

Lecture 24

**Advanced
Error Handling**

Announcements for This Lecture

Prelim 2

- **Prelim, Dec 5th at 7:30**
 - See webpage for rooms
 - Review **Wed Dec. 4 (TBA)**
- **Material up to Last Week**
 - Recursion + Loops + Classes
 - Study guide is now posted
- **Conflict with Prelim?**
 - Prelim 2 Conflict on CMS
 - SDS students must submit!

Assignments

- A5 is now graded
 - **Mean:** 47.2 **Median:** 49
 - **A:** 47 (73%), **B:** 40 (22%)
- A6 graded by Sunday
- Keep working on A7
 - Make a big push this week
 - Time in lab Thu/Fri/Tues!
 - Get ship, asteroids moving

Error Types in Python

```
def foo():
```

```
    assert 1 == 2, 'My error'
```

```
    ...
```

```
>>> foo()
```

AssertionError: My error

```
def foo():
```

```
    x = 5 / 0
```

```
    ...
```

```
>>> foo()
```

ZeroDivisionError: integer division or modulo by zero

Class Names



Error Types in Python

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

```
>>> foo()
```

```
AssertionError: My error
```

Class Names

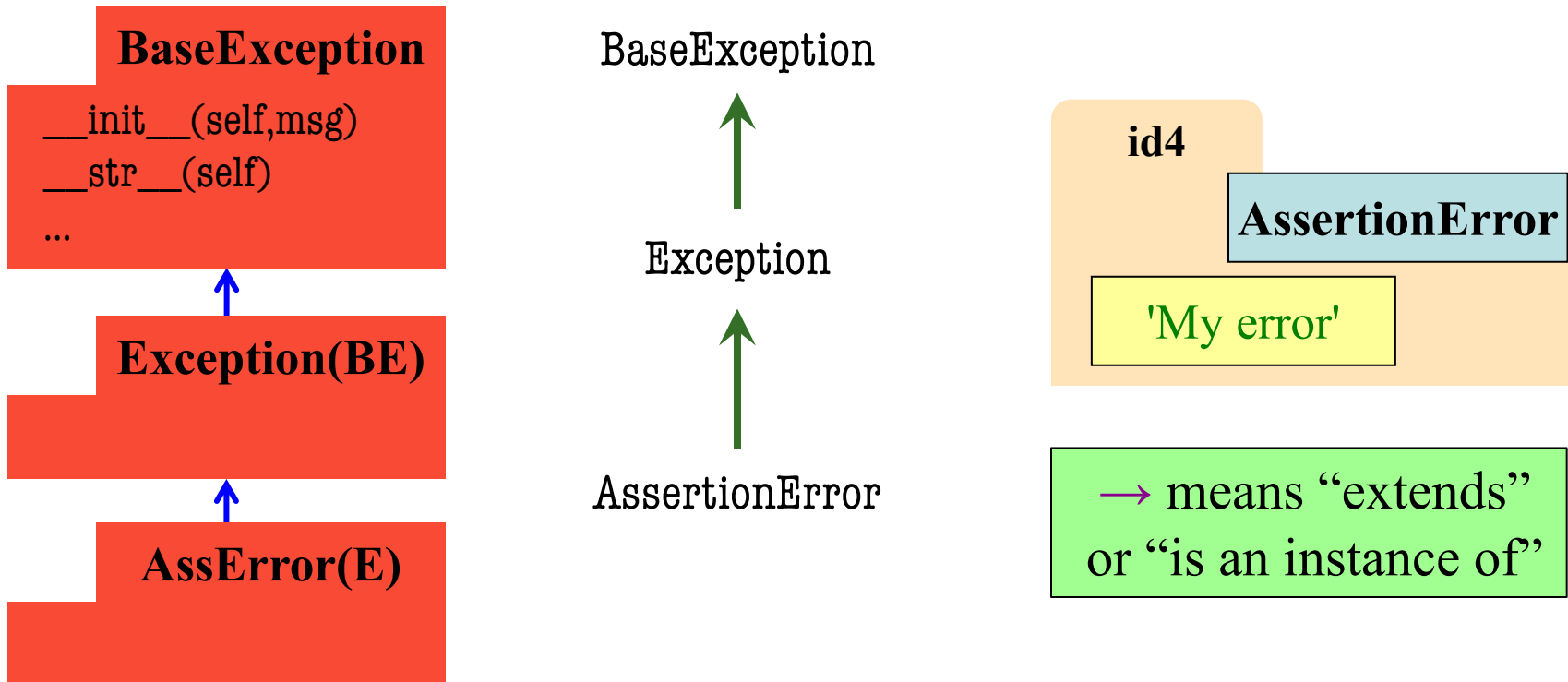
Information about an error is stored inside an **object**. The error type is the **class** of the error object.

```
>>> foo()
```

```
ZeroDivisionError: integer  
division or modulo by zero
```

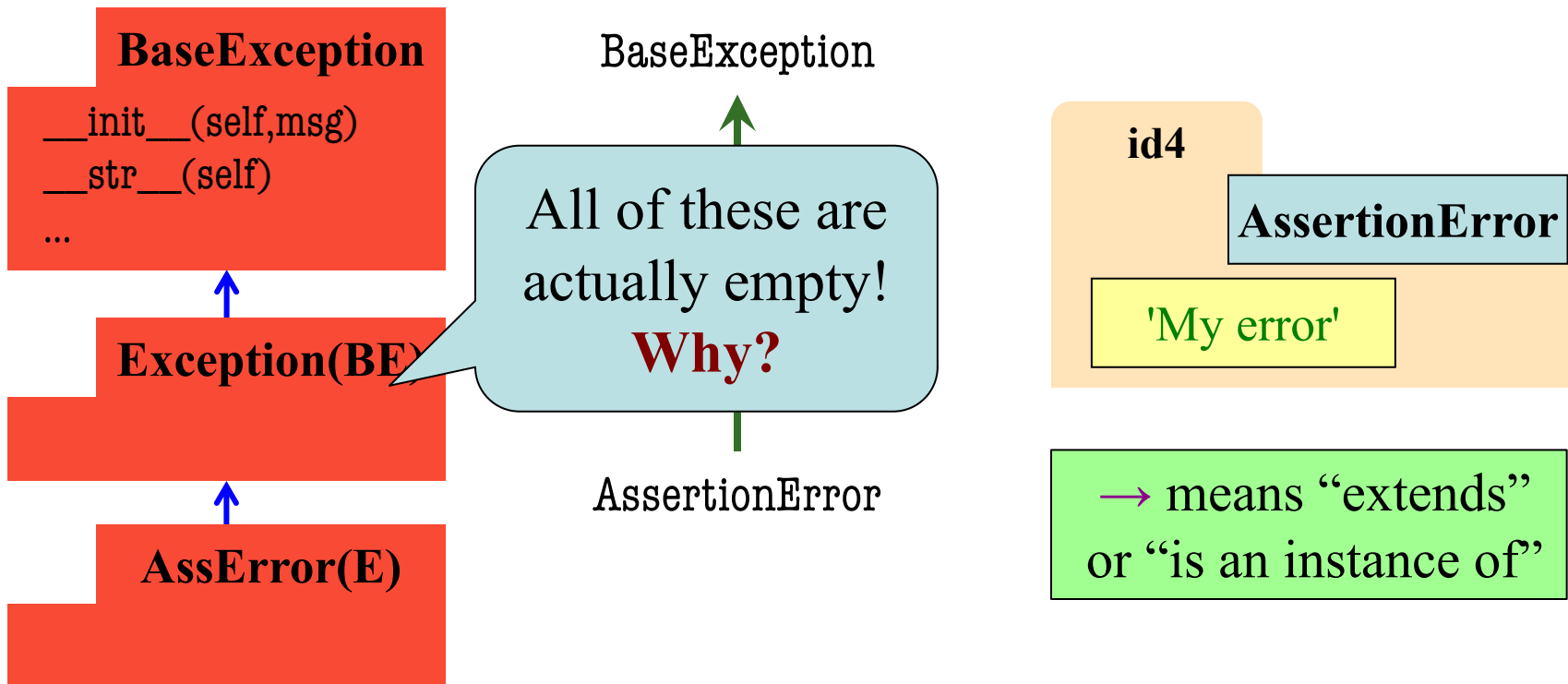
Error Types in Python

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

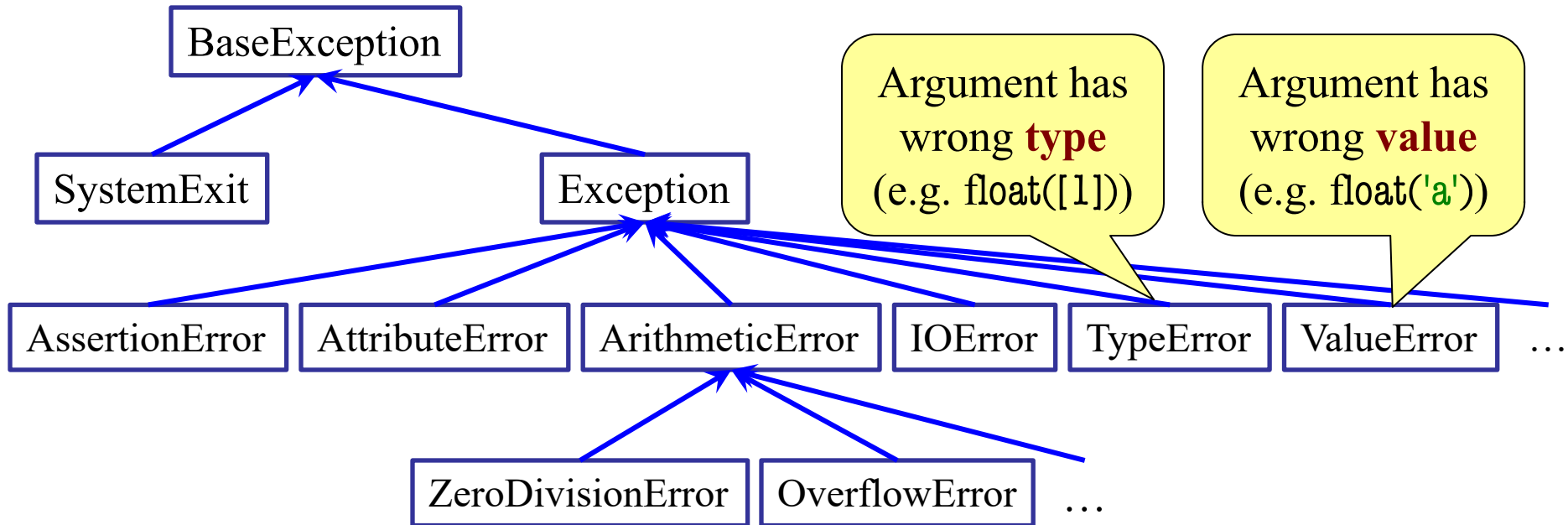


Error Types in Python

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



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 except
- **Example:**

try:

```
val = input()      # get number from user
x = float(val)     # convert string to float
print('The next number is '+str(x+1))
```

might have an error



except:

```
print('Hey! That is not a number!')
```

executes if have an error



Handling Errors by Type

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

- **Example:**

try:

```
val = input()      # get number from user
```

```
x = float(val)    # convert string to float
```

```
print('The next number is '+str(x+1))
```

May have
KeyboardInterrupt



May have ValueError

except ValueError:

```
print('Hey! That is not a number!')
```

Only recovers ValueError.
Other errors ignored.



Handling Errors by Type

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

- **Example:**

try:

```
val = input()      # get number from user
x = float(val)     # convert string to float
print('The next number is '+str(x+1))
```

except KeyboardInterrupt:

```
print('Check your keyboard!')
```

May have
KeyboardInterrupt



May have ValueError

Only recovers
KeyboardInterrupt.
Other errors ignored.



Handling Errors by Type

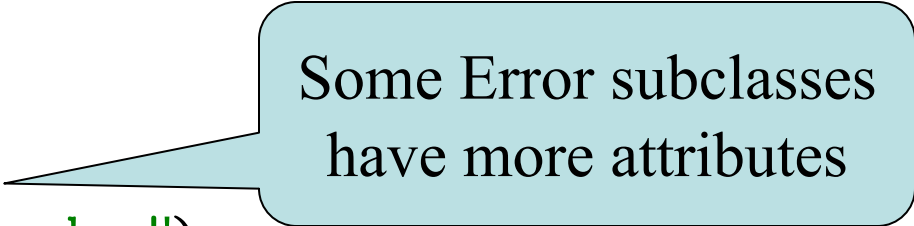
- try-except can put the error in a variable
- **Example:**

try:

```
val = input()      # get number from user
x = float(val)     # convert string to float
print('The next number is '+str(x+1))
```

except ValueError as e:

```
print(e.args[0])
print('Hey! That is not a number!')
```



Some Error subclasses
have more attributes

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'
```

```
    ...
```

```
def foo(x):
```

```
    if x >= 2:
```

```
        m = 'My error'
```

```
        err = AssertionError(m)
```

```
        raise err
```

Identical



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'
```

```
    ...
```

```
def foo(x):
```

```
    if x >= 2:
```

```
        m = 'My error'
```

```
        err = ValueError(m)
```

```
        raise err
```

Identical

Raising and Try-Except

```
def foo():  
    x = 0  
  
    try:  
        raise Exception()  
        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 Exception()  
        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 Exception()  
        x = 2  
    except BaseException:  
        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 Exception()  
        x = 2  
    except BaseException:  
        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 Exception()  
        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 Exception()  
        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(Exception):  
    """An instance is a custom exception"""  
    pass
```

This is all you need!

- No extra attributes
- No extra methods
- No constructors

Inherit everything

Only issue is choice of parent error class. Use `Exception` if you are unsure what.

Case Study: Files

- Can read the contents of any file with `open()`
 - Returns a file object with method `read()`
 - Method `read()` returns contents as a string
 - Remember to `close()` file when done
- There are **SO** many errors that can happen
 - **FileNotFoundError**: File does not exist
 - **PermissionError**: You are not allowed to read it
 - Other errors possible when processing data

Recall: JSON Files

```
{
  "wind" : {
    "speed" : 13.0,
    "crosswind" : 5.0
  },
  "sky" : [
    {
      "cover" : "clouds",
      "type" : "broken",
      "height" : 1200.0
    },
    {
      "type" : "overcast",
      "height" : 1800.0
    }
  ]
}
```

- Look like a nested dict
 - But read in as a string
 - You have to **convert** it
- Python module json
 - Function `loads()`
Converts str -> dict
 - Function `dumps()`
Convert dict -> str
- Conversion is sensitive
 - Stray commas crash it

Reading a JSON File

```
def read_json(fname):  
    try:  
        file = open(fname)  
        data = file.read()  
        file.close()  
        result = json.loads(data)  
        return result  
    except FileNotFoundError:  
        print(fname + ' not found')  
    except JsonDecodeError:  
        print(fname + ' is invalid')  
    return None
```

Open file
with name

Close file
when done

Note that we can
chain excepts like
an if-elif statement

Could not
find file

JSON contents
are not valid

If failed

Reading a File in General

```
def read_foo(fname):  
    try:  
        file = open(fname)  
        data = file.read()  
        file.close()  
        result = convert(data)  
        return result  
    except FileNotFoundError:  
        print(fname + ' not found')  
    except MyConversionError:  
        print(fname + ' is invalid')  
    return None
```

All the work is
in conversion step

Custom helper
for this file type

Error specific
to the file format

Aside: Pathnames

- Files obey the same rule as other modules
 - To read a file, it must be in the same folder
 - Otherwise, you must use a pathname for file
- **Relative path**: directions from current folder
 - **macOS**: `'../..../lec22/file.txt'`
 - **Windows**: `'..\..\lec22\file.txt'`
- **Absolute path**: directions that work anywhere
 - **macOS**: `'/Users/white/cs1110/lect22/file.txt'`
 - **Windows**: `'C:\Users\white\cs1110\lect22\file.txt'`

Like navigating
command shell

Aside: Pathnames

- Files obey the same rule as other modules
 - To read a file, it must be in the same folder
 - Otherwise, you must specify the full path for file
- **Relative path:** directions from current folder
 - **macOS:** `'../..../lec22/file.txt'`
 - **Windows:** `'..\..\lec22\file.txt'`
- **Absolute path:** directions that work anywhere
 - **macOS:** `'/Users/white/cs1110/lect22/file.txt'`
 - **Windows:** `'C:\Users\white\cs1110\lect22\file.txt'`

Note the change
in slash direction

Like navigating
command shell

Pathnames are OS Specific

- This makes reading files harder
 - May work on Windows but crash on macOS!
 - Yet another error message we need to handle
- **Solution:** Use the module `os.path`
 - Builds a pathname string for current os
- **Example:** `os.path('..', 'cs1110', 'lec22', 'file.txt')`
 - **macOS:** `'../cs1110/lec22/file.txt'`
 - **Windows:** `'..\cs1110\lec22\file.txt'`
- Absolute paths are a little trickier, but similar

Final Word on Error Handling

- Versions of **try-except** exist in most languages
 - Java, C++, C#, Objective-C all have it
- But those languages try to **minimize** its use
 - Give application a way to crash “nicely”
 - Because processing a try-except is quite slow
- Python has a very **different** philosophy
 - Python is sort-of slow; exceptions are not slower
 - It is okay to use **try-except** all the time
 - Encourages its use as much as **if**-statements

Final Word on Error Handling

- Versions of **try-except** exist in most languages
 - Java, C++, C#, Objective-C all have it
 - But those languages try to **minimize** its use
 - Give application a way to crash “nicely”
 - Because processing a try-except it quite slow
 - Python has **try-except** why
 - Python is not slower
 - It is okay
 - Encourages developers
- Developers refer to coding styles unique to python as **pythonic** programming