Python language: exceptions

The FOSSEE Group

Department of Aerospace Engineering
IIT Bombay

Mumbai, India

Motivation

- How do you signal errors to a user?

Exceptions

- Python's way of notifying you of errors
- Several standard exceptions: `SyntaxError`, `IOError` etc.
- Users can also `raise` errors
- Users can create their own exceptions
- Exceptions can be "caught" via `try/except` blocks

Exceptions: examples

```
In []: while True print('Hello world')

File "<stdin>", line 1, in ?
    while True print('Hello world')
                    ^
SyntaxError: invalid syntax
```

Exceptions: examples

```
In []: while True print('Hello world')

File "<stdin>", line 1, in ?
    while True print('Hello world')
                   ^
SyntaxError: invalid syntax
```

Exceptions: examples

```
In []: print(spam)

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

Exceptions: examples

```
In []: print(spam)

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

Exceptions: examples

```
In []: 1 / 0
```

```
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ZeroDivisionError: integer division
or modulo by zero
```

Exceptions: examples

```
In []: 1 / 0

Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ZeroDivisionError: integer division
or modulo by zero
```

Exceptions: examples

```
In []: '2' + 2

Traceback (most recent call last):
  File "<stdin>", line 1, in ?
TypeError: cannot concatenate 'str' and '
```

Exceptions: examples

```
In []: '2' + 2

Traceback (most recent call last):
  File "<stdin>", line 1, in ?
TypeError: cannot concatenate 'str' and '
```

Processing user input

```
prompt = 'Enter a number(Q to quit): '
a = input(prompt)

num = int(a) if a != 'Q' else 0
```

What if the user enters some other alphabet?

Handling Exceptions

Python provides a **`try`** and **`except`** clause.

```
prompt = 'Enter a number(Q to quit): '
a = input(prompt)
try:
    num = int(a)
    print(num)
except:
    if a == 'Q':
        print("Exiting ...")
    else:
        print("Wrong input ...")
```

Handling Exceptions a little better

Use specific exceptions; avoid blanket except clauses

```
prompt = 'Enter a number(Q to quit): '
a = input(prompt)
try:
    num = int(a)
    print(num)
except ValueError:
    if a == 'Q':
        print("Exiting ...")
    else:
        print("Wrong input ...")
```

Exceptions: examples

```
prompt = "Enter a number: "
while True:
    try:
        x = int(input(prompt))
        break
    except ValueError:
        print("Invalid input, try again...")
```

Catching multiple exceptions

```
while True:
    try:
        data = input()
        x = int(data.split(',')[1])
        break
    except IndexError:
        print('Input at least 2 values.')
    except ValueError:
        print("Invalid input, try again...")
```

Catching multiple exceptions

```
data = input()
try:
    x = int(data.split(',')[1])
except (ValueError, IndexError):
    print("Invalid input ...")
```

try, except, else

```
while True:
    try:
        data = input()
        x = int(data.split(',')[1])
    except (ValueError, IndexError):
        print("Invalid input ...")
    else:
        print('All is well!')
        break
```

Some comments

- In practice NEVER use blanket except clauses
- Always catch specific exceptions

Exceptions: raising your exceptions

```
>>> raise ValueError("your error message")
Traceback (most recent call last):
  File "<stdin>", line 2, in ?
ValueError: your error message
```

Exceptions: try/finally

```
while True:
    try:
        x = int(input(prompt))
        break
    except ValueError:
        print("Invalid number, try again...")
    finally:
        print("All good!")
```

Always runs the finally clause!

Exceptions: try/finally

```
def f(x):
    try:
        y = int(x)
        return y
    except ValueError:
        print(x)
    finally:
        print('finally')

>>> f(1)
>>> f('a')
```

Always runs the finally clause!

Summary

- Catching exceptions with **`try/except`**
- Catching multiple exceptions
- Cleanup with **`finally`**
- Raising your own exceptions

What next?

- Only covered the very basics
- More advanced topics remain
- Read the official Python tutorial: `docs.python.org/tutorial/`