Attributes
Announcements
Method Calls
Dot Expressions

Methods are invoked using dot notation

<expression> . <name>

The <expression> can be any valid Python expression.

The <name> must be a simple name.

Evaluates to the value of the attribute looked up by <name> in the object that is the value of the <expression>

(Demo)
Looking Up Attributes by Name

Both instances and classes have attributes that can be looked up by dot expressions

<expression> . <name>

To evaluate a dot expression:

1. Evaluate the <expression> to the left of the dot, which yields the object of the dot expression

2. <name> is matched against the instance attributes of that object; if an attribute with that name exists, its value is returned

3. If not, <name> is looked up in the class, which yields a class attribute value

4. That value is returned unless it is a function, in which case a bound method is returned instead
Discussion Question: Where's Waldo?

For each class, write an expression with no quotes or + that evaluates to 'Waldo'

```
class Town:
    def __init__(self, w, aldo):
        if aldo == 7:
            self.street = {self.f(w): 'Waldo'}

    def f(self, x):
        return x + 1

>>> Town(1, 7).street[2]
'Waldo'
```

```
class Beach:
    def __init__(self):
        sand = ['Wal', 'do']
        self.dig = sand.pop

    def walk(self, x):
        self.wave = lambda y: self.dig(x) + self.dig(y)
        return self

>>> Beach().walk(0).wave(0)
'Waldo'
```

Reminder: s.pop(k) removes and returns the item at index k
Class Attributes
The Class Statement

A class statement creates a new class and binds that class to `<name>` in the first frame of the current environment.

Assignment & def statements in `<suite>` create attributes of the class (not names in frames)

```python
>>> class Clown:
...     nose = 'big and red'
...     def dance():
...         return 'No thanks'
... >>> Clown.nose
'big and red'
>>> Clown.dance()
'No thanks'
>>> Clown
<class '__main__.Clown'>
```
Class Attributes

Class attributes are "shared" across all instances of a class because they are attributes of the class, not the instance

class Account:

    interest = 0.02  # A class attribute

    def __init__(self, account_holder):
        self.balance = 0
        self.holder = account_holder

    # Additional methods would be defined here

>>> tom_account = Account('Tom')
>>> jim_account = Account('Jim')
>>> tom_account.interest
0.02
>>> jim_account.interest
0.02

The interest attribute is not part of the instance; it's part of the class!
Bound Methods
Terminology: Attributes, Functions, and Methods

All objects have attributes, which are name-value pairs
A class is a type (or category) of objects
Classes are objects too, so they have attributes
Instance attribute: attribute of an instance
Class attribute: attribute of the class of an instance

Terminology:

Python object system:
Functions are objects
Bound methods are also objects: a function that has its first parameter "self" already bound to an instance
Dot expressions evaluate to bound methods for class attributes that are functions
<instance>.<method_name>
Methods and Functions

Python distinguishes between:

- **Functions**, which we have been creating since the beginning of the course, and
- **Bound methods**, which couple together a function and the object on which that method will be invoked

\[ \text{Object} + \text{ Function} = \text{Bound Method} \]

```python
>>> type(Account.deposit)
<class 'function'>
>>> type(tom_account.deposit)
<class 'method'>
```

```python
>>> Account.deposit(tom_account, 1001)
1011
>>> tom_account.deposit(1007)
2018
```

**Function**: all arguments within parentheses

**Method**: One object before the dot and other arguments within parentheses
Attribute Assignment
Attribute Assignment Statements

Account class attributes

interest: 0.02 0.04 0.05
(withdraw, deposit, __init__)

Instance attributes of jim_account

balance: 0
holder: 'Jim'
interest: 0.08

Instance attributes of tom_account

balance: 0
holder: 'Tom'

>>> jim_account = Account('Jim')
>>> tom_account = Account('Tom')
>>> tom_account.interest
0.02
>>> jim_account.interest
0.02
>>> Account.interest = 0.04
>>> tom_account.interest
0.04
>>> jim_account.interest
0.04

>>> jim_account.interest = 0.08
>>> jim_account.interest 0.08
>>> tom_account.interest
0.04
>>> Account.interest = 0.05
>>> tom_account.interest
0.05
>>> jim_account.interest
0.08