Editor

Exceptions

Section 22.10

Completed

< back to module

Skip straight to editor/terminal

As we've been working through learning Python, you've probably noticed a certain pattern in the way we get errors shown to us.

Let's cause an error on purpose and take a closer look at the format of our errors.

```
print(5/0)
```

This code will create an error, because we can't divide 5 by 0: mathematically, it doesn't make sense to divide a number by 0. That doesn't stop us from writing in code though, so what happens when we try to run this program?

```
$ python program.py
Traceback (most recent call last):
  File "program.py", line 1, in <module>
    print(5/0)
ZeroDivisionError: integer division or modulo by zero
```

This is a *traceback error*. These are helpful to us, because they give us additional information about what went wrong in our code. Looking at this error, here's what we know:

- The error happened in the "program.py" file, on line 1.
- It happened somewhere around the `print(5/0)` code execution.
- A particular exception was thrown, `ZeroDivisionError`, and a more friendly message follows this to explain what went wrong.

An *exception* is a special kind of object that Python uses as part of its error management system. Whenever an error happens, Python creates an exception object and, by default, halts the execution of the program as soon as the exception is generated. This is great when we're building code... but not so great when code is in production. We don't really want our users seeing our exceptions. Also, sometimes we don't necessarily *want* our code to just stop immediately: in some situations it's better to log the error, then allow the program to keep executing. Or sometimes the detection of an exception can be built into the program itself, using an exception as a decision point to move the user down a different logic path.

We can use a *try-catch block* to *handle* exceptions that we suspect might crop up in our code.

```
try:
    print(5/0)
except ZeroDivisionError:
    print("You can't divide by zero!")
```

Now, instead of seeing the exception code and traceback, when we run this the error will still occur, but we'll see our custom message printed out.

```
$ python program.py
You can't divide by zero!
```

Let's try something more complex to show how useful catching exceptions can be, by building a program that takes 2 numbers from user input and divides them.

```
print('Please give me 2 numbers and I will divide them.')
print('Enter "q" to quit.\n')

while True:
    first_number = raw_input('First Number: ')
    if first_number == 'q':
        break

    second_number = raw_input('Second Number: ')
    if second_number == 'q':
        break

    answer = int(first_number) / int(second_number)
    print('Your answer is: ' + str(answer))
    print('Give me another!\n')

print('Ok, bye!')
```

This code accepts 2 numbers from the user, and then divides them and returns the answer. After it is successful, it will go back and run again, prompting the user to start over, until the user types 'q' to quit.

So now, if we enter "5" as our first number and "0" as our second number, we once again get our exception and the program crashes. Let's add a try-catch block so our program can keep running in this situation.

```
print('Please give me 2 numbers and I will divide them.')
print('Enter "q" to quit.\n')

while True:
    first_number = raw_input('First Number: ')
    if first_number == 'q':
        break

    second_number = raw_input('Second Number: ')
    if second_number == 'q':
        break

    try:
        answer = int(first_number) / int(second_number)
    except ZeroDivisionError:
        print('You can\'t divide by zero! Let\'s start again.')
    else:
        print('Your answer is: ' + str(answer))
        print('Give me another!\n')

print('Ok, bye!')
```

Here we've added a try-catch block. First we try to do our division. If it doesn't work and there's a ZeroDivisionError thrown, we catch that exception and print a nicer error message to the user, then invite them to start again. If we see this message, the while loop will then pick up from the beginning for a fresh run.

If the division is successful, then the user moves down to the `else` block, where their answer gets printed and we see the 'Give me another!' message, before the while loop restarts.

Here's how it looks to a user now if they run this program and try to divide 5 by 0.

```
$python program.py
Please give me 2 numbers and I will divide them.
Enter "q" to quit.

First Number: 5
Second Number: 0
You can't divide by zero! Let's start again.
First Number:
```

That's a lot nicer!

Sorry, there has been a server issue, message: container limit reached [Info]

Time Remaining: Loading... Language: Python 2 | Python 3 | C Theme: Dark | Light Status: Editor Terminal Server Server: Reset Log: Download

Steps

Step 1

There is at least one more exception that can easily happen in the last example code on this page. What happens if, instead of a number, we type a letter or a word, like "five" or "apple"? Revise the code to handle this user error as well so the program doesn't crash.

Editor

Loading editor...

Save

Terminal

Loading terminal...

Completed

< back to module