61A Lecture 16

Friday, October 11

Announcements

Announcements

- Homework 5 is due Tuesday 10/15 @ 11:59pm
- Project 3 is due Thursday 10/24 @ 11:59pm
- Midterm 2 is on Monday 10/28 7pm–9pm

Attributes

Terminology: Attributes, Functions, and Methods

Terminology: Attributes, Functions, and Methods

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

Terminology: Attributes, Functions, and Methods

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

Classes are objects too, so they have attributes
```

Terminology: Attributes, Functions, and Methods

All objects have attributes, which are name-value pairs

Classes are objects too, so they have attributes

Instance attribute: attribute of an instance

Terminology: Attributes, Functions, and Methods

All objects have attributes, which are name-value pairs

Classes are objects too, so they have attributes

Instance attribute: attribute of an instance

Class attribute: attribute of the class of an instance

Terminology: Attributes, Functions, and Methods

All objects have attributes, which are name-value pairs

Classes are objects too, so they have attributes

Instance attribute: attribute of an instance

Class attribute: attribute of the class of an instance

Terminology:

Terminology: Attributes, Functions, and Methods

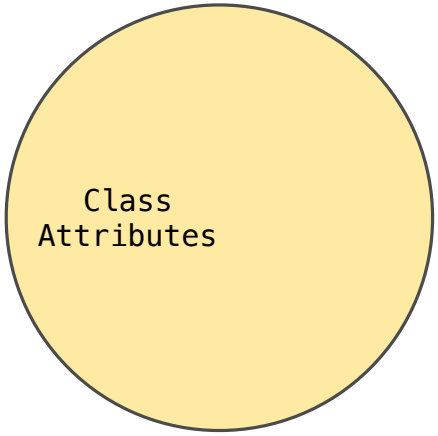
All objects have attributes, which are name-value pairs

Classes are objects too, so they have attributes

Instance attribute: attribute of an instance

Class attribute: attribute of the class of an instance

Terminology:



Terminology: Attributes, Functions, and Methods

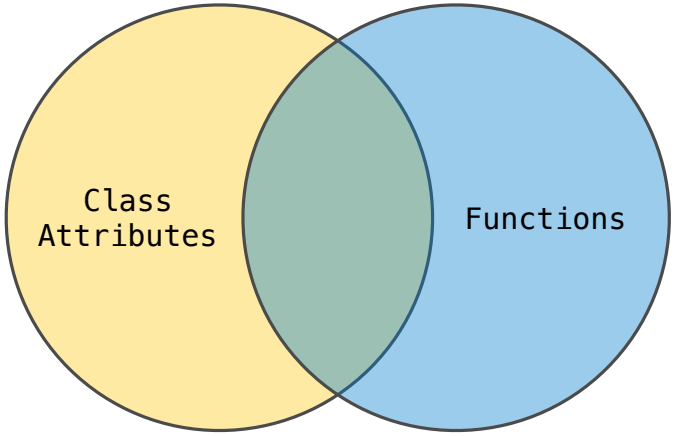
All objects have attributes, which are name-value pairs

Classes are objects too, so they have attributes

Instance attribute: attribute of an instance

Class attribute: attribute of the class of an instance

Terminology:



Terminology: Attributes, Functions, and Methods

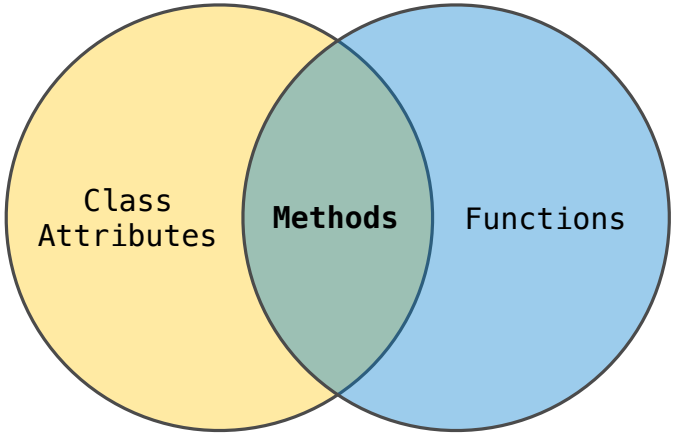
All objects have attributes, which are name-value pairs

Classes are objects too, so they have attributes

Instance attribute: attribute of an instance

Class attribute: attribute of the class of an instance

Terminology:



Terminology: Attributes, Functions, and Methods

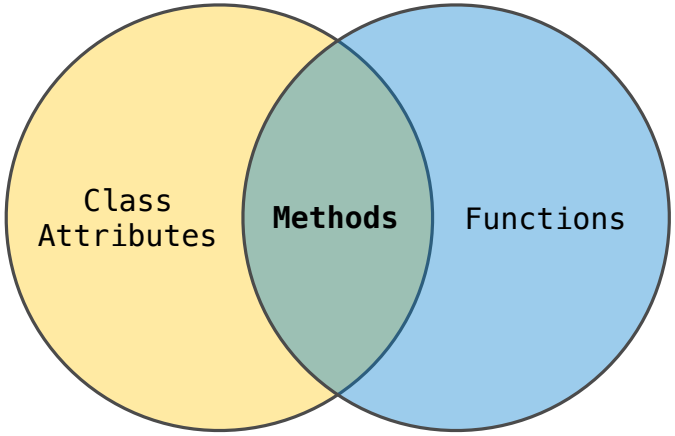
All objects have attributes, which are name-value pairs

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:

Terminology: Attributes, Functions, and Methods

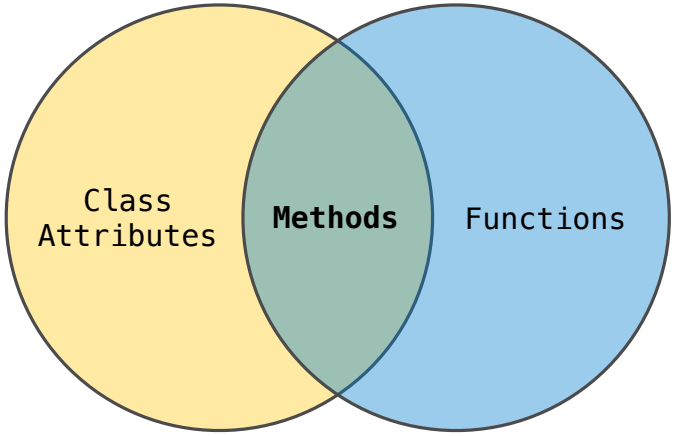
All objects have attributes, which are name-value pairs

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.

Terminology: Attributes, Functions, and Methods

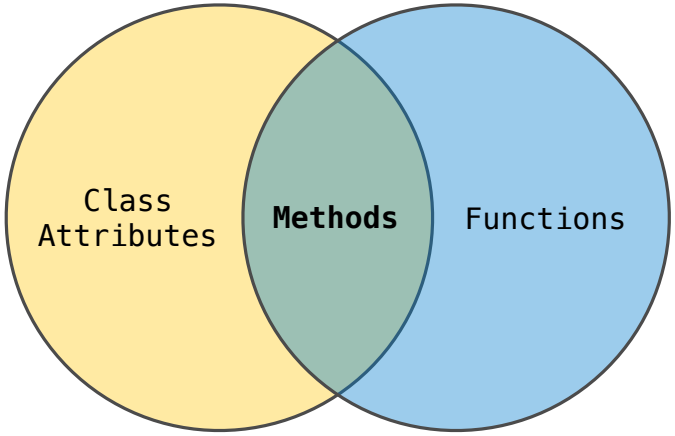
All objects have attributes, which are name-value pairs

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.

Terminology: Attributes, Functions, and Methods

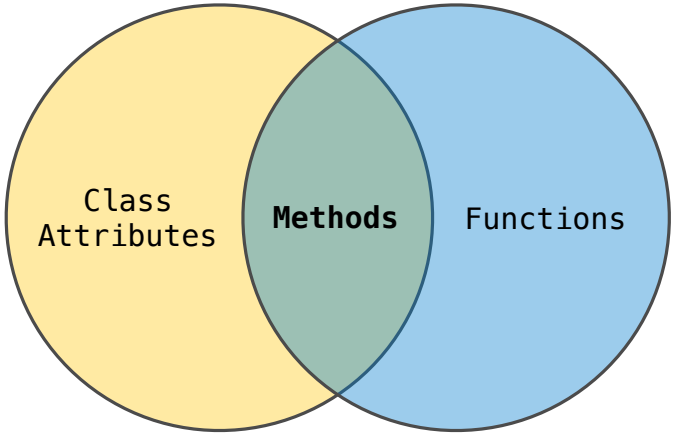
All objects have attributes, which are name-value pairs

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.

Terminology: Attributes, Functions, and Methods

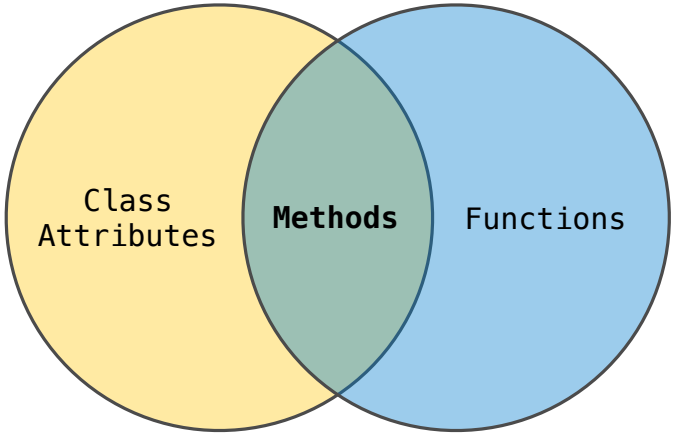
All objects have attributes, which are name-value pairs

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>

Looking Up Attributes of an Object

```
<expression> . <name>
```

Looking Up Attributes of an Object

```
<expression> . <name>
```

To evaluate a dot expression:

Looking Up Attributes of an Object

```
<expression> . <name>
```

To evaluate a dot expression:

1. Evaluate the <expression>.

Looking Up Attributes of an Object

```
<expression> . <name>
```

To evaluate a dot expression:

1. Evaluate the <expression>.
2. <name> is matched against the instance attributes.

Looking Up Attributes of an Object

```
<expression> . <name>
```

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.

Looking Up Attributes of an Object

```
<expression> . <name>
```

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.
4. That class attribute value is returned **unless it is a function**, in which case a *bound method* is returned.

Looking Up Attributes of an Object

<expression> . <name>

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.
4. That class attribute value is returned **unless it is a function**, in which case a *bound method* is returned.

Attribute Assignment

Assignment to Attributes

Assignment to Attributes

```
Assignment statements with a dot expression on their left-hand side affect attributes
for the object of that dot expression
```

Assignment to Attributes

Assignment statements with a dot expression on their left-hand side affect attributes for the object of that dot expression

- If the object is an instance, then assignment sets an instance attribute

Assignment to Attributes

Assignment statements with a dot expression on their left-hand side affect attributes for the object of that dot expression

- If the object is an instance, then assignment sets an instance attribute
- If the object is a class, then assignment sets a class attribute

Assignment to Attributes

Assignment statements with a dot expression on their left-hand side affect attributes for the object of that dot expression

- If the object is an instance, then assignment sets an instance attribute
- If the object is a class, then assignment sets a class attribute

```
tom_account.interest = 0.08
```

Assignment to Attributes

Assignment statements with a dot expression on their left-hand side affect attributes for the object of that dot expression

- If the object is an instance, then assignment sets an instance attribute
- If the object is a class, then assignment sets a class attribute

```
tom_account.interest = 0.08
```

This expression evaluates to an object

Assignment to Attributes

Assignment statements with a dot expression on their left-hand side affect attributes for the object of that dot expression

- If the object is an instance, then assignment sets an instance attribute
- If the object is a class, then assignment sets a class attribute

```
tom_account.interest = 0.08
```

This expression evaluates to an object

But the name (“interest”) is **not** looked up

Assignment to Attributes

Assignment statements with a dot expression on their left-hand side affect attributes for the object of that dot expression

- If the object is an instance, then assignment sets an instance attribute
- If the object is a class, then assignment sets a class attribute

```
tom_account.interest = 0.08
```

This expression evaluates to an object

But the name (“interest”) is **not** looked up

Attribute assignment statement adds or modifies the attribute named “interest” of tom_account

Assignment to Attributes

Assignment statements with a dot expression on their left-hand side affect attributes for the object of that dot expression

- If the object is an instance, then assignment sets an instance attribute
- If the object is a class, then assignment sets a class attribute

Instance Attribute Assignment :

```
tom_account.interest = 0.08
```

This expression evaluates to an object

But the name (“interest”) is **not** looked up

Attribute assignment statement adds or modifies the attribute named “interest” of tom_account

Assignment to Attributes

Assignment statements with a dot expression on their left-hand side affect attributes for the object of that dot expression

- If the object is an instance, then assignment sets an instance attribute
- If the object is a class, then assignment sets a class attribute

Instance Attribute Assignment :

```
tom_account.interest = 0.08
```

This expression evaluates to an object

But the name (“interest”) is **not** looked up

Attribute assignment statement adds or modifies the attribute named “interest” of tom_account

Class Attribute Assignment :

```
Account.interest = 0.04
```

Attribute Assignment Statements

Account class attributes

```
interest: 0.02
(withdraw, deposit, __init__)
```

Attribute Assignment Statements

Account class attributes

```
interest: 0.02
(withdraw, deposit, __init__)
```

```
>>> jim_account = Account('Jim')
```

Attribute Assignment Statements

Account class attributes

```
interest: 0.02
(withdraw, deposit, __init__)
```

```
balance:  0
holder:   'Jim'
```

```
>>> jim_account = Account('Jim')
```

Attribute Assignment Statements

Account class attributes

```
interest: 0.02
(withdraw, deposit, __init__)
```

Instance attributes of jim_account

```
balance:  0
holder:   'Jim'
```

```
>>> jim_account = Account('Jim')
```

Attribute Assignment Statements

Account class attributes

```
interest: 0.02
(withdraw, deposit, __init__)
```

Instance attributes of jim_account

```
balance:  0
holder:   'Jim'
```

```
>>> jim_account = Account('Jim')
>>> tom_account = Account('Tom')
```

Attribute Assignment Statements

Account class attributes

```
interest: 0.02
(withdraw, deposit, __init__)
```

Instance attributes of jim_account

```
balance:  0
holder:   'Jim'
```

Instance attributes of tom_account

```
balance:  0
holder:   'Tom'
```

```
>>> jim_account = Account('Jim')
>>> tom_account = Account('Tom')
```

Attribute Assignment Statements

Account class attributes

```
interest: 0.02
(withdraw, deposit, __init__)
```

Instance attributes of jim_account

```
balance:  0
holder:   'Jim'
```

Instance attributes of tom_account

```
balance:  0
holder:   'Tom'
```

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

Attribute Assignment Statements

Account class attributes

```
interest: 0.02
(withdraw, deposit, __init__)
```

Instance attributes of jim_account

```
balance:  0
holder:   'Jim'
```

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
```

Attribute Assignment Statements

Account class attributes

```
interest: 0.02
(withdraw, deposit, __init__)
```

Instance attributes of jim_account

```
balance:  0
holder:   'Jim'
```

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
>>> tom_account.interest
0.02
```

Attribute Assignment Statements

Account class attributes

```
interest: 0.02
(withdraw, deposit, __init__)
```

Instance attributes of jim_account

```
balance:  0
holder:   'Jim'
```

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
>>> tom_account.interest
0.02
>>> Account.interest = 0.04
```

Attribute Assignment Statements

Account class attributes

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

Instance attributes of jim_account

balance: 0
holder: 'Jim'

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
>>> tom_account.interest
0.02
>>> Account.interest = 0.04
```

Attribute Assignment Statements

Account class attributes

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

Instance attributes of jim_account

balance: 0
holder: 'Jim'

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
>>> tom_account.interest
0.02
>>> Account.interest = 0.04
>>> tom_account.interest
0.04
```

Attribute Assignment Statements

Account class attributes

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

Instance attributes of jim_account

balance: 0
holder: 'Jim'

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
>>> tom_account.interest
0.02
>>> Account.interest = 0.04
>>> tom_account.interest
0.04
```

```
>>> jim_account.interest = 0.08
```

Attribute Assignment Statements

Account class attributes

interest: ~~0.02~~ 0.04
(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
>>> tom_account.interest
0.02
>>> Account.interest = 0.04
>>> tom_account.interest
0.04
```

```
>>> jim_account.interest = 0.08
```

Attribute Assignment Statements

Account class attributes

interest: ~~0.02~~ 0.04
(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
>>> tom_account.interest
0.02
>>> Account.interest = 0.04
>>> tom_account.interest
0.04
```

```
>>> jim_account.interest = 0.08
>>> jim_account.interest
0.08
```

Attribute Assignment Statements

Account class attributes

interest: ~~0.02~~ 0.04
(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
>>> tom_account.interest
0.02
>>> Account.interest = 0.04
>>> tom_account.interest
0.04
```

```
>>> jim_account.interest = 0.08
>>> jim_account.interest
0.08
>>> tom_account.interest
0.04
```

Attribute Assignment Statements

Account class attributes

interest: ~~0.02~~ 0.04
(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
>>> tom_account.interest
0.02
>>> Account.interest = 0.04
>>> tom_account.interest
0.04
```

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

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
>>> tom_account.interest
0.02
>>> Account.interest = 0.04
>>> tom_account.interest
0.04
```

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

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
>>> tom_account.interest
0.02
>>> Account.interest = 0.04
>>> tom_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
```

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
>>> tom_account.interest
0.02
>>> Account.interest = 0.04
>>> tom_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
```

Inheritance

Inheritance

Inheritance

```
Inheritance is a method for relating classes together.
```

Inheritance

Inheritance is a method for relating classes together.

A common use: Two similar classes differ in their degree of specialization.

Inheritance

Inheritance is a method for relating classes together.

A common use: Two similar classes differ in their degree of specialization.

The specialized class may have the same attributes as the general class, along with some special-case behavior.

Inheritance

Inheritance is a method for relating classes together.

A common use: Two similar classes differ in their degree of specialization.

The specialized class may have the same attributes as the general class, along with some special-case behavior.

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

Inheritance

Inheritance is a method for relating classes together.

A common use: Two similar classes differ in their degree of specialization.

The specialized class may have the same attributes as the general class, along with some special-case behavior.

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

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

Inheritance

Inheritance is a method for relating classes together.

A common use: Two similar classes differ in their degree of specialization.

The specialized class may have the same attributes as the general class, along with some special-case behavior.

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

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

The subclass may *override* certain inherited attributes.

Inheritance

Inheritance is a method for relating classes together.

A common use: Two similar classes differ in their degree of specialization.

The specialized class may have the same attributes as the general class, along with some special-case behavior.

```
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 differences from the the base class.

Inheritance Example

```
A CheckingAccount is a specialized type of Account.
```

Inheritance Example

A CheckingAccount is a specialized type of Account.

```
>>> ch = CheckingAccount('Tom')
```

Inheritance Example

A CheckingAccount is a specialized type of Account.

```
>>> ch = CheckingAccount('Tom')
>>> ch.interest     # Lower interest rate for checking accounts
0.01
```

Inheritance Example

A CheckingAccount is a specialized type of Account.

```
>>> ch = CheckingAccount('Tom')
>>> ch.interest     # Lower interest rate for checking accounts
0.01
>>> ch.deposit(20)  # Deposits are the same
20
```

Inheritance Example

A CheckingAccount is a specialized type of Account.

```
>>> ch = CheckingAccount('Tom')
>>> 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
```

Inheritance Example

A CheckingAccount is a specialized type of Account.

```
>>> ch = CheckingAccount('Tom')
>>> 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

Inheritance Example

A CheckingAccount is a specialized type of Account.

```
>>> ch = CheckingAccount('Tom')
>>> 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

```
class CheckingAccount(Account):
```

Inheritance Example

A CheckingAccount is a specialized type of Account.

```
>>> ch = CheckingAccount('Tom')
>>> 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

```
class CheckingAccount(Account):
    """A bank account that charges for withdrawals."""
```

Inheritance Example

A CheckingAccount is a specialized type of Account.

```
>>> ch = CheckingAccount('Tom')
>>> 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

```
class CheckingAccount(Account):
    """A bank account that charges for withdrawals."""
    withdraw_fee = 1
```

Inheritance Example

A CheckingAccount is a specialized type of Account.

```
>>> ch = CheckingAccount('Tom')
>>> 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

```
class CheckingAccount(Account):
    """A bank account that charges for withdrawals."""
    withdraw_fee = 1
    interest = 0.01
```

Inheritance Example

A CheckingAccount is a specialized type of Account.

```
>>> ch = CheckingAccount('Tom')
>>> 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

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

Inheritance Example

A CheckingAccount is a specialized type of Account.

```
>>> ch = CheckingAccount('Tom')
>>> 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

```
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 Example

A CheckingAccount is a specialized type of Account.

```
>>> ch = CheckingAccount('Tom')
>>> 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

```
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!

Looking Up Attribute Names on Classes

Base class attributes *aren't copied* into subclasses!

To look up a name in a class.

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.

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.

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.

```
>>> ch = CheckingAccount('Tom')  # Calls Account.__init__
```

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.

```
>>> ch = CheckingAccount('Tom')  # Calls Account.__init__
>>> ch.interest     # Found in CheckingAccount
0.01
```

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.

```
>>> ch = CheckingAccount('Tom')  # Calls Account.__init__
>>> ch.interest     # Found in CheckingAccount
0.01
>>> ch.deposit(20)  # Found in Account
20
```

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.

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

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.

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

(Demo)

Object-Oriented Design

Designing for Inheritance

```
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)
```

Designing for Inheritance

Don't repeat yourself; use existing implementations.

```
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)
```

Designing for Inheritance

Don't repeat yourself; use existing implementations.

Attributes that have been overridden are still accessible via class objects.

```
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)
```

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.

```
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)
```

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.

```
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)
```

Attribute look-up
on base class

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.

```
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)
```

Attribute look-up on base class

Preferred to CheckingAccount.withdraw_fee to allow for specialized accounts

Inheritance and Composition

Inheritance and Composition

```
Object-oriented programming shines when we adopt the metaphor.
```

Inheritance and Composition

Object-oriented programming shines when we adopt the metaphor.

Inheritance is best for representing *is-a* relationships.

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.

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.
- So, CheckingAccount inherits from Account.

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.
- So, CheckingAccount inherits from Account.

Composition is best for representing *has-a* relationships.

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.

So, CheckingAccount inherits from Account.

Composition is best for representing *has-a* relationships.

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

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.

So, CheckingAccount inherits from Account.

Composition is best for representing *has-a* relationships.

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

So, A bank has a list of accounts as an attribute.

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.

So, CheckingAccount inherits from Account.

Composition is best for representing *has-a* relationships.

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

So, A bank has a list of accounts as an attribute.

(Demo)

Multiple Inheritance

Multiple Inheritance

Multiple Inheritance

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

Multiple Inheritance

```
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.

Multiple Inheritance

```
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.

CleverBank marketing executive wants:

Multiple Inheritance

```
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.

CleverBank marketing executive wants:

- Low interest rate of 1%

Multiple Inheritance

```
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.

CleverBank marketing executive wants:

- Low interest rate of 1%
- A $1 fee for withdrawals

Multiple Inheritance

```
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.

CleverBank marketing executive wants:

- Low interest rate of 1%
- A $1 fee for withdrawals
- A $2 fee for deposits

Multiple Inheritance

```
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.

CleverBank 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

Multiple Inheritance

```
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.

CleverBank 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

```
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!
```

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")
```

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
```

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!
```

Instance attribute

```
>>> such_a_deal = AsSeenOnTVAccount("John")
>>> such_a_deal.balance
1
```

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!
```

Instance attribute

```
>>> such_a_deal = AsSeenOnTVAccount("John")
>>> such_a_deal.balance
1
>>> such_a_deal.deposit(20)
19
```

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!
```

Instance attribute

SavingsAccount method

```
>>> such_a_deal = AsSeenOnTVAccount("John")
>>> such_a_deal.balance
1
>>> such_a_deal.deposit(20)
19
```

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!
```

Instance attribute

SavingsAccount method

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

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!
```

Instance attribute

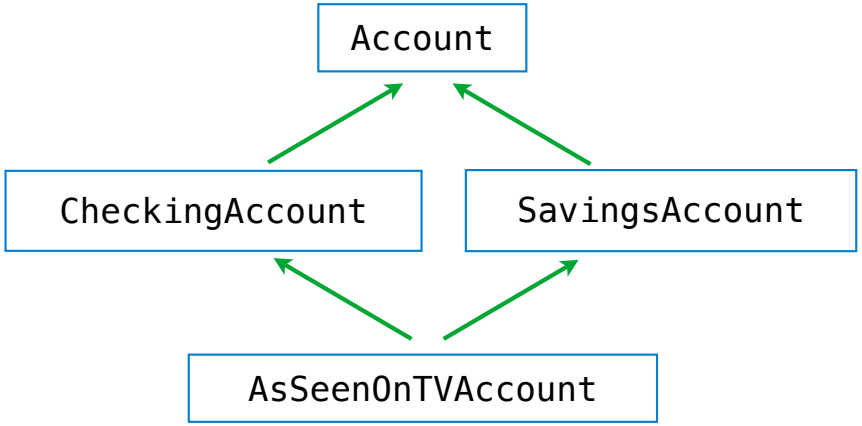
SavingsAccount method

CheckingAccount method

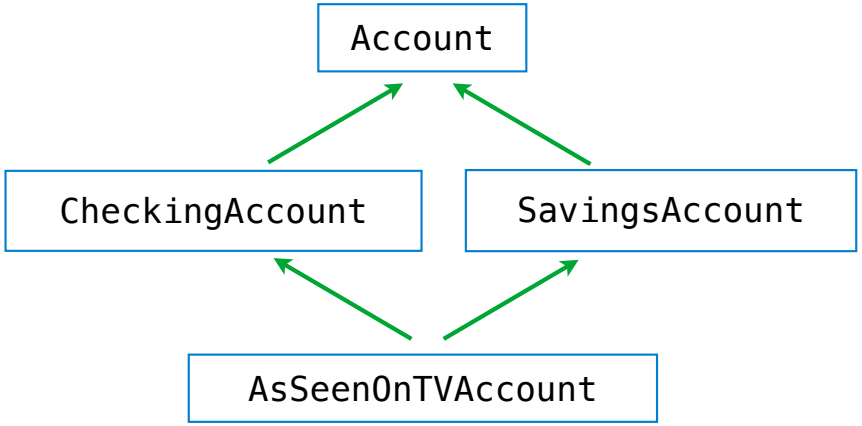
```
>>> such_a_deal = AsSeenOnTVAccount("John")
>>> such_a_deal.balance
1
>>> such_a_deal.deposit(20)
19
>>> such_a_deal.withdraw(5)
13
```

Resolving Ambiguous Class Attribute Names

Resolving Ambiguous Class Attribute Names

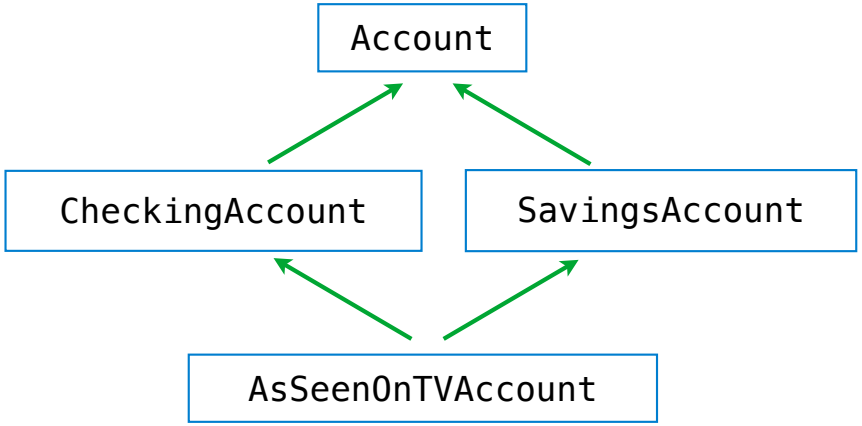


Resolving Ambiguous Class Attribute Names



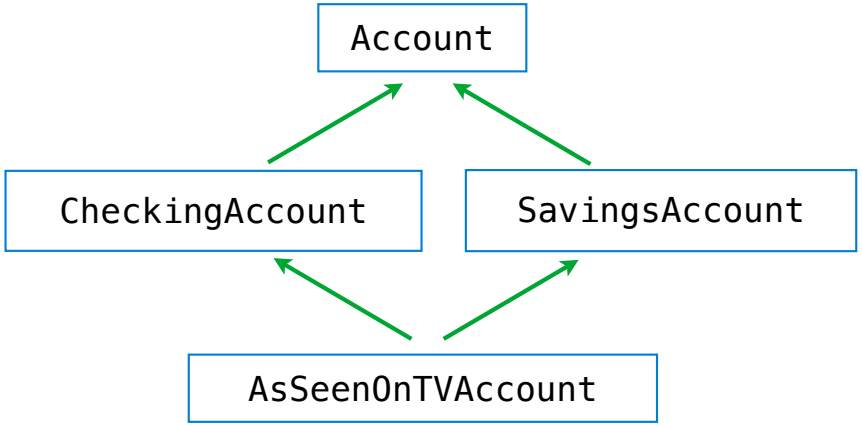

```
>>> such_a_deal = AsSeenOnTVAccount("John")
```

Resolving Ambiguous Class Attribute Names




```
>>> such_a_deal = AsSeenOnTVAccount("John")
>>> such_a_deal.balance
1
```

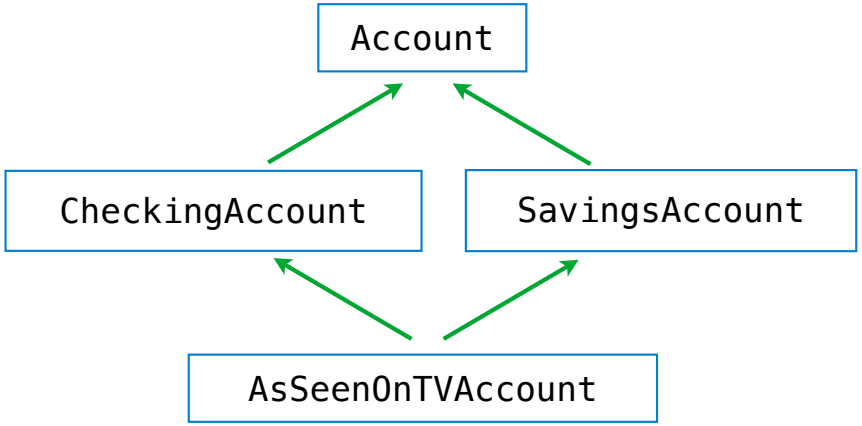
Resolving Ambiguous Class Attribute Names



Instance attribute

```
>>> such_a_deal = AsSeenOnTVAccount("John")
>>> such_a_deal.balance
1
```

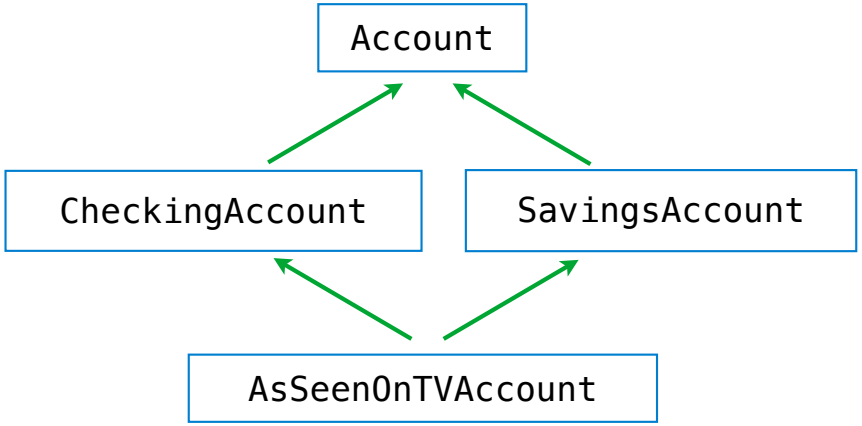
Resolving Ambiguous Class Attribute Names



Instance attribute

```
>>> such_a_deal = AsSeenOnTVAccount("John")
>>> such_a_deal.balance
1
>>> such_a_deal.deposit(20)
19
```

Resolving Ambiguous Class Attribute Names

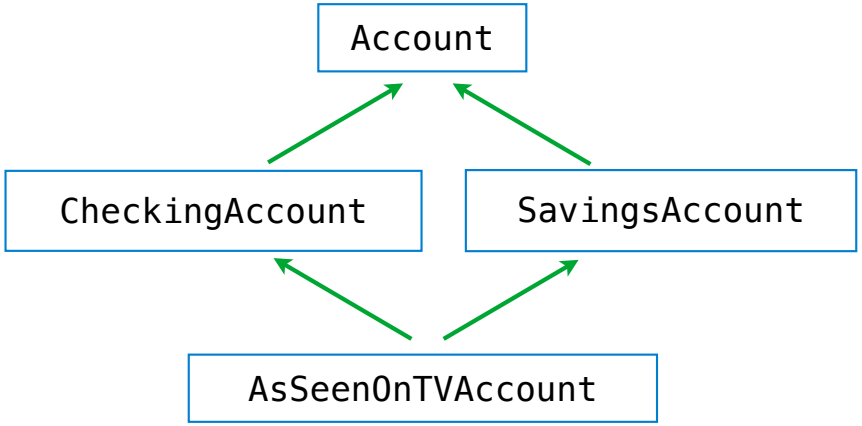


Instance attribute

SavingsAccount method

```
>>> such_a_deal = AsSeenOnTVAccount("John")
>>> such_a_deal.balance
1
>>> such_a_deal.deposit(20)
19
```

Resolving Ambiguous Class Attribute Names

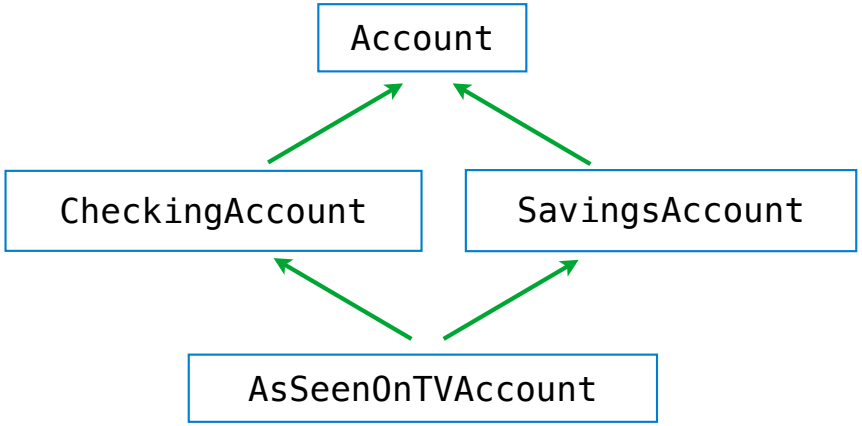


Instance attribute

SavingsAccount method

```
>>> 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



Instance attribute

SavingsAccount method

CheckingAccount method

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

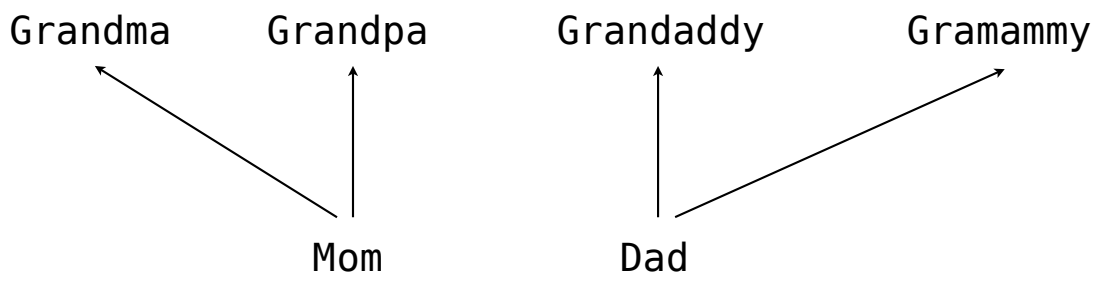
Complicated Inheritance

Biological Inheritance

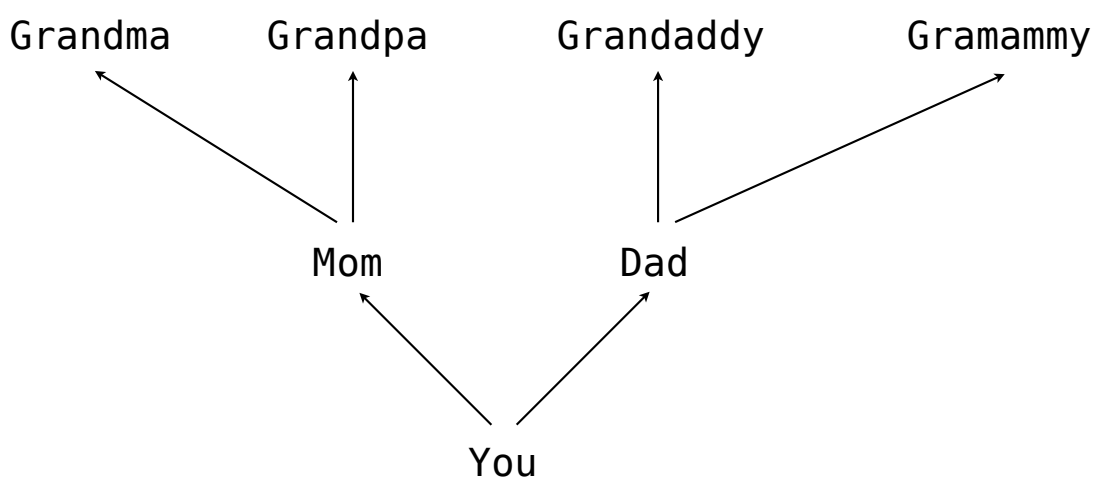
Biological Inheritance

Grandma Grandpa Grandaddy Gramammy

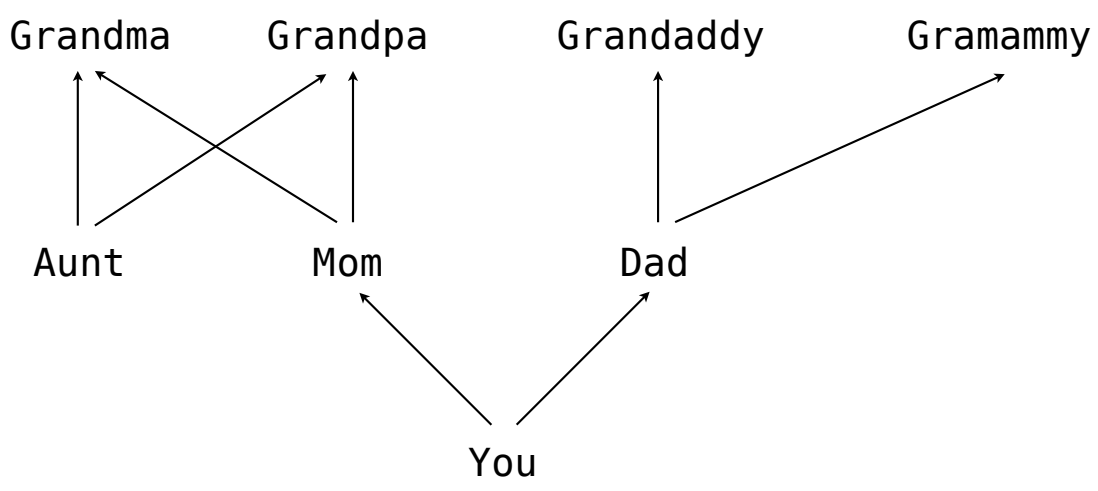
Biological Inheritance



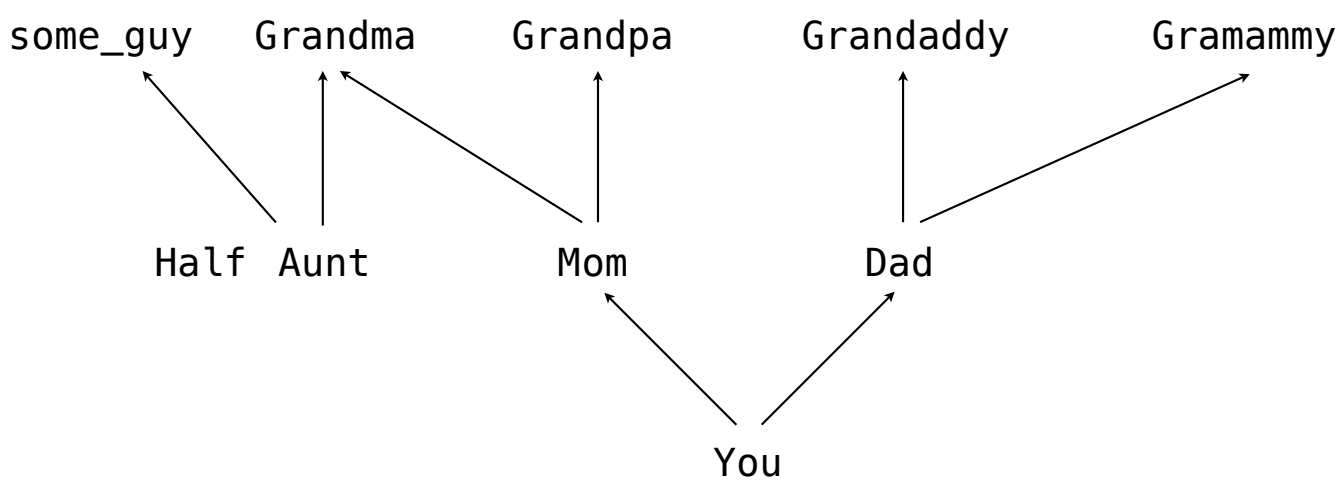
Biological Inheritance



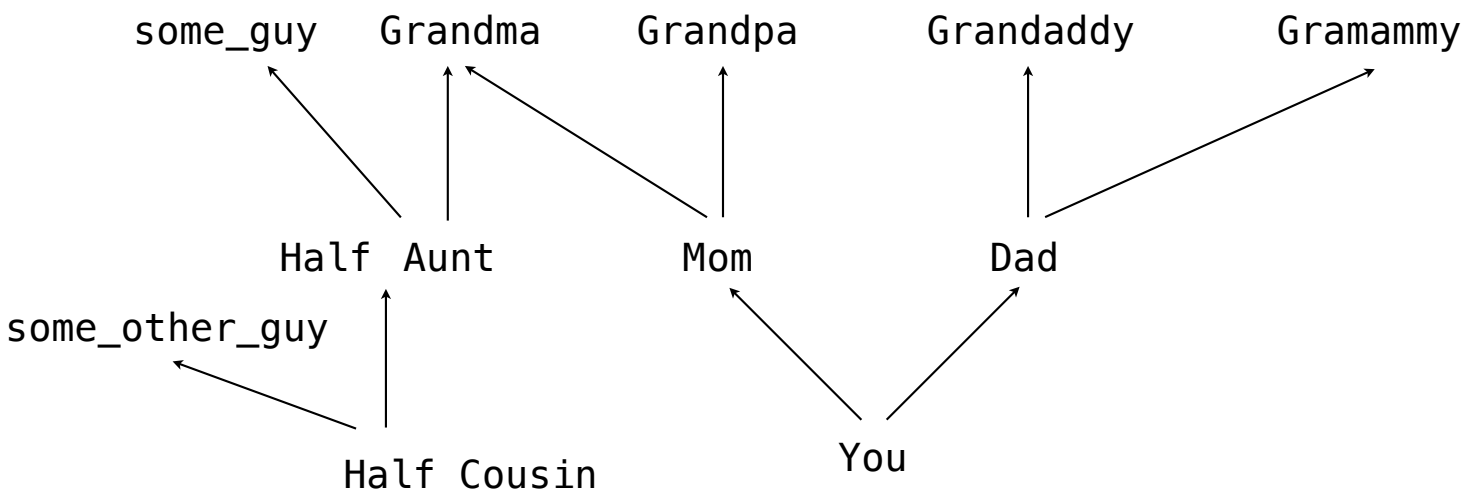
Biological Inheritance



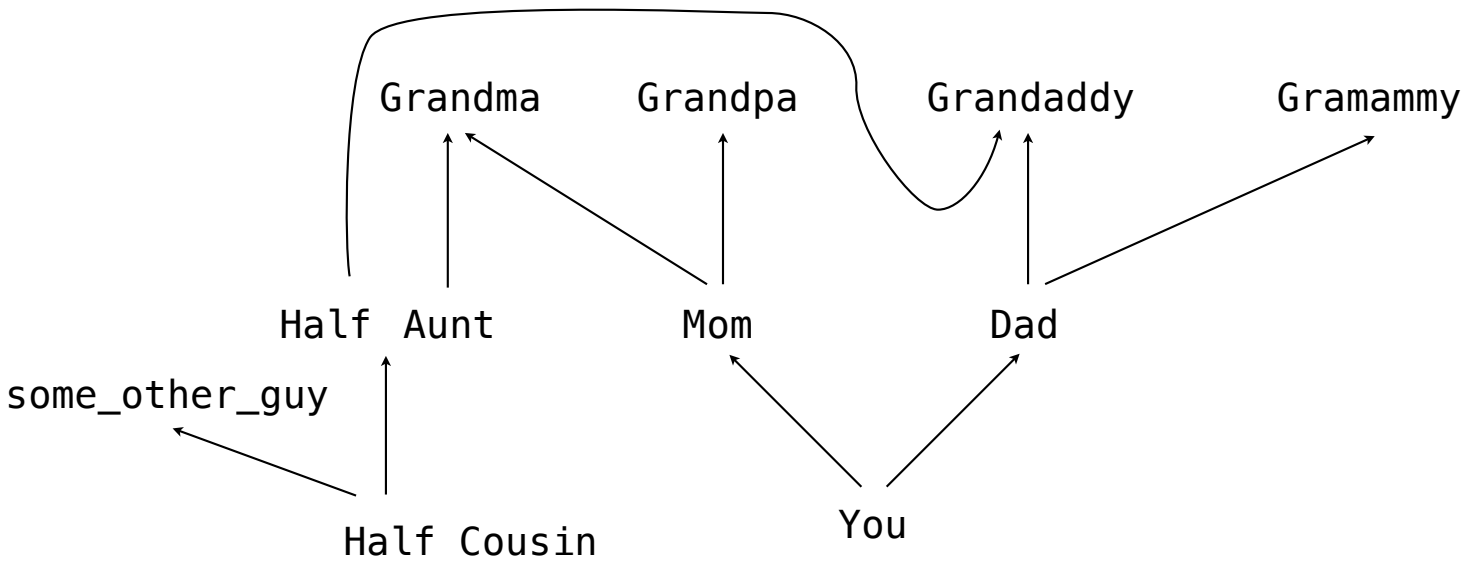
Biological Inheritance



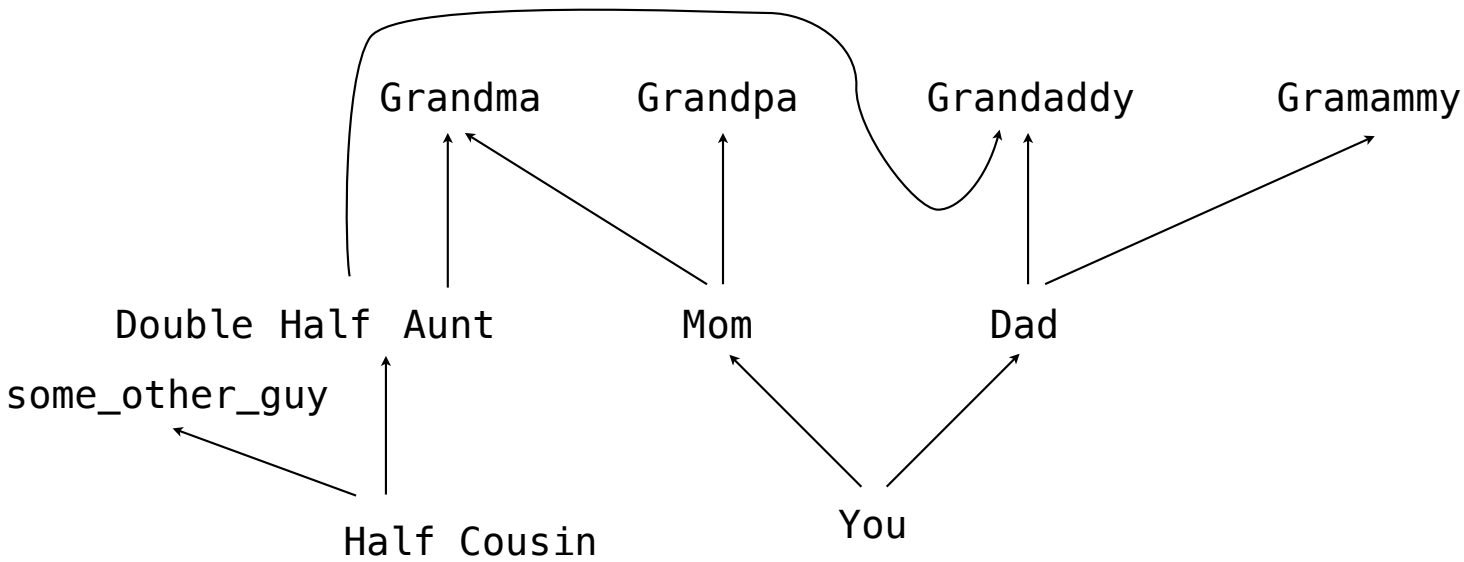
Biological Inheritance



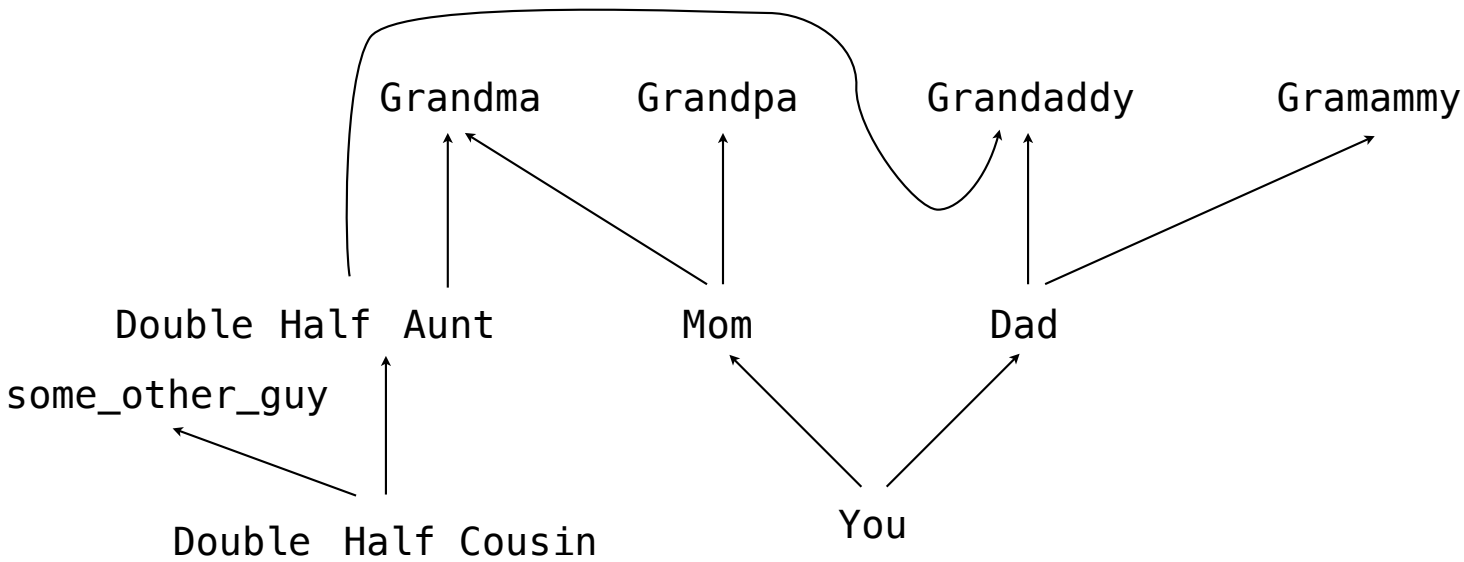
Biological Inheritance



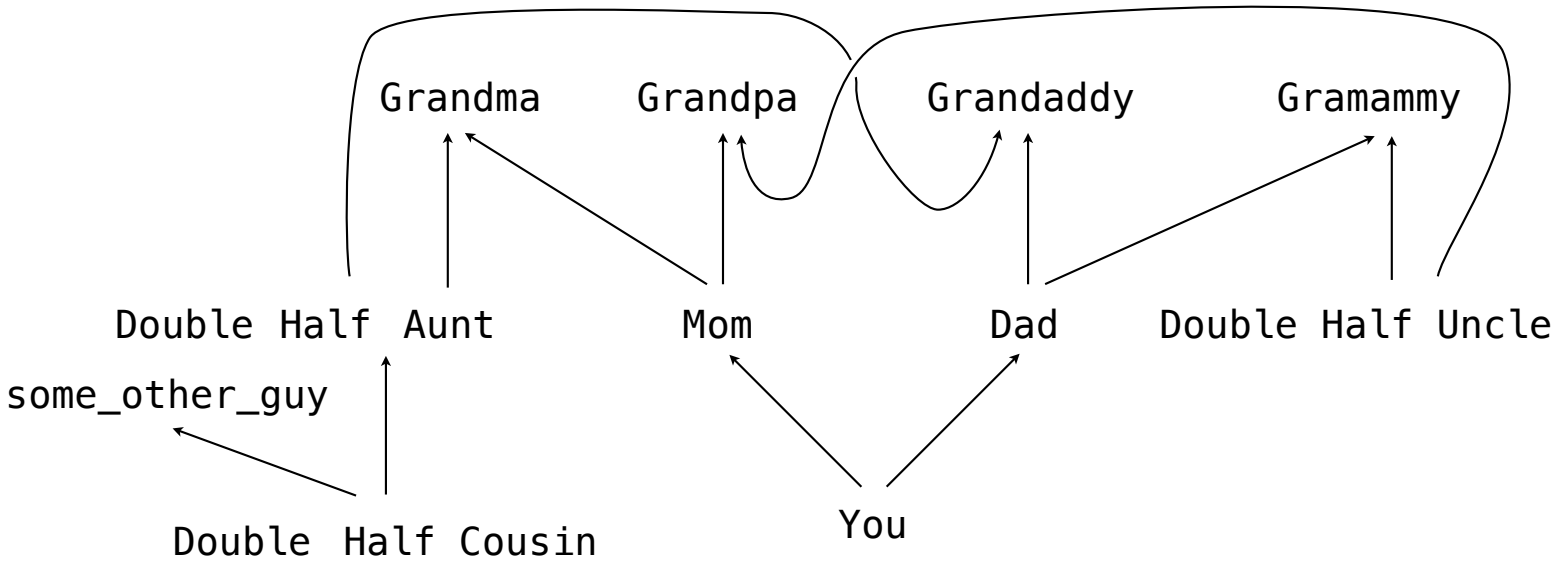
Biological Inheritance



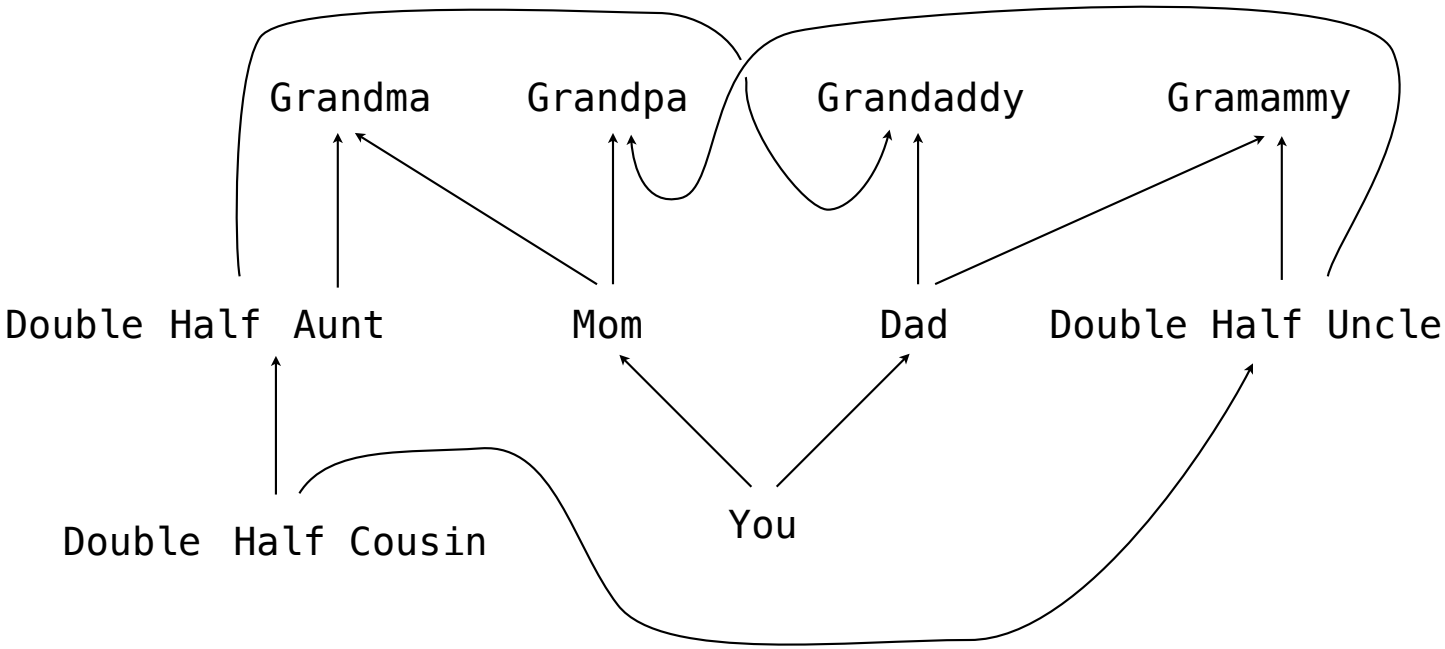
Biological Inheritance



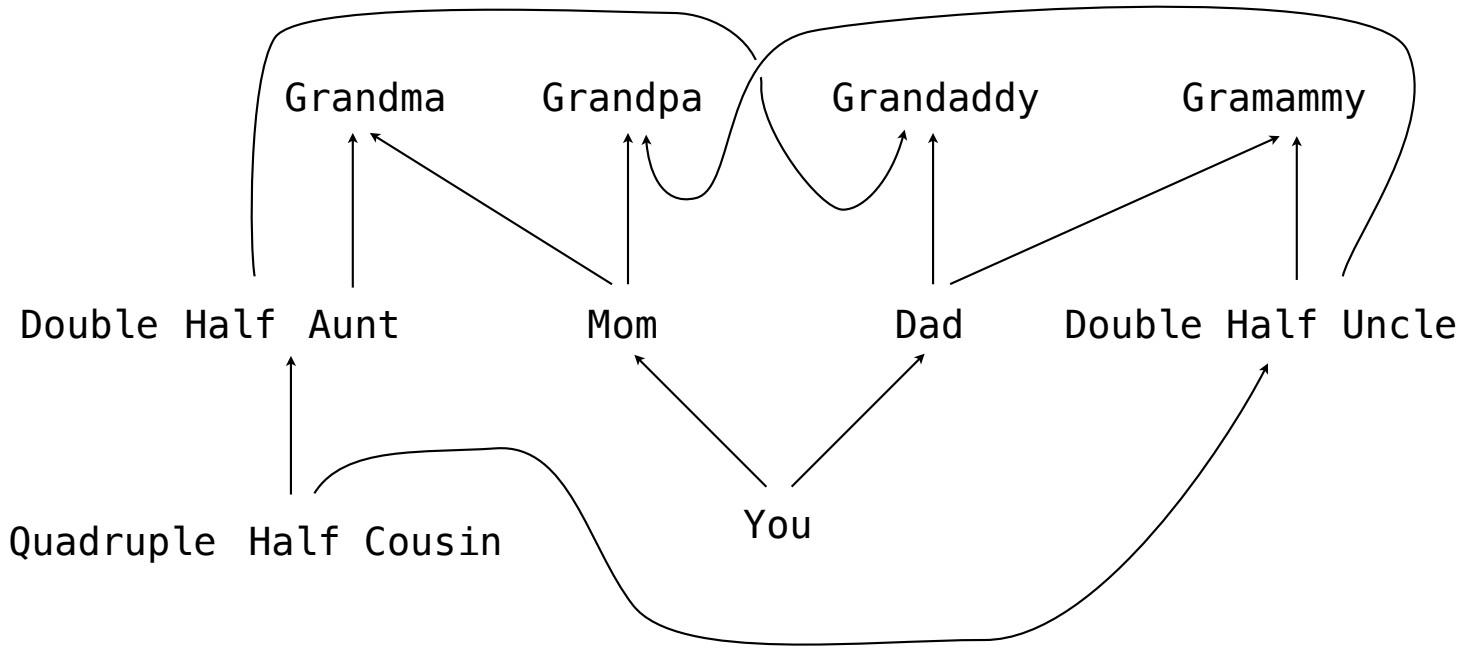
Biological Inheritance



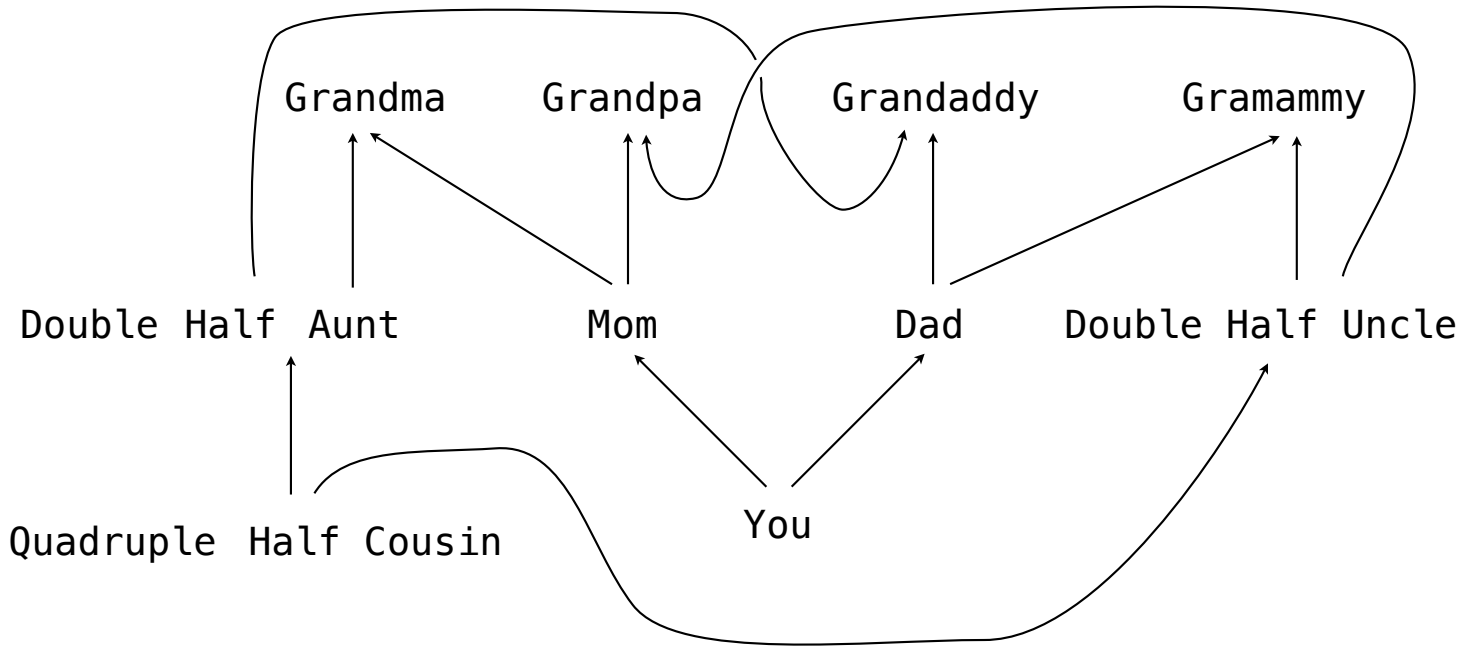
Biological Inheritance



Biological Inheritance



Biological Inheritance



Moral of the story: Inheritance can be complicated, so don't overuse it!