CSCI 145 · Introduction to Computing · Fall 2026

Objects

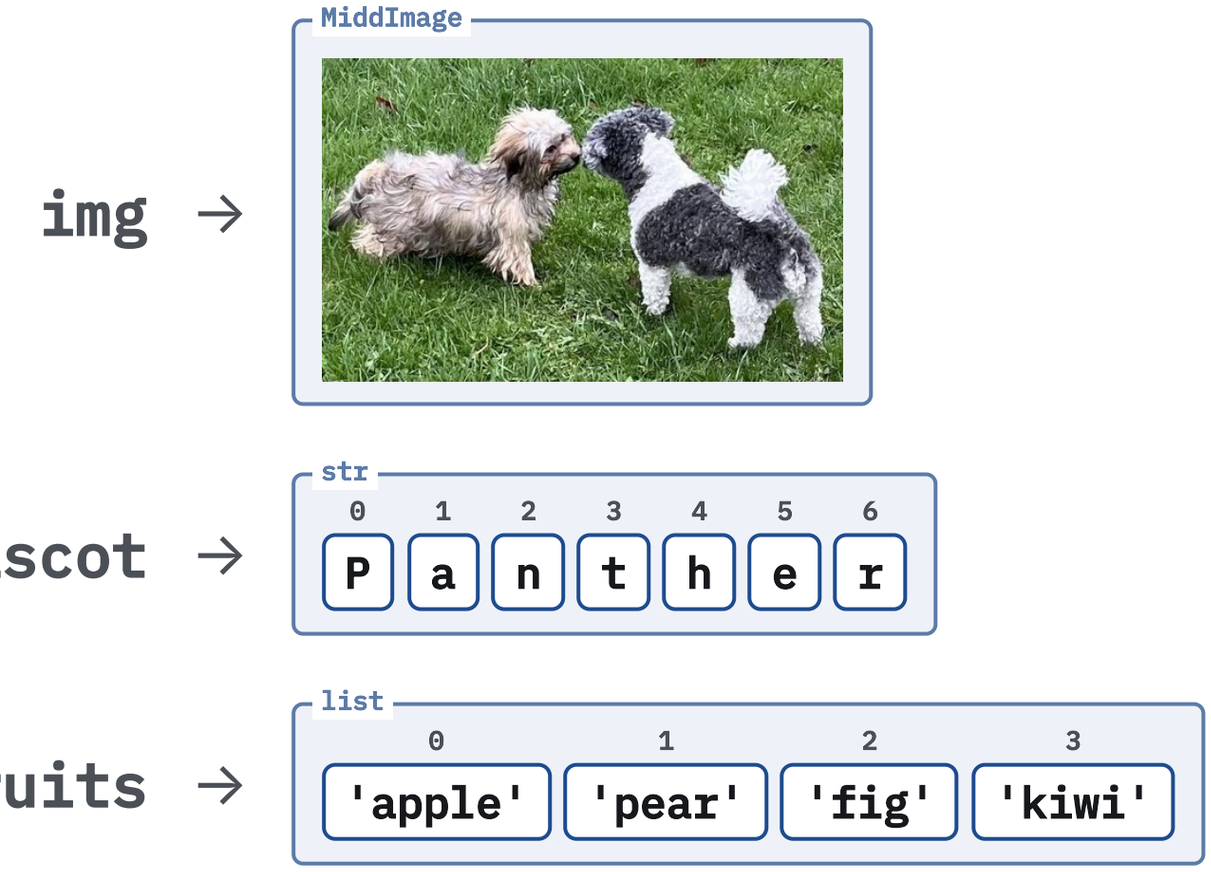
Lecture 18 — classes, attributes, and methods

Registration

Spring 2027:

- CSCI 200 – Mathematical Foundations of Computing
- CSCI 201 – Data Structures

We've been using objects all along



An image, a string, a list: each is an **object**

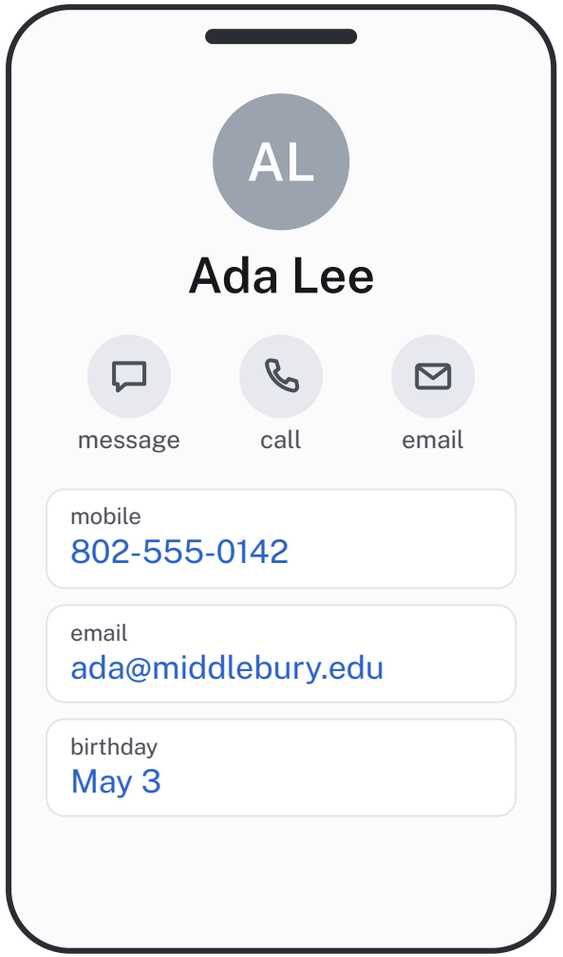
Today: make our own kinds of objects

Part 1 of 3

Objects bundle data and functions

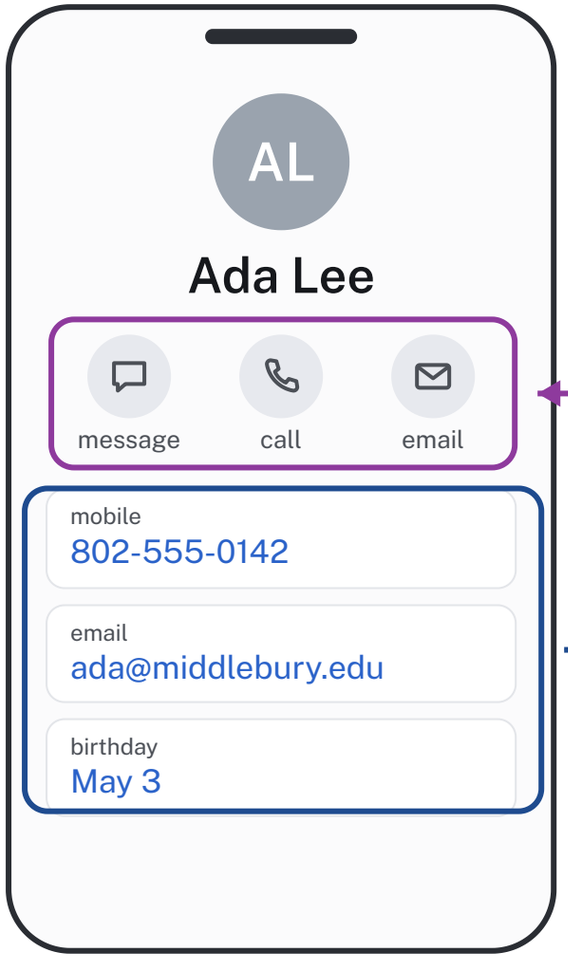
> An object stores its data in **attributes**
> and has functions for that data, called **methods**
> Programming with objects is **object-oriented programming**

Think about the contacts on your phone



1. What **data** does a contact store?
2. What can you **do** with a contact?

A contact is an object



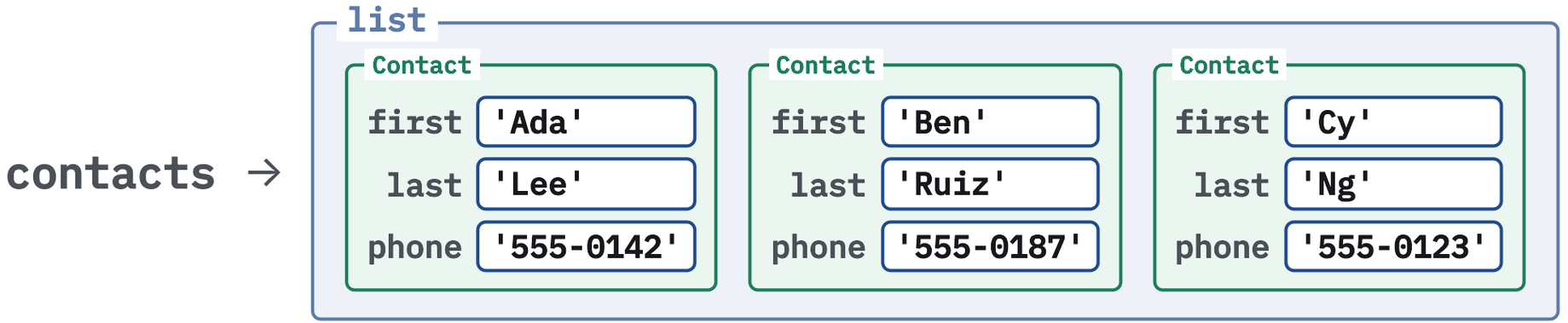
Methods
functions that work on the data

Attributes
data stored in the object

An **object** bundles attributes and methods together

Storing many contacts

One list of `Contact` objects: each keeps its data together



WITHOUT OBJECTS: ONE LIST PER KIND OF DATA

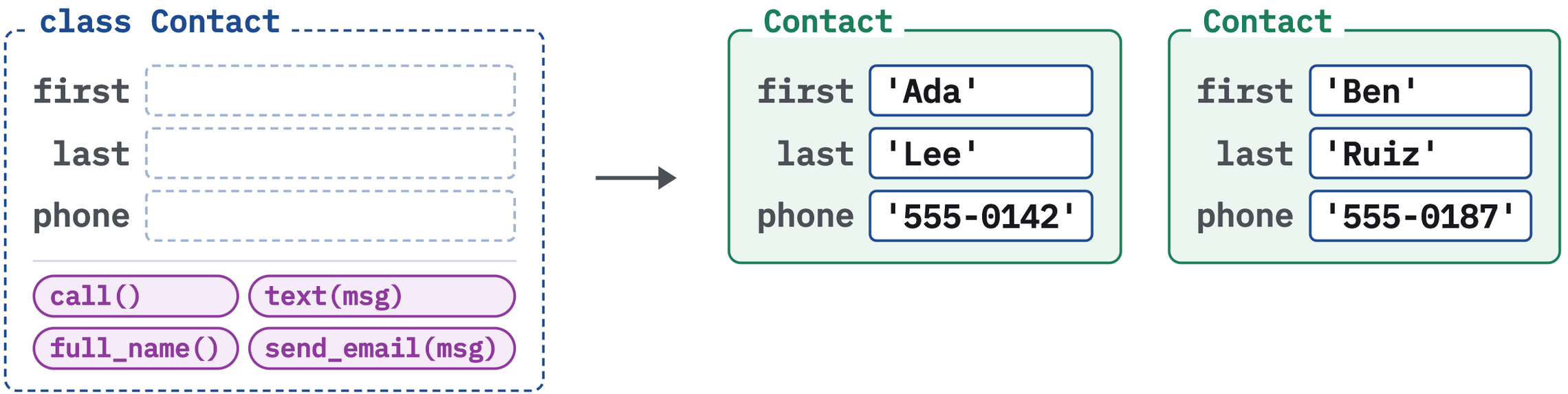
```
first_names → 'Ada' 'Ben' 'Cy'
 last_names → 'Lee' 'Ruiz' 'Ng'
     phones → '555-0142' '555-0187' '555-0123'
```

Ben is spread over three lists

Sort or delete a contact: change all three

Classes and instances

A **class** is a blueprint for a kind of object



- Each object made from it is an **instance** of the class
- Each instance has its own attributes
- The methods belong to the class: every instance shares them

"A is a B": A is an instance of class B

INSTANCE		CLASS
Ada Lee	*is a*	contact
Willa	*is a*	dog
the queen of spades	*is a*	card
`"hi"`	*is a*	`str`
`[4, 5]`	*is a*	`list`
`img`	*is a*	`MiddImage`

```
>>> type("hi")
<class 'str'>
>>> type([4, 5])
<class 'list'>
>>> type(img)
<class 'midd_media.middimage.MiddImage'>
```

Class names are in **CamelCase**

`Contact` `MiddImage`

Classes you already know

`str`

Attributes

its characters

Methods

`upper()`

`split()`

`strip()`

`list`

Attributes

its items

Methods

`append(item)`

`pop()`

`sort()`

`MiddImage`

Attributes

`width`, **`height`**, **`mode`**

the pixels

Methods

`show()`

`save(filename)`

`copy()`

Designing a class

`Contact` A contact in a phone's address book

What does it have?

Attribute	Example value
`first`	'Ada'
`last`	'Lee'
`phone`	'802-555-0142'
`email`	'ada@midd.edu'

What can it do?

Method call	What it does or returns
`contact.call()`	calls the number
`contact.text(msg)`	texts **`msg`**
`contact.full_name()`	returns **`'Ada Lee'`**
`contact.send_email(msg)`	emails **`msg`**

Attributes are nouns — what it has

Methods are verbs — what it does

Activity A — design a class

Worksheet Part A: a music app needs a `Song` class

Song A song in a music app

Attribute	Example value
`title`	'Clair de Lune'

Method call	What it does or returns
`song.play()`	plays the song

Then do the same for a `Playlist`

Activity A — some answers

`Song`

Attribute	Example value
`title`	'Clair de Lune'
`artist`	'Debussy'
`seconds`	303
`plays`	12

Method call	What it does or returns
`song.play()`	plays it
`song.length()`	returns **`'5:03'`**
`song.longer_than(other)`	**`True`** or **`False`**
`song.add_play()`	adds 1 to **`plays`**

`Playlist`

Attribute	Example value
`name`	'Practice'
`songs`	a list of **`Song`** objects

Method call	What it does or returns
`playlist.add(song)`	adds a song
`playlist.total_seconds()`	returns 664
`playlist.longest()`	returns a **`Song`**

Objects inside objects!

Part 2 of 3

Writing a class

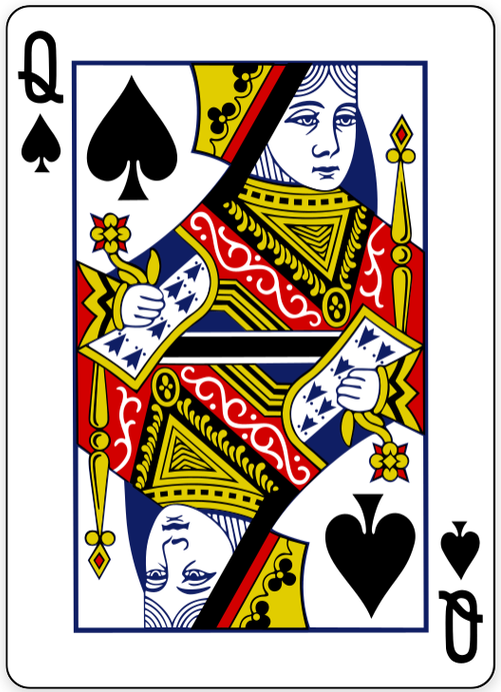
> `class` defines a new kind of object
> `__init__` gives each new object its attributes
> Every method's first parameter is `self`, the object itself

Let's write a `Card` class

`Card` A playing card, aces high

Attribute	Example value
`rank`	'Q'
`suit`	'spades'

Method call	What it does or returns
`card.value()`	returns 12 (2 up to 14 for an ace)
`card.beats(other)`	**`True`** if **`card`** is higher
`print(card)`	prints **`Q of spades`**



Anatomy of a class

```
class Card:
    """A playing card"""

    def __init__(self, rank, suit):
        self.rank = rank
        self.suit = suit
```

class · __init__ · self · attributes

`class Card:`

- The keyword `class`, then the name
- The name is in CamelCase
- The body is indented
- The docstring says what a Card is

Anatomy of a class

```
class Card:
    """A playing card"""

    def __init__(self, rank, suit):
        self.rank = rank
        self.suit = suit
```

class · __init__ · self · attributes

`def __init__(…):`

- The **constructor**
- Two underscores on each side
- Runs when we make a new `Card`
- Sets up the new card's attributes

Anatomy of a class

```
class Card:
    """A playing card"""

    def __init__(self, rank, suit):
        self.rank = rank
        self.suit = suit
```

class · __init__ · self · attributes

`self`

- The new object itself
- Always the first parameter
- Python passes it for us

Anatomy of a class

```
class Card:
    """A playing card"""

    def __init__(self, rank, suit):
        self.rank = rank
        self.suit = suit
```

class · __init__ · self · **attributes**

`self.rank = rank`

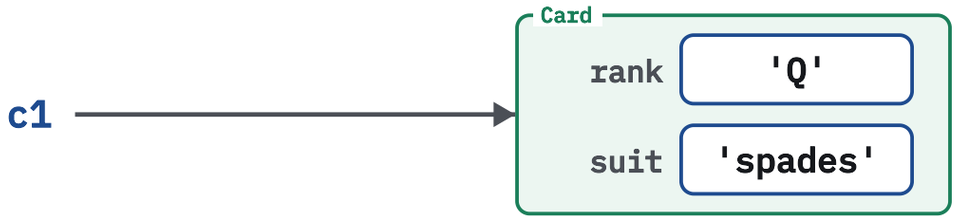
- Gives the object an **attribute**
- Parameter `rank` is gone when `__init__` ends
- The attribute stays with the object

Making a Card

```
class Card:
    """A playing card"""

    def __init__(self, rank, suit):
        self.rank = rank
        self.suit = suit

c1 = Card("Q", "spades")
```

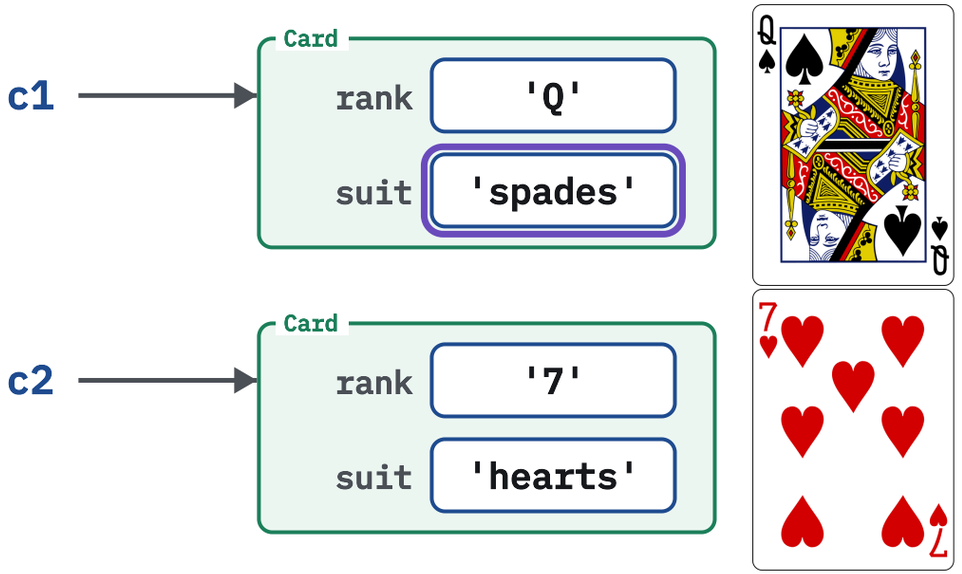


__init__ is done: c1 names the new Card

- We pass 2 arguments, not 3: Python passes `self` for us

Two cards, two objects

```
>>> c1 = Card("Q", "spades")
>>> c2 = Card("7", "hearts")
>>> c1.rank
'Q'
>>> c2.suit
'hearts'
>>> c1.suit
'spades'
```



- One class, two instances
- `c1.rank` reads c1's attribute
- Each object has its own attributes

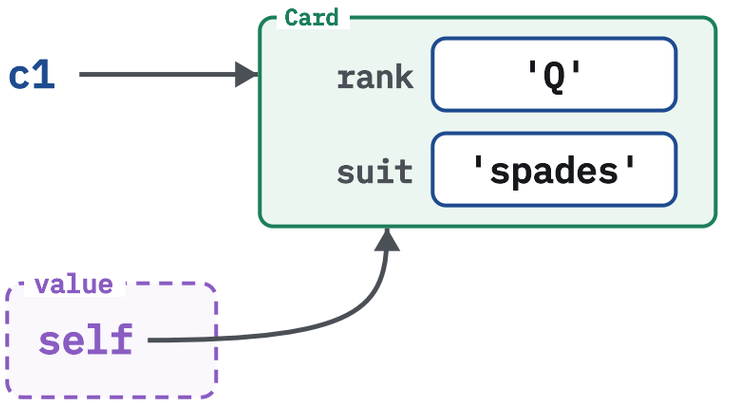
A method

```
RANKS = ["2", "3", "4", "5", "6", "7", "8",
         "9", "10", "J", "Q", "K", "A"]

class Card:
    # __init__ as before

    def value(self):
        """Returns the card's value, 2 to 14"""
        return RANKS.index(self.rank) + 2
```

```
>>> c1.value()
12
>>> c2.value()
7
```



- A function inside the class
  Its first parameter is `self`
- `c1.value()` runs
  the `value` method
  with `self` naming `c1`
- `RANKS.index('Q')` is 10
  the position of `'Q'` in the list
  10 + 2 is 12
- Same method, other object,
  other answer

Printing a Card

```
>>> print(c1)
<__main__.Card object at 0x10e2f3b50>
```

- Python doesn't know how to show a Card

  It prints its class and where it is in memory

- We'd rather see `Q of spades`



`Q of spades`

SOLUTION

Give the class a method that tells Python how to show a Card

Printing a Card: `__str__`

```
    def __str__(self):
        """Returns the card as a string"""
        return self.rank + " of " + self.suit
```

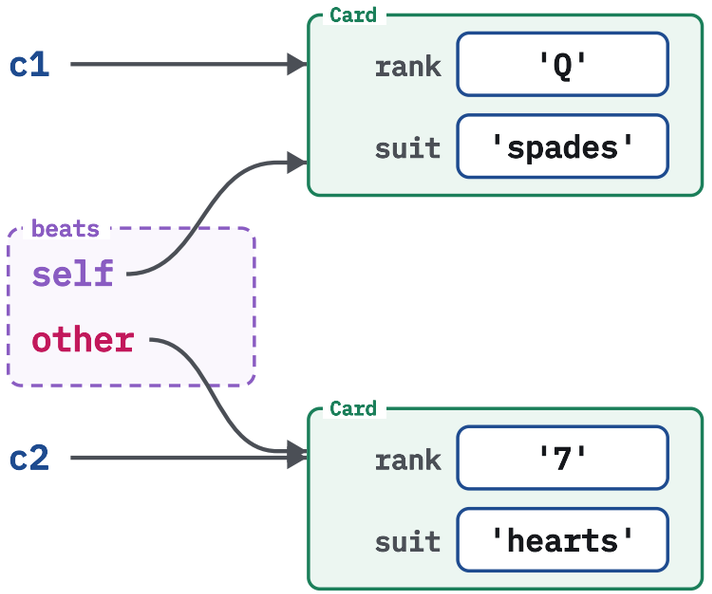
```
>>> print(c1)
Q of spades
>>> str(c1)
'Q of spades'
>>> print(c1, c2)
Q of spades 7 of hearts
```

- A special method: two underscores on each side
- `print` and `str` call `__str__` for us
- `__str__` **returns** the string
  It doesn't print it
- In the Shell, `c1` alone still shows the address

A method that takes another object

```
    def beats(self, other):
        """Returns True if this card beats other"""
        return self.value() > other.value()
```

```
>>> c1.beats(c2)
True
>>> c2.beats(c1)
False
```



- `self` is the card before the dot
  `other` is the argument
- Inside the class, call a method on `self`:
  `self.value()`
- 12 > 7

Activity B — what does this print?

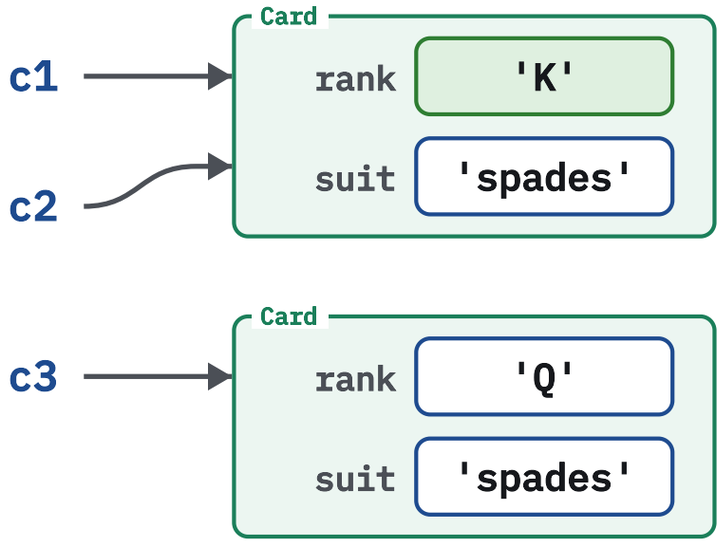
```
c1 = Card("Q", "spades")
c2 = c1
c3 = Card("Q", "spades")
c2.rank = "K"
print(c1, c3)
```

Worksheet Part B

- Draw the variables and objects
- Then write what it prints

Activity B — answer

```
c1 = Card("Q", "spades")
c2 = c1
c3 = Card("Q", "spades")
c2.rank = "K"
print(c1, c3)
```



OUTPUT

```
K of spades Q of spades
```

- `c2 = c1` makes a second name for the same Card
- `c3` is a new Card that happens to match
- `c2.rank = "K"` changes the card `c1` names too
- `c3` is unchanged

Part 3 of 3

Objects that hold objects

> An attribute can be any value, even a list of objects
> A `Deck` **has** 52 `Card` objects

A deck of cards

`Deck` A deck of playing cards

Attribute	Example value
`cards`	a list of 52 **`Card`** objects

Method call	What it does or returns
`deck.size()`	returns 52
`deck.shuffle()`	shuffles the cards
`deck.deal()`	removes and returns a **`Card`**

An attribute can be any value

even a list of objects

A deck: 52 cards



4 suits (clubs, diamonds, hearts, spades)

× 13 ranks (2, 3, 4, 5, 6, 7, 8, 9, 10, J, Q, K, A)

= 52 cards

A deck: a list of 52 `Card` objects



- A `Deck` object has one attribute, `cards`
- It's a list that holds 52 `Card` objects

The `Deck` class: `__init__`

```
import random

RANKS = ["2", "3", "4", "5", "6", "7", "8",
         "9", "10", "J", "Q", "K", "A"]
SUITS = ["clubs", "diamonds", "hearts", "spades"]

class Deck:
    def __init__(self):
        self.cards = []
        for suit in SUITS:
            for rank in RANKS:
                self.cards.append(Card(rank, suit))
```

- The attribute `cards` starts as an empty list
- Outer loop: one suit, one row
  Inner loop: 13 ranks across
- Each new `Card` is appended to `self.cards`

The `Deck` class: methods

```
class Deck:
    # __init__ as before

    def size(self):
        return len(self.cards)

    def shuffle(self):
        random.shuffle(self.cards)

    def deal(self):
        return self.cards.pop()
```

- `size` returns how many cards are left
- `random.shuffle` shuffles a list in place
- `pop()` removes and returns the last item, so `deal` returns a `Card`

A round of War

```
>>> deck = Deck()
>>> deck.size()
52
>>> deck.shuffle()
>>> mine = deck.deal()
>>> yours = deck.deal()
>>> print(mine, yours)
K of diamonds 4 of clubs
>>> mine.beats(yours)
True
>>> deck.size()
50
```

In War, each player turns over a card

The higher card wins



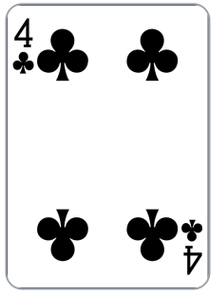
50 cards

shuffled

mine



yours



- `Deck()`: `__init__` has only `self`, so we pass nothing
- `deal` returns a Card, so we can call Card methods on it
- Run `day18_cards.py` to play a whole game

  Try it after class

Common errors

```
def value():
```

```
TypeError: Card.value() takes 0 positional arguments but 1 was given
```

Forgot `self`: Python always passes the object

```
rank = rank
```

```
AttributeError: 'Card' object has no attribute 'rank'
```

Forgot `self.` in `__init__`: no attribute was made

```
RANKS.index(rank)
```

```
NameError: name 'rank' is not defined
```

Forgot `self.` when using an attribute in a method

```
Card("Q")
```

```
TypeError: Card.__init__() missing 1 required positional argument: 'suit'
```

Arguments must match `__init__`'s parameters

```
c1.value
```

```
<bound method Card.value of <__main__.Card object at 0x10e2f3b50>>
```

Forgot the parentheses: no call, so no 12

```
def __str__(self):
    print(self.rank)
```

```
TypeError: __str__ returned non-string (type NoneType)
```

`__str__` must **return** a string, not print it

Activity C

Download day18.zip and extract it in your `class` folder

In `day18_songs.py`, write the `Song` class:

`__init__`, `__str__`, `length`, and `longer_than`

Uncomment the Activity C tests at the bottom and run the file

PATTERNS

Store a parameter	`self.title = title`
Use an attribute	`self.seconds`
Whole minutes, seconds left over	`self.seconds // 60` `self.seconds % 60`
Number to string	`str(minutes)`
5 seconds as 05	`"0" + str(secs)`
The other song's attribute	`other.seconds`

Activity C — answers

```
class Song:
    def __init__(self, title, artist, seconds):
        self.title = title
        self.artist = artist
        self.seconds = seconds

    def __str__(self):
        return self.title + " by " + self.artist

    def longer_than(self, other):
        return self.seconds > other.seconds
```

```
    def length(self):
        minutes = self.seconds // 60
        secs = self.seconds % 60
        if secs < 10:
            return str(minutes) + ":0" + str(secs)
        return str(minutes) + ":" + str(secs)
```

On your own for practice

In `day18_songs.py`, write the `Playlist` class: `__init__`, `add`, `total_seconds`, and `longest`
Challenge: write `__str__`
The solution is posted after class

PATTERNS

Start with no songs	`self.songs = []`
Add a song	`self.songs.append(song)`
Every song	`for song in self.songs:`
Compare two songs	`song.longer_than(best)`
A song as a string	`str(song)`
A new line	`"\n"`

To do

- Review this slide deck and Notes 15
- Homework 5 (files and data) is due Thursday 11/12
- Read Lab 5 before Friday 11/13
- Register for CSCI 200 and/or CSCI 201 for Spring 2027
- Quiz 9 (Lectures 17–18) is at the start of next class
- Homework 6 (images and objects) is due Thursday 11/19

Next time

Lecture 19 — encapsulation, inheritance, and polymorphism

Two queens of spades: is `c1 == c3` *True?*