Lecture #13: Objects and Classes

Data Abstraction vs. Function Abstraction

• Functions perform computations; their specifications abstract from possible implementations of a particular computation.

• In the old days, programs tended to be organized around functions or modules comprising related functions. The data were just the operands.

• Now we tend to organize instead around the data—around objects or types (classes) of objects.

• Objects have state, which is accessed and manipulated by means of attributes.

• The set of attributes and their behavior is analogous to the syntactic and semantic specification of a function.

• In previous lectures, we've seen standard Python objects and ways to get (in effect) new kinds of objects using functions and non-local variables. We've defined data types using them by defining a set of functions to be used to construct, query, and modify them.

• Python also provides standard new types of data: classes.

Extending the Mutable Objects: Classes

• In languages such as Python, Java, and C++, an object is an instance of a class; the class is called the object's type.

• The Python class statement defines new classes or types, creating new, vaguely dictionary-like varieties of object.

Simple Classes: Bank Account

```python
class Account:
    def __init__(self, initial_balance):
        self.balance = initial_balance
    def balance(self):
        return self.balance
    def deposit(self, amount):
        if amount < 0:
            raise ValueError("negative deposit")
        self.balance += amount
    def withdraw(self, amount):
        if 0 <= amount <= self.balance:
            self.balance -= amount
        else:
            raise ValueError("bad withdrawal")

>>> mine = Account(1000)
>>> mine.deposit(100)
>>> mine.balance()
1100
>>> mine.withdraw(200)
>>> mine.balance()
900
```

Class Concepts

• Just as def defines functions and allows us to extend Python with new operations, class defines types and allows us to extend Python with new classes of objects.

Class Machinery

• The Account type illustrated how we do each of these:

  - A way of defining named new types of data.
  - A means of defining and accessing state for these objects.
  - A means of defining operations specific to these objects.

  - A means of creating new objects.

In particular, an operation for initializing the state of an object.

In languages such as Python, Java, and C++, an object is an instance of a class; the class is called the object's type.

• The Python class statement defines new classes or types, creating new, vaguely dictionary-like varieties of object.

• Python also provides standard new types of data: classes.

Extending the Mutable Objects: classes
In general, the notation $X.Y$ means "The value named $Y$ in the object pointed to by $X".

Unlike C++ or Java, Python takes a very dynamic approach.

Classes and class instances behave rather like environment frames.

Given a pointer to some object, $\text{obj}$,

- $\text{obj}.x = \text{value}$ looks for a definition of $x$ in the object referenced by $\text{obj}$, creating one if it doesn't exist, and assigning $\text{value}$ to it.

- When not being assigned to, $\text{obj}.x$ returns the definition of $x$ in the object referenced by $\text{obj}$, if any, . . . and if there is no such definition, it returns the value defined for $x$ in the class itself, if any.

### Attributes in Python

```python
class Account:
    total_deposits = 0
    def __init__(self, initial_balance):
        self.balance = initial_balance
        Account.total_deposits += initial_balance

acct1 = Account(1000)
acct2 = Account(10000)
acct1.deposit(300)
```

### Assigning to Attributes

Assigning to an attribute of an object (including a class) is like assigning to a local variable: it creates a new binding for that attribute in the object selected from (i.e., referenced by the expression on the left of the dot).

```python
class Value:
    value = 0

val1 = Value()
val2 = Value()
val2.value = 3
val1.value
```

### Methods

```python
class Foo:
    def set(self, x):
        self.value = x

aFoo = Foo(10)
```

### Class Attributes in Python

Sometimes, a quantity applies to a type as a whole, not a specific instance.

```python
def set_deposits(self, total):
    self.total_deposits = total

class Account:
    total_deposits = 0
    def __init__(self, initial_balance):
        self.balance = initial_balance
        self.set_deposits(initial_balance)
```

```python
acct1 = Account(1000)
acct1.set_deposits(10000)
acct2 = Account(1000)
```

```python
>>> Account.total_deposits
12000
>>> acct1.balance
1000
>>> acct2.balance
2000
```
Class Attribute Example

```python
class Account:
    total_deposits = 0  # Define/initialize a class attribute

    def __init__(self, initial_balance):
        self.balance = initial_balance
        Account.total_deposits += initial_balance

    def deposit(self, amount):
        self.balance += amount
        Account.total_deposits += amount

    @classmethod
    def total_deposits(cls):
        return Account.total_deposits
```

```python
>>> acct1 = Account(1000)
>>> acct2 = Account(10000)
>>> acct1.deposit(300)
>>> Account.total_deposits()
11300
```

Classes and Operators

- Many standard operators defined in Python are essentially "syntax sugar" for method calls.

Class Machinery: Summary

- Classes have attributes, created by assignment statements and defs in the class body.
- Function-values attributes of classes are called methods.
- Classes beget objects called instances, created by "calling" the class: `Account(1000)`.
- Each such Account object initially shares the attributes of its class.
- Attributes can be accessed using object.attribute notation.
- A method call essentially the same as `Account.deposit(mine, 100)`.
- By convention, we call the first argument of a method `self` to indicate that it is the object from which we got the method.
- Assigning to an attribute of an object (`a.a = v`) gives that object its own attribute (not shared with the class) if it doesn't have it already.
- When an object is created, the special `__init__` method is called on the object using object.attribute notation.
- By convention, we call the first argument of a method `self` to indicate that it is the object from which we got the method.
- Assigning to an attribute of an object (`a.a = v`) gives that object its own attribute (not shared with the class) if it doesn't have it already.

Class Attribute Example