Lecture 10

Exceptions and Error Handling

Recovering from Errors

- try-except blocks allow us to recover from errors
  - Do the code that is in the try-block
  - Once an error occurs, jump to the catch
- **Example**:

```
try:
    input = raw_input() # get number from user
    x = float(input)    # convert string to float
    print 'The next number is '+str(x+1)
except:
    print 'Hey! That is not a number!'
```

might have an error

executes if error happens

Recovering from Errors

- try-except blocks allow us
  - Do the code that is in the tr
  - Once an error occurs, jump
- **Example**:

Similar to if-else
- But always does try
- Just might not do **all** of the try block

```
try:
    input = raw_input() # get number from user
    x = float(input)       # convert string to float
    print 'The next number is '+str(x+1)
except:
    print 'Hey! That is not a number!'
```

might have an error

executes if error happens

Try-Except is Very Versatile

```
def isfloat(s):
    """Returns: True if string
    s represents a float"""
    try:
        x = float(s)
        return True
    except:
        return False
```

Conversion to a float might fail

If attempt succeeds, string s is a float

Otherwise, it is not

Try-Except and the Call Stack

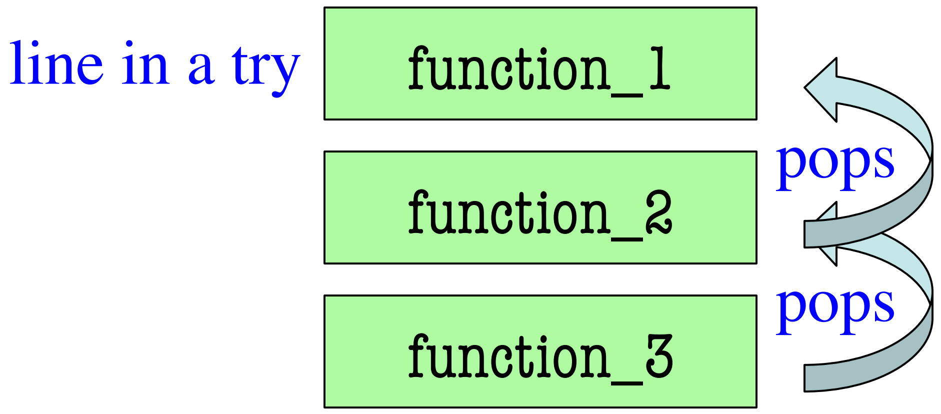
```
# recover.py

def function_1(x,y):
    try:
        return function_2(x,y)
    except:
        return float('inf')

def function_2(x,y):
    return function_3(x,y)

def function_3(x,y):
    return x/y # crash here
```

- Error "pops" frames off stack
  - Starts from the stack bottom
  - Continues until it sees that current line is in a **try**-block
  - Jumps to **except**, and then proceeds as if no error



Try-Except and the Call Stack

```
# recover.py

def function_1(x,y):
    try:
        return function_2(
    except:
        return float('inf')

def function_2(x,y):
    return function_3(x,y)

def function_3(x,y):
    return x/y # crash here
```

How to return
∞ as a float.

- Error "pops" frames off stack
  - om the stack bottom
  - es until it sees that current line is in a try-block
  - Jumps to except, and then proceeds as if no error
- **Example**:

```
>>> print function_1(1,0)
inf
>>>
```

No traceback!

Tracing Control Flow

```
def first(x):
    print 'Starting first.'
    try:
        second(x)
    except:
        print 'Caught at first’
    print 'Ending first’


def second(x):
    print 'Starting second.'
    try:
        third(x)
    except:
        print 'Caught at second’
    print 'Ending second’
```

```
def third(x):
    print 'Starting third.'
    assert x < 1
    print ’Ending third.'
```

What is the output of first(2)?

Tracing Control Flow

```
def first(x):
    print 'Starting first.'
    try:
        second(x)
    except:
        print 'Caught at first’
    print 'Ending first’

def second(x):
    print 'Starting second.'
    try:
        third(x)
    except:
        print 'Caught at second’
    print 'Ending second’
```

```
def third(x):
    print 'Starting third.'
    assert x < 1
    print ’Ending third.'
```

What is the output of first(2)?

```
'Starting first.'
'Starting second.'
'Starting third.'
'Caught at second'
'Ending second'
'Ending first'
```

Tracing Control Flow

```
def first(x):
    print 'Starting first.'
    try:
        second(x)
    except:
        print 'Caught at first’
    print 'Ending first’

def second(x):
    print 'Starting second.'
    try:
        third(x)
    except:
        print 'Caught at second’
    print 'Ending second’
```

```
def third(x):
    print 'Starting third.'
    assert x < 1
    print ’Ending third.'
```

What is the output of first(0)?

Tracing Control Flow

```
def first(x):
    print 'Starting first.'
    try:
        second(x)
    except:
        print 'Caught at first’
    print 'Ending first’

def second(x):
    print 'Starting second.'
    try:
        third(x)
    except:
        print 'Caught at second’
    print 'Ending second’
```

```
def third(x):
    print 'Starting third.'
    assert x < 1
    print ’Ending third.'
```

What is the output of first(0)?

```
'Starting first.'
'Starting second.'
'Starting third.'
'Ending third'
'Ending second'
'Ending first'
```

Error Types in Python

```
def foo():
    assert 1 == 2, 'My error'
    …
```

```
def foo():
    x = 5 / 0
    …
```

```
>>> foo()
AssertionError: My error
```

```
>>> foo()
ZeroDivisionError: integer
division or modulo by zero
```

Class Names

Error Types in Python

```
def foo():
    assert 1 == 2, 'My error'
    …
```

d

Information about an error is stored inside an **object**. The error type is the **class** of the error object.

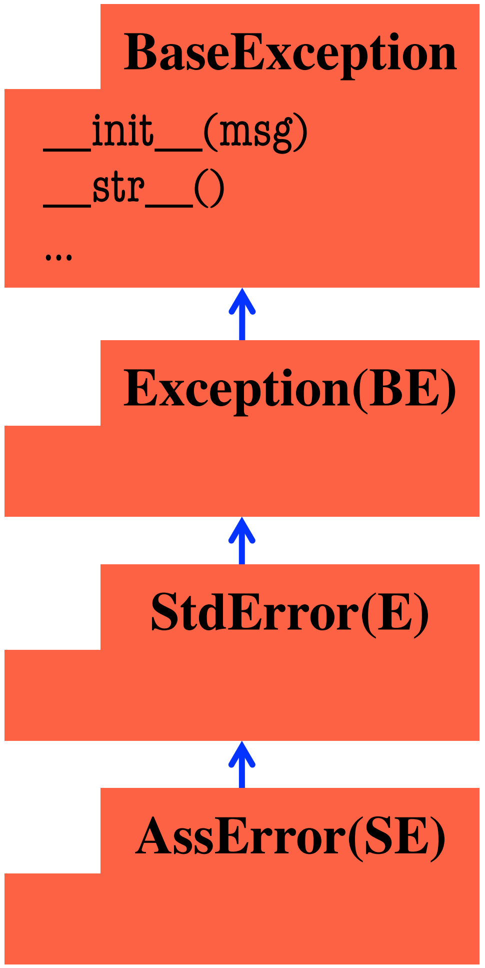
```
>>> foo()
AssertionError: My error
```

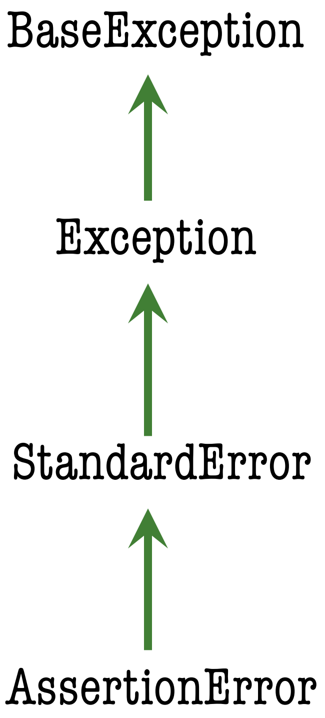
```
>>> foo()
ZeroDivisionError: integer division or modulo by zero
```

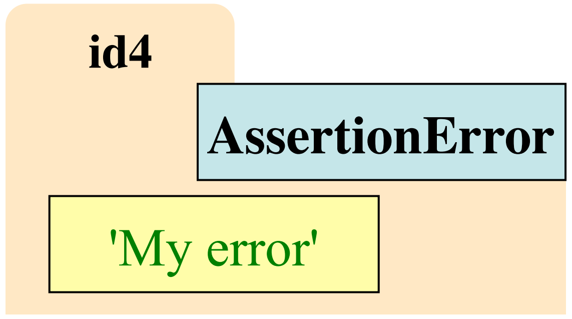
Class Names

Error Types in Python

- All errors are instances of class BaseException
- This allows us to organize them in a hierarchy

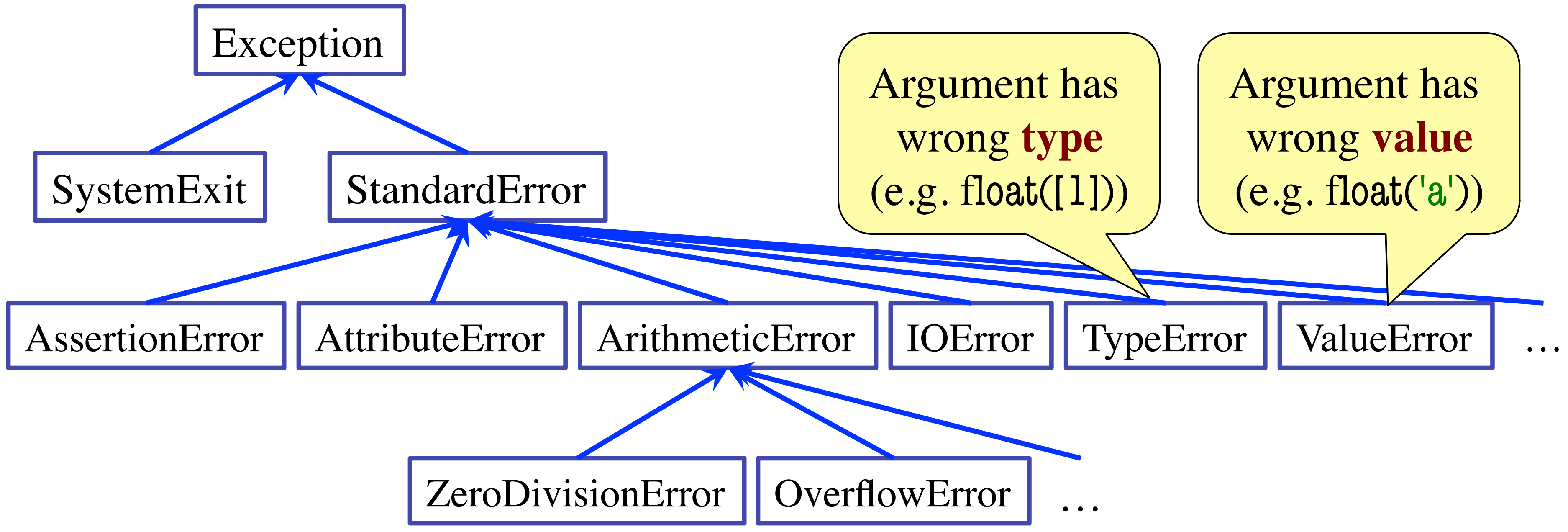






→ means "extends" or "is an instance of"

Python Error Type Hierarchy



http://docs.python.org/library/exceptions.html

Why so many error types?

Recall: Recovering from Errors

- try-except blocks allow us to recover from errors
  - Do the code that is in the try-block
  - Once an error occurs, jump to the catch
- **Example**:

```
try:
    input = raw_input() # get number from user
    x = float(input)     # convert string to float
    print 'The next number is '+str(x+1)
except:
    print 'Hey! That is not a number!'
```

might have an error

executes if have an error

Errors and Dispatch on Type

- try-except blocks can be restricted to **specific** errors
  - Doe except if error is **an instance** of that type
  - If error not an instance, do not recover
- **Example**:

```
try:
    input = raw_input() # get number from user
    x = float(input)     # convert string to float
    print 'The next number is '+str(x+1)
except ValueError:
    print 'Hey! That is not a number!'
```

May have IOError

May have ValueError

Only recovers ValueError.
Other errors ignored.

Errors and Dispatch on Type

- try-except blocks can be restricted to **specific** errors
  - Doe except if error is **an instance** of that type
  - If error not an instance, do not recover
- **Example**:

```
try:
    input = raw_input() # get number from user
    x = float(input)    # convert string to float
    print 'The next number is '+str(x+1)
except IOError:
    print 'Check your keyboard!'
```

May have IOError

May have ValueError

Only recovers IOError. Other errors ignored.

Creating Errors in Python

- Create errors with `raise`
  - **Usage**: `raise <exp>`
  - `exp` evaluates to an object
  - An instance of Exception
- Tailor your error types
  - **ValueError**: Bad value
  - **TypeError**: Bad type
- Still prefer **asserts** for preconditions, however
  - Compact and easy to read

```
def foo(x):
    assert x < 2, 'My error'
    ...
```

Identical

```
def foo(x):
    if x >= 2:
        m = 'My error'
        raise AssertionError(m)
    ...
```

Raising and Try-Except

```
def foo():
    x = 0
    try:
        raise StandardError()
        x  = 2
    except StandardError:
        x = 3
    return x
```

- The value of foo()?

A: 0
B: 2
C: 3
D: No value. It stops!
E: I don't know

Raising and Try-Except

```
def foo():
    x = 0
    try:
        raise StandardError()
        x  = 2
    except StandardError:
        x = 3
    return x
```

- The value of foo()?

A: 0
B: 2
C: 3   Correct
D: No value. It stops!
E: I don’t know

Raising and Try-Except

```
def foo():
    x = 0
    try:
        raise StandardError()
        x  = 2
    except Exception:
        x = 3
    return x
```

- The value of foo()?

A: 0
B: 2
C: 3
D: No value. It stops!
E: I don't know

Raising and Try-Except

```
def foo():
    x = 0
    try:
        raise StandardError()
        x  = 2
    except Exception:
        x = 3
    return x
```

- The value of foo()?

A: 0
B: 2
C: 3 Correct
D: No value. It stops!
E: I don't know

Raising and Try-Except

```
def foo():
    x = 0
    try:
        raise StandardError()
        x  = 2
    except AssertionError:
        x = 3
    return x
```

- The value of foo()?

A: 0
B: 2
C: 3
D: No value. It stops!
E: I don't know

Raising and Try-Except

```
def foo():
    x = 0
    try:
        raise StandardError()
        x  = 2
    except AssertionError:
        x = 3
    return x
```

- The value of foo()?

A: 0
B: 2
C: 3
D: No value. Correct
E: I don't know

Python uses isinstance to match Error types

Creating Your Own Exceptions

```
class CustomError(StandardError):
    """An instance is a custom exception"""
    pass
```

Only issues is choice of parent Exception class. Use `StandardError` if you are unsure what.

This is all you need

- No extra fields
- No extra methods
- No constructors

Inherit everything

Errors and Dispatch on Type

- try-except can put the error in a variable
- **Example**:

```
try:
    input = raw_input() # get number from user
    x = float(input)     # convert string to float
    print 'The next number is '+str(x+1)
except ValueError as e:
    print e.message
    print 'Hey! That is not a number!'
```

Some Error subclasses have more attributes