INHERITANCE

LECTURE 07–1

JIM FIX, REED COLLEGE CSCI 121

TODAY

- More object orientation in Python
  - ➡ Inheritance
- **Quiz** on lists
- **IOU**:
  - ➡ some OO examples
  - ➡ practice midterm
  - ➡ graded Project 1

TODAY

- More object orientation in Python
  - ➡ Inheritance
- **Quiz** on lists
- **IOU**:
  - ➡ some OO examples: ***this afternoon***
  - ➡ practice midterm: ***this afternoon***
  - ➡ graded Project 1: ***tomorrow evening***

RECALL: OBJECT ORIENTATION

- Many languages support coding up data abstractions in this style.
  - ➡ They allow you to invent your own type of object.
  - ➡ They let you define its attributes, the information each object stores.
  - ➡ They allow you to define a set of operations on that type.
- Your code is organized as a ***class definition*** for that object type.

EXAMPLE: A GIFT CARD OBJECT

- Here is a gift card object:

```
>>> gc = GiftCard(100)
>>> gc.spend(20)
80
>>> gc.spend(45)
35
>>> gc.spend(50)
'Insufficient funds'
>>> gc.addFunds(20)
55
>>> gc.spend(50)
5
>>>
```

EXAMPLE: GIFT CARD CLASS

- Here is the class definition of a new **`GiftCard`** type:

```
class GiftCard:

    def __init__(self, amount): # used by the constructor
        self.balance = amount

    def addFunds(self, amount): # a method definition
        self.balance = self.balance + amount
        return self.balance

def spend(self, amount):    # another method definition
    if amount > self.balance:
        return “Insufficient funds”
    self.balance = self.balance – amount
    return self.balance
```

EXAMPLE: GIFT CARD CLASS

- Here is the class definition of a new **`GiftCard`** type:

```
class GiftCard:

    def __init__(self, amount): # used by the constructor
        self.balance = amount

    def addFunds(self, amount): # a method definition
        self.balance = self.balance + amount
        return self.balance

    def spend(self, amount):    # another method definition
        if amount > self.balance:
            return “Insufficient funds”
        self.balance = self.balance – amount
        return self.balance

    def getBalance(self):       # a balance “getter”
        return self.balance
```

EXAMPLE: USING A GIFT CARD OBJECT

- Here is a gift card object's use, assuming there is a `"GiftCard.py"` file:

```
>>> from GiftCard import GiftCard
>>> gc = GiftCard(100) # use the constructor; it calls __init__
>>> gc.spend(20)
80
>>> gc.spend(45)
35
>>> gc.spend(50)
'Insufficient funds'
>>> gc.getBalance()
35
>>> gc.addFunds(20)
55
>>> gc.spend(50)
5
>>> gc.balance   # Python lets a client access attributes EEK!
5
```

GIFT CARD SUMMARY

- We made a gift card object that responds to two kinds of request:
  - ➡ We could spend money from the card.
  - ➡ We could add funds to the card.
- We built these as two different functions.

OBJECT TERMINOLOGY

- **`spend`** and **`addFunds`** are *messages* to which gift card objects respond.
- Their code are the gift card's ***methods*** for handling each request.
- The suite of messages that an object supports is its ***interface***.

OBJECT ORIENTATION

- Many languages support coding up data abstractions in this style.
  - ➡ They allow you to invent your own type of object.
  - ➡ They let you define its attributes, the information each object stores.
  - ➡ They allow you to define a set of operations on that type.
- Your code is organized as a ***class definition*** for that object type.

OBJECT ORIENTATION

- These are called ***class-based object-oriented*** languages.
  - **Python** is an example, as is **C++** and **Java**.

- Object-oriented languages have special syntax for:
  - constructors
  - attribute access
  - method definition

EXAMPLE: ACCOUNT CLASS

- Here is the class definition of a new **`Account`** type:

```
class Account:

    rate = .02

    def __init__(self, amount):
        self.balance = amount

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

    def payInterest(self):
        self.balance *= 1.0 + self.rate
```

EXAMPLE: ACCOUNT

- Here is `Account` in use:

```
>>> a = Account(100)
```

```
class Account:

    rate = .02

    def __init__(self, amount):
        self.balance = amount

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

    def payInterest(self):
        self.balance *= 1.0 + self.rate
```

EXAMPLE: ACCOUNT

- Here is **`Account`** in use:

```
>>> a = Account(100)
```

```
class Account:

    rate = .02

    def __init__(self, amount):
        self.balance = amount

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

    def payInterest(self):
        self.balance *= 1.0 + self.rate
```

COMMENTARY

- The class name is used like a function. We're calling the *constructor*.
  - ➡ This creates a new **`Account`** object.
- **`__init__`** runs with **`self`** as this new object.
- 100 is passed as the other parameter to **`__init__`**.

EXAMPLE: ACCOUNT

- Here is **`Account`** in use:

```
>>> a = Account(100)
```

```
class Account:

    rate = .02

    def __init__(self, amount):
        self.balance = amount

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

    def payInterest(self):
        self.balance *= 1.0 + self.rate
```

COMMENTARY

- The class name is used like a function. We're calling the *constructor*.
  - ➡ This creates a new **`Account`** object.
- **`__init__`** runs with **`self`** as this new object.
- 100 is passed as the other parameter to **`__init__`**.

EXAMPLE: ACCOUNT

- Here is **`Account`** in use:

```
>>> a = Account(100)
```

```
class Account:

    rate = .02

    def __init__(self, amount):
        self.balance = amount

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

    def payInterest(self):
        self.balance *= 1.0 + self.rate
```

COMMENTARY

- The class name is used like a function. We're calling the *constructor*.
  - ➡ This creates a new **`Account`** object.
- **`__init__`** runs with **`self`** as this new object.
- 100 is passed as the other parameter to **`__init__`**.

EXAMPLE: ACCOUNT

- Here is **`Account`** in use:

```
>>> a = Account(100)
```

```
class Account:

    rate = .02

    def __init__(self, amount):
        self.balance = amount

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

    def payInterest(self):
        self.balance *= 1.0 + self.rate
```

COMMENTARY

- The class name is used like a function. We're calling the *constructor*.
  - ➡ This creates a new **`Account`** object.
- **`__init__`** runs with **`self`** as this new object.
- 100 is passed as the other parameter to **`__init__`**.

EXAMPLE: ACCOUNT

- Here is **`Account`** in use:

```
>>> a = Account(100)
```

```
class Account:

    rate = .02

    def __init__(self, amount):
        self.balance = amount

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

    def payInterest(self):
        self.balance *= 1.0 + self.rate
```

COMMENTARY

- The class name is used like a function. We're calling the *constructor*.
  - ➡ This creates a new **`Account`** object.
- **`__init__`** runs with **`self`** as this new object.
- 100 is passed as the other parameter to **`__init__`**.

EXAMPLE: ACCOUNT

- Here is `Account` in use:

```
>>> a = Account(100)
>>> a.balance
100
```

```
class Account:

    rate = .02

    def __init__(self, amount):
        self.balance = amount

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

    def payInterest(self):
        self.balance *= 1.0 + self.rate
```

EXAMPLE: ACCOUNT

- Here is **`Account`** in use:

```
>>> a = Account(100)
>>> a.balance
100
```

```
class Account:

    rate = .02

    def __init__(self, amount):
        self.balance = amount

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

    def payInterest(self):
        self.balance *= 1.0 + self.rate
```

COMMENTARY

- This accesses an ***instance variable***.
  - Syntax: ***object . attribute-name***
- Gets that attribute's value.

EXAMPLE: ACCOUNT

- Here is **`Account`** in use:

```
>>> a = Account(100)
>>> a.balance
100
```

```
class Account:

    rate = .02

    def __init__(self, amount):
        self.balance = amount

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

    def payInterest(self):
        self.balance *= 1.0 + self.rate
```

COMMENTARY

- This accesses an ***instance variable***.
  - Syntax: ***object . attribute-name***
- Gets that attribute's value.

EXAMPLE: ACCOUNT

- Here is **`Account`** in use:

```
>>> a = Account(100)
>>> a.balance
100
```

```
class Account:

    rate = .02

    def __init__(self, amount):
        self.balance = amount

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

    def payInterest(self):
        self.balance *= 1.0 + self.rate
```

COMMENTARY

- This accesses an ***instance variable***.
  - Syntax: ***object . attribute-name***
- Gets that attribute's value.

EXAMPLE: ACCOUNT

- Here is **`Account`** in use:

```
>>> a = Account(100)
>>> a.balance
100
```

```
class Account:

    rate = .02

    def __init__(self, amount):
        self.balance = amount

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

    def payInterest(self):
        self.balance *= 1.0 + self.rate
```

COMMENTARY

- This accesses an ***instance variable***.
  - Syntax: ***object . attribute-name***
- Gets that attribute's value.

EXAMPLE: ACCOUNT

- Here is `Account` in use:

```
>>> a = Account(100)
>>> a.balance
100
>>> a.rate
0.02
```

```
class Account:

    rate = .02

    def __init__(self, amount):
        self.balance = amount

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

    def payInterest(self):
        self.balance *= 1.0 + self.rate
```

EXAMPLE: ACCOUNT

- Here is `Account` in use:

```
>>> a = Account(100)
>>> a.balance
100
>>> a.rate
0.02
```

```
class Account:

    rate = .02

    def __init__(self, amount):
        self.balance = amount

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

    def payInterest(self):
        self.balance *= 1.0 + self.rate
```

COMMENTARY

- The same notation is used to access a ***class variables***.
- If an object is missing an attribute, the class is checked instead.
- You can also access it directly inside the class.

EXAMPLE: ACCOUNT

- Here is **`Account`** in use:

```
>>> a = Account(100)
>>> a.balance
100
>>> a.rate
0.02
```

```
class Account:

    rate = .02

    def __init__(self, amount):
        self.balance = amount

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

    def payInterest(self):
        self.balance *= 1.0 + self.rate
```

COMMENTARY

- The same notation is used to access a ***class variables***.
- If an object is missing an attribute, the class is checked instead.
- You can also access it directly inside the class.

EXAMPLE: ACCOUNT

- Here is **`Account`** in use:

```
>>> a = Account(100)
>>> a.balance
100
>>> Account.rate
0.02
```

```
class Account:

    rate = .02

    def __init__(self, amount):
        self.balance = amount

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

    def payInterest(self):
        self.balance *= 1.0 + self.rate
```

COMMENTARY

- The same notation is used to access a ***class variables***.
- If an object is missing an attribute, the class is checked instead.
- You can also access a class variable by "dotting" with the class.

EXAMPLE: ACCOUNT

- Here is `Account` in use:

```
>>> a = Account(100)
>>> a.balance
100
>>> a.rate
0.02
>>> a.deposit(50)
```

```
class Account:

    rate = .02

    def __init__(self, amount):
        self.balance = amount

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

    def payInterest(self):
        self.balance *= 1.0 + self.rate
```

EXAMPLE: ACCOUNT

- Here is `Account` in use:

```
>>> a = Account(100)
>>> a.balance
100
>>> a.rate
0.02
>>> a.deposit(50)
```

```
class Account:

    rate = .02

    def __init__(self, amount):
        self.balance = amount

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

    def payInterest(self):
        self.balance *= 1.0 + self.rate
```

COMMENTARY

- This expression requests execution of a ***method***.
  - Similar syntax: ***object . method-name (...arguments...)***
- This behaves a lot like a function call.

EXAMPLE: ACCOUNT

- Here is **`Account`** in use:

```
>>> a = Account(100)
>>> a.balance
100
>>> a.rate
0.02
>>> a.deposit(50)
```

```
class Account:

    rate = .02

    def __init__(self, amount):
        self.balance = amount

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

    def payInterest(self):
        self.balance *= 1.0 + self.rate
```

COMMENTARY

- This expression requests execution of a ***method***.
  - Similar syntax: ***object . method-name (...arguments...)***
- This behaves a lot like a function call.

EXAMPLE: ACCOUNT

- Here is **`Account`** in use:

```
>>> a = Account(100)
>>> a.balance
100
>>> a.rate
0.02
>>> a.deposit(50)
```

```
class Account:

    rate = .02

    def __init__(self, amount):
        self.balance = amount

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

    def payInterest(self):
        self.balance *= 1.0 + self.rate
```

COMMENTARY

- This expression requests execution of a ***method***.
  - Similar syntax: ***object . method-name (...arguments...)***
- This behaves a lot like a function call.

EXAMPLE: ACCOUNT

- Here is **`Account`** in use:

```
>>> a = Account(100)
>>> a.balance
100
>>> a.rate
0.02
>>> a.deposit(50)
```

```
class Account:

    rate = .02

    def __init__(self, amount):
        self.balance = amount

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

    def payInterest(self):
        self.balance *= 1.0 + self.rate
```

COMMENTARY

- This expression requests execution of a ***method***.
  - Similar syntax: ***object . method-name (...arguments...)***
- This behaves a lot like a function call.
  - ➡ The argument value is passed as the parameter **`amount`**.

EXAMPLE: ACCOUNT

- Here is **`Account`** in use:

```
>>> a = Account(100)
>>> a.balance
100
>>> a.rate
0.02
>>> a.deposit(50)
```

```
class Account:

    rate = .02

    def __init__(self, amount):
        self.balance = amount

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

    def payInterest(self):
        self.balance *= 1.0 + self.rate
```

COMMENTARY

- This expression requests execution of a ***method***.
  - Similar syntax: ***object . method-name (...arguments...)***
- This behaves a lot like a function call.
  - ➡ The ***message receiver object*** is passed as **`self`**.

EXAMPLE: ACCOUNT

- Here is **`Account`** in use:

```
>>> a = Account(100)
>>> a.balance
100
>>> a.rate
0.02
>>> a.deposit(50)
>>> a.payInterest()
```

```
class Account:

    rate = .02

    def __init__(self, amount):
        self.balance = amount

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

    def payInterest(self):
        self.balance *= 1.0 + self.rate
```

COMMENTARY

- This expression requests execution of a ***method***.
  - Similar syntax: ***object . method-name (...arguments...)***
- Methods with no arguments just have a receiver parameter **`self`**.

EXAMPLE: ACCOUNT

- Here is `Account` in use:

```
>>> a = Account(100)
>>> a.balance
100
>>> a.rate
0.02
>>> a.deposit(50)
>>> Account.rate
0.02
```

```
class Account:

    rate = .02

    def __init__(self, amount):
        self.balance = amount

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

    def payInterest(self):
        self.balance *= 1.0 + self.rate
```

COMMENTARY

- In a way, a class is like an object. It can have attributes.
- There is only one "class object", so only one `Account.rate`
- But there is a different `balance` for every `Account` **instance**.

EXAMPLE: ACCOUNT

- Here is `Account` in use:

```
>>> a = Account(100)
>>> a.balance
100
>>> a.rate
0.02
>>> a.deposit(50)
>>> Account.rate
0.02
>>> Account.deposit(a,10)
```

```
class Account:

    rate = .02

    def __init__(self, amount):
        self.balance = amount

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

    def payInterest(self):
        self.balance *= 1.0 + self.rate
```

COMMENTARY

- You can also call an instance's method using its class name:
  - Syntax: *class-name* . *instance-method-name* (*receiver* , *arguments*)

EXAMPLE: ACCOUNT

- Here is `Account` in use:

```
>>> a = Account(100)
>>> a.balance
100
>>> a.rate
0.02
>>> a.deposit(50)
>>> Account.rate
0.02
>>> Account.deposit(a,10)
```

```
class Account:

    rate = .02

    def __init__(self, amount):
        self.balance = amount

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

    def payInterest(self):
        self.balance *= 1.0 + self.rate
```

COMMENTARY

- You can also call an instance's method using its class name:
  - Syntax: *class-name* `.` *instance-method-name* `(` *receiver* `,` *arguments* `)`
- It is as if `deposit` is a function attached to the `Account` class.

EXAMPLE: ACCOUNT

- Here is **`Account`** in use:

```
>>> a = Account(100)
>>> a.balance
100
>>> a.rate
0.02
>>> a.deposit(50)
>>> Account.rate
0.02
>>> Account.deposit(a,10)
```

```
class Account:

    rate = .02

    def __init__(self, amount):
        self.balance = amount

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

    def payInterest(self):
        self.balance *= 1.0 + self.rate
```

COMMENTARY

- You can also call an instance's method using its class name:
  - Syntax: *class-name* . *instance-method-name* (*receiver* , *arguments*)
- You pass the receiver as the first argument to that "function."

SUMMARY: ACCOUNT CLASS

- Here is the class definition of a new **`Account`** type:

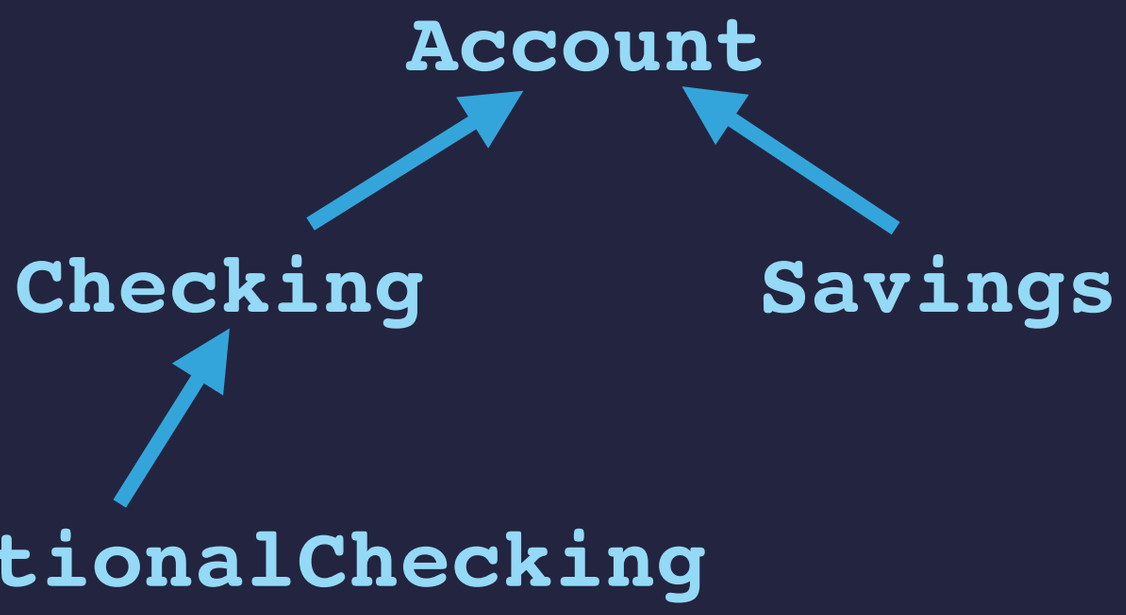
```
class Account:
    interest_rate = .02
    def __init__(self, amount):
        self.balance = amount
    def deposit(self, amount):
        self.balance += amount
    def payInterest(self):
        self.balance *= 1.0 + self.interest_rate
    def getBalance(self):
        return self.balance
```

- Here is **`Account`** in use:

```
>>> a = Account(150)
>>> a.deposit(50)
>>> a.payInterest()
>>> a.getBalance()
204.0
```

AN ACCOUNT CLASS HIERARCHY

- We can build ***hierarchies*** of different accounts:



- We make ***subclasses*** that ***inherit*** the attributes of their "***superclasses***"
  - A `Savings` account has all the info and operations of an `Account`.
  - But it has features and behavior more specific to checking accounts
    - This is called subclass ***specialization***.
    - We ***extend*** the superclass with additional attributes.
  - It also ***overrides*** some of the behavior it inherits from `Account`.

INHERITANCE EXAMPLE: A SAVINGS ACCOUNT

```
class Account:
    interest_rate = .02
    def __init__(self, amount):
        self.balance = amount
    def deposit(self, amount):
        self.balance += amount
    def withdraw(self, amount):
        self.balance -= amount
    def payInterest(self):
        self.balance *= 1.0 + self.interest_rate

class Savings(Account):
    interest_rate = 0.04
    withdraw_fee = 1.0
    def withdraw(self, amount):
        Account.withdraw(self, amount + self.withdraw_fee)
```

INHERITANCE EXAMPLE: A SAVINGS ACCOUNT

```
class Account:
    interest_rate = .02
    def __init__(self, amount):
        self.balance = amount
    def deposit(self, amount):
        self.balance += amount
    def withdraw(self, amount):
        self.balance -= amount
    def payInterest(self):
        self.balance *= 1.0 + self.interest_rate

class Savings(Account): # inherit the methods and class variables of Account
    interest_rate = 0.04
    withdraw_fee = 1.0
    def withdraw(self, amount):
        Account.withdraw(self, amount + self.withdraw_fee)
```

INHERITANCE EXAMPLE: A SAVINGS ACCOUNT

```
class Account:
    interest_rate = .02
    def __init__(self, amount):
        self.balance = amount
    def deposit(self, amount):
        self.balance += amount
    def withdraw(self, amount):
        self.balance -= amount
    def payInterest(self):
        self.balance *= 1.0 + self.interest_rate

class Savings(Account):
    interest_rate = 0.04 # overrides the class variable inherited from Account
    withdraw_fee = 1.0
    def withdraw(self, amount):
        Account.withdraw(self, amount + self.withdraw_fee)
```

INHERITANCE EXAMPLE: A SAVINGS ACCOUNT

```
class Account:
    interest_rate = .02
    def __init__(self, amount):
        self.balance = amount
    def deposit(self, amount):
        self.balance += amount
    def withdraw(self, amount):
        self.balance -= amount
    def payInterest(self):
        self.balance *= 1.0 + self.interest_rate

class Savings(Account):
    interest_rate = 0.04
    withdraw_fee = 1.0 # extends with a specializing class variable
    def withdraw(self, amount):
        Account.withdraw(self, amount + self.withdraw_fee)
```

INHERITANCE EXAMPLE: A SAVINGS ACCOUNT

```
class Account:
    interest_rate = .02
    def __init__(self, amount):
        self.balance = amount
    def deposit(self, amount):
        self.balance += amount
    def withdraw(self, amount):
        self.balance -= amount
    def payInterest(self):
        self.balance *= 1.0 + self.interest_rate

class Savings(Account):
    interest_rate = 0.04
    withdraw_fee = 1.0
    def withdraw(self, amount): # overrides a method inherited from Account
        Account.withdraw(self, amount + self.withdraw_fee)
```

INHERITANCE EXAMPLE: A SAVINGS ACCOUNT

```
class Account:
    interest_rate = .02
    def __init__(self, amount):
        self.balance = amount
    def deposit(self, amount):
        self.balance += amount
    def withdraw(self, amount):
        self.balance -= amount
    def payInterest(self):
        self.balance *= 1.0 + self.interest_rate

class Savings(Account):
    interest_rate = 0.04
    withdraw_fee = 1.0
    def withdraw(self, amount): # overrides a method inherited from Account
        Account.withdraw(self, amount + self.withdraw_fee)
        # explicitly invokes the method of its superclass
```

ACCOUNT VERSUS SAVINGS

- Here is **`Account`** in use:

```
>>> a = Account(100)
>>> a.balance
100.0
>>> a.payInterest()
>>> a.balance
102.0
>>> a.withdraw(20)
>>> a.balance
82.0
```

- Here is **`Savings`** in use:

```
>>> a = Savings(100)
>>> a.balance
100.0
>>> a.payInterest()
>>> a.balance
104.0
>>> a.withdraw(20)
>>> a.balance
83.0
```

INHERITANCE EXAMPLE: A CHECKING ACCOUNT

```
class Account:
    interest_rate = .02
    def __init__(self, amount):
        self.balance = amount
    def deposit(self, amount):
        self.balance += amount
    def withdraw(self, amount):
        self.balance -= amount
    def payInterest(self):
        self.balance *= 1.0 + self.interest_rate

class Checking(Account):

    min_balance = 1000.0

    def payInterest(self):
        if self.balance >= self.min_balance:
            Account.payInterest(self)
```

CHECKING ACCOUNT INTERACTION

- Here is **`Checking`** in use:

```
>>> a = Checking(1000.0)
>>> a.balance
1000.0
>>> a.payInterest()
>>> a.balance
1040.0
>>> a.withdraw(50.0)
>>> a.balance
990.0
>>> a.payInterest()
>>> a.balance
990.0
```

INHERITANCE EXAMPLE: A PROMOTIONAL CHECKING ACCOUNT

```
class Checking(Account):

    min_balance = 1000.0

    def payInterest(self):
        if self.balance >= self.min_balance:
            Account.payInterest(self)
```

```
class PromotionalChecking(Checking):

    reward = 50

    def __init__(self,amount):
        Checking.__init__(self,amount+self.reward)
        # The code above explicitly uses the initializer code from Checking
```

INHERITANCE EXAMPLE: A PROMOTIONAL CHECKING ACCOUNT

```
class Checking(Account):

    min_balance = 1000.0

    def payInterest(self):
        if self.balance >= self.min_balance:
            Account.payInterest(self)
```

```
class PromotionalChecking(Checking):

    reward = 50

    def __init__(self,amount):
        super().__init__(amount+self.reward)
        # The code above uses the initializer code from Checking that was inherited from Account
        # Using super() references self as though it is an instance of its superclass
```

OBJECT TAKEAWAYS

- New object types are defined with `class`.
- Within the class you define these things:
  - `__init__`
  - other methods
  -
- Method parameters are `self` followed by the others.
- Object dot notation:
  - Methods are called using `receiver.method(...)`
  - Object attributes are accessed by `receiver.variable`
  - We use `self.` notation inside a method to access these things too.
- New instances are built with `class-name(...)`

OBJECT TAKEAWAYS

- New object types are defined with `class`.
- Within the class you define these things:
  - `__init__`
  - other methods
  - (maybe) class attributes
- Method parameters are `self` followed by the others.
- Object dot notation:
  - Methods are called using `receiver.method(...)`
  - Object attributes are accessed by `receiver.variable`
  - We use `self.` notation inside a method to access these things too.
- New instances are built with `class-name(...)`

INHERITANCE TAKEAWAYS

- A class inherits from its superclass with
  - `class class-name(super-class-name):`
- You can call the superclass initializer with the syntax:
  - `super-class-name.__init__(self,...)`
- You can call the superclass methods with the syntax:
  - `super-class-name.method(self,...)`
- Subclasses inherit the methods of their superclass.
- They can be ***specialized*** in two ways:
  - You can add additional attributes and methods.
  - You can override super-class methods.

QUIZ ON LISTS