

Error (Exception) Handling (Try / Except)

Chapter 03b

Lesson Objectives

- Examples of Possible Errors
- Types of Errors
- Where to put in **(Try / Except)** Error Handling in Code
- Many Exception Errors

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

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

Types of Errors – Syntax Errors

- Misspelled words, quotes missing
 - TIPS:
 - Sometimes look right before where error occurred.
 - Look at the error message.
 - Use Internet to explain error message.

Types of Errors – Logic Errors

- Syntax is correct, but program doesn't do what you want it to do.
- Output is not correct answer.

```
salary = '5000'
```

```
bonus = '500'
```

```
paycheck = salary + bonus
```

```
print(paycheck)
```

```
#OUTPUT
```

```
5000500
```

Types of Errors – Runtime Errors

- Code basically works but something out of the ordinary ‘crashes’ the code
- You write a calculator program and the user tries to divide 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.

EXAMPLE: Let's create a calculator program that will take two numbers inputted by the user and divide them, **save as calc.py**

```
firstNumber = float(input('Enter first number'))
```

```
secondNumber = float(input('Enter second number'))
```

```
result = firstNumber / secondNumber
```

```
print(result)
```

ADD a **try/except** around the code that **generates the error**

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

try:

```
    result = firstNumber / secondNumber  
    print(result)
```

#indent try clause

#if error occurred anywhere

#in try clause, jumps to except

except:

```
    print('I am very sorry something went wrong')
```


try / except

try block – lets you test a block of code for errors.

except block – lets you handle the error.

To display the error, use function `sys.exc_info()`

import sys

firstNumber = float(input('Enter first number'))

secondNumber = float(input('Enter second number'))

try:

result = firstNumber / secondNumber

print(result)

except:

error = sys.exc_info()[0]

print('I am very sorry something went wrong')

print(error)

MANY EXCEPTIONS: Handle one or more specific errors, and then have a generic error handler as well

import sys

```
firstNumber = float(input('Enter first number'))
```

```
secondNumber = float(input('Enter second number'))
```

try:

```
    result = firstNumber / secondNumber
```

```
    print(result)
```

except ZeroDivisionError:

```
    print('The answer is infinity')
```

except:

```
    error = sys.exc_info()[0]
```

```
    print('I am very sorry something went wrong')
```


```
    print(error)
```

ERROR HANDLING

- Errors can be handled with try and except statements.
- The **code that could potentially have an error** is put in a **try clause**.
- The program execution moves to the start of an **except clause** if an error happens.

ERROR HANDLING

```
def spam(divideBy):  
    try:  
        return 42 / divideBy  
    except ZeroDivisionError:  
        print('Error: Invalid argument')
```



#Calls to the spam function, passes number as argument

```
print(spam(2))  
print(spam(12))  
print(spam(0))  
print(spam(1))
```

The most important thing to do is **to test** with different values!

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 scenario** such as missing files
4. Try anything you can think of that might **crash your code**
 - Enter really big numbers
 - Enter 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.
- Depends on how the code will be used.
 - If you are writing a system for **air traffic control**, I would want a 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.