Exceptions

Contents

1. Getting started with exceptions
2. Additional exception techniques

Demo folder: **AppxA-Exceptions**

1. Getting Started with Exceptions

- Overview
- Standard exceptions in Python
- Simple exception example
- Accessing the exception object

Overview

- Exceptions are a run-time mechanism for indicating exceptional conditions in Python
  - If you detect an "exceptional" condition, you can throw an exception
  - An exception is an object that contains relevant error info
- Somewhere up the call stack, the exception is caught and dealt with
  - If the exception is not caught, your application terminates

Standard Exceptions in Python

- There are lots of things that can go wrong in a Python app
  - Therefore, there are lots of different exception classes
  - Each exception class represents a different kind of problem

- Here are some of the standard exception classes in Python:
  - KeyboardInterrupt
  - OSError
  - EOFError
  - ValueError
  - … etc.

Simple Exception Example

- Here's a simple example of how to deal with exceptions in a Python app
  - The `try` block contains code that might cause an exception
  - The `except` block catches a particular type of exception

```
# Keep on looping until the user enters a number.

while True:

    try:
        inp = input("What's your favourite number? ")
        num = int(inp)
        print("Thanks, your favourite number is %d" % num)
        break

    except ValueError:
        print("Eek, that's not valid a number!")
```

simpleExceptions.py

Accessing the Exception Object

- In your `except` clause, you can specify a name for the exception object you just caught
  - Allows you to use the exception object in your `except` block

- Example
  - Catch `ValueError` and display error message on console

```
# Keep on looping until the user enters a number.
while True:

    try:
        inp = input("What's your favourite number? ")
        num = int(inp)
        print("Thanks, your favourite number is %d" % num)
        break

    except ValueError as err:
        print("ValueError occurred: %s" % err)
```

usingExceptionObject.py

```
What's your favourite number? d
ValueError occurred: invalid literal for int() with base 10: 'd'
```

2. Additional Exception Techniques

- Catching multiple exception types
- The "all ok" scenario
- Unconditional "wrap-up" code
- Exception hierarchies
- Defining custom exception classes
- Raising exceptions

Catching Multiple Exception Types (1 of 2)

- If your `try` block contains complex code, then multiple different types of exception might occur
  - You can define multiple `except` blocks, to catch each type of error
  - Optionally the last `except` block can be a catch-all (omit the type)

- Example

```
import sys

try:
    fh = open('favNum.txt')
    str = fh.readline()
    num = int(str.strip())
    print("The number in the file is %d" % num)

except OSError as err:
    print("OSError occurred: %s" % err)

except ValueError as err:
    print("ValueError occurred: %s" % err)

except:
    print("Some other error occurred")
```

multipleExceptionTypes1.py

Catching Multiple Exception Types (2 of 2)

- If you want to perform the same processing for several types of exception:
  - Group the exceptions together in a single `except` block
  - Specify the exception types as a tuple

- Example

```
import sys

try:
    fh = open('favNum.txt')
    str = fh.readline()
    num = int(str.strip())
    print("The number in the file is %d" % num)

except (OSError, ValueError) as err:
    print("Error occurred: %s" % err)

except:
    print("Some other error occurred")
```

multipleExceptionTypes2.py

The "All OK" Scenario

- You can add an `else` block at the end of `try…except`
  - Executed only if the `try` block completed successfully

- Example

```
import sys

try:
    fh = open('favNum.txt')
    str = fh.readline()
    num = int(str.strip())
    print("The number in the file is %d" % num)

except OSError as err:
    print("OSError occurred: %s" % err)
…

else:
    print("All completed OK!")
    fh.close()
```

allOK.py

Unconditional "Wrap-Up" Code

- You can add a `finally` block at the end of everything
  - Always executed at the end of the `try…except…else` construct
  - Whether an exception occurred or not

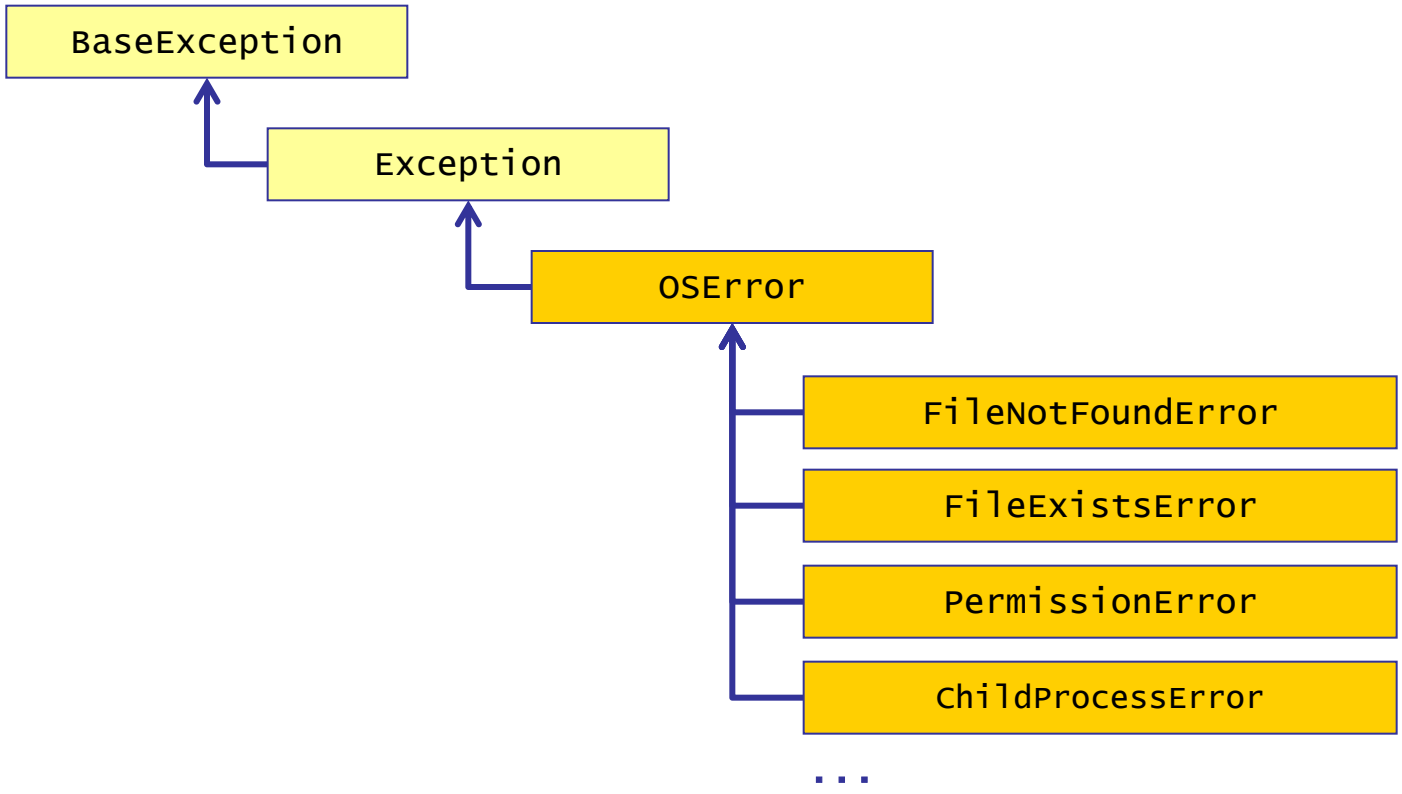
- Example

```
import sys

try:
    fh = open('favNum.txt')
    str = fh.readline()
    num = int(str.strip())
    print("The number in the file is %d" % num)

except OSError as err:
    print("OSError occurred: %s" % err)
…

else:
    print("All completed OK!")
    fh.close()

finally:
    print("That's all folks. This message will always appear!")
```

finally.py

Exceptions Hierarchies (1 of 2)

- Python organizes exceptions into an inheritance hierarchy
  - Represents specializations of general error conditions
- Example
  - There are several subclasses of `OSError`



Exceptions Hierarchies (2 of 2)

- When you define an `except` block…
  - It will catch that exception type, plus any subclasses

- Example:
  - "Special" processing for `FileNotFoundError` exceptions
  - "Generic" processing for any other kind of `OSError` exceptions

```
import sys

try:
    fh = open('favNum.txt')
    str = fh.readline()
    num = int(str.strip())
    print("The number in the file is %d" % num)

except FileNotFoundError as err:
    print("File not found: %s" % err)

except OSError as err:
    print("More general OSError occurred: %s" % err)

… plus other except blocks and an else block, as appropriate …
```

allOK.py

Defining Custom Exception Classes

- You can define custom exception classes
  - To represent important types of error in your application
- How to do it:
  - Define a class that inherits from `Exception` (or a subclass)
  - Implement `__init__` and `__str__` methods
- Example:

```
class MyError(Exception):

    def __init__(self, value):
        self.value = value

    def __str__(self):
        return repr(self.value)
```

customExceptions.py

Raising Exceptions

- To raise (i.e. trigger) an exception:
  - Use the `raise` keyword
  - Specify the type of exception you want to raise
  - Pass in any constructor arguemnts as appropriate

- Example:

```
try:
    raise MyError("EEK ERROR ERROR ERROR")

except MyError as err:
    print("It appears my exception occurred, the value is %s" % err.value)
```

customExceptions.py

Any Questions?