Representation
Announcements
Inheritance Example

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

```python
>>> 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`

```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)
        # or
        return super().withdraw(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')  # Calls Account.__init__
>>> ch.interest                # Found in CheckingAccount
0.01
>>> ch.deposit(20)             # Found in Account
20
>>> ch.withdraw(5)             # Found in CheckingAccount
14
```
Example: Three Attributes

class A:
    x, y, z = 0, 1, 2

    def f(self):
        return [self.x, self.y, self.z]

class B(A):
    """What would Python Do?"

    >>> A().f()
    [0, 1, 2]

    >>> B().f()
    [6, 1, 'A']

    x = 6
    def __init__(self):
        self.z = 'A'
String Representations
String Representations

In Python, all objects produce two string representations:

• The `str` is legible to humans
• The `repr` is legible to the Python interpreter

The `str` and `repr` strings are often the same, but not always

```python
>>> from fractions import Fraction
>>> half = Fraction(1, 2)
>>> str(half)
'1/2'
>>> repr(half)
'Fraction(1, 2)'
```
Class Practice
class Letter:
    def __init__(self, contents):
        self.contents = contents
        self.sent = False

    def send(self):
        if self.sent:
            print(self, 'was already sent.')
        else:
            print(self, 'has been sent.')
            self.sent = True
        return Letter(self.contents.upper())

    def __repr__(self):
        return self.contents

Implement the `Letter` class. A `Letter` has two instance attributes: `contents` (a `str`) and `sent` (a `bool`). Each `Letter` can only be sent once. The `send` method prints whether the letter was sent, and if it was, returns the reply, which is a new `Letter` instance with the same contents, but in all caps.

*Hint:* `hi`.upper() evaluates to 'HI'.

"""A letter receives an all-caps reply.

```python
>>> hi = Letter('Hello, World!')
>>> hi.send()
Hello, World! has been sent.
HELLO, WORLD!
>>> hi.send()
Hello, World! was already sent.
>>> Letter('Hey').send().send()
Hey has been sent.
HEY has been sent.
HEY
"""
Implement the `Numbered` class. A `Numbered` letter has a `number` attribute equal to how many numbered letters have previously been constructed. This `number` appears in its `repr` string. Assume `Letter` is implemented correctly.

A numbered letter has a different `repr` method that shows its number.

```python
>>> hey = Numbered('Hello, World!')
>>> hey.send()
#0 has been sent.
HELLO, WORLD!
>>> Numbered('Hi!').send()
#1 has been sent.
HI!
>>> hey
#0
```