CS 61A SU26

Lecture 13: Objects and Attributes

July 14, 2026

Rebecca Dang

Join pollev.com/rebeccadang831 (or scan QR code) on your phone or laptop

We'll begin at Berkeley time (10 minutes after), as per tradition!

Potpourri

10 min

- Announcements
- Midterm Survey
- Computing in the News

Announcements (1 of 2)

- **Midterm exam grades will be released later this week**
  - Please refrain from discussing the exam until then
  - **Remember there is the exam recovery (aka clobber) policy** so if you improve on the final exam you can make up some points on the midterm
- **Your grades do not define you — focus on a growth mindset!**
  - Example raw scores I have actually received in the past
    - CS 61C: 35/60 (58%) on the final
    - CS 70: 136/242 (56%) on the midterm
    - CS 170: 65/170 (38%) on the final
  - This is not to scare you into not pursuing CS, this is to show you that:
    - (1) You are more than your exam scores
    - (2) You can have a great college experience/GPA/career even if you "bomb" a few exams, *as long as you put in the work for the rest of the course*
  - **"It's not about your y-intercept, it's about your slope" -Berkeley professor**

Announcements (2 of 2)

- **Quiz 3 reservations are open**
- **Grades have been published** for various assignments and regrade requests are open
- New instructor tea hours location: Cory 529
- Mid-semester feedback form due tomorrow (see HW 3)
- To give y'all a break after the midterm and more time to work on Cats, we'll be ending at 6 pm (and no post-lecture questions)

Midterm Exam Survey

PollEv

Getting Started Videos

- In case you aren't aware, our lovely TAs and tutors create **Getting Started videos for all labs, HWs, and projects** which can be found by clicking on the Videos button underneath the assignment on the course website!
- They also create **walkthrough videos for past exams and discussions**
- Demo

Computing in the News

- New York City moves to adopt ban of deceptive subscription practices (The Guardian)
  - Bans "junk fees" and adopts "click to cancel" rule
- Apple accuses OpenAI of using stolen trade secrets to create its upcoming AI gadgets in new lawsuit (CNN)
- The End of Reading Is Here (The Atlantic)
- A Leak of San Francisco Police Drone Footage Exposes the New Reality of Urban Surveillance (WIRED) [Web Archive Link]
  - "...two security researchers, Sam Curry and Maik Robert, discovered that the SFPD was leaking all of the real-time footage from five of its surveillance drones, including both color and thermal imaging, accompanying location metadata, and the drone pilots' names and email addresses, to anyone who merely found the public web address where the videos were hosted."

Objects

20 min

- Motivation
- Definitions
- Defining Classes
- Working With Instances
- Instance vs. Class Attributes
- Attribute/Method Lookup
- Identity vs. Equality Revisited
- Misc. Functions to Know

Motivation

- It would be nice to be able to represent real-world entities in code, e.g. a person, bank account, college course, etc.
  - So far we've learned one way to do this: ADTs
- However, ADTs have limitations
  - Can't mutate without violating abstraction barrier (only constructors and selectors)
  - Abstraction barrier is a convention, not a formal barrier
  - No formal way to check if something is a particular ADT type
  - Attributes and behaviors are separated into multiple functions instead of encapsulated in a single entity
- **Enter... object-oriented programming (OOP)!**

Definitions

Object-oriented programming (OOP) is a way to organize programs (e.g. a type of programming paradigm) by bundling together data abstraction and behavior.

A **class** defines how **objects** of a particular type behave.

An **object** is an **instance** of a class.

Analogy: A class is like a blueprint, an object is like the house that's built.

A **method** is a function associated with a class.

Ex: The `list` class has an `append` method

Defining Classes

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

    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
```

Working With Instances

```
>>> spock_account = Account('Spock')
>>> spock_account.balance
0
>>> spock_account.holder
'Spock'
>>> spock_account.deposit(100)
100
>>> spock_account.withdraw(90)
10
>>> spock_account.withdraw(90)
'Insufficient funds'
>>> picard_account = Account('Picard')
>>> picard_account.balance += 100
>>> Account.withdraw(picard_account, 75)
25
```

When poll is active respond at **PollEv.com/rebeccadang831**

Activate this Poll with the Poll Everywhere Live app.

If video conferencing, you must be sharing your full screen to share the activated poll.

Methods: What would Python display?

Instance vs. Class Attributes (1 of 2)

```
class Account:
    account_number = 1

    def __init__(self, account_holder):
        self.balance = 0
        self.holder = account_holder
        self.account_number = Account.account_number
        Account.account_number += 1

    ...
```

Instance vs. Class Attributes (2 of 2)

```
>>> kaz_account = Account('Kaz')
>>> kaz_account.account_number
1
>>> kaz_account.account_number += 10
>>> kaz_account.account_number
11
>>> Account.account_number
2
>>> inej_account = Account('Inej')
>>> inej_account.account_number
2
```

When poll is active respond at **PollEv.com/rebeccadang831**

Activate this Poll with the Poll Everywhere Live app.

If video conferencing, you must be sharing your full screen to share the activated poll.

Attribute Lookup: What would Python display?

Attribute/Method Lookup (1 of 5)

Both instances and classes have attributes and methods* (e.g. names) that can be accessed or modified with **dot notation**:

```
<instance or class expression>.<name>
```

*Note: **For the purposes of lookup/evaluation, Python does not actually differentiate between attributes or methods.** If you want to get *really* technical, methods are both attributes and functions. However, we *typically* refer to attributes and methods as separate things: attributes refer only to local state associated with an object/class, while methods refer only to functions associated with an object/class.

Attribute/Method Lookup (2 of 5)

To evaluate a **dot expression:**

1. Evaluate the expression to the left of the dot
2. To determine what `<name>` evaluates to:
   a. If step 1 evaluates to an instance, first check if the instance has an instance attribute/method that matches `<name>`, and if it does, return its value
   b. Otherwise, if there is no matching instance name **OR** if step 1 evaluates to a class, check if there is a class attribute* that matches `<name>`
   c. If there's no match, raise an `AttributeError`

*Note: **A method can be bound to a particular class, even though it is an instance method** (e.g. a method that acts on an instance of the class). We'll go over an example later to explain this. However, `classmethod` and `staticmethod` are out of scope.

Attribute/Method Lookup (3 of 5)

```
>>> class Car:
...     def __init__(self):
...         self.fuel = 10
...     def drive(self, distance):
...         if distance > self.fuel:
...             return "Can't drive that far"
...         self.fuel -= distance
...         return self.fuel
...
>>> a = Car()
>>> b = Car()
```

Attribute/Method Lookup (4 of 5)

```
>>> a.drive = lambda distance: "Car overheating!"
>>> a.drive(1)
'Car overheating!'
>>> b.drive(5)
5
>>> Car.drive = lambda self: "Tire popped!"
>>> a.drive(100)
'Car overheating!'
>>> b.drive()
'Tire popped!'
>>> f = a.drive
>>> f(1)
'Car overheating!'
```

Attribute/Method Lookup (5 of 5)

Just because you *can* doesn't mean you *should*.

- By convention, programmers typically do not define attributes/methods or reassign methods outside the class definition
- We're showing you this syntax for the same reason why we teach environment diagrams:
  - So you can form a mental model of how the computer stores/handles data
  - So that you read code for *what it actually does*, not *what you think it does*

Identity vs. Equality Revisited

```
>>> class Person:
...     def __init__(self, name):
...         self.name = name
...
>>> rebecca1 = Person('Rebecca')
>>> rebecca2 = Person('Rebecca')
>>> rebecca1 is rebecca2
False
>>> rebecca1 == rebecca2
False
```

```
>>> class Person:
...     def __init__(self, name):
...         self.name = name
...     def __eq__(self, other):
...         if type(other) is not Person:
...             return NotImplemented
...         return self.name == other.name
...
>>> rebecca1 = Person('Rebecca')
>>> rebecca2 = Person('Rebecca')
>>> rebecca1 is rebecca2
False
>>> rebecca1 == rebecca2
True
```

Misc. Functions to Know

```
>>> class Person:
...     def __init__(self, name):
...         self.name = name
...
>>> rebecca = Person('Rebecca')
>>> type(rebecca)
<class 'Person'>
>>> isinstance(rebecca, Person)
True
>>> isinstance(rebecca, str)
False
>>> hasattr(rebecca, 'name')
True
>>> getattr(rebecca, 'name')
'Rebecca'
```

Let's Design Spotify!

(reminder to self to use high contrast theme)

20 min

- Classes Brainstorm
- Implement the Classes

Classes Brainstorm

Suppose we want to create our own music streaming app, similar to Spotify.

What classes should we have?

Possible answers: User, Artist, Song, Album, Playlist

What should these classes be able to do? What are the relationships between classes?

Possible answers:

- A `User` should be able to follow an `Artist`
- A `User` should be able to add a `Song` to their Liked Songs `Playlist`
- An `Album` should be created by an `Artist` and is composed of multiple `Songs`
- A `Playlist` is composed of multiple `Songs`
- A `Song` should be created by an `Artist`

Implement the Classes

- Implement all classes
  - Some classes depend on other classes to be implemented fully before you can run the doctests
  - I recommend implementing each class in order, then running the doctests at the end
- Download starter code `13.py` from the course website under Lecture 13
- Run doctests with `python3 -m doctest 13.py`
- 5 min: Try implementing it yourself
- 5 min: Discuss with someone next to you and compare your implementations
  - What worked?
  - What didn't?
  - Why?