnostic information. Using a try and except block, we can check the type of error and take appropriate action based on the type. We also saw with that inserting a value into a dictionary example that we can exploit exception handling to develop new styles of programming and finally, what we will see is that, as we go ahead and we start dealing with input output and files exceptions will be rather more common as we saw earlier one of the examples we mentioned was a file is not found or a disk is full.

Input and output inherently involves a lot of interaction with outside things outside the program and hence is much more prone to errors and therefore, is useful to be able have this mechanism within our bag of tricks.