Meeting 21: 3/31/2015: Objects

Announcements

- HW6 due 4/11

HW6 Objects - Questions

- Use of while
- Calling constructor

P3 Objects, Classes, While Loops, If Statements

Python is very different than Java!
→ Attributes of both classes and objects can be dynamically extended.

→ Classes "live" at run time.

→ Classes/objects are first-class values.

→ Methods that are first-class:
  → Functions + receiver object "self"
  → Closure
    "Bound method"

→ "Unbound method" is a "bound method" waiting for its receiver.

Closure + class
expression ::= ... 
  | expression " . " identifier 
  | attribute lookup 
  | x.f 
  | C.f 
  | T 
  | convention only

statement ::= ... 
  | "class" name ["(" expression_list ")"] : suite 

suite ::= statement+ 

"class" ::= like "def" 

class C: 
  C = class lambda:
It introduces a new scope

"class locals" = class attributes

```python
class C:
    x = 5
```

Simple statement `ii = ...`

```
1 target = "expression"

target = expression" identifier
1 ...
```

`x.f = 10`

Statement `ii = ...`

```
1 if ... 
1 while 
```
```python
class C:
    x = 42
    print C.x

class C:
    print 4*10 + 2
C.x = 10

class C:
    f=lambda o,dx: o.x+dx

class C:
    def f(o, dx):
        return o.x+dx

What is stored in the attribute is a normal closure

x = C.f  # an unbound method gets stored in x
```
class C:
    x = 42

class D:
    x = 0

print(C if x == 2 else D).x

---

Inheritance

class A:
    x = 4

class B:
    x = 0
    y = 2

class C(A, B):
    z = 3
    B.y = 5
    print(C.x * 10 + C.y)

resolves attributes
depth-first, left-to-right

Yes, prints 45

So inheritance implemented
by keeping references to
super classes
Objects

```
class C:
    o = C(42)
    o.x = 10
```

```
class C:
    def __init__(o, n):
        print(n)

o = C(42)
```
\[ p = c(42) \]
\[ p.x = 42 \]

\[
\text{print } o.x, p.x
\]

\[ q = 0 \]
\[ q.x = 1 \]

\[
\text{print } o.x \quad \text{I printed}
\]

\[ o.f = \lambda x: \_ \]

**Attribute Resolution for objects**

```python
class C:
    def move(o, dx):
        o.x = o.x + dx

y = C()
y.move
```

`create board method with closure from C.move`
\( x(10) \)

```
big_py_obj

class: list of base classes
and a dictionary of attributes

object: its class
and a dictionary of attributes

unbound method: closure + class

\[
y = C \cdot x \quad \xrightarrow{\text{closure}} \quad \Box
\]

bound method: closure + object

\[
y = o \cdot x \quad \xrightarrow{\text{closure}} \quad \Box
\]
```