

### A Problem with Subclasses

```

class Fraction(object):
    """Instances are normal fractions n/d"""
    # INSTANCE ATTRIBUTES
    # _numerator: int
    # _denominator: int > 0

class FractionalLength(Fraction):
    """Instances are fractions with units"""
    # INSTANCE ATTRIBUTES same but
    # _unit: one of 'in', 'ft', 'yd'
    def __init__(self,n,d,unit):
        """Make length of given units"""
        assert unit in ['in', 'ft', 'yd']
        super().__init__(n,d)
        self._unit = unit
    
```

Python converts to

```

>>> p = Fraction(1,2)
>>> q = FractionalLength(1,2,'ft')
>>> r = p*q

>>> r = p.__mul__(q) # ERROR
    
```

mul has precondition type(q) == Fraction

1

### The isinstance Function

- `isinstance(<obj>, <class>)`
  - True if <obj>'s class is same as or a subclass of <class>
  - False otherwise
- Example:**
  - `isinstance(e, Executive)` is True
  - `isinstance(e, Employee)` is True
  - `isinstance(e, object)` is True
  - `isinstance(e, str)` is False
- Generally preferable to `type`
  - Works with base types too!

e id4

Executive
_name: Fred
_start: 2012
_salary: 0.0
_bonus: 0.0

2

### Fixing Multiplication

```

class Fraction(object):
    """Instances are fractions n/d"""
    # _numerator: int
    # _denominator: int > 0

    def __mul__(self,q):
        """Returns: Product of self, q
        Makes a new Fraction; does not
        modify contents of self or q
        Precondition: q a Fraction"""
        assert isinstance(q, Fraction)
        top = self.numerator*q.numerator
        bot = self.denominator*q.denominator
        return Fraction(top,bot)
    
```

Python converts to

```

>>> p = Fraction(1,2)
>>> q = FractionalLength(1,2,'ft')
>>> r = p*q

>>> r = p.__mul__(q) # OKAY
    
```

Can multiply so long as it has numerator, denominator

3

### Error Types in Python

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

id4

AssertionError
'My error'

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

4

### Python Error Type Hierarchy

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

Why so many error types?

5

### 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 ValueError:
    print("Hey! That is not a number!")
    
```

Annotations: May have IOError, May have ValueError, Only recovers ValueError. Other errors ignored.

6

### 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'
        err = AssertionError(m)
        raise err

```

7

### Creating Your Own Exceptions

```

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

```

This is all you need

- No extra fields
- No extra methods
- No constructors

Inherit everything

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

8

### 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

9

### 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

Could not find file

JSON contents are not valid

If failed

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

10

### 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:** '../..lect22/file.txt'
  - **Windows:** '..\..\lect22\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

11

### 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

12