NOVEMBER 1, 2022 / #ERROR HANDLING

How to Handle Errors in Python – the try, except, else, and finally Keywords Explained

By P S Mohammed Ali

ADVERTISEMENT

"It's hard enough to find an error in your code when you're looking for it; it's even harder when you've assumed your code is error-free."

— Steve McConnell

Errors are inevitable in a programmer's life. In fact, while writing programs, errors can be really helpful in identifying the logic bugs and syntax errors in your code.

But, if you can anticipate an error in a particular set of code lines before execution, then you can handle those errors and make the code error free.

Why Error Handling is Important

Handling or taking care of errors that you're aware of helps the code flow and execute smoothly without any interruptions. If errors occur in any lines of code, the error handling takes care of them and then the code resumes execution.

Let's take an example and understand why we need error handling:

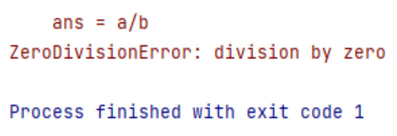
```
a = 12
b = 6
result = a/b
print(result)
print("I have reached the end of the line")
```

From the above code, what do you expect ?. Well, the `result` variable prints `2.0` and on the next line, the console prints `I have reached the end of the line`. That's what we are excepting.

Let's change value of `b` from `b = 6` to `b = 0` and run.

```
1. a = 12
2. b = 0
3. result = a/b
4. print(result)
5. print("I have reached the end of the line")
```

When this code gets executed, we will get an error as below:



```
    ans = a/b
ZeroDivisionError: division by zero

Process finished with exit code 1
```

Error message displayed when b is set to 0

The code didn't print the `result` value and it also didn't print `I have reached the end of the line`

The above error messages displays `division by zero`, which means that if we try to divide any number by `0`, we will get this error.

The problem is in line `3`. Even though the code didn't print the `result` value, it should have printed `I have reached the end of the line`. But, it didn't – why ?

Well, because the Python interpreter stopped at line 3 when the `a` got divided by `0`. At this point, it raised an error in the console and exited the code.

ADVERTISEMENT

One of the naive solutions to solve this problem can be hard coding the values. If the values of `a` and `b` are hard-coded, then running the code will solve this error to some extent.

But the other major problem that may arise is when a user wants to give values of `a` and `b` at the time of execution.

```
a = int(input())
b = int(input())
result = a/b
print(result)
print("I have reached the end of the line")
```

At this time, there's a high probability that the user will give `0` as the input to `b`. In order to handle this kind of expected error, we will use certain methods of error handling in order to avoid interrupting the execution flow (even though the user might give any invalid input like `0` as input to `b`).

How to Use the Try and Except Keywords in Python

Any lines of code that are more prone to errors are kept in `try` block. If any error occurs, then the `except` block will take care of those errors.

The code structure looks something like this:

ADVERTISEMENT

```
try:
    code that may/may not produce errors
except:
    when error arises, then this block of code exceutes.
    Otherwise, this block of code doesn't exceute
```

Let's come back to the standard example that we have been discussing.

We will handle the `division by zero` problem using `try/except` blocks.

Let's insert the lines of code that have a high probability of producing an error. In our case, lines `1-4` of our code have high potential to produce an error. So, we put these four lines in the `try` block:

```
try:
  a = int(input())
  b = int(input())
  result = a/b
  print(result)
except:
  print("We caught an error")

print("I have reached the end of the line")
```

Now, when we give `b` a value of `0`, an error occurs. So, the `except` block executes and the interpreter prints `We caught an error` and comes out of the except block and resumes printing `I have reached the end of the line`.

On the other hand, when we give `b` a non-zero value, then we print the `result` value. The code comes out of the try block and resumes printing `I have reached the end of the line`.

In both cases, we are able to execute until the last line of code without any interruptions.

ADVERTISEMENT

Apart from try and except, it's quite important to understand the `else` and `finally` keywords that come along with `try` and `except` .

The `else` block of code comes after the `try` and `except` blocks and executes when no error is raised from the `try` code block. Similarly, the `finally` code block comes after the `else` block and executes whether errors occur or not – this block will execute for sure.

Now that you understand how the `try` , `except` , `else` , and `finally` code blocks work, the order of flow will be:

```
try:
    code that may/may not produce errors
except:
    when error arises, then this block of code exceutes
else:
    when error doesn't arise, then this block of code exceute
finally:
    This block will exceute whether error occurs or not.
```

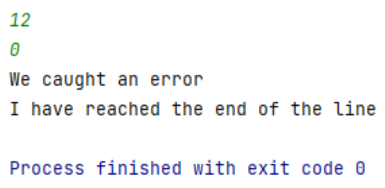
On applying the same structure to the number division problem, we get this:

```
try:
   a = int(input())
```

```
    b = int(input())
    result = a/b
    print(result)

except:
    print("We caught an error")

else:
    print("Hurray, we don't have any errors")

finally:
    print("I have reached the end of the line")
```

ADVERTISEMENT

When `b` is assigned `0`, then we get an error. So, the except block executes and prints `We caught an Error` and finally the code block executes and prints `I have reached the end of the line`.

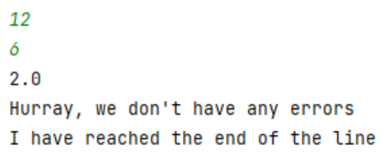


```
12
0
We caught an error
I have reached the end of the line

Process finished with exit code 0
```

Code execution flow when error occurs

On the other hand, if `b` gets `6` for example (or any non-zero value), then we divide the `a` value by `6` and store it in the `result` variable. The code then prints the `result` value.

Then, the `else` block executes and prints `Hurray, we don't have any errors` and finally the code block executes and prints `I have reached the end of the line`.



```
12
6
2.0
Hurray, we don't have any errors
I have reached the end of the line
```

Code execution flow when no error raises

ADVERTISEMENT

Summary

Now I hope you understand how you can implement error handling in Python in order to catch potential errors with `try/except` blocks.

You've also learned how to use the `else` and `finally` code blocks that are associated with these error handling methods.

Happy Programming...