Object-Oriented Programming

A method for organizing programs
- Bundling together information and related behavior
- A metaphor for computation using distributed state
- Each object has its own local state
- Each object also knows how to manage its own local state, based on method calls
- Method calls are messages passed between objects
- General objects may all be instances of a common type
- Different types may relate to each other

Object Identity

Every object that is an instance of a user-defined class has a unique identity:

```python
>>> a = Account('Jack')
>>> b = Account('Jack')
>>> a.balance = 15
>>> b.balance = 15
>>> a.balance == b.balance
True
```

Identity operators "is" and "is not" test if two expressions evaluate to the same object:

```python
>>> a is b
False
>>> a is not b
True
```

Binding an object to a new name using assignment does not create a new object:

```python
>>> c = a
>>> c is a
True
```

Class Statements

A class statement creates a new class and binds that class to <name>

```python
class Account:
    def __init__(self, account_holder):
        self.balance = 0
        self.account_holder = account_holder
```

Methods

Methods are functions defined in the suite of a class statement

```python
class Account:
    def __init__(self, account_holder):
        self.balance = 0
        self.account_holder = account_holder
    def deposit(self, amount):
        self.balance += amount
    def withdraw(self, amount):
        if self.balance >= amount:
            self.balance -= amount
        else:
            raise ValueError('Insufficient funds')
```

Each object also knows how to manage its own local state,

```python
a = Account('Jack')
a.deposit(15)
a.balance
```

Each object can also tell who it is,

```python
a.account_holder
```

A class serves as a template for its instances

```python
class Account:
    def __init__(self, account_holder):
        self.balance = 0
        self.account_holder = account_holder

>>> a = Account('Jack')
>>> a.balance
0
```

```python
>>> a.deposit(15)
>>> a.balance
15
>>> a.withdraw(15)
>>> a.balance
0
```

A class statement creates function objects as always, but their names are bound as attributes of the class.
Invoking Methods

All invoked methods have access to the object via the self parameter, and so they can all access and manipulate the object’s state.

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

Invoking with one argument
Invoked with two parameters

Dot notation automatically supplies the first argument to a method

Accessing Attributes

By using getattr, we can look up an attribute using a string.

```python
>>> getattr(tom_account, 'balance')
10
>>> hasattr(tom_account, 'deposit')
True
```

getattr and dot expressions look up a name in the same way.

Looking up an attribute name in an object may return:
• One of its instance attributes, or
• One of the attributes of its class.

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.

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

Function: all arguments within parentheses
Method: One object before the dot and other arguments within parentheses

Class Attributes

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

```python
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
```

The interest attribute is not part of the instance; it’s part of the class!

Dot Expressions

Objects receive messages via dot notation.

Dot notation accesses attributes of the instance or its class.

```python
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> on the object
that is the value of the <expression>.

```python
tom_account.deposit(100)
```

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

Attributes

Looking Up Attributes by 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.

```python
Class Attributes
```

```python
Accessing Attributes
```

```python
Methods and Functions
```

```python
Dot Expressions
```

```python
Looking Up Attributes by Name
```