# Terminology: Attributes, Functions, and Methods

All objects have attributes, which are name-value pairs.

- Classes are objects too, so they have attributes.
- Instance attributes: attributes of instance objects.
- Class attributes: attributes of class objects.

## Attribute Assignment Statements

To evaluate a dot expression:

1. Evaluate the `<expression>`...
2. `<name>` is matched against the instance attributes...
3. If not found, `<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.

## Inheritance

A technique for relating classes together.

Common use: Similar classes differ in amount of specialization.

Two classes have overlapping attribute sets, but one represents a special case of the other.

```
class <name>(<base class>):
    <suite>
```

Conceptually, the new subclass "shares" attributes with its base class.

The subclass may override certain inherited attributes.

Using inheritance, we implement a subclass by specifying its difference from the base class.
Inheritance Example

A CheckingAccount is a specialized type of Account

```python
>>> ch = CheckingAccount('Tom')  # Tom's checking account
>>> ch.interest  # Lower interest rate for checking accounts
0.01
>>> ch.deposit(20)  # Deposits are the same
20
>>> ch.withdraw(5)  # Withdrawals incur a $1 fee
14
```

Most behavior is shared with the base class Account

```python
class CheckingAccount(Account):
    """A bank account that charges for withdrawals."""
    withdraw_fee = 1
    interest = 0.01
    def withdraw(self, amount):
        return Account.withdraw(self, amount + self.withdraw_fee)
```

Looking Up Attribute Names on Classes

Base class attributes aren't copied into subclasses!

To look up a name in a class.

1. If it names an attribute in the class, return the attribute value.
2. Otherwise, look up the name in the base class, if there is one.

```python
>>> ch = CheckingAccount('Tom')
>>> ch.interest  # Found in CheckingAccount
0.01
>>> ch.deposit(20)  # Found in Account
20
>>> ch.withdraw(5)  # Found in CheckingAccount
14
```

Designing for Inheritance

Don't repeat yourself; use existing implementations

Attributes that have been overridden are still accessible via class objects

Look up attributes on instances whenever possible

```python
class CheckingAccount(Account):
    """A bank account that charges for withdrawals."""
    withdraw_fee = 1
    interest = 0.01
    def withdraw(self, amount):
        return Account.withdraw(self, amount + self.withdraw_fee)
```

Base Class Generality

Base classes may contain logic that is meant for subclasses

Example: Same CheckingAccount behavior; different approach

```python
class CheckingAccount(Account):
    """A bank account that charges for withdrawals."""
    withdraw_fee = 1
    interest = 0.01
    def withdraw(self, amount):
        return Account.withdraw(self, amount + self.withdraw_fee)
```

Inheritance and Composition

Object-oriented programming shines when we adopt the metaphor

Inheritance is best for representing is-a relationships

E.g., a checking account is a specific type of account

Composition is best for representing has-a relationships

E.g., a bank has a collection of bank accounts it manages

No local state at all? Just write a function!

Multiple Inheritance

```python
class SavingsAccount(Account):
    deposit_fee = 2
    def deposit(self, amount):
        return Account.deposit(self, amount - self.deposit_fee)
```

A class may inherit from multiple base classes in Python

Bank of America marketing executive wants:

- Low interest rate of 1%
- A $1 fee for withdrawals
- A $2 fee for deposits
- A free dollar when you open your account

```python
class AsSeenOnTVAccount(CheckingAccount, SavingsAccount):
    def __init__(self, account_holder):
        self.holder = account_holder
        self.balance = 1  # A free dollar!
```

```
Multiple Inheritance

A class may inherit from multiple base classes in Python.

class AsSeenOnTVAccount(CheckingAccount, SavingsAccount):
    def __init__(self, account_holder):
        self.holder = account_holder
        self.balance = 1  # A free dollar!

>>> such_a_deal = AsSeenOnTVAccount("John")
>>> such_a_deal.balance
1
>>> such_a_deal.deposit(20)
19
>>> such_a_deal.withdraw(5)
13

Resolving Ambiguous Class Attribute Names

Human Relationships

- Grandma
- Grandpa
- Gramammy
- Grandaddy
- Aunt
- Double Half Aunt
- Mom
- Dad
- Double Half Uncle
- Some_Guy
- Double Half Cousin
- Some_Dude
- You
- Double
- Quaduple