Announcements

The End is Nigh!

1. Next (last) lecture will be recap and final exam review
2. A5 due Wednesday night
3. Final exam **7pm Thursday May 15** in Barton Hall
   (East section)
== versus is

class Card(object):
    def __init__(self, s, r):
        self.suit = s
        self.rank = r

    def __eq__(self, other):
        return (isinstance(other, Card) and
                (self.suit, self.rank) ==
                (other.suit, other.rank))

    def __ne__(self, other):
        return not self.__eq__(other)

c = Card(3,2)
d = Card(3,2)
e = c
print c    # <Card object at 0x100497b10>
print d    # <Card object at 0x100497b50>
print c == d    # True
print c is d    # False
print e is c    # True

- When you define a class, you might like to define what it means for instances to be equal.
- To do this you define the __eq__ and __ne__ methods (overriding the default ones in the class object).
- But now what if you want to tell if two cards are the same object? Use is instead of ==.
- And, whenever you are really talking about equality of object identity, use is (e.g. is None).

\[1\] Before you do this for real, read about __hash__.
Dispatch on Type

- Sometimes you have an object that might be one of several types, and you need to know which it is.
- Example: have list of GObject instances, want to turn all the ellipses green and the rectangles blue. Leave the other shapes alone. Simple enough:

```python
for shape in shapes:
    if type(shape) is GEllipse:
        shape.fillcolor = colormodel.GREEN
    elif type(shape) is GRectangle:
        shape.fillcolor = colormodel.BLUE
```

Finds any object whose class is GEllipse.
Dispatch on Type

• Problem: some of your shapes might actually be subclasses (in A7, a GEllipse might be the Ball).
• Solution: the built-in function isinstance. It answers the question, “Is this object an instance of this class?” and an instance of a subclass counts.

for shape in shapes:
    if isinstance(shape, GEllipse):
        shape.fillcolor = colormodel.GREEN
    elif isinstance(shape, GRectangle):
        shape.fillcolor = colormodel.BLUE

Finds any object whose class is GEllipse or a subclass of GEllipse.
Dispatch on Type vs. Method Overriding

class Ball(object):
    ...

class SuperBall(Ball):
    ...

class BallOfClay(Ball):
    ...

... 

if isinstance(my_ball, SuperBall):
    ball.vy = -0.99 * ball.vy 
elif isinstance(my_ball, BallOfClay):
    ball.vy = -0.05 * ball.vy 
else:
    ball.vy = -0.5 * ball.vy 

my_ball.rebound()
(Runtime) errors are exception objects

When various bad things happen, Python creates an exception object.

If that object is not otherwise "handled", the system halts, printing the stack trace and info about the exception object.

ZeroDivisionError: integer division or modulo by zero

name of the type of the exception object

string kept in the exception object
Hierarchy of exceptions

Exception

SystemExit
StandardError

ArgumentException
AttributeError
ArithmeticError
IOError
TypeError

ZeroDivisionError
OverflowError

Argument has wrong **type** (e.g. `float([1])`)
Argument has wrong **value** (e.g. `float('a')`)

http://docs.python.org/library/exceptions.html
Recovering from errors: Try-except

Try-except blocks allow us to recover from errors
- Do the code that is in the try-block
- If an error occurs, jump to the except-block (skip it o.w.)

```python
def recip(x):
    """Return 1.0/x, or inf if x is 0. Pre: x is a number""
    try:
        return 1.0/x
    except:
        return float('Inf')
```

executes if an error occurs
Recovering from specific error types

You can have except-blocks that are executed only if the exception is an instance of a particular class.

def recip(x):
    """Return 1.0/x, or inf if x is 0""
    try:
        return 1.0/x
    except ZeroDivisionError:
        return float('Inf')
Creating exceptions: raise

You can signal errors by creating exceptions with `raise`.

```python
def speed(x):
    if x > 3e8:
        raise ValueError('speed: input > light speed')
```

- As usual, creates a new object.
- The type is informative to the user (or enclosing except-blocks).
- You can choose an informative output message.
Important example of type-based dispatch

try:
    input = raw_input()
    x = float(input)
    print 'The next number is ' + str(x+1)
except ValueError:
    print 'Hey! That is not a number!'

try:
    input = raw_input()
    x = float(input)
    print 'The next number is ' + str(x+1)
except StandardError:
    print 'Hey! We had a problem!'

• Exception handling effectively uses isinstance to match exceptions to exception handlers.

Matches any exception whose class is ValueError or any subclass of ValueError.

Matches any exception whose class is StandardError or any subclass of StandardError (including ValueError or IOError).
Recall: Hierarchy of exceptions

- Exception
  - SystemExit
  - StandardError
    - ArgumentError
    - AttributeError
    - ArithmeticError
    - AssertionError
    - IOError
    - OverflowError
    - ValueError
    - TypeError
    - ZeroDivisionError

Argument has wrong **type** (e.g. `float([1])`)
Argument has wrong **value** (e.g. `float('a')`)

http://docs.python.org/library/exceptions.html
# Dictionaries (Type `dict`)

<table>
<thead>
<tr>
<th>Description</th>
<th>Python Syntax</th>
</tr>
</thead>
</table>
| - List of **key-value** pairs  
  - Keys are unique  
  - Values need not be  
- Example: net-ids  
  - net-ids are **unique** (a key)  
  - names need not be (values)  
  - `js1` is John Smith (class ’13)  
  - `js2` is John Smith (class ’16)  
- Many other applications | - Create with format:  
  `{k1:v1, k2:v2, ...}`  
- Keys must be non-mutable  
  - ints, floats, bools, strings  
  - **Not** lists or custom objects  
- Values can be anything  
- Example:  
  `d = { 'js1':'John Smith',  
        'js2':'John Smith',  
        'wmw2':'Walker White' }` |
Using Dictionaries (Type `dict`)

- Access elts. like a list
  - `d['js1']` evaluates to `'John'`
  - But cannot slice ranges!
- Dictionaries are **mutable**
  - Can reassign values
    - `d['js1'] = 'Jane'`
  - Can add new keys
    - `d['aa1'] = 'Allen'`
  - Can delete keys
    - `del d['wmw2']`

```python
d = {'js1': 'John', 'js2': 'John', 'wmw2': 'Walker'}
```

Key-Value order in folder is not important
Using Dictionaries (Type `dict`)

- Access elts. like a list
  - `d['js1']` evaluates to 'John'
  - But cannot slice ranges!
- Dictionaries are **mutable**
  - Can reassign values
    - `d['js1'] = 'Jane'`
  - Can add new keys
    - `d['aal'] = 'Allen'`
  - Can delete keys
    - `del d['wmw2']`

```python
d = {'js1':'John','js2':'John',
    'wmw2':'Walker'}
```

Key-Value order in folder is not important
Using Dictionaries (Type dict)

- Access elts. like a list
  - `d['js1']` evaluates to 'John'
  - But cannot slice ranges!
- Dictionaries are **mutable**
  - Can reassign values
  - `d['js1'] = 'Jane'`
  - Can add new keys
  - `d['aa1'] = 'Allen'`
  - Can delete keys
  - `del d['wmw2']`

```python
import id0

d = {'js1': 'John', 'js2': 'John', 'wmw2': 'Walker'}
```

![Diagram of dictionary with keys and values]

- **id0**
- **dict**
- `'js1'` `'Jane'`
- `'js2'` `'John'`
- `'wmw2'` `'Walker'`
- `'aa1'` `'Allen'`
Using Dictionaries (Type `dict`)

- Access elts. like a list
  - `d['js1']` evaluates to 'John'
  - But cannot slice ranges!
- Dictionaries are **mutable**
  - Can reassign values
    - `d['js1'] = 'Jane'`
  - Can add new keys
    - `d['aa1'] = 'Allen'`
  - Can delete keys
    - `del d['wmw2']`

``` python
d = {'js1': 'John', 'js2': 'John', 'wmw2': 'Walker'}
```

Deleting key deletes both
Dictionaries and For-Loops

- Dictionaries != sequences
  - Cannot slice them
  - Cannot use in for-loop
- But have methods to give you related sequences
  - Seq of keys: `d.keys()`
  - Seq of values: `d.values()`
  - Seq of key-value pairs: `d.items()`
- Use these in for-loops

```python
for k in d.keys():
    print k
    print d[k]

for v in d.values():
    print v

for k, v in d.items():
    print k
    print v
```