Attributes

Announcements

Method Calls

Dot Expressions

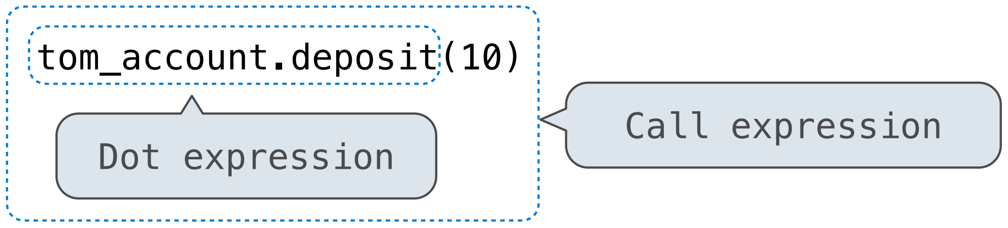
Methods are invoked using dot notation

<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>



```
tom_account.deposit(10)
```

(Demo)

Attribute Lookup

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
```

Reminder: `s.pop(k)` removes and returns the item at index k

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

Class Attributes

Class Attributes

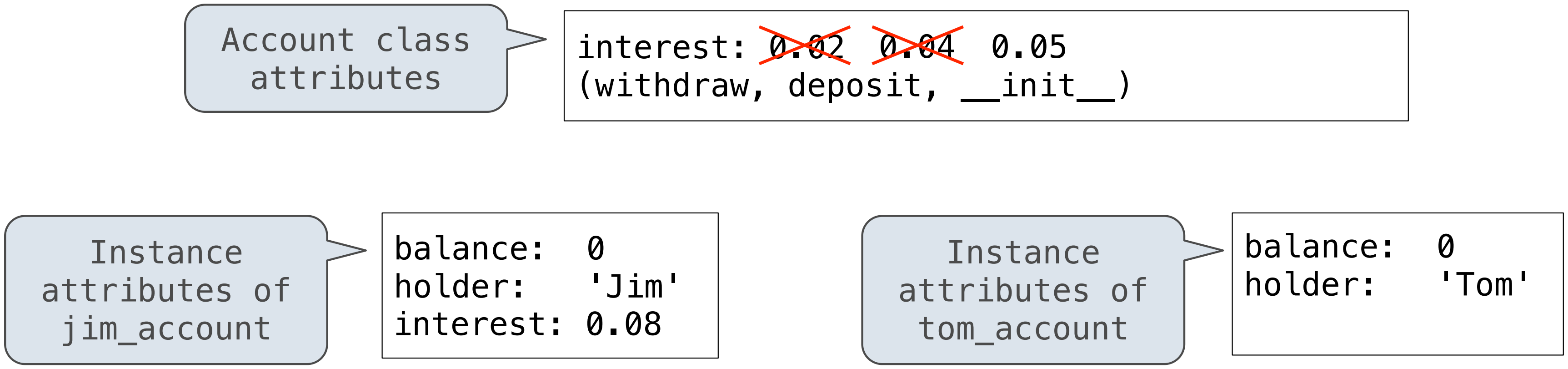
Class attributes are "shared" across all instances of a class because they are attributes of the class, not the instance

```
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!

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
```

Discussion Question: Class Attribute Assignment

Implement the **Place** class, which takes a **name.** Its **print_history**() method prints the **name** of the **Place** and then the names of all the **Place** instances that were created before it.

```
class Place:

    last = None

    def __init__(self, n):

        self.name = n

        self.then = Place.last

        Place.last = self


    def print_history(self):

        print(self.name)

        if self.then is not None:

            self.then.print_history()
```

OK to write **self.last** or **type(self.last)**

Not ok to write **self.last**

```
>>> places = [Place(x*2) for x in range(10)]
>>> places[4].print_history()
8
6
4
2
0
>>> places[6].print_history()
12
10
8
6
4
2
0
```