Intro to Coding with Python– Classes Pt 1

Dr. Ab Mosca (they/them)

Slides based off slides courtesy of Jordan Crouser (https://jcrouser.github.io/)

Plan for Today

- Recap: **`lists`** and **`dictionaries`**
- Activity: **`functions`** vs. **`methods`**
- Creating our first **`class`**
  - **`attributes`**
  - **`methods`**
  - **`self`**
  - getting started

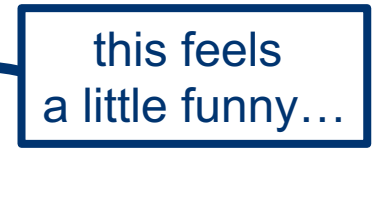
RECAP: using `lists` and `dictionaries`

```
def addSong(library):
    # Initialize an empty dictionary
    song = {}

    # Fill in details
    song["title"] = input("Song title: ")
    song["artist"] = input("Artist: ")
    song["album"] = input("Album: ")

    # Append song to library
    library.append(song)
```

RECAP: `lists` and `dictionaries`




```
def printSongs(library):
    # A counter is one way to number the songs
    counter = 0

    # Loop over all the songs in the library
    for song in library:
        counter += 1
        # String formatting to the rescue!
        print("{}. '{}' by {} ({})".format(counter, song
        ["title"], song["artist"], song["album"]))
```

Discussion

Compare this with other operations we can perform on **`lists`** and **`dictionaries`**; what do you **notice**?

```
animals.append('guinea pig')
vowels.insert(3, 'o')
animals.remove('rabbit')
pets.count('dog')
numbers.reverse()
names.sort()
numbers.copy()
```

Discussion

So what's the difference between a **`function`** and a **`method`**?

Activity: `functions` vs. `methods`

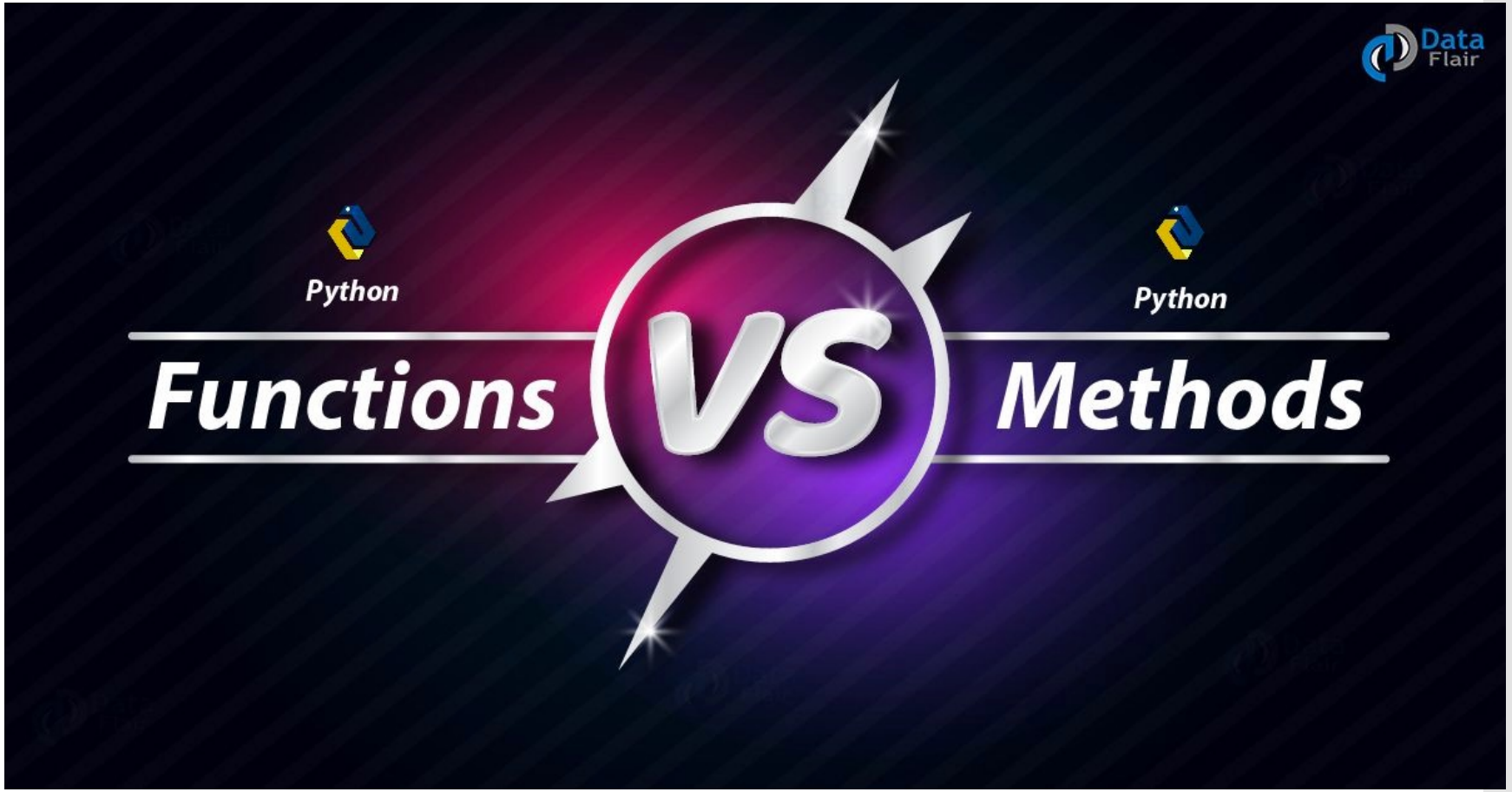
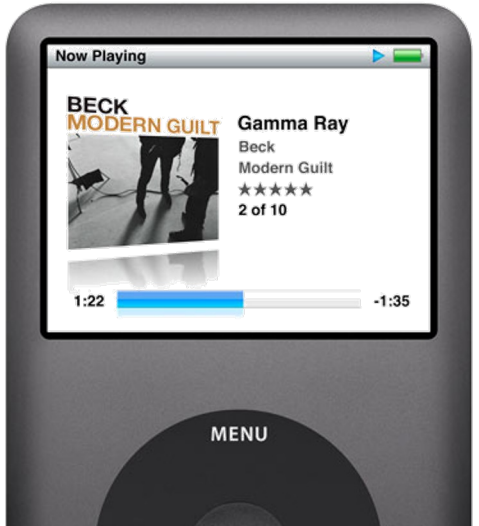


Image courtesy: https://data-flair.training/blogs/python-method-vs-function/

Back to the `playlist`: what we want

- We'd like to be able to ask a **`Song`** to **`print()`** or **`play()`** itself (since it already has access to all the information)
- That way we don't have to waste time **passing everything** from function to function
- To do this, we'll need a way to combine functions (**`methods`**) and variables (**`attributes`**)
- Solution: **`classes`**



Building a `Die` class

How this might look

*dice.py - /Users/jcrouser/Google Drive/Teaching/Course Material/SCS-Noona...

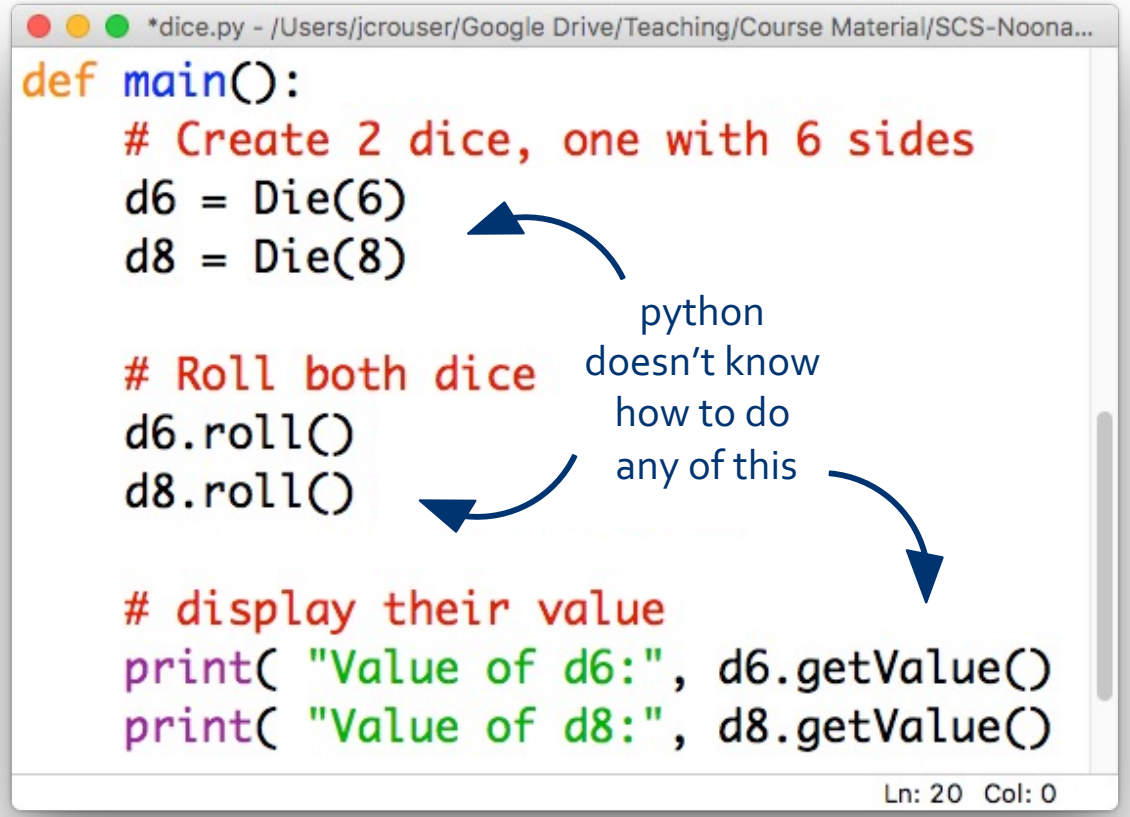
```
def main():
    # Create 2 dice, one with 6 sides
    d6 = Die(6)
    d8 = Die(8)

    # Roll both dice
    d6.roll()
    d8.roll()

    # display their value
    print( "Value of d6:", d6.getValue()
    print( "Value of d8:", d8.getValue()
```

Ln: 20 Col: 0

Just one problem…



we need to build it a **blueprint**

1. a way to build a **`Die`** given # sides

*dice.py - /Users/jcrouser/Google Drive/Teaching/Course Material/SCS-Noona...

```
def main():
    # Create 2 dice, one with 6 sides
    d6 = Die(6)
    d8 = Die(8)

    # Roll both dice
    d6.roll()
    d8.roll()

    # display their value
    print( "Value of d6:", d6.getValue()
    print( "Value of d8:", d8.getValue()
```

Ln: 20 Col: 0

2. to be able to `.roll()` them

*dice.py - /Users/jcrouser/Google Drive/Teaching/Course Material/SCS-Noona...

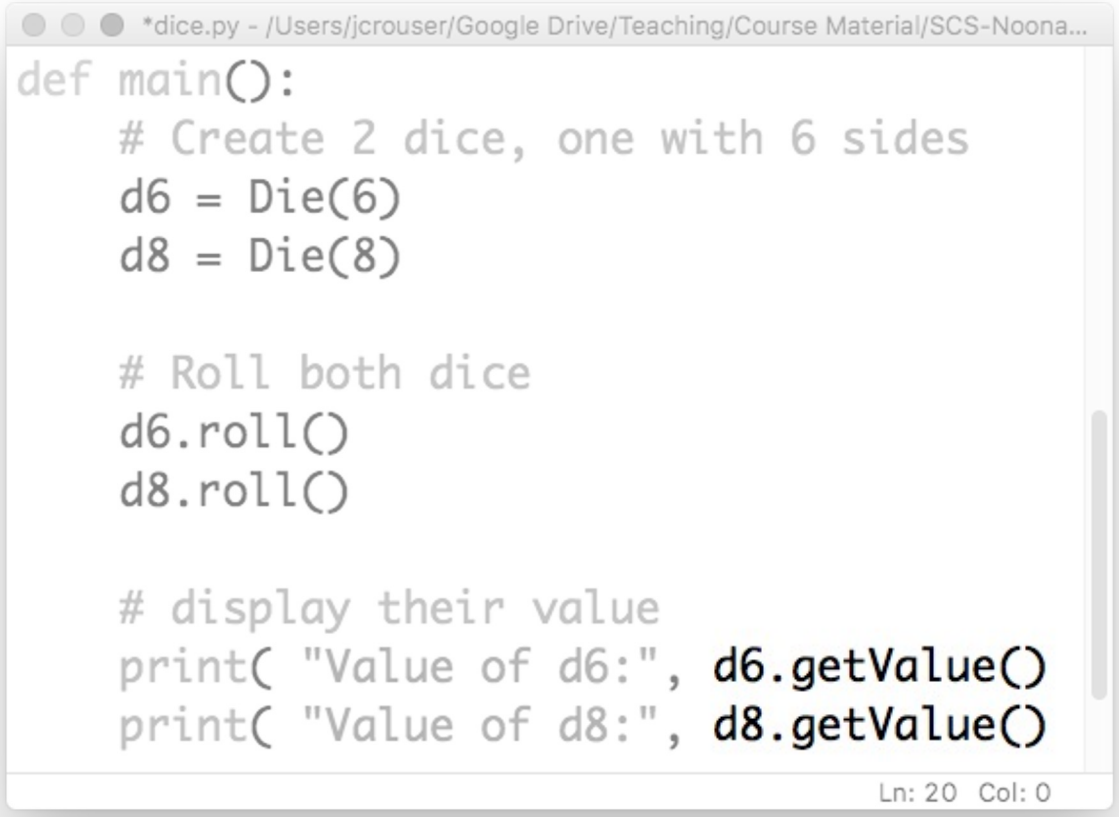
```
def main():
    # Create 2 dice, one with 6 sides
    d6 = Die(6)
    d8 = Die(8)

    # Roll both dice
    d6.roll()
    d8.roll()

    # display their value
    print( "Value of d6:", d6.getValue()
    print( "Value of d8:", d8.getValue()
```

Ln: 20 Col: 0

3. to be able to `.getValue()`



Blueprint for a `Die`

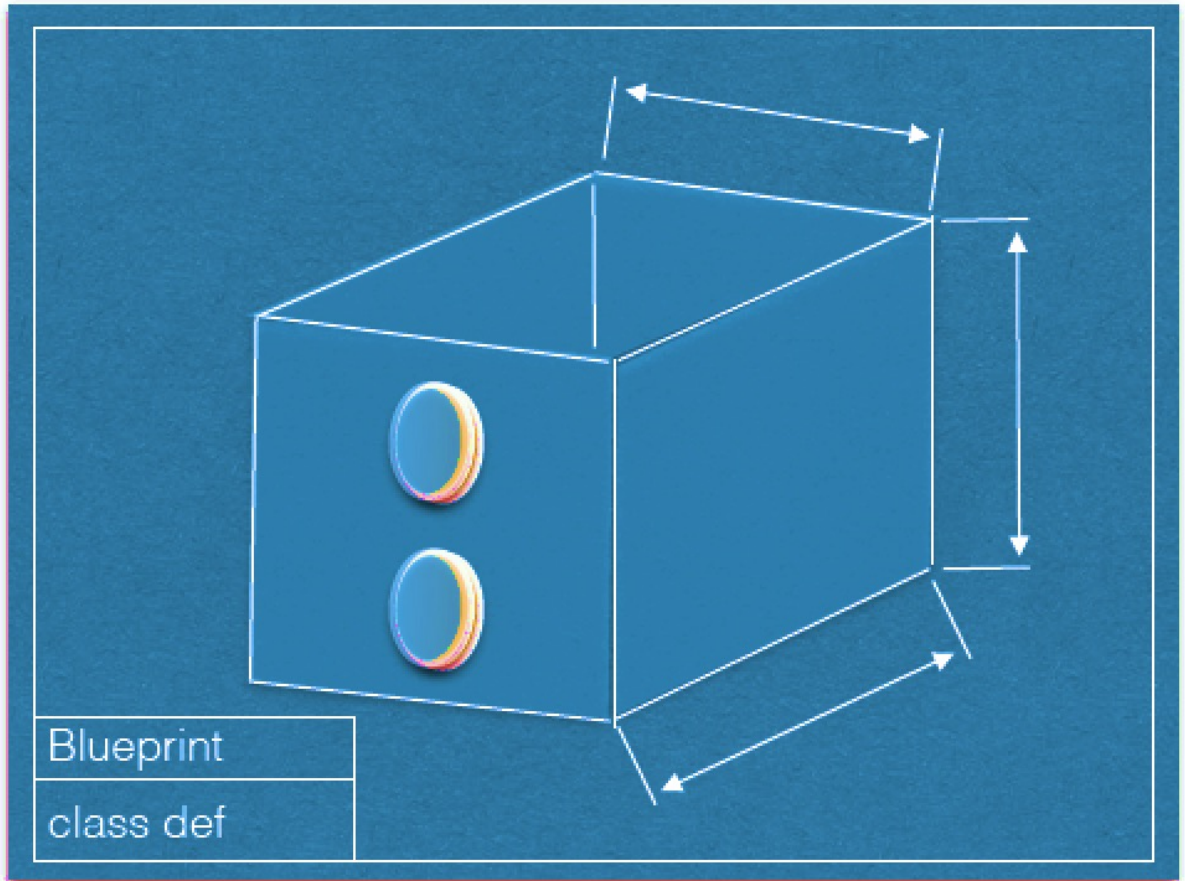


Image courtesy Dominique Thiebaut, Smith College

Blueprint for a `Die`: `attributes`

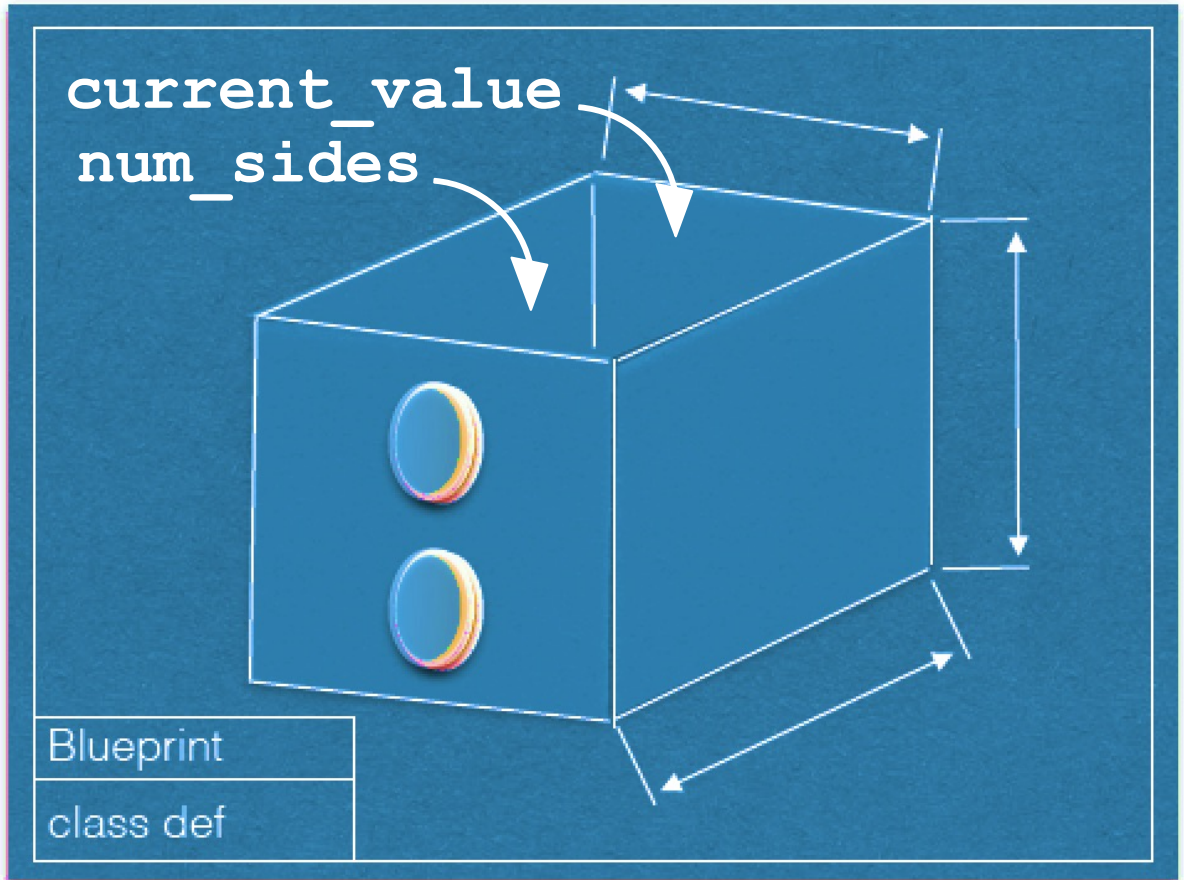


Image courtesy Dominique Thiebaut, Smith College

Blueprint for a `Die`: `methods`

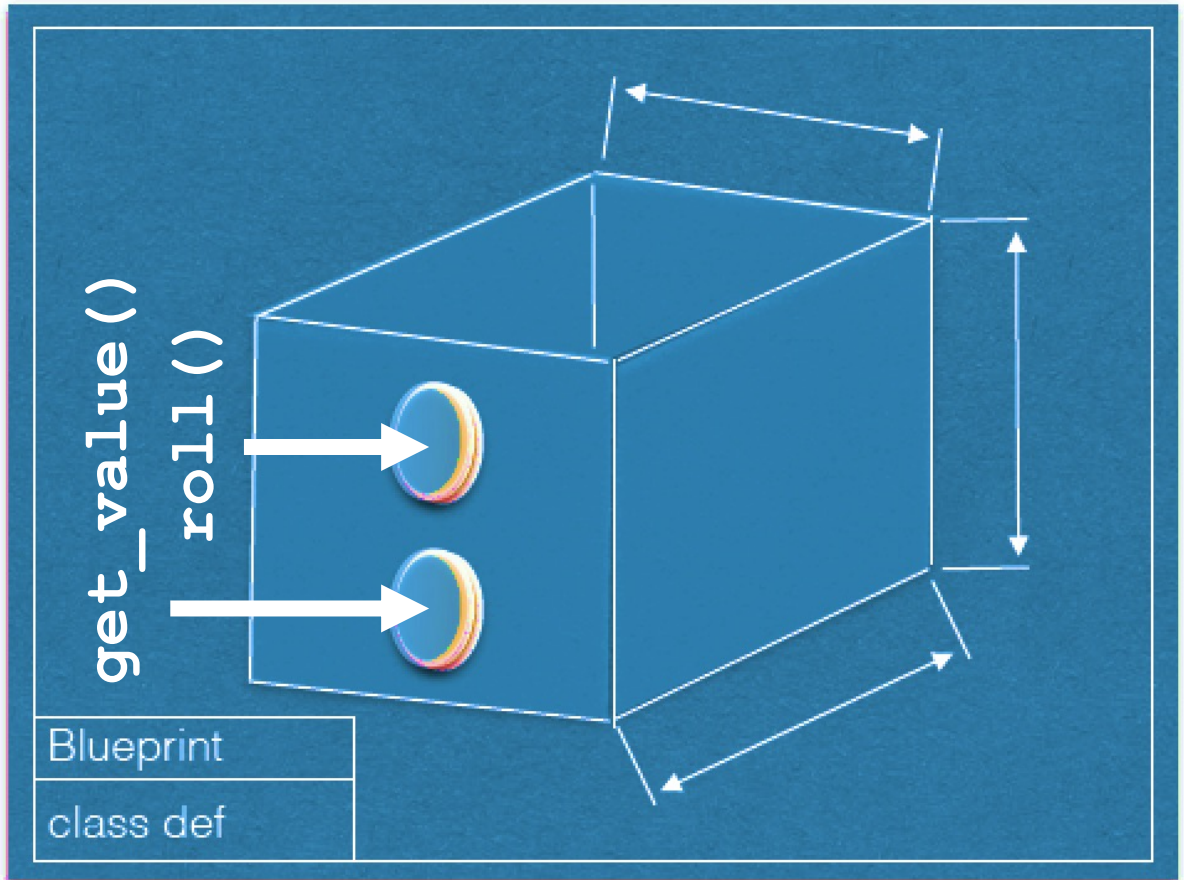
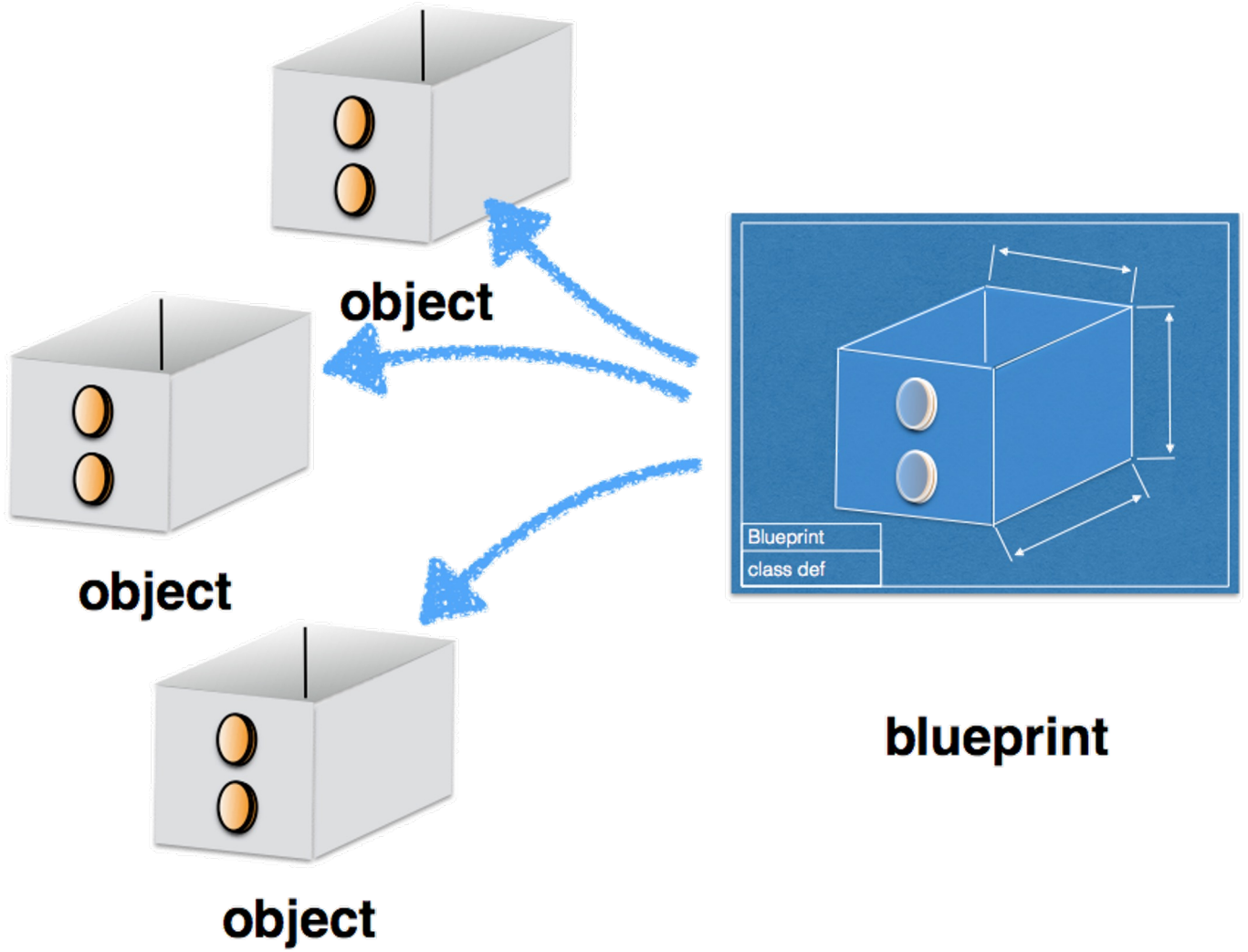
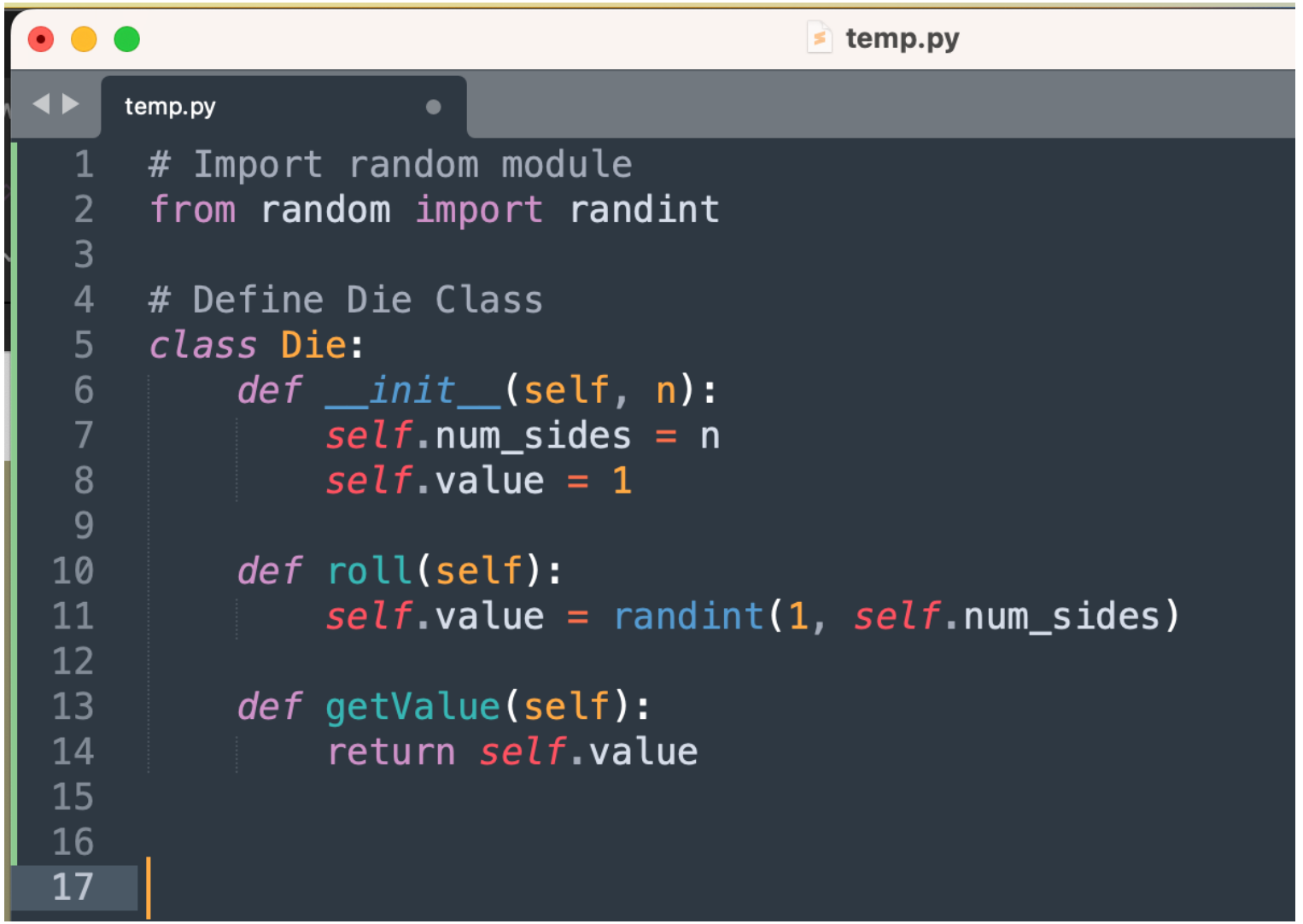


Image courtesy Dominique Thiebaut, Smith College

Given a blueprint, **replication** is easy

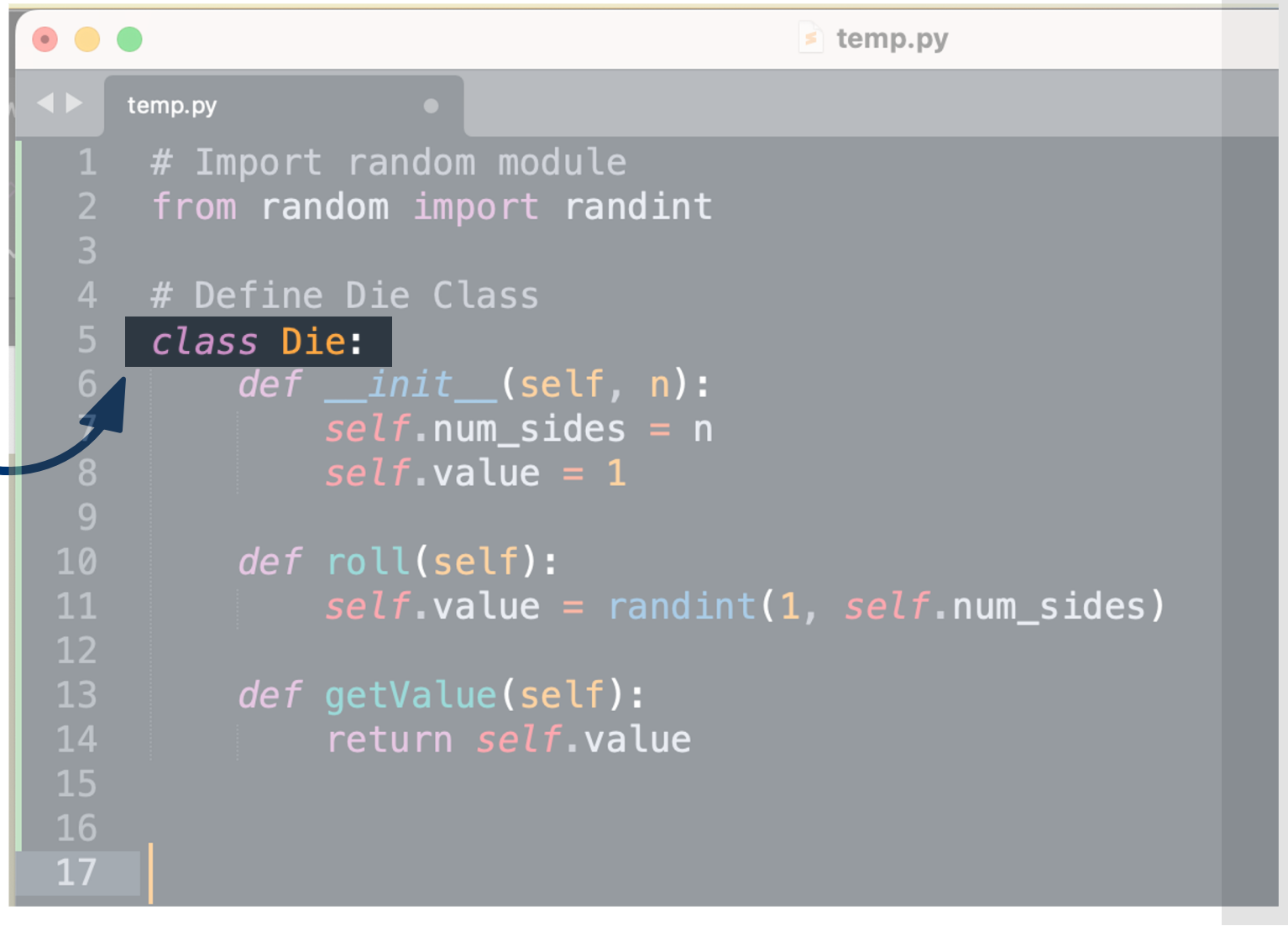


Coding the `Die` `class`



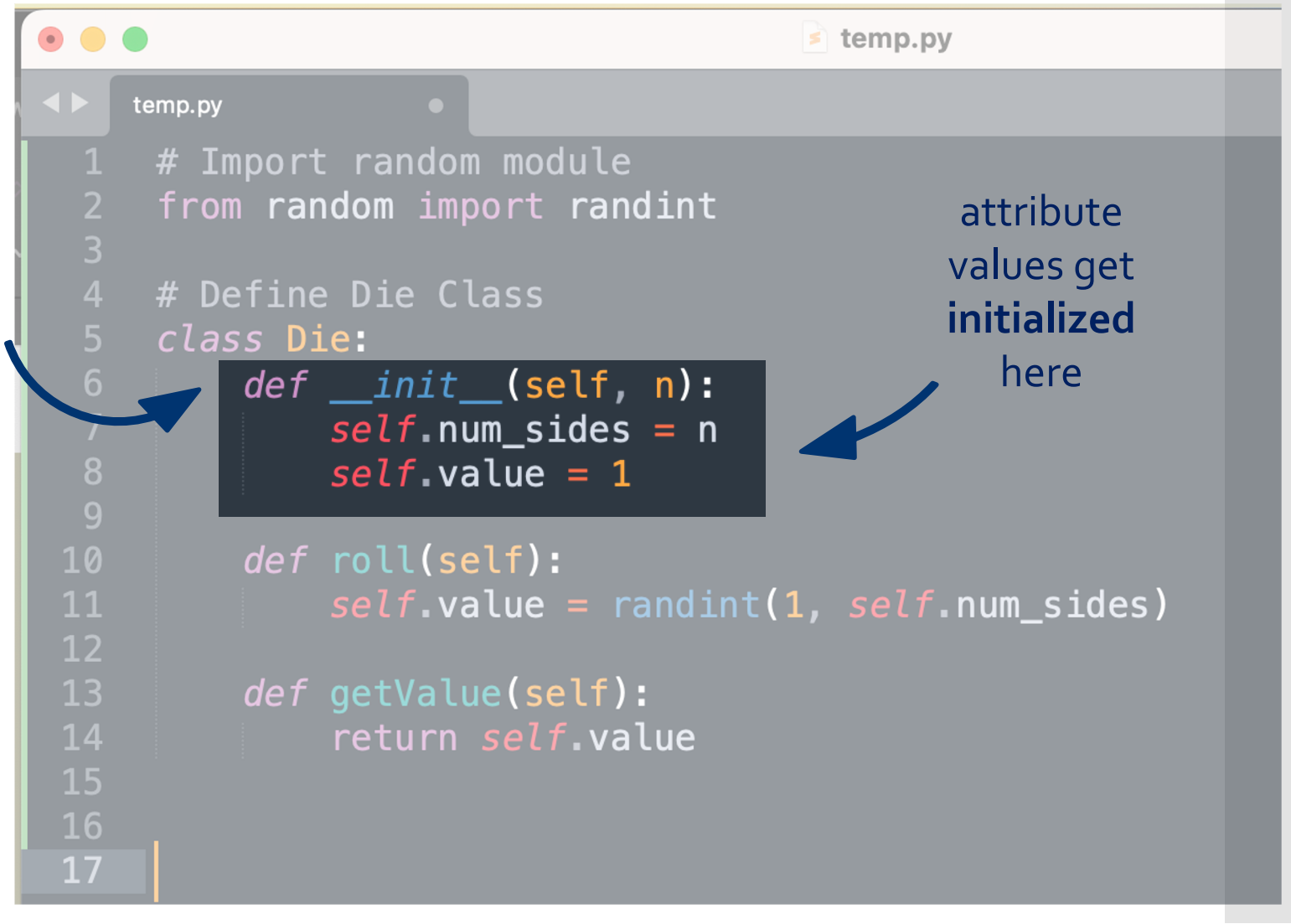
`classes` are defined using **`class`**

convention: class names start with a capital letter



All **classes** need a **constructor**

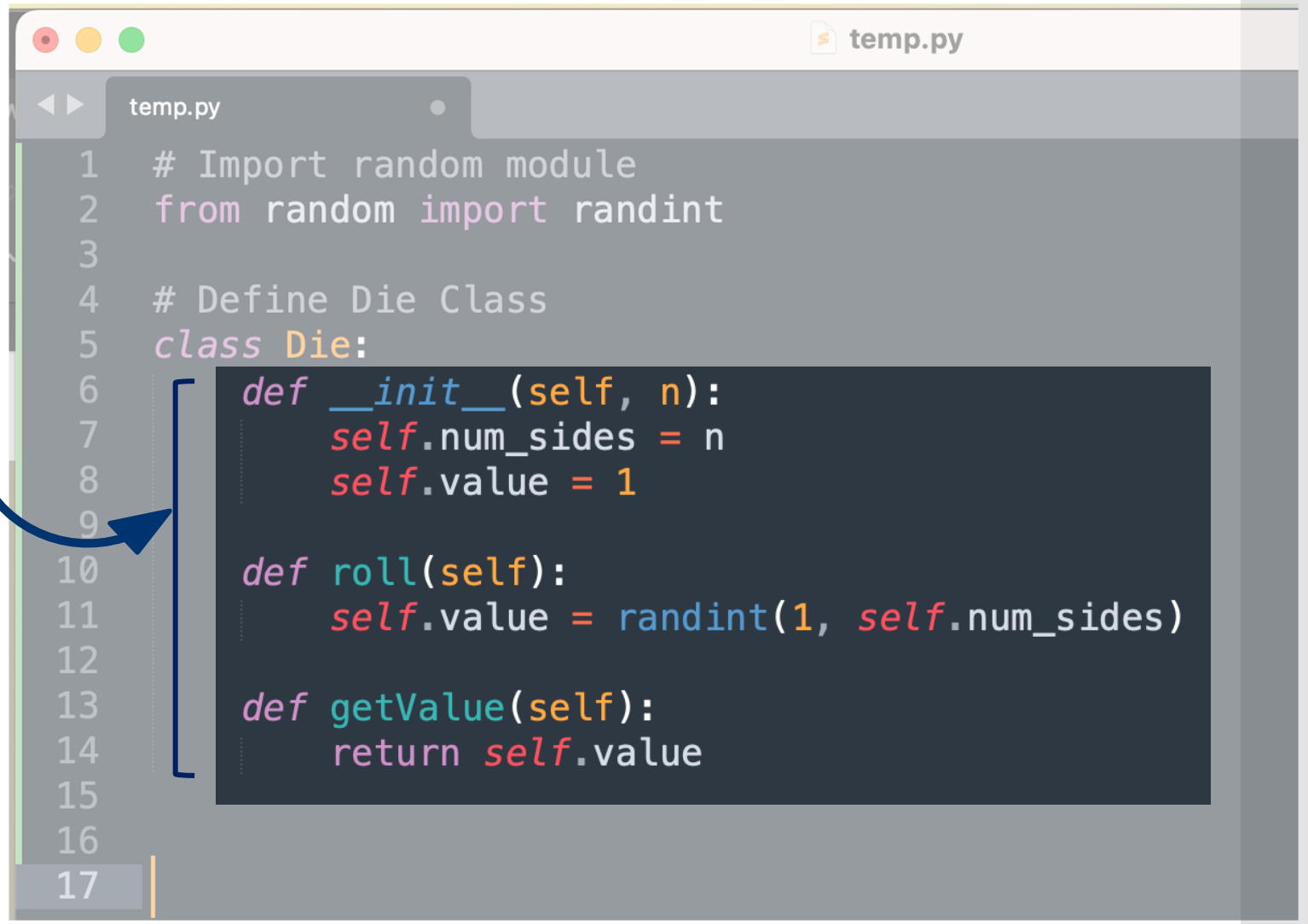
python constructors are always called **__init__**



attribute values get **initialized** here

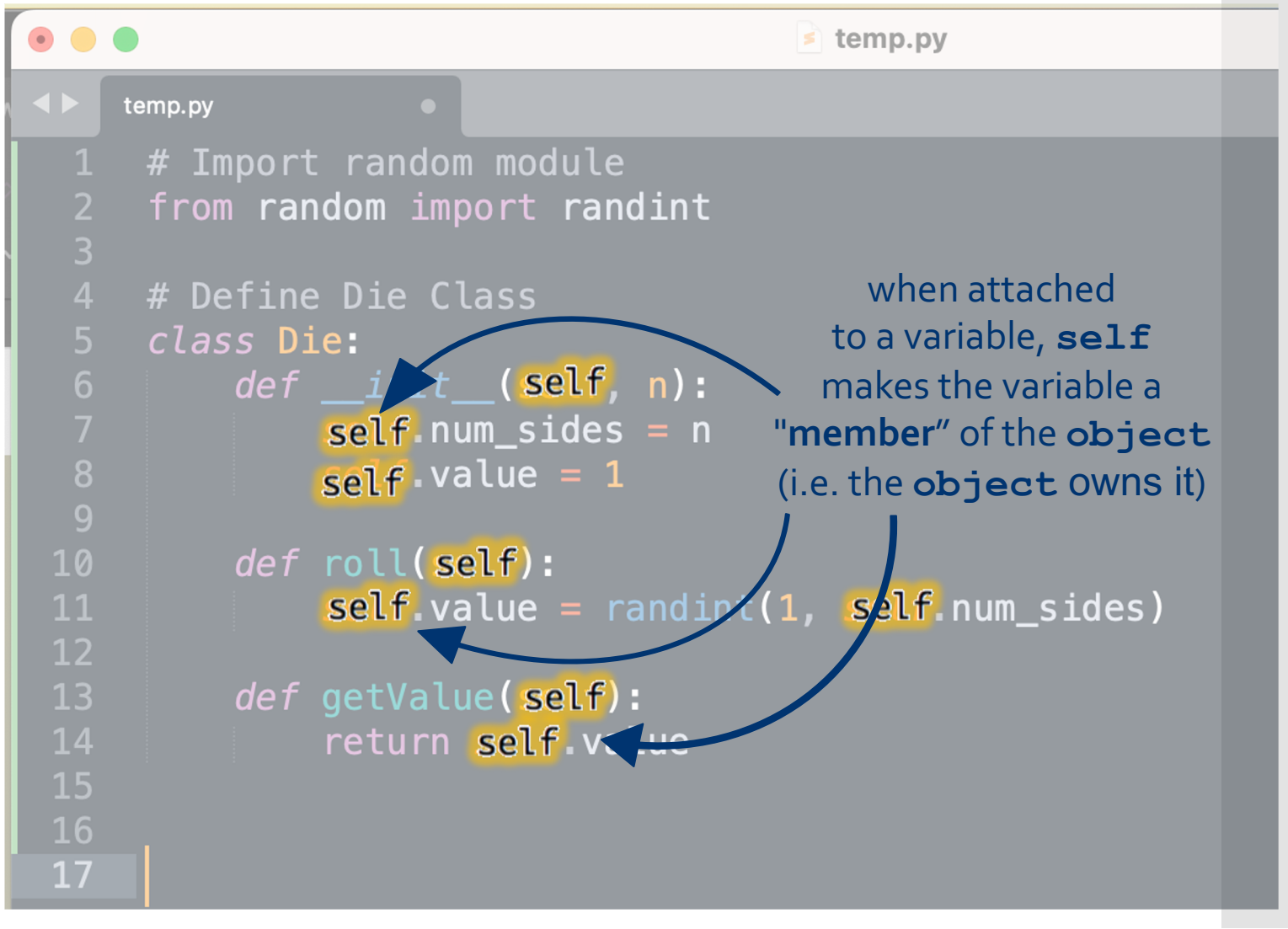
methods are **def**ined inside the **class**

nicely indented

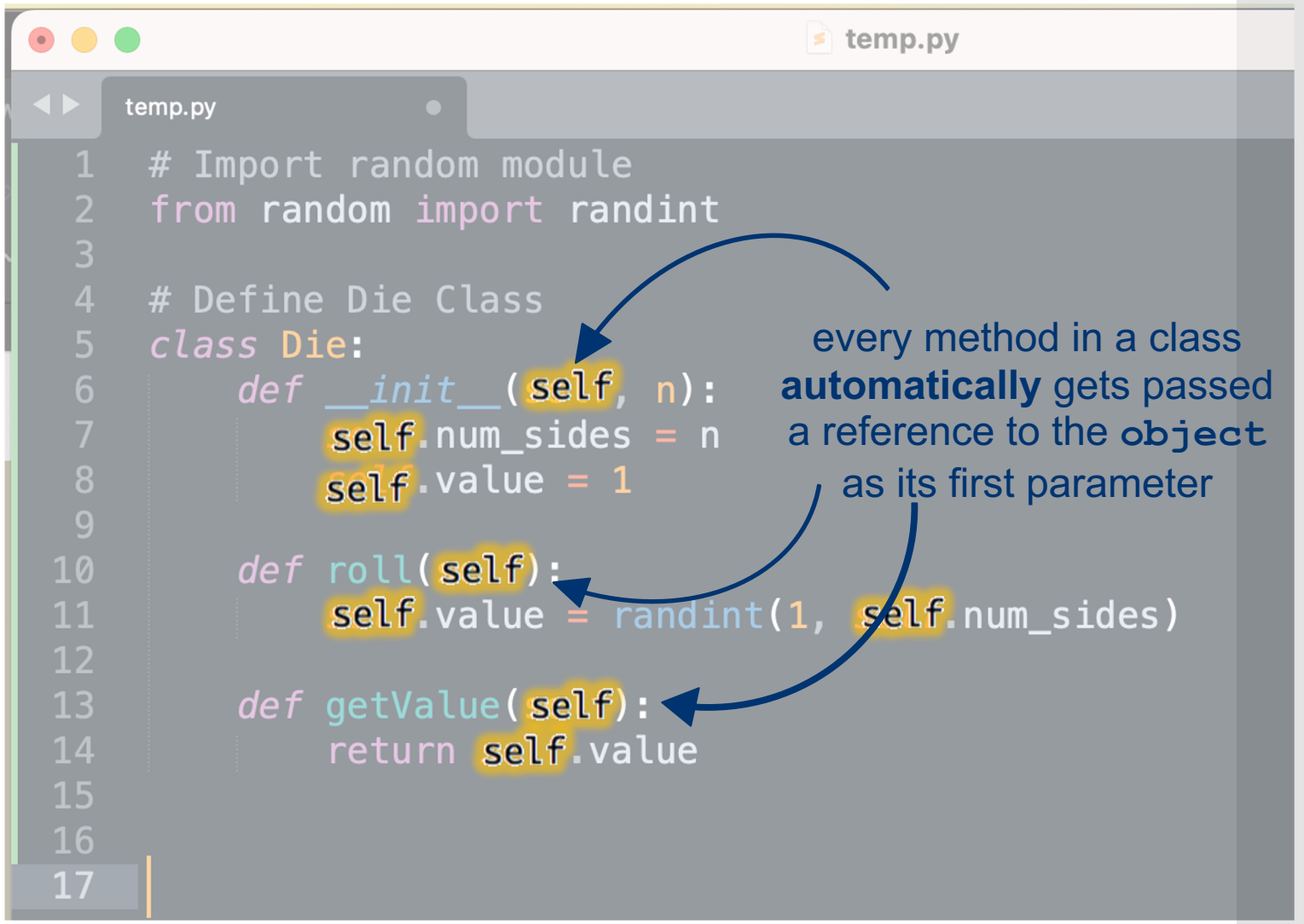


Question: what's with all the `self`s?

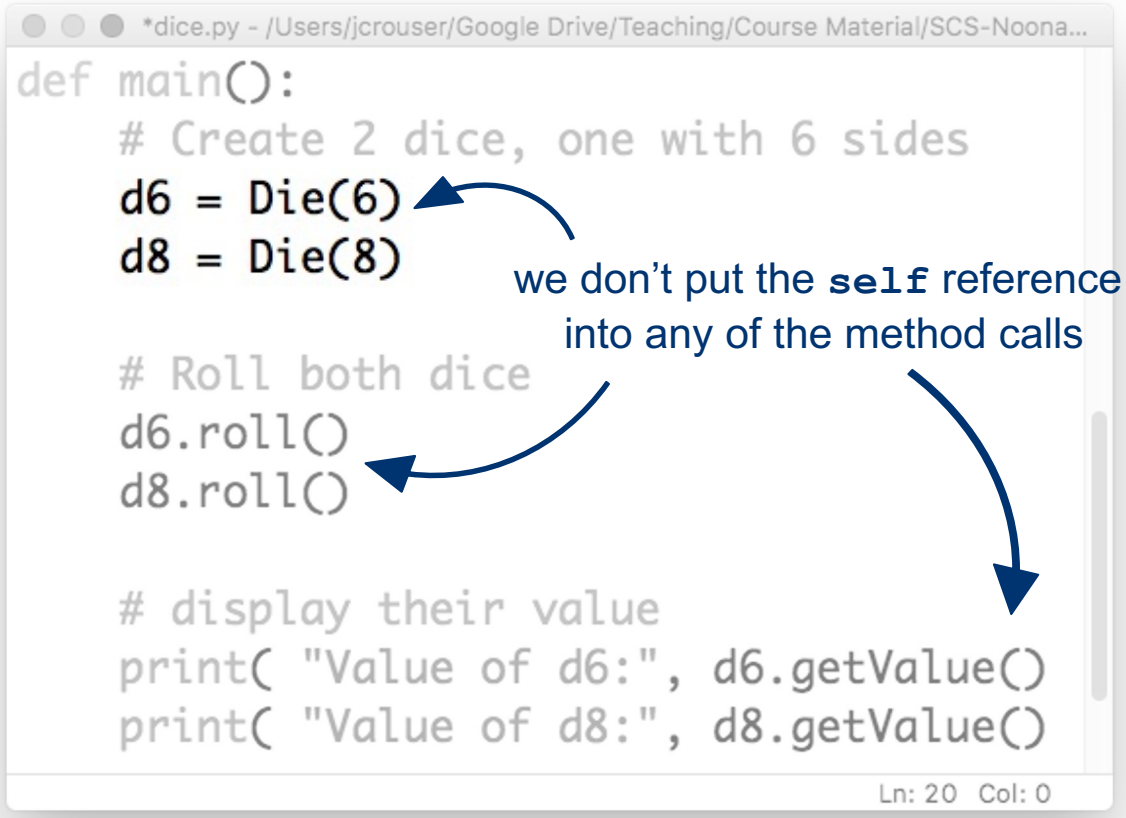
Question: what's with all the `self`s?



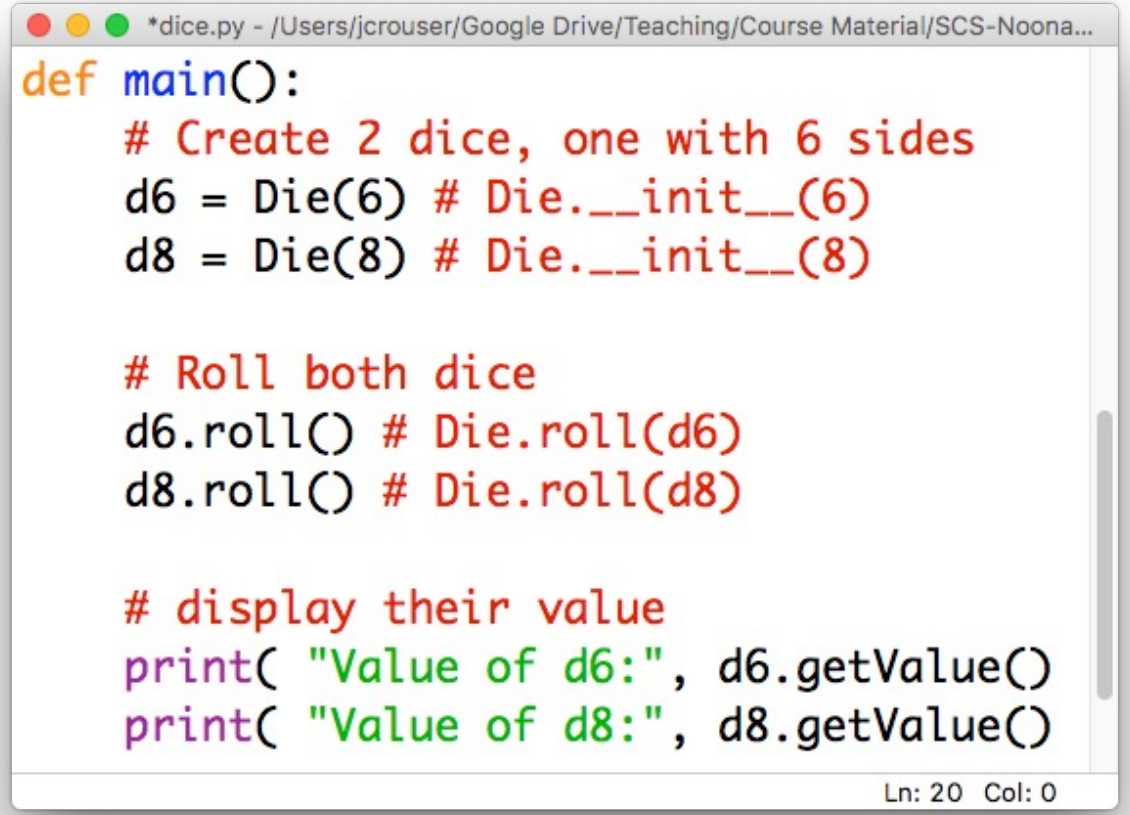
Question: what's with all the `self`s?



Again, this happens **automatically**



But the effect is **really cool**



15-minute exercise: Classy Playlist

open “playlist” on repl.it and fill in the missing code to create a playlist class

What did you come up with?