



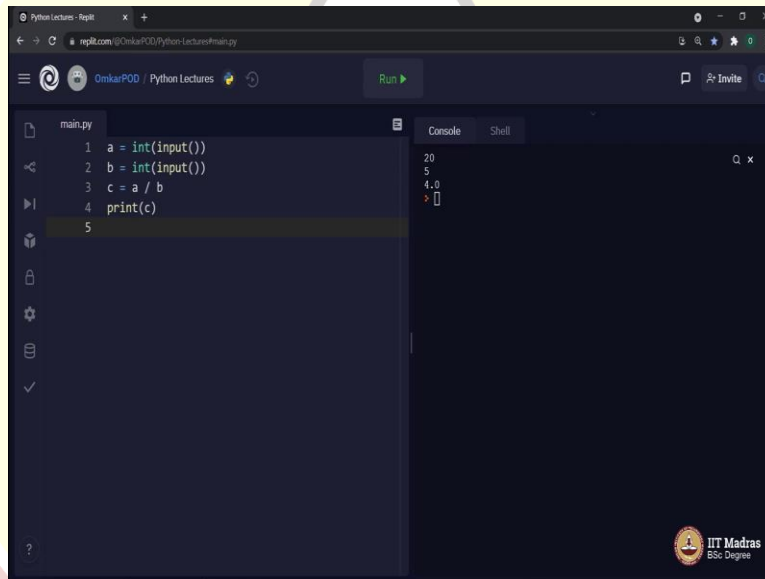
# IIT Madras

ONLINE DEGREE

**Programming in Python**  
**Professor Sudarshan Iyengar**  
**Department of Computer Science & Engineering**  
**Indian Institute of Technology, Ropar**  
**Exception handling**

(Refer Slide Time: 00:16)

Hello Python students. Let us start this lecture with a question which you all must be having from the moment when you heard this term exception handling. What do you mean by exceptions and why Python programs have exceptions? Is it similar to errors or is it something else? So, let us first address these common questions about exceptions and then we will move to handling part of it.

A screenshot of a web-based Python IDE. The editor shows a file named 'main.py' with the following code:

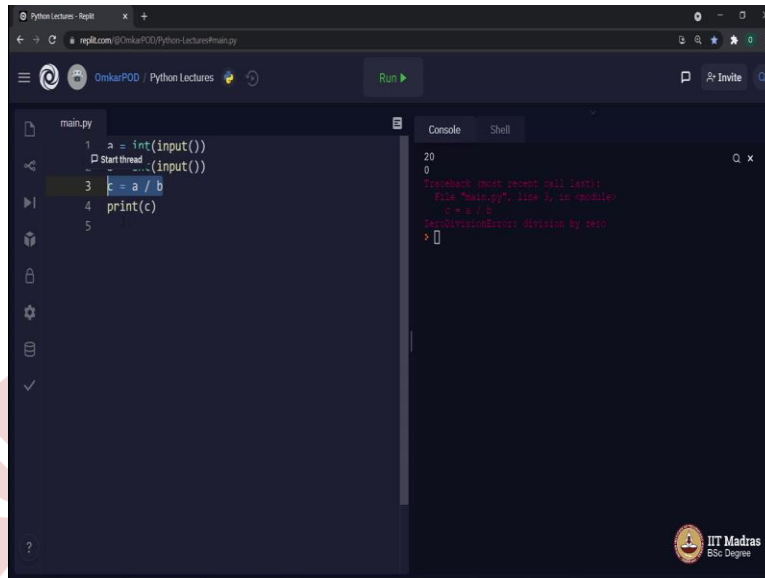
```
1 a = int(input())
2 b = int(input())
3 c = a / b
4 print(c)
5
```

The 'Run' button is visible above the editor. To the right, the 'Console' tab is active, showing the input and output:

```
20
5
4.0
```

The IDE interface includes a file explorer on the left, a settings gear, and a search icon. The bottom right corner features the IIT Madras logo and text.

Observe this particular piece of Python code carefully. As you can see, it is very simple Python program where we are taking two inputs from user, dividing those numbers, storing the result in third variable and printing the third variable. Let us execute, let us say 20, 5 and we have the division as 4.0, which is perfectly fine. And as you can see, there is no error in this particular program. Computer is executing this code and also giving us the expected output. And this will continue to happen for every single piece of input which we can give to this particular program except one situation and which is, correct.



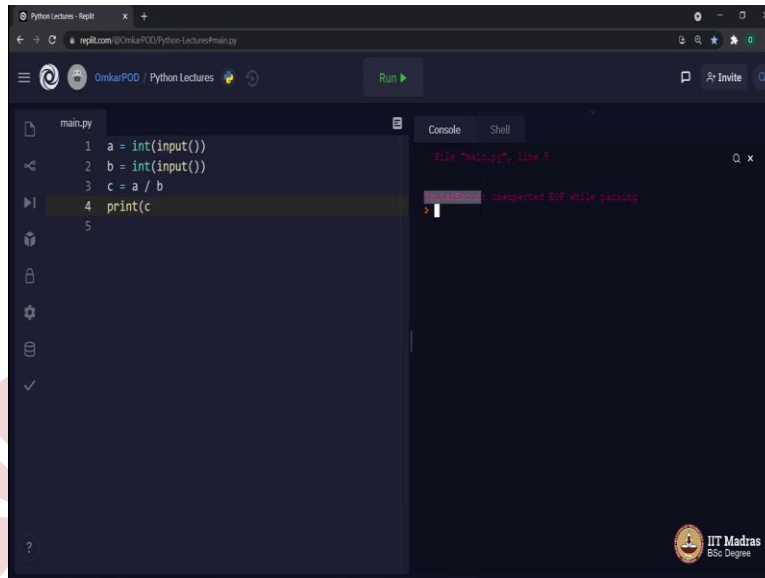
The screenshot shows a web-based Python REPL interface. The code editor on the left contains the following Python code:

```
1 a = int(input())
2 b = int(input())
3 c = a / b
4 print(c)
5
```

The 'Run' button is highlighted. The console on the right shows the input '20' and '0', followed by a `ZeroDivisionError: division by zero` exception. The error message is displayed in red text. The interface also includes a 'Shell' tab and a 'Python Lectors' header.

If we execute it again and instead of 20 and 5, let us say if we gave input as 20 and 0. This is the place computer will give us a particular error message. It says `ZeroDivisionError`, division by 0, because in this case, the value for variable `b` is 0. And mathematically, we cannot have 0 as divisor. Hence, computer is saying I cannot execute this particular line number 3, even though it is syntactically correct, `c` is equal to `a` divided by `b`. It is a correct Python statement.

But still I cannot execute it only because the value of `b` is not an appropriate one for this particular operation. And this is what the exception is all about, which means this particular Python program executes in every single situation for every single input for `a` and `b` except only when `b` is equal, equal to 0. And that is why these type of errors are referred as exceptions, because these errors has nothing to do with our code. The mistake is somewhere else. So, as a programmer, we have not made any mistake. The mistake is in input, at least in this particular case.



The screenshot shows a web-based Python REPL interface. The code editor on the left contains the following Python code:

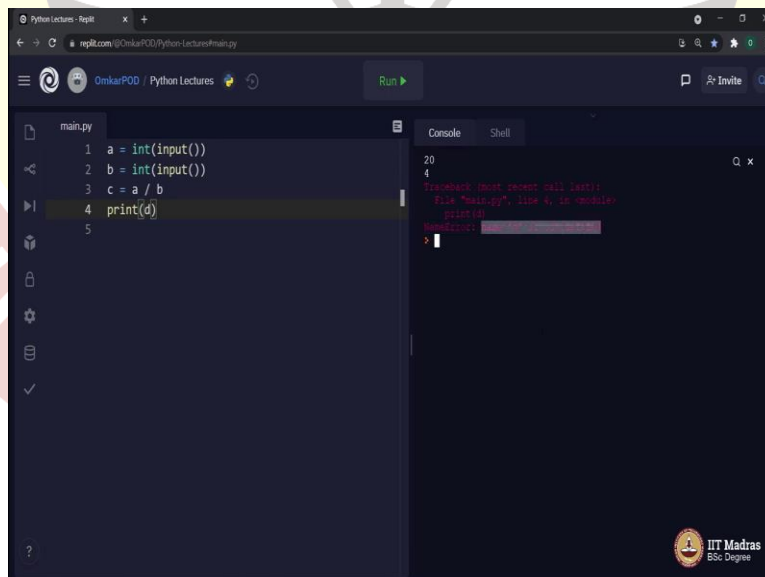
```
1 a = int(input())
2 b = int(input())
3 c = a / b
4 print(c)
5
```

The 'Run' button is highlighted. The console on the right displays the error message:

```
File "main.py", line 4
    c = a / b
          ZeroDivisionError: division by zero
```

The IIT Madras logo is visible in the bottom right corner.

Whereas let us say if I forget this particular bracket and executed this program, then it says `SyntaxError`. This means this is a mistake from our end. This is a mistake in the program. But in this case, there is no mistake in the program. The mistake is in this particular 0 input. This is one example of an exception. Let us look at a few more such examples of exceptions then we will see how to avoid such exceptions or can we do something else about these exceptions in order to avoid these kind of error messages. Even though the input is like this.



The screenshot shows the same web-based Python REPL interface. The code editor on the left contains the following Python code:

```
1 a = int(input())
2 b = int(input())
3 c = a / b
4 print(d)
5
```

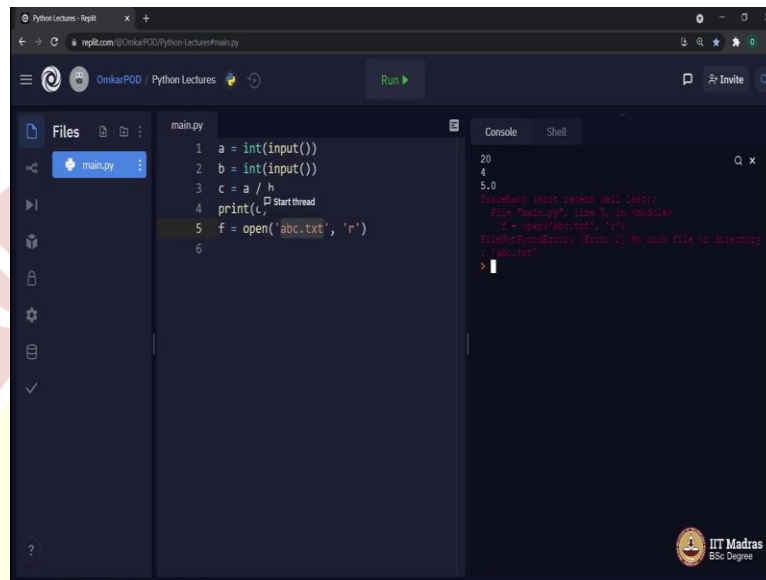
The 'Run' button is highlighted. The console on the right displays the error message:

```
20
4
Traceback (most recent call last):
  File "main.py", line 4, in module:
    print(d)
NameError: name 'd' is not defined
```

The IIT Madras logo is visible in the bottom right corner.

Let us say by mistake instead of `c` I use `d` over here. And now let us say 20 and 4. Both are valid inputs and I executed the code. It says, `NameError`. Name `d` is not defined. Once again, there is nothing wrong as such in this particular program. `Print d` is a valid statement. The problem is the

variable `d` is not defined. So, as per Python syntax, there is nothing wrong in this code once again, but still we will end up getting this kind of error because variable `d` is not defined. Once again, this is another type of exception which we can handle in some way.



The screenshot shows a web-based Python IDE (Replit) with a file named `main.py`. The code in the editor is as follows:

```
1 a = int(input())
2 b = int(input())
3 c = a / b
4 print(c)
5 f = open('abc.txt', 'r')
6
```

The console output shows the results of the first three lines of code:

```
20
4
5.0
```

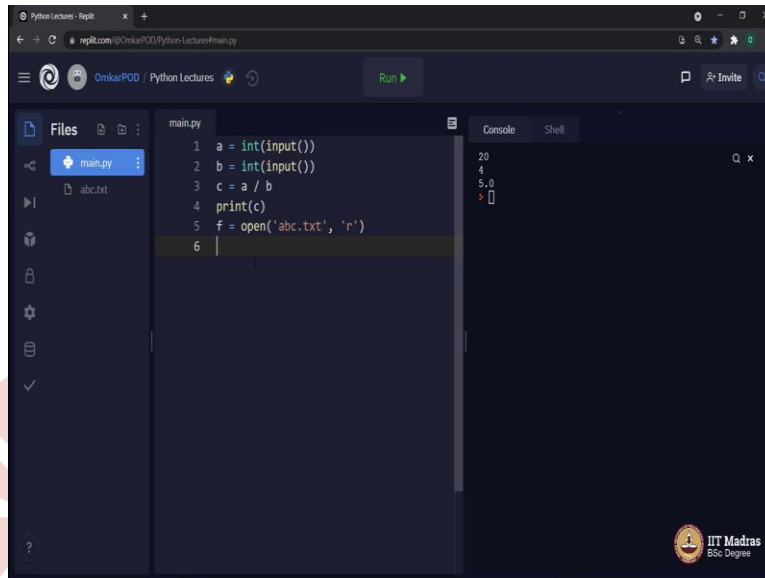
Following the output, a red error message is displayed in the console:

```
Traceback (most recent call last):
  File "main.py", line 5, in <module>
    f = open('abc.txt', 'r')
FileNotFoundError: [Errno 2] No such file or directory: 'abc.txt'
```

The error indicates that the program attempted to open a file named `abc.txt` in read mode, but the file does not exist in the current directory.

Let me correct this code and try something else now, `f` is equal to open "`abc.txt`" in read mode and this is it. I am not going to do any further operations with respect to this particular code. And the reason is, if I look here, there is no such file called "`abc.txt`", which means I am asking computer to read a particular file which does not exist in this particular files list in repl.it. So, let us first execute this then I will try to explain what happens in such a case.

Once again, let us say 20 and 4, we got the output as 5.0 for this print statement, but as soon as computer reach to this particular line number 5, where we are trying to open this particular file "`abc.txt`", which does not exist in this particular directory, we are getting an error called `FileNotFoundError`, no such file or directory "`abc.txt`". Once again, there is absolutely nothing wrong with this particular line. This is exactly how we read a text file in Python.

A screenshot of a web-based Python IDE (Replit) showing a file named 'main.py' with the following code:

```
1 a = int(input())
2 b = int(input())
3 c = a / b
4 print(c)
5 f = open('abc.txt', 'r')
6
```

The console on the right shows the output of the code: '20', '4', and '5.0'. Below the output, there is a red error message: 'FileNotFoundError: [Errno 2] No such file or directory: \'abc.txt\''. The background of the image features a large, faint watermark of the IIT Madras logo.

So, the problem is not with Python code. The problem is over here. The file is not available. If I create a file over here, let us say "abc.txt" and now if I execute it once again 20, 4, now there is no error. So, going back to my previous point, there is nothing wrong in this code. The problem was this file was not available like this.

And in such a case, we will get an error message which says `FileNotFoundError`. This is another type of exception only. You will not come across such errors on a regular basis. This will be an exceptional situation where you are trying to read a file which is not available at least at that particular location.

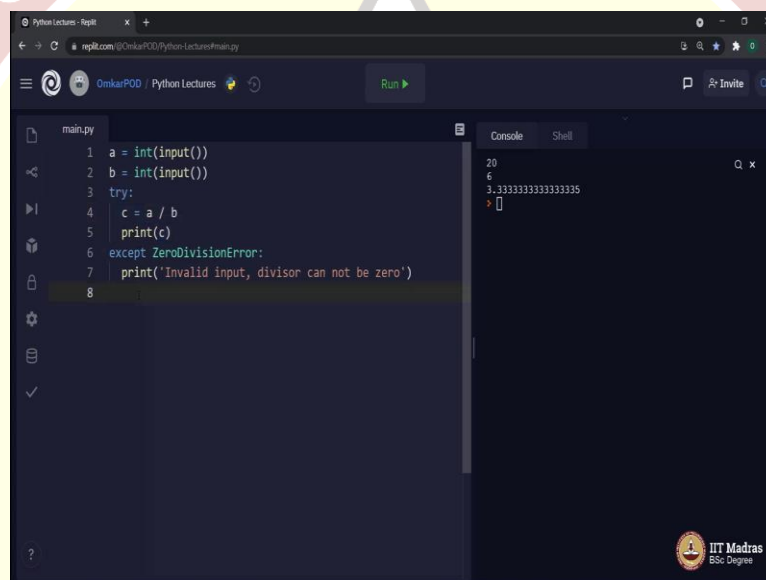
So, before moving to handling part of it, let me repeat the 3 types of errors which we saw; first was because of divide by 0; second variable not defined; and then third file not found. All such types of errors or more precisely exceptions can be managed, can be handled without such an error message from computer.

Now, you must be thinking, how can we handle this particular situation where computer is trying to read a file which does not exist? How can we solve this problem? And the answer is, obviously we cannot resolve this. But at least we will not end up getting this kind of a message. We can avoid this thing. That is what exception handling is all about.

We cannot avoid this situation. We cannot force computer to divide a number by 0, which is not possible. That is not the motive behind exception handling. The motive is our program should

not throw an error like this. It should terminate with some proper message which will help user who can correct its input for a next run.

For example, in this run if we say 20, 0, then instead of getting this error from computer, we can show some guided message to the user, maybe saying your input is wrong please enter a valid input. Some kind of informative message is the way you handle these exceptions. So, now enough of this. Now, let us go to the exact code through which we can manage all these exceptions.

A screenshot of a web-based Python IDE interface. The editor shows a file named 'main.py' with the following code:

```
1 a = int(input())
2 b = int(input())
3 try:
4     c = a / b
5     print(c)
6 except ZeroDivisionError:
7     print('Invalid input, divisor can not be zero')
8
```

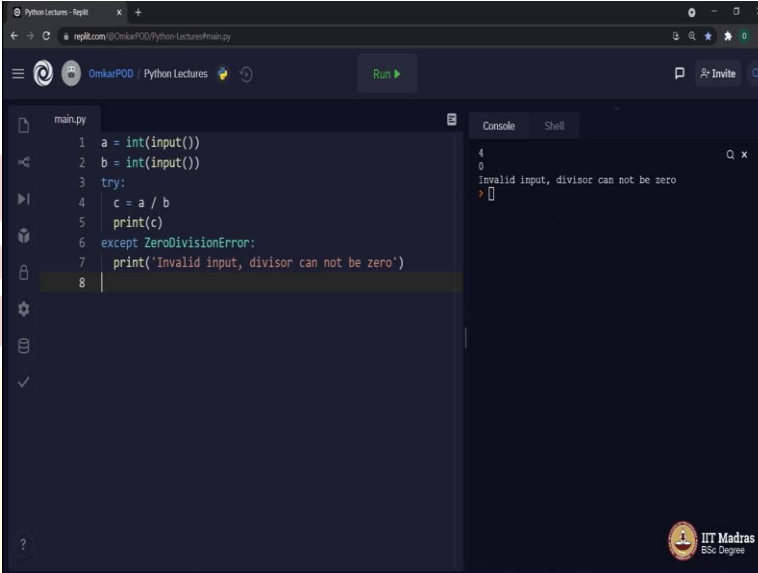
The interface includes a 'Run' button and a 'Console' panel on the right. The console shows the output of the program: the number '20' followed by a newline, then '6', and finally a long string of '3's representing a repeating decimal. The background features a large, semi-transparent watermark of the IIT Madras logo.

And the way this code is written is like, it says, try, try what,  $c$  is equal to  $a$  divided by  $b$ . Let us delete this first. And if this is possible, then print that value as well. If you are able to do this, that is well and good, if not which means if there is an exceptional case and the keyword used for that is except, what you will do, except what, except as in what is that exceptional case. And the exceptional case is this, `ZeroDivisionError`. Let us say except `ZeroDivisionError` print invalid input, divisor cannot be 0, some message like this.

Now, let us execute this program again with 0 input and non-zero input. When we say 20 and 6, it gives us the expected output by executing these two lines 4 and 5. Let us execute again. Let us say 4, 0. And now instead of computer throwing some error, we terminated our program with a valid output. And the output is an indicative message to the user which says your input was



wrong. You should not have 0 as a value for the divisor. And this is exactly how we handle exceptions.



The screenshot shows a web-based Python REPL interface. The code editor on the left contains the following Python code:

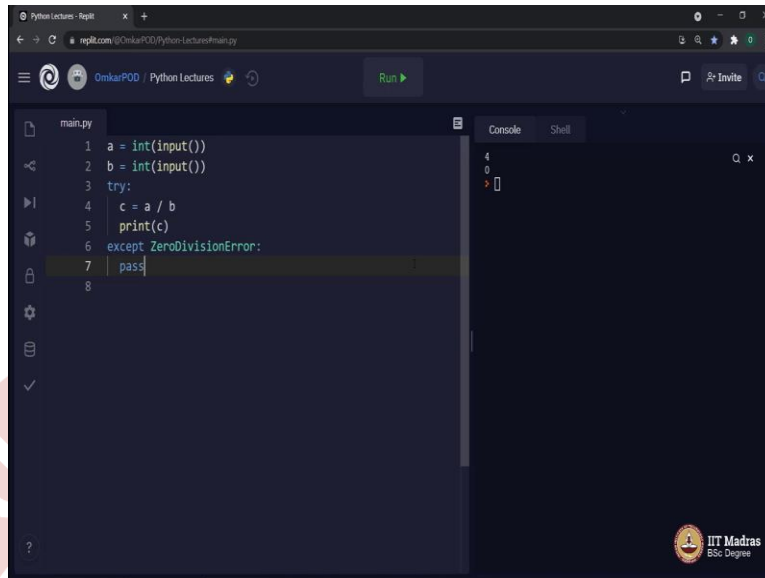
```
1 a = int(input())
2 b = int(input())
3 try:
4     c = a / b
5     print(c)
6 except ZeroDivisionError:
7     print('Invalid input, divisor can not be zero')
8
```

The console on the right shows the output of the program. It displays the number 4, followed by a newline, and then the message "Invalid input, divisor can not be zero". The interface includes a "Run" button and a "Shell" tab.

And as I said earlier, we cannot force computer to do 4 divided by 0. That is not possible. But at least we can tell computer whenever you come across that situation do not throw an error, do not print your regular error message, instead give an indicative message, give an helpful message to the user so that user will avoid such mistakes in future runs of this particular code. So, going back to syntax, whatever code which may or may not result into an error that particular code block should be written inside this new block called try.

As in, we are telling computer, try to execute this. If you are able to do it, that is good. If not, which means if you come across an exceptional situation, which is `ZeroDivisionError`, please print this message and then terminate without any error. As you can see, this is not an error. This is simple print function. Even though we do not write it and we say pass, still it will work and program will terminate without any error and of course, without any output.

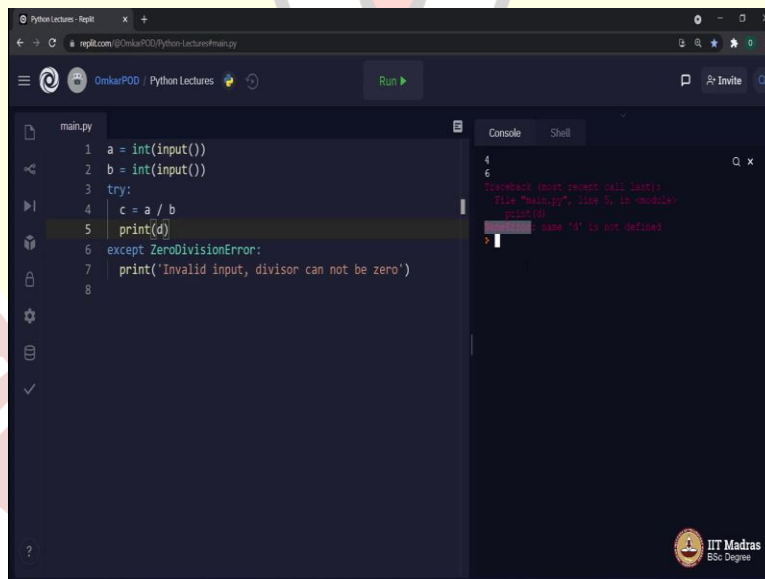




```
1 a = int(input())
2 b = int(input())
3 try:
4     c = a / b
5     print(c)
6 except ZeroDivisionError:
7     pass
8
```

The console shows the input '4' and '0', and the output is an empty line.

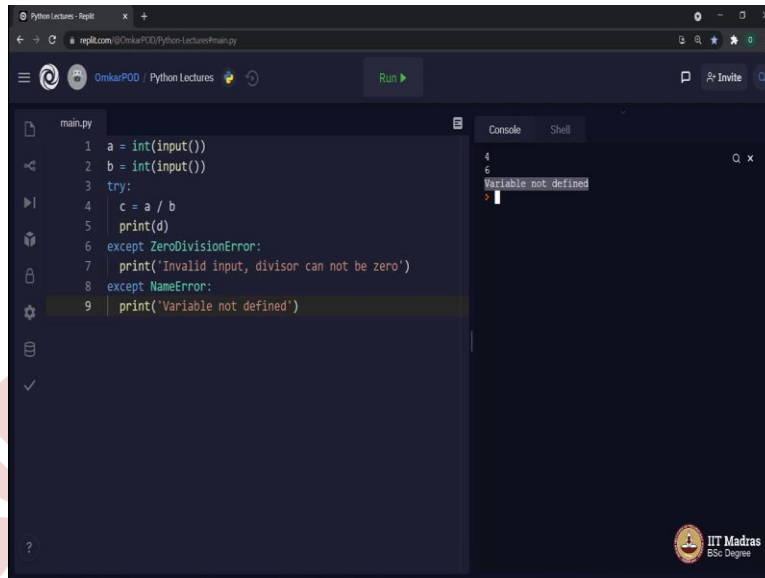
But this will not serve our purpose. The purpose behind this whole concept of exception handling is to indicate user, is to help the user to avoid such mistakes. Hence, it is best to have such a print statement which will be a helpful message. Similarly, instead of this c if I have d and now I executed this program with let us say 4 and 6, now it says NameError.



```
1 a = int(input())
2 b = int(input())
3 try:
4     c = a / b
5     print(d)
6 except ZeroDivisionError:
7     print('Invalid input, divisor can not be zero')
8
```

The console shows the input '4' and '6', and the output is a NameError: name 'd' is not defined.

Now, you must be thinking why. I did exception handling still I am getting some kind of error. And the reason is now this line is not giving us any exception because now we have valid inputs 4 and 6, but this time, this particular line, print d is giving us that error. And the type of error is now different.



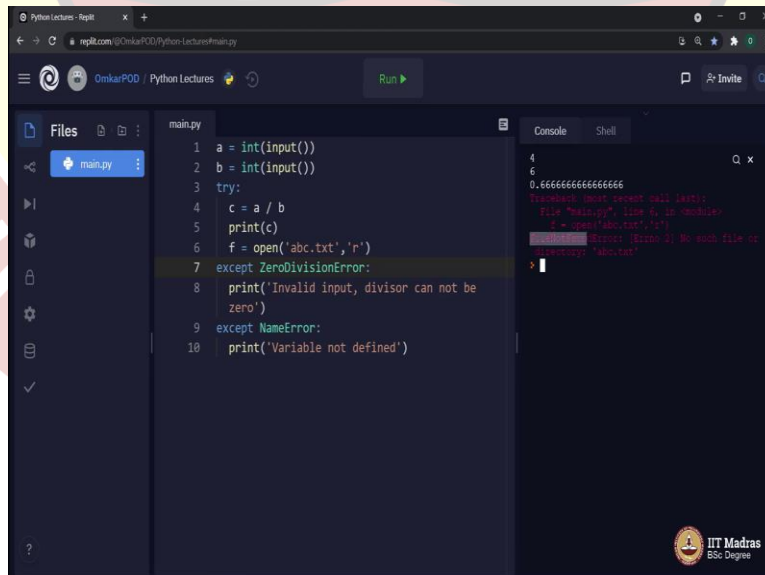
The screenshot shows a Python REPL window titled "Python Lectures - Repl.it". The code in the editor is as follows:

```
1 a = int(input())
2 b = int(input())
3 try:
4     c = a / b
5     print(d)
6 except ZeroDivisionError:
7     print('Invalid input, divisor can not be zero')
8 except NameError:
9     print('Variable not defined')
```

The console output shows the execution of the code, resulting in a `NameError` because the variable `d` is not defined:

```
4
6
Variable not defined
>
```

We handled one type of error which is `ZeroDivisionError`, but now we are getting a different type of error which is `NameError`. In such a case, we should handle that error as well, except `NameError` and we can print some message over here, let us say, variable not defined. Let us execute again. And now we are getting that helpful message which says variable not defined instead of an error from computer.



The screenshot shows the same Python REPL window, but the code has been updated to include a `FileNotFoundError` exception:

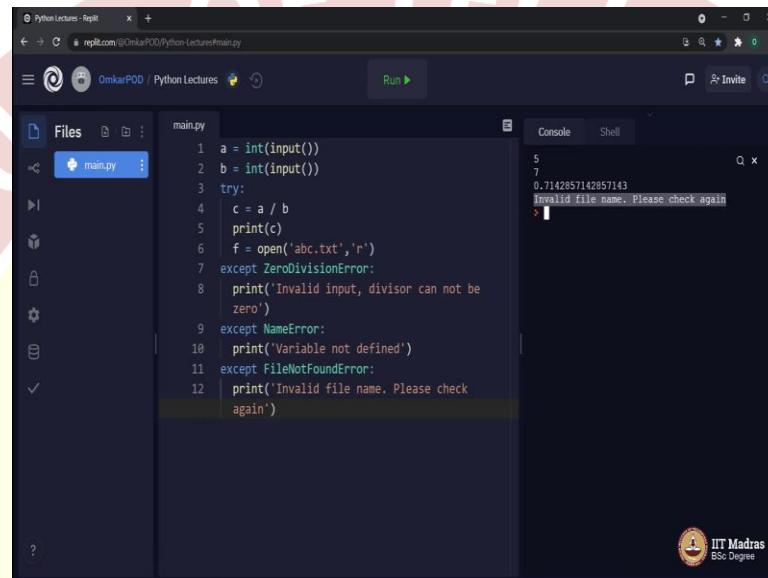
```
1 a = int(input())
2 b = int(input())
3 try:
4     c = a / b
5     print(c)
6     f = open('abc.txt', 'r')
7 except ZeroDivisionError:
8     print('Invalid input, divisor can not be zero')
9 except NameError:
10    print('Variable not defined')
```

The console output shows the execution of the code, resulting in a `FileNotFoundError` because the file `abc.txt` does not exist:

```
4
6
0.6666666666666666
Traceback (most recent call last):
  File "main.py", line 6, in <module>
    f = open('abc.txt', 'r')
FileNotFoundError: [Errno 2] No such file or directory: 'abc.txt'
>
```

Now, let us see what happens in the case of the third exception as well. First, let me make it `c`. Now, `f` is equal to open "abc.txt" in read mode. Once again, just to make sure, we do not have

any "abc.txt", available over here. Let us execute this code. Some valid inputs. We got the output. But now once again, we got an exceptional message which says FileNotFoundError. Because once again, we told computer or we have managed ZeroDivisionError, we have managed NameError, but we have not written any code to manage FileNotFoundError.

A screenshot of a Python IDE window titled 'Python Lectures - Replit'. The code editor shows a file named 'main.py' with the following Python code:

```
1 a = int(input())
2 b = int(input())
3 try:
4     c = a / b
5     print(c)
6     f = open('abc.txt', 'r')
7 except ZeroDivisionError:
8     print('Invalid input, divisor can not be
9         zero')
9 except NameError:
10    print('Variable not defined')
11 except FileNotFoundError:
12    print('Invalid file name. Please check
13        again')
```

The console on the right shows the execution output: '5', '7', and '0.7142857142857143'. Below this, the error message 'Invalid file name. Please check again' is displayed, indicating that the program has reached the FileNotFoundError exception handler.

In order to manage that error as well, we should write except FileNotFoundError. Maybe we can say print invalid file name. Please check again. And this time, we will get that particular message invalid file name, please check again. And this is exactly how you get those error messages when you use different softwares or different websites where you try to do something and computer gives you a pop-up dialog box or some kind of a instructive message which helps you identify your mistake.

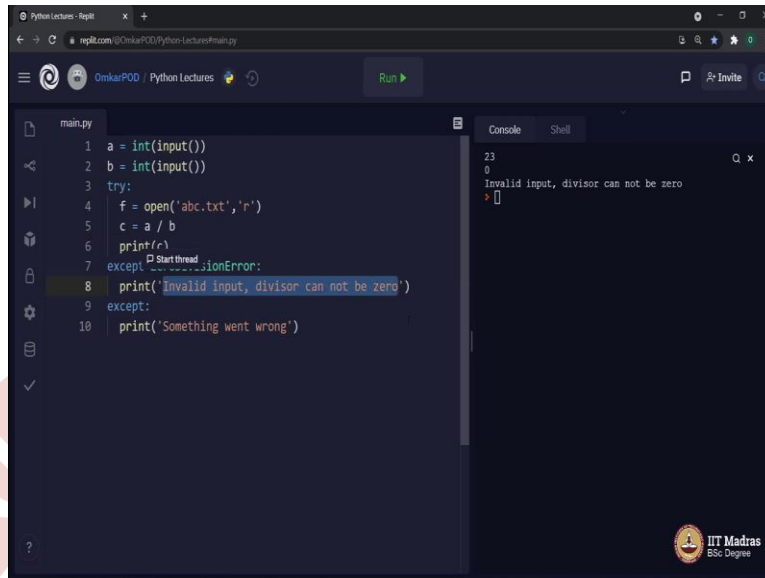
I am sure you must have used simple Google form where a particular question is mandatory. And without giving any answer for that question, you click submit and you get a pop-up message which says this is a mandatory question or this is a compulsory question where you have to give your answers without that you cannot submit. So, those kind of informative warnings are nothing but such print statements where the intention behind those warning messages is to help user rather than throw up some random errors which user may not even understand.

I hope you all must have understood the idea and the logic behind this whole concept of exception handling and the usefulness of this concept. And as I said earlier, because of all this, this is considered as a industry standard. Good programmers never write any program in any programming language without using proper exception handling in it. Now, we are set with the general idea of exception handling.

But still you must be wondering what if I come across some exception which is not listed over here, something other than `ZeroDivisionError`, something other than `NameError`, something other than `FileNotFoundError`, because we may end up getting some different error message, some other exceptions, which is not even listed over here. Then in that case computer will give me a regular error. Then what should I do? In such a case, there is a way where we can say simply `except` and do not mention any specific type of exception. And then we can say `print`. Something went wrong, a generic message.

Now, in this case, first computer will check whether a specific exception is listed over here or not computer comes with some error. It will check whether that exception matches with `ZeroDivisionError`, no, then it will check with `NameError`, no, then it will check with `FileNotFoundError`, no, then ultimately it will execute this particular block. And this is required, because in Python there are approximately around 30 such exceptions and no programmer can remember or write a code to handle all such exceptions.

So, it is advised to add all those exceptions which you think may appear during execution of these 3 lines. But still, there are certain things which are out of our control. And in such a case, it is best way to add such a particular common exception handling block at the end, which will handle anything which is outside these 3 types. So, now moving on with this exception handling concept now let me change this code a little bit and then you tell me what will be the output of this particular program.

A screenshot of a Python REPL window titled "Python Lectures - Repl.it". The code in the editor is as follows:

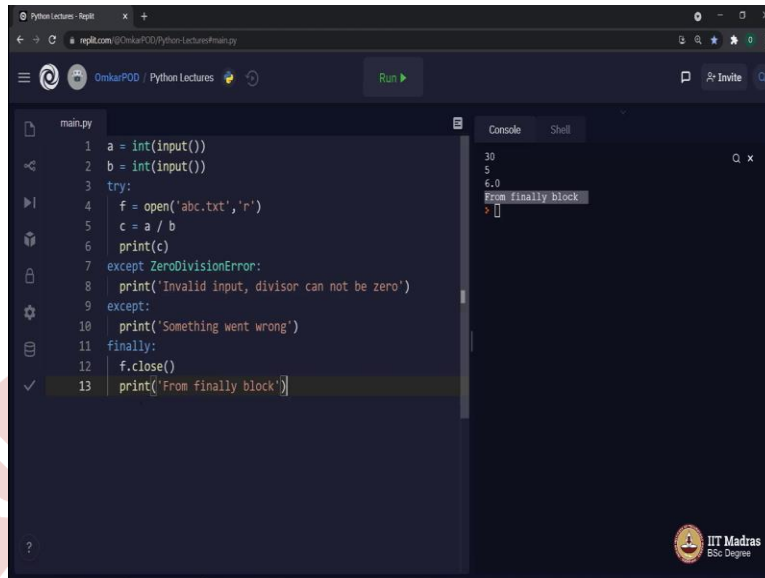
```
1 a = int(input())
2 b = int(input())
3 try:
4     f = open('abc.txt', 'r')
5     c = a / b
6     print(c)
7 except ZeroDivisionError:
8     print('Invalid input, divisor can not be zero')
9 except:
10    print('Something went wrong')
```

The console output shows the input "23" and "0", followed by the error message "Invalid input, divisor can not be zero". The IIT Madras logo is visible in the bottom right corner of the window.

I have decided to move this particular line from here to here, just a change in sequence. First, I will open this "abc.txt", then I am trying to divide a by b store in c, then print c. Now, as I am printing c over here, this is not required. Let us remove this. But let us retain ZeroDivisionError and FileNotFoundError and let me add that file as well, "abc.txt". Now, as you know, this particular line will not give us any exception, but still there is a possibility of exception because of a divided by b. So, in this case, we can even remove this part.

Now, let us execute this code. Let us say 23, 0, and as expected, we got this particular exception which says invalid input divisor cannot be 0, which is absolutely correct. There is nothing new about this. And that is when I will ask you this question, what happened to this f, because this particular line number 4 executed successfully as we know because we created "abc.txt", which means computer opened this particular file in read mode and then when computer tried to execute this particular line number 5, it came across an exception and it printed this particular warning message.

But what happened to this f. It is still opened. And if you remember, it is always a good practice to close a particular opened file like this. Even though this particular statement being there or not, if we get exception here, these two lines anyway will not execute then what happens to this particular file. This is still open within Python program, because no one closed it because this particular line did not get executed at all.



```
main.py
1 a = int(input())
2 b = int(input())
3 try:
4     f = open('abc.txt', 'r')
5     c = a / b
6     print(c)
7 except ZeroDivisionError:
8     print('Invalid input, divisor can not be zero')
9 except:
10    print('Something went wrong')
11 finally:
12    f.close()
13    print('From finally block')
```

Console

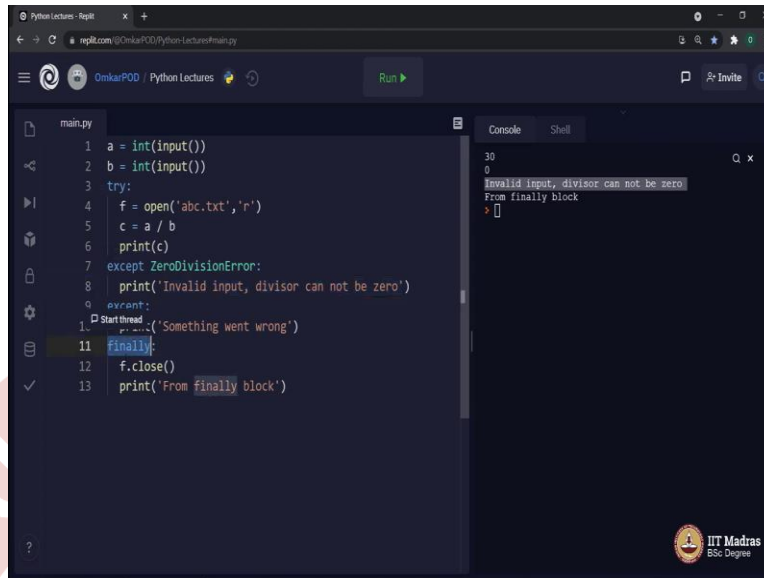
```
30
5
6.0
From finally block
```

Now, what. What to do in such a case? In such a case, Python has a provision and that provision is called as finally block and this is the place where we close any opened resources instead of closing it over here. Now, let us see what is this particular finally block is all about.

Finally is a special block which executes irrespective of whether we get exception or we do not get exception. So, irrespective of occurrence of exception, this particular block will always execute before terminating that particular program. So, let us execute and we will see. Just to be sure, let us say some print statement here which says from finally block, let us execute.

This time, let us say 30 and 5, no exception, we got the output and it is from finally block, which means this statement got executed. And if this statement got executed, that means this statement also got executed, which means we closed this opened file.





```
main.py
1 a = int(input())
2 b = int(input())
3 try:
4     f = open('abc.txt','r')
5     c = a / b
6     print(c)
7 except ZeroDivisionError:
8     print('Invalid input, divisor can not be zero')
9 except:
10    print('Something went wrong')
11 finally:
12    f.close()
13    print('From finally block')
```

```
Console
30
0
Invalid input, divisor can not be zero
From finally block
```

Now, let us check second version of it, where our input is, let us say 30 and 0. In such a case, we will get exception. And once again, even though it gives this particular warning message, still after this particular print, computer jumps to this finally block, executes `f dot close` and print this particular message which says from finally block. And this is exactly what I was trying to explain earlier.

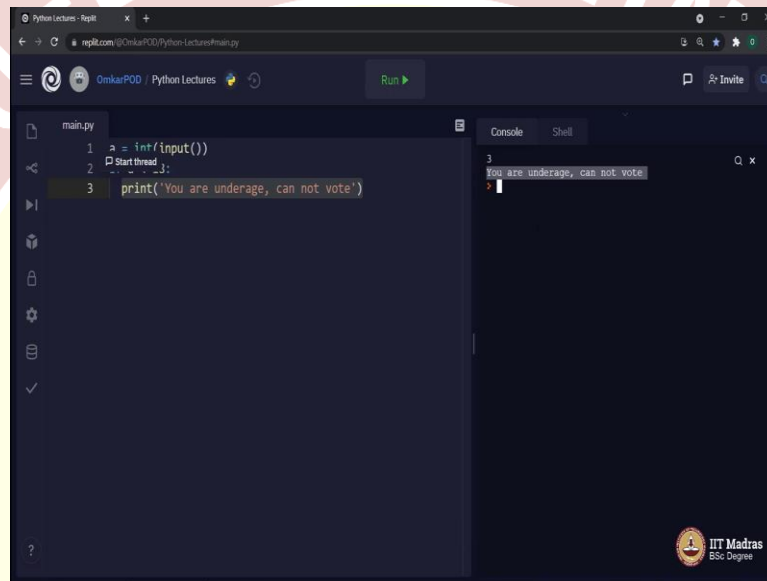
Either this code executes without exception or we might get exception. Irrespective of that, this particular block will execute. And that will make sure that this closing of a file happens properly. Hence, it is a standard programming practice to close all such resources or to execute all such important operations in the finally block. So that, even in case of an exception, we will be able to execute these lines.

So, now, I have one follow up question with this. And now it is your responsibility to find answer to this particular question. And the question is, what happens if exception occurs in a code which is written in finally block? I hope you all must have understood this concept of exceptions.



So, now, let us move to a completely different type of exception, a type of exception, which does not exist in Python. As I said earlier, in Python, there are roughly around 30 such exception classes or such exception types which we can use with except block, but what if I want to create an exception of my own with some condition which I want.

(Refer Slide Time: 27:42)



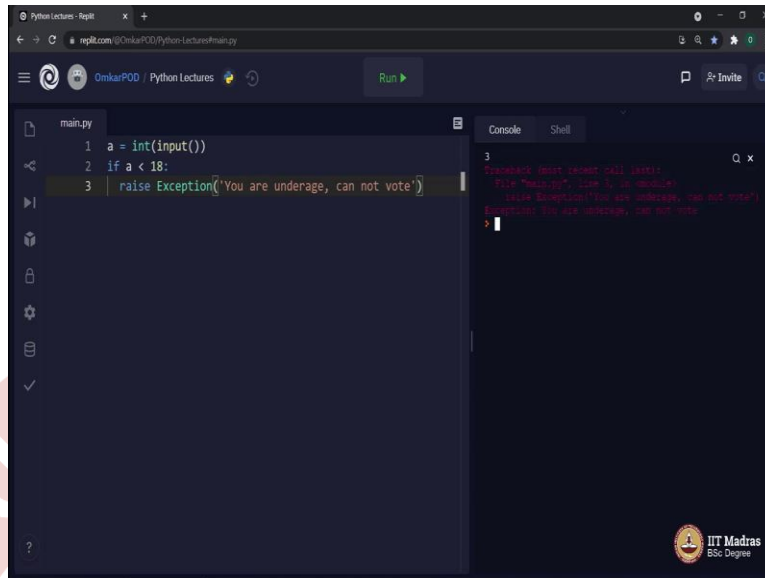
```
main.py
1 a = int(input())
2 start thread
3 print('you are underage, can not vote')
```

Console

```
3
you are underage, can not vote
```

And that is why I said, let us explore some exceptions which are not even known to Python. Let us look at this particular code, a is equal to int of input, a is less than 18, print 'you are underage, cannot vote'. Let us execute, let us say 50, no output, as expected. But when I say let us say 3, it will print this particular message, 'you are underage, cannot vote'.

But in this case, this is just a print function where we are printing one string. I do not want that. In such a case, computer should give some kind of a warning to its user saying you are not supposed to vote because you are underage. And this is the place where we can create our own exceptions and such type of exceptions are referred as user defined exceptions.



```
Python Lectures - Repl
repl.it.com/@OmkarPOD/Python-Lectures/main.py
OmkarPOD / Python Lectures
Run
main.py
1 a = int(input())
2 if a < 18:
3   raise Exception('You are underage, can not vote')
Console
3
Traceback (most recent call last):
  File "main.py", line 3, in <module>:
    raise Exception('You are underage, can not vote')
Exception: You are underage, can not vote
IIT Madras
BSc Degree
```

So, as a programmer, we will create our own exceptions rather than using some existing exceptions which are available in Python. Because according to Python, a less than 18 has no meaning whatsoever because this is the condition which we humans develop, which has nothing to do with computer or Python. And this particular number will also vary depending on the type of voting and from country to country.

So, in such a case, instead of this print, I will write something like, raise exception. And then if I execute it with same input, let us say, 3, it will throw an exception. And this is the difference between a print message and an exception. Now, this time, it is not the Python or not the computer who gave this particular error message or this particular exception, it is us as in our program triggered this particular exception.

We told computer if this condition satisfies it is your job to raise an exception and this is what we are doing which says exception, 'you are underage, cannot vote' or whatever message whichever you want to print.

So, before closing this lecture, let me quickly revise whatever we have seen so far. Exceptions are a different type of errors which are not because of something wrong in the program, those errors occur because of something went wrong which is outside that program. It can be the input

or some external resource like a file. So, in an ideal scenario, the program runs perfectly fine. But in that one odd, as in one exceptional case, the program throws some kind of error. And that kind of error is referred as exception.

And using two special keywords, two special blocks, try and except, we can handle such exceptions. If we want to make sure some lines of code has to be executed irrespective of exception or no exception, we should write those lines of codes in a third type of a block called finally, which always executes. And then this was our last concept where we saw user defined exception where we on our own using our Python program created some exceptions based on some condition like this.

So, now it is your responsibility to go back to all those Python programs which you have done so far in last 10 weeks and try to convert those programs into a more standardized code using exception handling. And for this, you do not even require any additional problem statements. You can use n number of problem statements which are already available with you from last 10 weeks. Thank you for watching this lecture. Happy learning.

