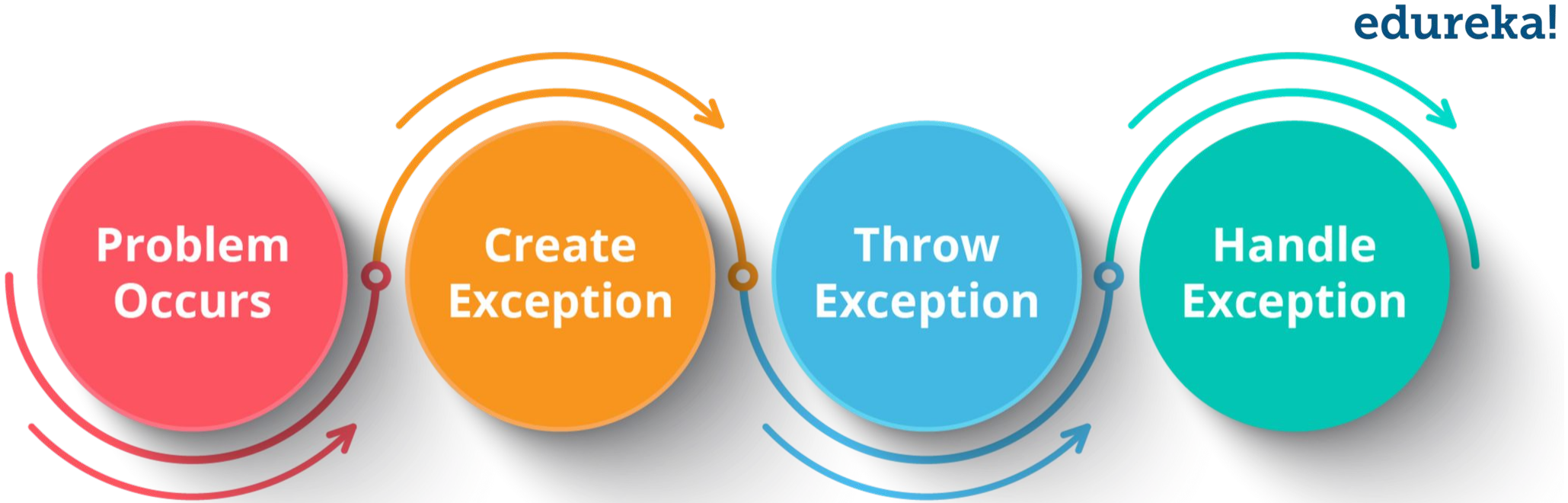
COMP 125 Programming with Python
Exception Handling



Mehmet Sayar

Koç University

MT4

Saturday, Dec 12 2020

6:15 PM

Division Example

- Write a program that will take two numbers as input

- It will print out the operation with all the numbers having 3 significant digits after the decimal point, if successful

- Your program should handle the case when the user enters a string that cannot be converted to a number (`ValueError`)
- Your program should handle the case when the user enters the denominator as 0 (`ZeroDivisionError`)

Division Example

```
try:
    number1 = input( "Enter numerator: " )
    number1 = float( number1 ) #may cause ValueError
    number2 = input( "Enter denominator: " )
    number2 = float( number2 ) #may cause ValueError
    result = number1 / number2 #may cause ZeroDivisionError
except ValueError: #Handling the ValueError
    print ("You must enter two numbers")
except ZeroDivisionError: #Handling the ZeroDivisionError
    print("Attempted to divide by zero")
else: #Run if there are no exceptions
    print(f"{number1:.3f} / {number2:.3f} = {result:.3f}")
finally: #Run regardless
    print("Goodbye!")
```

What If an Exception Is Not Handled?

- Two ways for exception to go unhandled:
  - No except clause specifying exception of the right type
  - Exception raised outside a try suite

- In both cases, exception will cause the program to halt (and a traceback)
  - Python documentation provides information about exceptions that can be raised by different functions

Raising an Exception

- You may want to raise your own exception, especially if you are writing a module/library
  - Example: checking the shapes of the matrices in matrix addition
- The raise statement is used towards this end. General format:

```
raise ExceptionType(message)
```

- For example

```
if row1 != row2:
    raise ValueError("Number of rows do not match")
```

- The parent of each exception type is the `Exception`. If unsure, just throw this. E.g.

```
if a == 0:
    raise Exception("Number cannot be zero.")
```

Custom Exceptions

- It is possible to define custom exceptions.
- This requires object-oriented programming to understand fully but for now you can directly do this:

```
class CustomException(Exception):
    pass
raise CustomException(message)
```

Example

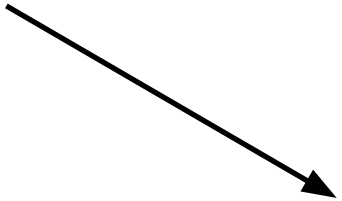
```
class NegativeNumberError( Exception ):
    pass

def squareRoot(number):
    if number < 0:
        raise NegativeNumberError("Negative numbers are not supported")
    else:
        return number ** 0.5
try:
    number = input( "Enter a positive number: " )
    number = float( number )
    result = squareRoot(number)
except (ValueError, NegativeNumberError) as err:
    print(err)
else:
    print(f"Square root of {number:.3f} is {result:.3f}")
```

Back to Files

- There are multiple potential issues working with files.
- One issue problem is not being able to call the `close` method
- This may result in loss of information if anything was written to the file
- This can be:
  - Due to programmer error
  - Due to an unhandled exception

Case

```
file = open('file_name', 'w')
file.write('hello world!\n')
some_statements_may_include_other_writes
file.close()
```



An exception here may result in loss of information, if it goes unhandled.
This results in failure to call the “file.close” statement.

How to Handle it?

```
file = open('file_name', 'w')
try:
    file.write('hello world!\n')
    some_statements_may_include_other_writes
finally:
   file.close()
```

This makes sure that the file is closed whatever happens.
However, the code is “clumsy” now

The `with` Statement for Working with Files

- A less clumsy alternative

```
with open('file_path', 'w') as file:
    file.write('hello world!')
    some_statements_may_include_other_writes
```

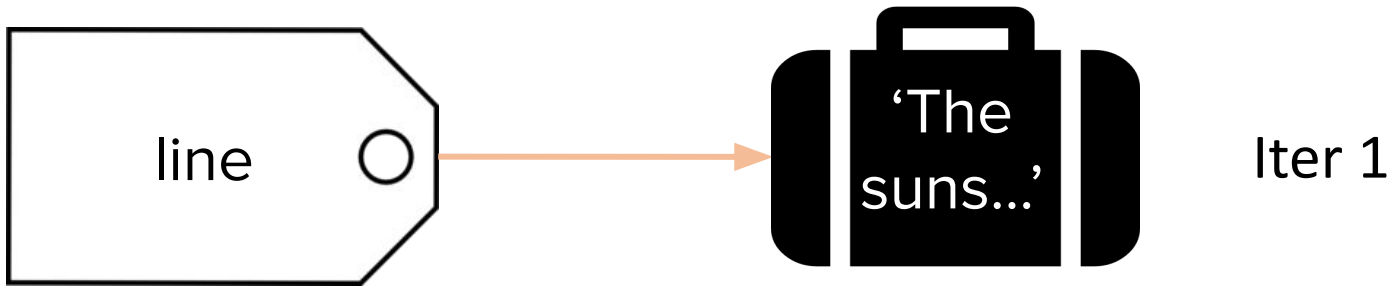
- This ensures that the close is called even if there is an exception
- More generally, this statement makes sure that the resources are acquired and released properly
- The goal is to replace `try-finally` statements

Reading a File Line-by-Line Using `with`

The contents of the file catullus.txt:

```
The suns are able to fall and rise:
When that brief light has fallen for us,
We must sleep a never ending night.
```

```
with open('catullus.txt', 'r') as file:
    for line in file:
        print(line)
```

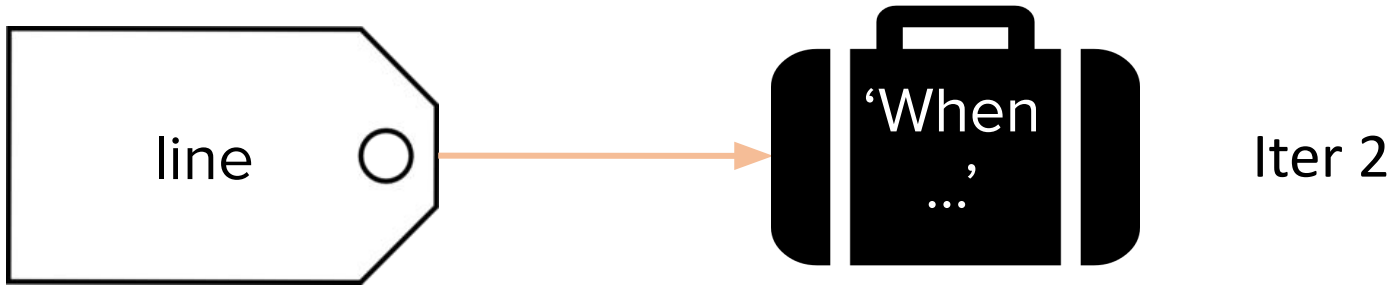


Reading a File Line-by-Line Using `with`

The contents of the file catullus.txt:

```
The suns are able to fall and rise:
When that brief light has fallen for us,
We must sleep a never ending night.
```

```
with open('catullus.txt', 'r') as file:
    for line in file:
        print(line)
```



Reading a File Line-by-Line Using `with`

The contents of the file catullus.txt:

```
The suns are able to fall and rise:
When that brief light has fallen for us,
We must sleep a never ending night.
```

```
with open('catullus.txt', 'r') as file:
    for line in file:
        print(line)
```