Objects

Announcements

Midterm Logistics:
https://edstem.org/us/courses/74610/discussion/6310007

Studying for the Midterm

* Do a few practice problems.
* Don’t time yourself at first.
* Go by topic rather than by exam
* Think through your hw/lab/projects.

Class Statements

Classes

A class describes the behavior of its instances

Idea: All bank accounts have a balance and an account holder; the Account class should add those attributes to each newly created instance

Idea: All bank accounts share a withdraw method and a deposit method

```
>>> a = Account('C88C')
>>> a.holder
'C88C'
>>> a.balance
0

>>> a.deposit(15)
15
>>> a.withdraw(10)
5
>>> a.balance
5
>>> a.withdraw(10)
'Insufficient funds'
```

balance and holder are ***attributes***

deposit and withdraw are ***methods***

The Account Class

```
class Account:
```

__init__ is a special method name for the function that constructs an Account instance

```
    def __init__(self, account_holder):
        self.balance = 0
        self.holder = account_holder
```

self is the instance of the Account class on which deposit was invoked: a.deposit(10)

```
    def deposit(self, amount):
        self.balance = self.balance + amount
        return self.balance
    def withdraw(self, amount):
        if amount > self.balance:
            return 'Insufficient funds'
        self.balance = self.balance - amount
        return self.balance
```

Methods are functions defined in a class statement

(Demo)

```
>>> a = Account('C88C')
>>> a.holder
'C88C'
>>> a.balance
0
>>> a.deposit(15)
15
>>> a.withdraw(10)
5
>>> a.balance
5
>>> a.withdraw(10)
'Insufficient funds'
```

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

```
>>> from fractions import Fraction
>>> half = Fraction(1, 2)
>>> str(half)
'1/2'
>>> repr(half)
'Fraction(1, 2)'
```

(Demo)

Expressions, Values, & Types (Classes)

Sample Lab Question: Email

```
class Email:
    def __init__(self, msg, sender, recipient_name):
        self.msg = msg
        self.sender = sender
        self.recipient_name = recipient_name

class Server:
    def __init__(self):
        self.clients = {}

    def send(self, email):
        # Append the email to the inbox of the client it is addressed to.
        self.clients[email.recipient_name].inbox.append(email)
    ...

class Client:
    def __init__(self, server, name):
        self.inbox = []
        self.server = server
        self.name = name
        ...
    ...
```

Server dict Client list

A **Client** can **send** an **Email** to its **Server.**

The Server then delivers it to the inbox of another Client.

To achieve this, a Server has a dictionary called clients that maps the name of the Client to the Client instance.

Class Practice

Spring 2023 Midterm 2 Question 2(a)

```
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.

    >>> 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
    """
```

Spring 2023 Midterm 2 Question 2(b)

```
class Numbered(Letter):

    number = 0

    def __init__(self, contents):

        super().__init__(contents)

        self.number = Numbered.number
        ____________________________

        Numbered.number += 1
        ____________________________

    def __repr__(self):

        return '#' + ___str(self.number)___
```

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.

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