Error Handling

Amit Kotlovski / @amitkot

Agenda

- Handling Exceptions
- Logging
- Use `os.exit()`
- Exception Arguments
- Raising Exceptions
- Finally Block
- Assert

Handling Exceptions

- Be optimistic, assume it's going to work
- EAFP - Easier to Ask for Forgiveness than Permission
- Handle Exceptions

Uncaught Exception

Printing a variable that hasn't been defined yet

```
>>> print f
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'f' is not defined
```

How can we handle this?

```python
def first():
    second()

def second():
    third()

first()
```

```
$ python simple_crash.py
Traceback (most recent call last):
  File "simple_crash.py", line 7, in <module>
    first()
  File "simple_crash.py", line 2, in first
    second()
  File "simple_crash.py", line 5, in second
    third()
NameError: global name 'third' is not defined
```

Catching the Exception

```python
def first():
    second()

def second():
    try:
        third()
    except NameError:
        print "Sorry, can't find the function third"

first()
```

```
$ python simple_crash_handled.py
Sorry, can't find the function third
```

Catching the Exception higher

```python
def first():
    try:
        second()
    except NameError:
        print "Sorry, can't find the function third"

def second():
    third()

first()
```

```python
def first():
    print 'before try+except'
    try:
        print 'before second'
        second()
        print 'after second'
    except NameError:
        print "Sorry, can't find the function third"
    print 'after try+except'
def second():
    third()

first()
```

```
$ python simple_crash_handled_2.py
before try+except
before second
Sorry, can't find the function third
after try+except
```

Raising Exceptions

```python
def print_small_numbers(data):
    if data > 40:
        raise ValueError("this method only prints small numbers")
    print 'here is a small number: ', data

print_small_numbers(7)
print ''
print_small_numbers(70)
```

```
$ python raise_exception.py
here is a small number:  7

Traceback (most recent call last):
  File "raise_exception.py", line 8, in <module>
    print_small_numbers(70)
  File "raise_exception.py", line 3, in print_small_numbers
    raise ValueError("this method only prints small numbers")
ValueError: this method only prints small numbers
```

Logging

Setup

To console:

```
import logging
# optional - set minimum log level to be handled
logging.basicConfig(level=logging.INFO)
```

To File:

```
import logging
logging.basicConfig(filename='example.log',level=logging.DEBUG)
```

Logging

Log Levels

Different log levels:

```
import logging
logging.info('Something happend')
logging.warning('A recoverable error happend - description')
logging.error('A serious problem happened - description')
logging.critical('A horrible problem happened - description')
```

Logging Exceptions

```
>>> try:
    raise ValueError('An invalid value could have been passed')
except ValueError, e:
    logging.exception(e)
...
ERROR:root:An invalid value could have been passed
Traceback (most recent call last):
  File "<ipython-input-144-74c741432256>", line 2, in <module>
    raise ValueError('An invalid value could have been passed')
ValueError: An invalid value could have been passed
```

Use `os.exit()`

- In case there is no way to solve the error
- Exits the script, returning a number to the process that started us
- Return a non-zero number for errors

Custom Exceptions

- Exceptions are classes
- The are organized in a hierarchy

Exception Class Hierarchy

```
BaseException
 +-- SystemExit
 +-- KeyboardInterrupt
 +-- GeneratorExit
 +-- Exception
      +-- StopIteration
      +-- StandardError
      |    +-- BufferError
      |    +-- ArithmeticError
      |    |    +-- FloatingPointError
      |    |    +-- OverflowError
      |    |    +-- ZeroDivisionError
      |    +-- AssertionError
      |    +-- AttributeError
```

Adding New Exceptions

```python
class BananaError(ValueError):
    def __init__(self, sweetness_level):
        self.sweetness = sweetness_level
    def __str__(self):
        return 'sweetness level is too low - ' + str(self.sweetness)

def second():
    raise BananaError(0.3)

second()
```

```
$ python custom_exception.py
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "<stdin>", line 2, in second
__main__.BananaError: sweetness level is too low - 0.3
```

Finally Block

- Code that will run no matter if an Exception was caught
- Common practice in other languages as well

```
try:
    # code
except:
    # handle exception
finally:
    # always run afterwards regardless
```

`assert`

- We can add asserts for things that must always be True
- A fail-early safety measure
- If they fail we get an AssertionError
- Assertions are removed in optimized code (`python -O`)

assert

```python
def run_method(limit):
    import random
    res = random.choice(range(limit))
    assert 0 < res < limit, "Result is outside parameters"
    return res
```

Summary

- Python programmers are optimistic - assume it will work
- But also pragmatic - be prepared when it doesn't
- Use tools for handling errors
- If you can't fix it - log it or exit
- You can define custom exceptions to match your needs

Questions

Thanks!

Amit Kotlovski / **@amitkot**