

# Handling errors

## try except

Susan Ibach | Technical Evangelist  
Christopher Harrison | Content Developer

# Even the best laid plans sometimes go wrong

- You create a shopping list then when you get to the grocery store realize you left the list at home
- You want to buy a pair of shoes, but your size is out of stock
- You need to call someone and your cell phone battery is dead

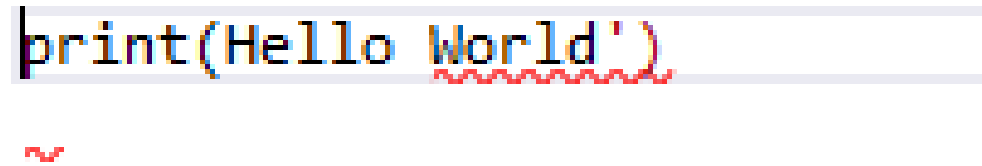
# Things go wrong in programs as well

- A program cannot find a file it needs
- A user enters a date in the wrong format
- You try to divide a number by zero

# Error types

Syntax errors are errors that the development tool can detect

- Visual Studio highlights syntax errors with the red squiggle




A screenshot of a code editor showing a Python print statement: `print(Hello World')`. The text is highlighted with a light blue background. A red squiggle is visible under the closing single quote, indicating a syntax error. The word "Hello" is in blue, "World" is in blue, and the closing quote is in red.

# Sometimes typing mistakes can't be detected until you run the program

```
prnit('Hello World')
```

```
prnit('Hello World')
```

 **NameError occurred** ✕

name 'prnit' is not defined

**Troubleshooting tips:**  
[Get general help for exceptions.](#)

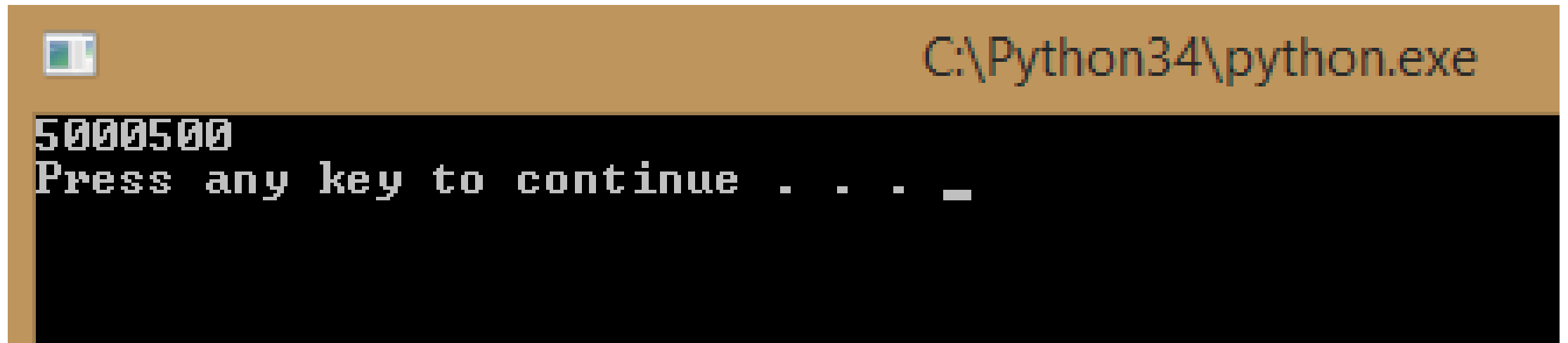
[Search for more Help Online...](#)

**Exception settings:**  
☐ Break when this exception type is thrown

**Actions:**  
[Copy exception detail to the clipboard](#)  
[Open exception settings](#)

Logic errors are syntactically correct, but the program doesn't do what we want it to do

```
salary = '5000'  
bonus = '500'  
payCheck = salary + bonus  
print(payCheck)
```



A screenshot of a Windows command prompt window. The title bar is brown and contains a small icon on the left and the text 'C:\Python34\python.exe' on the right. The command prompt area has a black background with white text. The first line shows the output '5000500'. The second line shows the prompt 'Press any key to continue . . . \_'.

```
C:\Python34\python.exe  
5000500  
Press any key to continue . . . _
```

# DEMO

---

Syntax and runtime errors



# Gracefully handling errors

Runtime errors occur when the code basically works but something out of the ordinary 'crashes' the code

- You write a calculator program and a user tries to divide a number by zero
- Your program tries to read a file, and the file is missing
- Your program is trying to perform a date calculation and the date provided is in the wrong format

# Having your code crash is a very poor experience for the user

- You can add error handling to your code to handle runtime errors gracefully

# Let's create a calculator program that will take two numbers and divide them for the user

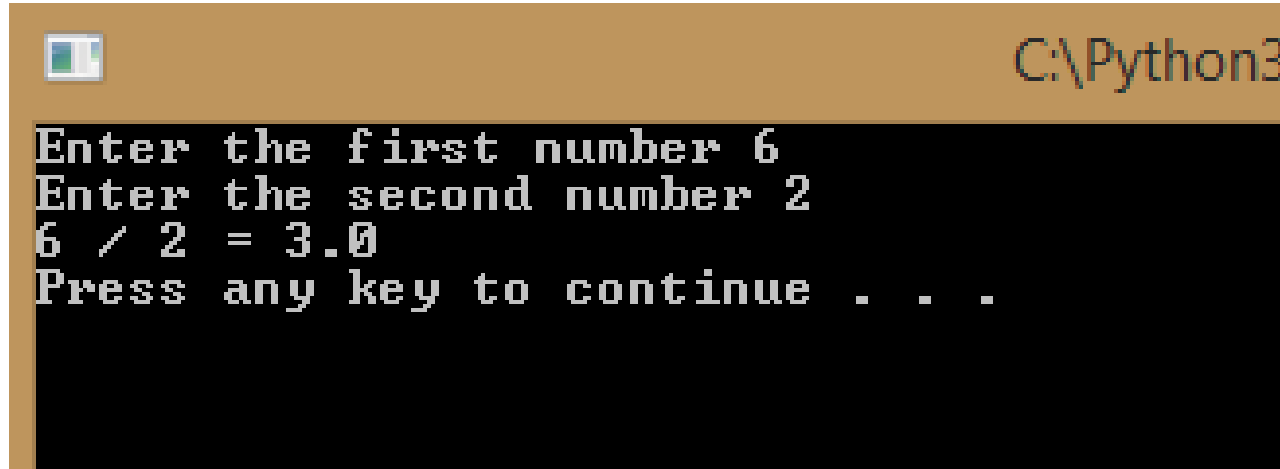
```
first = input("Enter the first number ")
second = input("Enter the second number ")

firstNumber = float(first)
secondNumber = float(second)

result = firstNumber / secondNumber

print (first + " / " + second + " = " + str(result))
```

- We test it and it works!



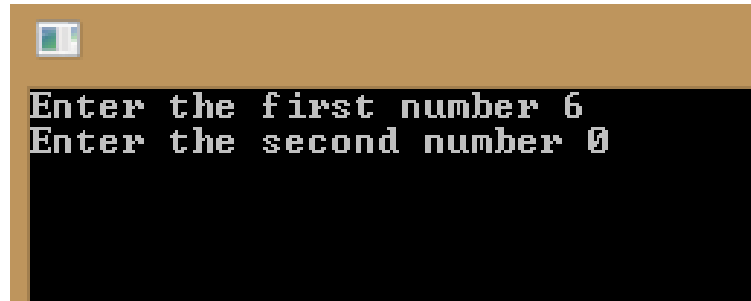
```
C:\Python3.6.4>python calculator.py
Enter the first number 6
Enter the second number 2
6 / 2 = 3.0
Press any key to continue . . .
```

# DEMO

---

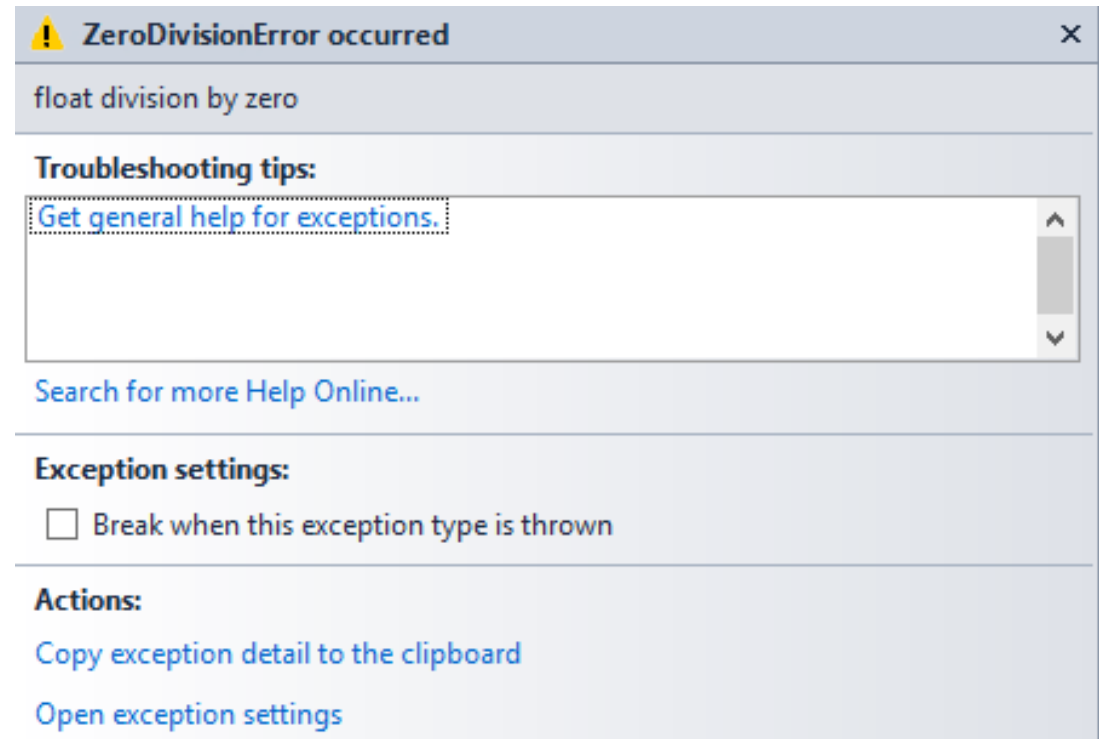
Create a calculator

# What happens you enter 0 as the second number



```
Enter the first number 6
Enter the second number 0
```

- You get an error at runtime



# Which line of code generated the error message?

```
first = input("Enter the first number ")  
second = input("Enter the second number ")
```

```
firstNumber = float(first)  
secondNumber = float(second)
```

```
result = firstNumber / secondNumber
```

```
print (first + " / " + second + " = " + str(result))
```

You can add a **try/except** around the code that generates the error to handle it gracefully

```
first = input("Enter the first number ")
second = input("Enter the second number ")
firstNumber = float(first)
secondNumber = float(second)
try :
    result = firstNumber / secondNumber
    print (first + " / " + second + " = " + str(result))
except :
    print("I am sorry something went wrong")
```

- The code in the except only runs if there is an error generated when executing the code in the try

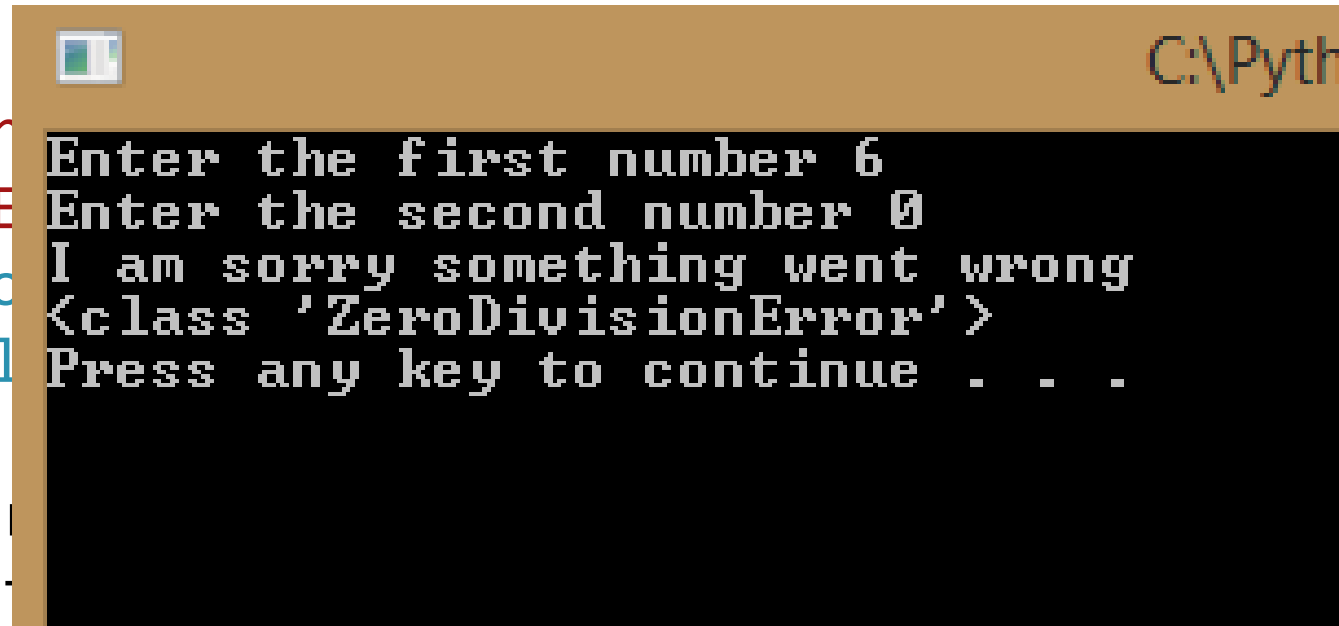


If you want to know what the error was, you can use the function `sys.exc_info()`

```
import sys
```

```
first = input("Enter the first number ")
second = input("Enter the second number ")
firstNumber = float(first)
secondNumber = float(second)

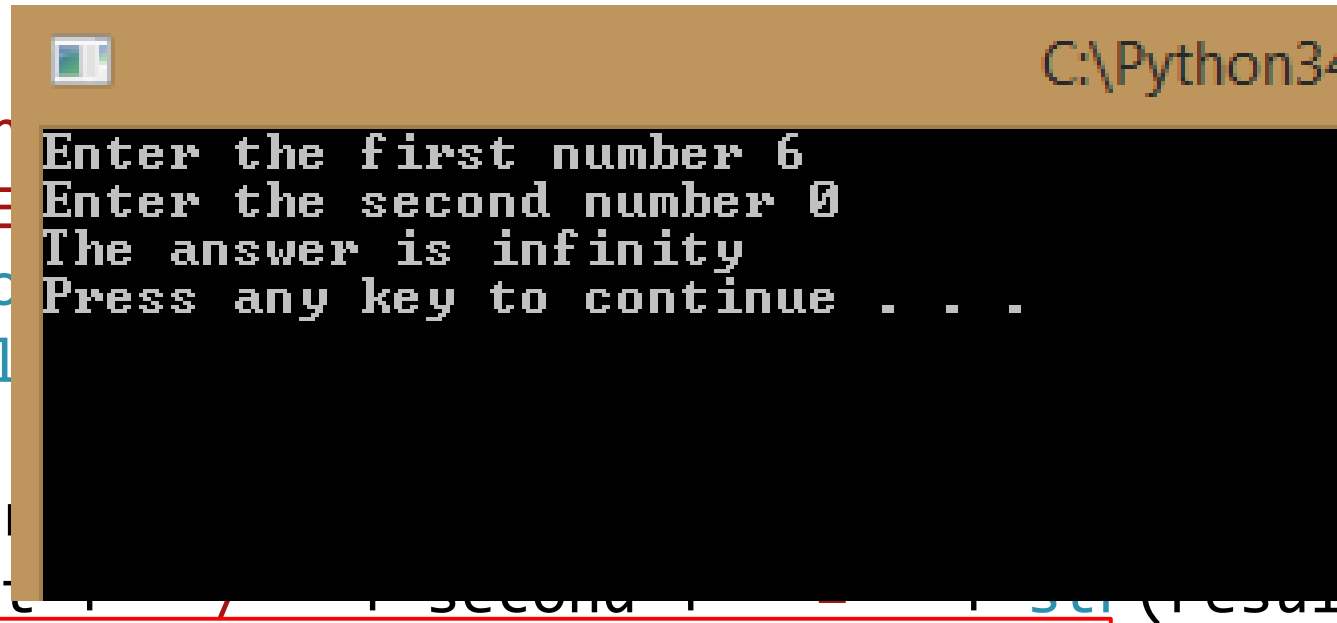
try :
    result = firstNumber / secondNumber
    print (firstNumber, "divided by", secondNumber, "is", result)
except :
    error = sys.exc_info()[0]
    print("I am sorry something went wrong")
    print(error)
```

A screenshot of a Python terminal window with a brown title bar. The window title is partially visible as 'C:\Pyth'. The terminal has a black background with white text. It shows the user entering '6' for the first number and '0' for the second number. An error message is displayed: 'I am sorry something went wrong', followed by the error details: '<class \'ZeroDivisionError\'>'. The prompt 'Press any key to continue . . .' is shown at the bottom of the terminal output.

```
C:\Pyth
Enter the first number 6
Enter the second number 0
I am sorry something went wrong
<class 'ZeroDivisionError'>
Press any key to continue . . .
```

If you know exactly what error is occurring, you can specify how to handle that exact error

```
first = input("Enter the first number ")
second = input("Enter the second number ")
firstNumber = float(first)
secondNumber = float(second)
try :
    result = firstNumber / secondNumber
    print (firstNumber / secondNumber, result)
except ZeroDivisionError :
    print("The answer is infinity")
```



```
C:\Python34>
Enter the first number 6
Enter the second number 0
The answer is infinity
Press any key to continue . . .
```

Ideally you should handle one or more specific errors and then have a generic error handler as well

```
first = input("Enter the first number ")
second = input("Enter the second number ")
firstNumber = float(first)
secondNumber = float(second)
try :
    result = firstNumber / secondNumber
    print (first + " / " + second + " = " + str(result))
except ZeroDivisionError :
    print("The answer is infinity")
except :
    error = sys.exc_info()[0]
    print("I am sorry something went wrong")
    print(error)
```

# DEMO

---

Trapping errors

Any code you place after the try except will always execute

```
first = input("Enter the first number ")
second = input("Enter the second number ")
firstNumber = float(first)
secondNumber = float(second)
try :
    result = firstNumber / secondNumber
    print (first + " / " + second + " = " + str(result))
except ZeroDivisionError :
    print("The answer is infinity")
except :
    error = sys.exc_info()[0]
    print("I am sorry something went wrong")
    print(error)
print("This message always displays!")
```

# How can I force my program to exit if an error occurs and I don't want to continue?

- You can use the function `sys.exit()` in the `sys` library

```
try :  
    result = firstNumber / secondNumber  
    print (first + " / " + second + " = " + str(result))  
except ZeroDivisionError :  
    print("The answer is infinity")  
    sys.exit()  
print("This message only displays if there is no error!")
```

You can also use variables and an if statement to control what happens after an error

```
try :  
    result = firstNumber / secondNumber  
    print (first + " / " + second + " = " + str(result))  
    errorFlag = False  
except ZeroDivisionError :  
    print("The answer is infinity")  
    errorFlag = True  
if not errorFlag :  
    print("This message only displays if there is no error!")
```

# DEMO

---

Controlling execution after an error



Is there any other code in our program that might give us an error at runtime?

```
first = input("Enter the first number ")  
second = input("Enter the second number ")
```

```
firstNumber = float(first)  
secondNumber = float(second)
```

```
result = firstNumber / secondNumber
```

```
print (first + " / " + second + " = " + str(result))
```

# There are a lot of different situations that can raise errors in our code

- Converting between datatypes
- Opening files
- Mathematical calculations
- Trying to access a value in a list that does not exist

# How do you know what errors will be raised?

- You can test it yourself and when an error occurs use the `sys.exc_info()` function to get the name of the error
- There is a list of standard Python errors
  - <https://docs.python.org/3/c-api/exceptions.html#standard-exceptions>

# The most important thing to do is to test!

1. Execute your code with everything running normally
2. Execute your code with incorrect user input
  - Enter letters instead of numbers
  - Enter 0 or spaces
  - Enter a value in the wrong format (e.g. dates)
3. Try other error scenarios such as missing files
4. Try anything you can think of that might crash your code
  - Entering really big numbers
  - negative numbers

# Do I need to handle EVERY possible error?

- Sometimes writing the code to handle the errors takes more time than writing the original program!
- Whether it is necessary to handle EVERY error depends on how the code will be used
- If you are writing a system for air traffic control I would want very thorough error handling!
- If you are writing a fun little app to tweet when your plant needs water, I wouldn't worry about it too much

# Your Challenge

- Write code to open and read a file
- Allow the user to specify the file name
- Add error handling to provide a suitable error message if the file specified by the user could not be found

# Congratulations



- You can now handle errors gracefully so your code doesn't crash



# Microsoft

©2013 Microsoft Corporation. All rights reserved. Microsoft, Windows, Office, Azure, System Center, Dynamics and other product names are or may be registered trademarks and/or trademarks in the U.S. and/or other countries. The information herein is for informational purposes only and represents the current view of Microsoft Corporation as of the date of this presentation. Because Microsoft must respond to changing market conditions, it should not be interpreted to be a commitment on the part of Microsoft, and Microsoft cannot guarantee the accuracy of any information provided after the date of this presentation. MICROSOFT MAKES NO WARRANTIES, EXPRESS, IMPLIED OR STATUTORY, AS TO THE INFORMATION IN THIS PRESENTATION.