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

Music Library

Objective:

- We want to write the code to manage a music library
- The music library is made up of many songs
- Each song has a title, artist, and album, and can be played

One way we could do this is by representing the library with a list and songs with dictionaries.

RECAP: using **lists** and **dictionaries**

What does this function do?

```
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

What does this function do?

```
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"]))
```

RECAP: lists and dictionaries

this feels
a little funny…

```
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`

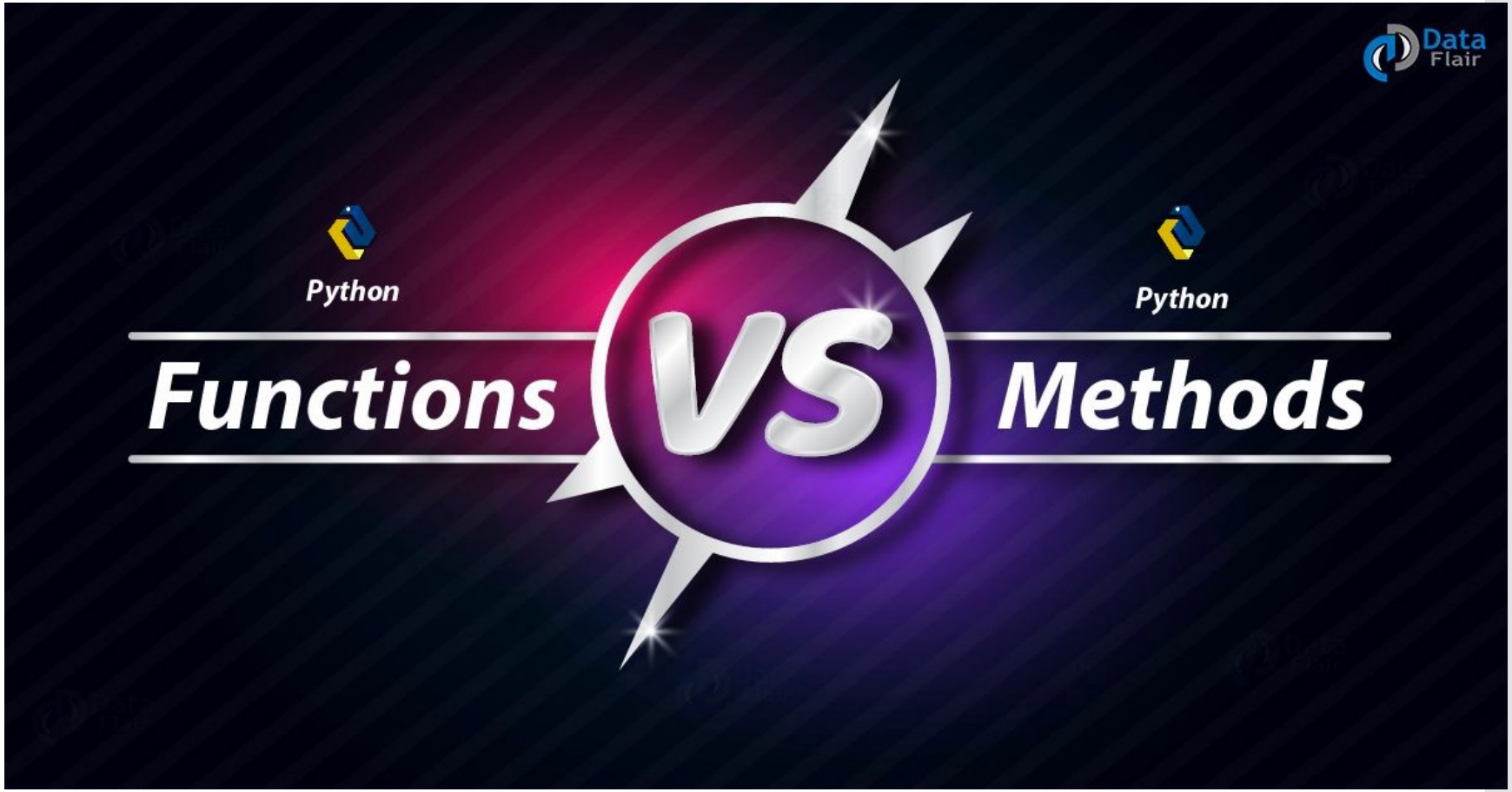
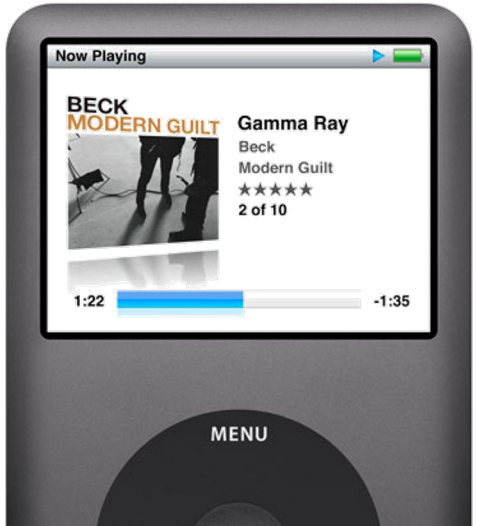


Image courtesy: https://data-flair.training/blogs/python-method-vs-function/

Back to the `music library`: 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 using the class 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…

*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

python doesn't know how to do any of this

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()`

*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

Blueprint for a `Die`

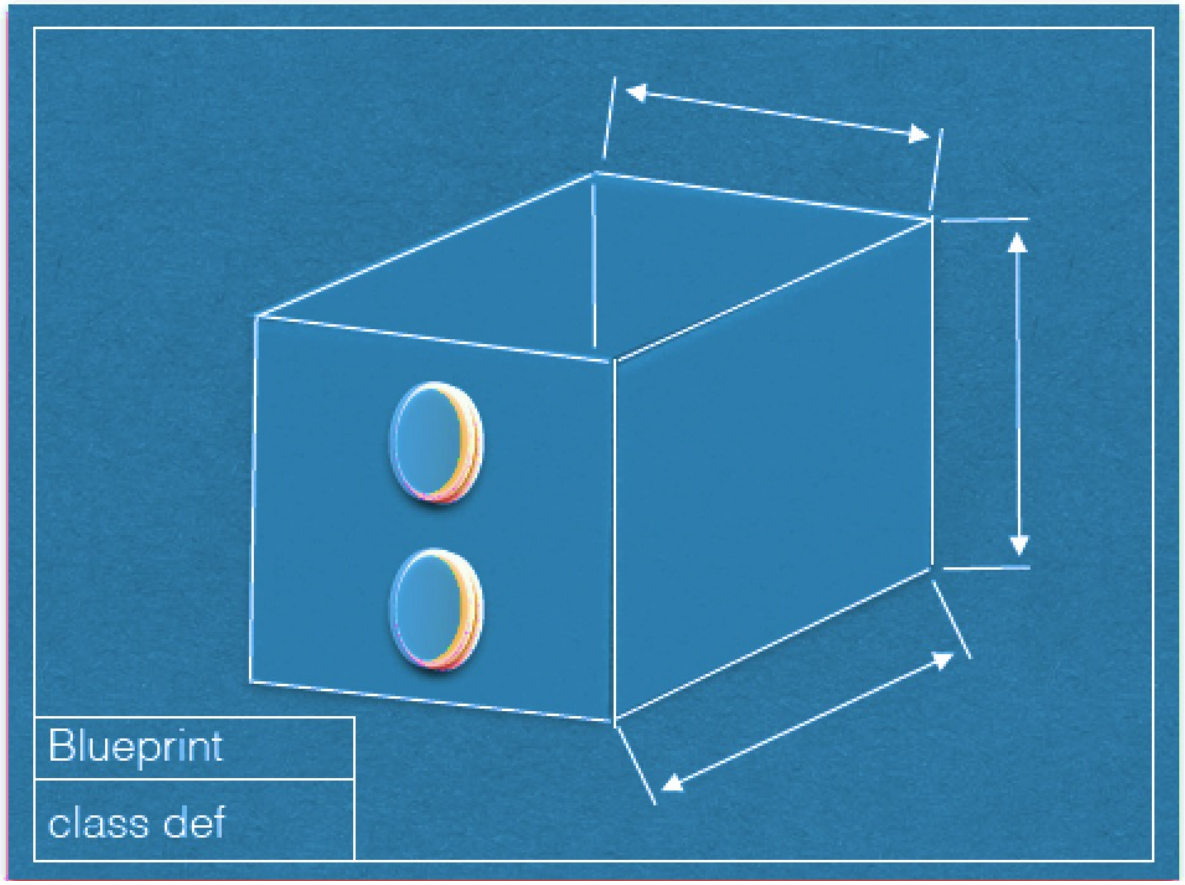


Image courtesy Dominique Thiebaut, Smith College

Blueprint for a `Die`: `attributes`

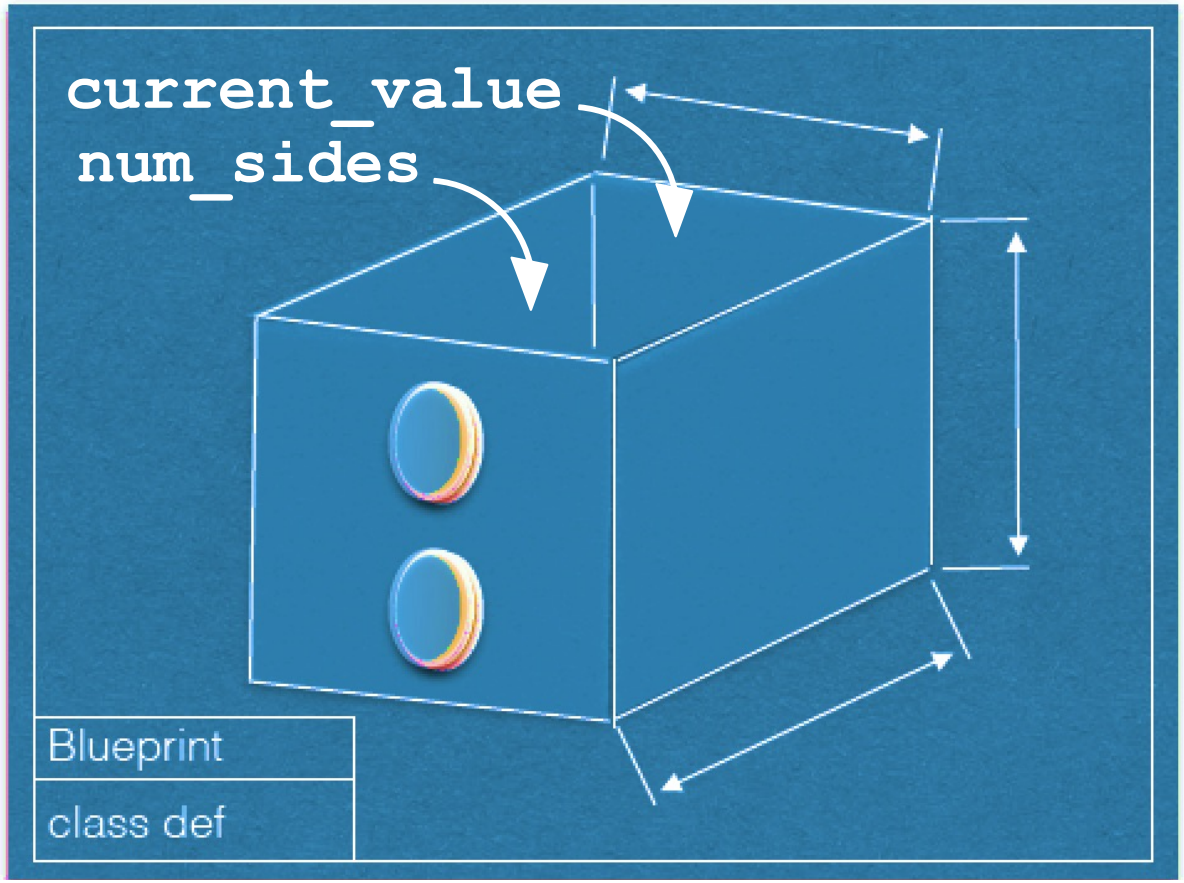


Image courtesy Dominique Thiebaut, Smith College

Blueprint for a `Die`: `methods`

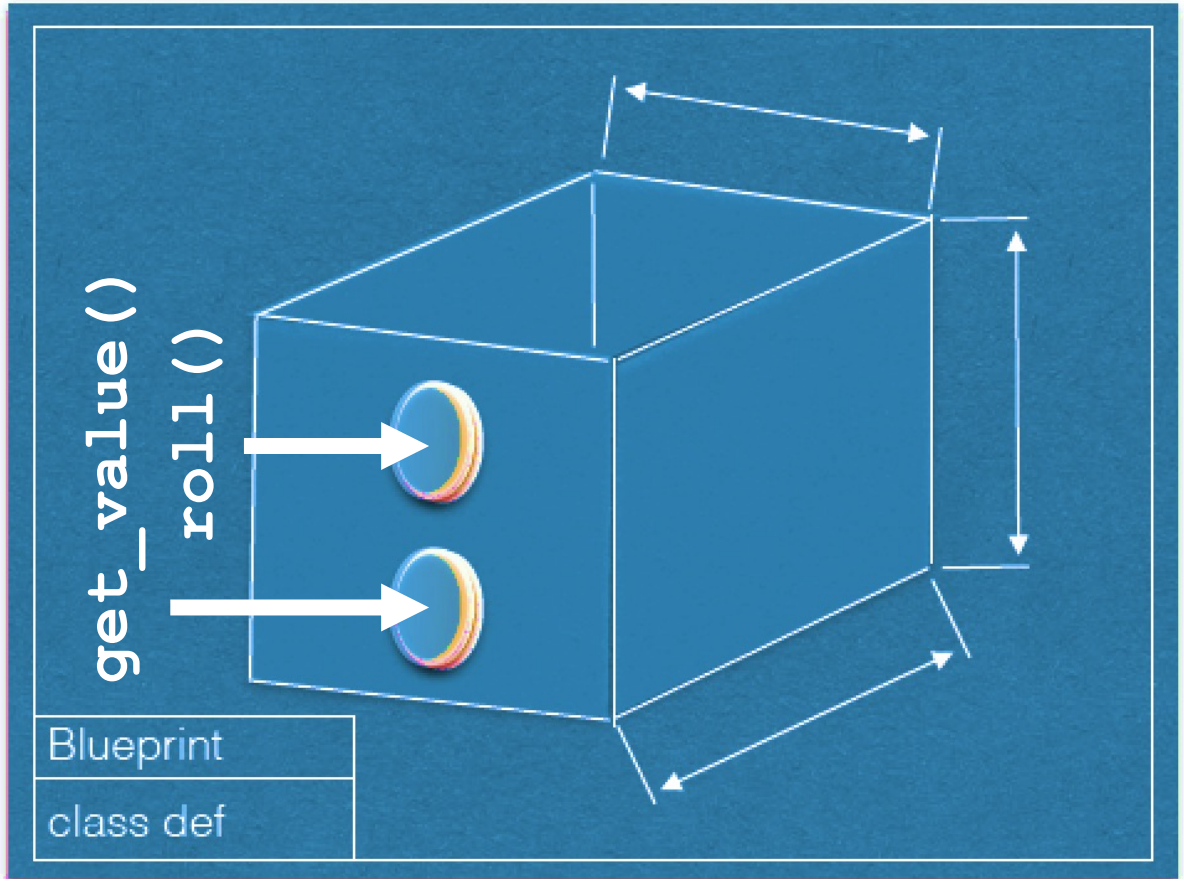
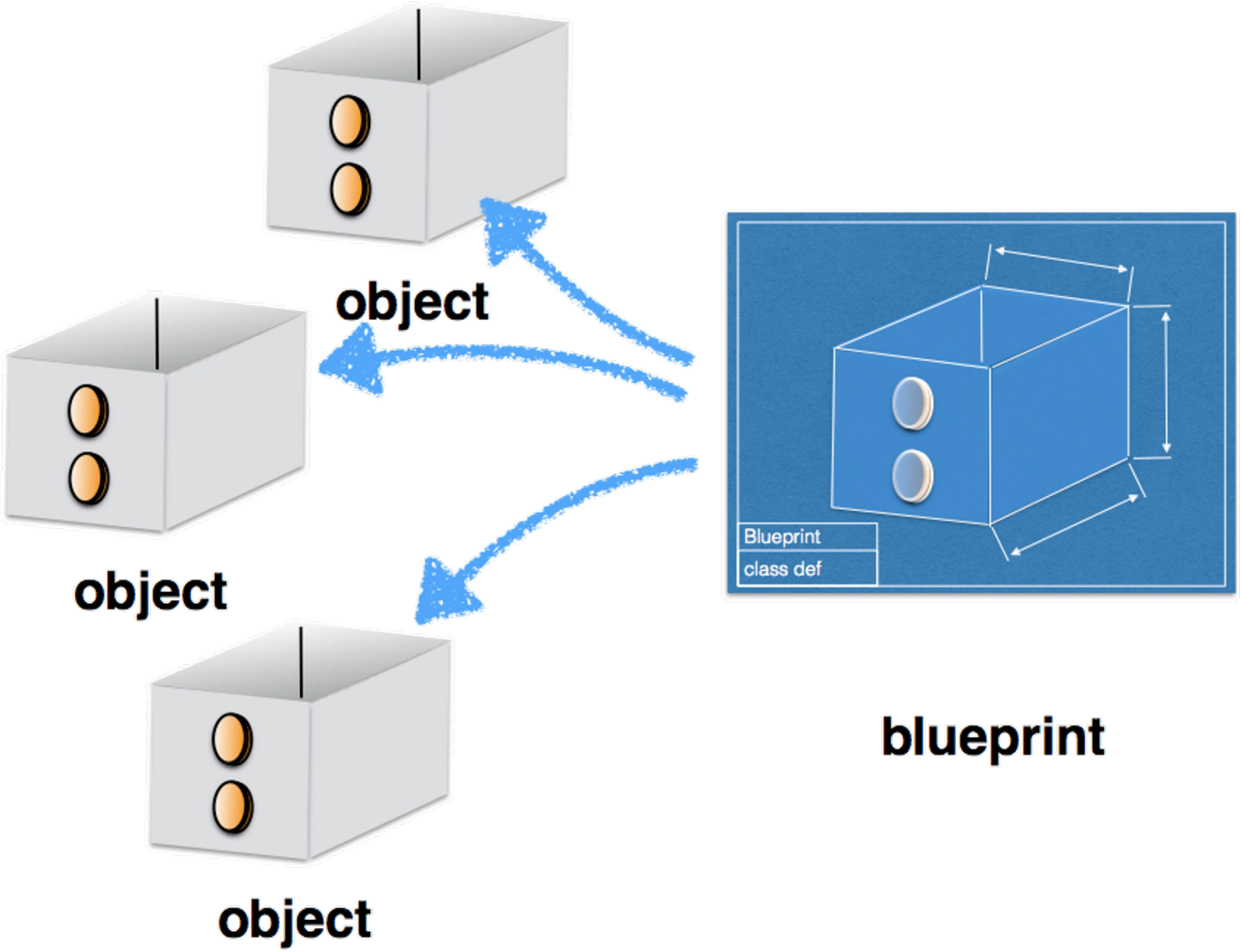


Image courtesy Dominique Thiebaut, Smith College

Given a blueprint, **replication** is easy



Coding the `Die class`

temp.py

```python
# Import random module
from random import randint

# Define Die Class
class Die:
    def __init__(self, n):
        self.num_sides = n
        self.value = 1

    def roll(self):
        self.value = randint(1, self.num_sides)

    def getValue(self):
        return self.value


```

classes are defined using **class**

convention: class names start with a capital letter

temp.py

```python
# Import random module
from random import randint

# Define Die Class
class Die:
    def __init__(self, n):
        self.num_sides = n
        self.value = 1

    def roll(self):
        self.value = randint(1, self.num_sides)

    def getValue(self):
        return self.value
```

All `classes` need a `constructor`

python constructors are always called **__init__**

```python
# Import random module
from random import randint

# Define Die Class
class Die:
    def __init__(self, n):
        self.num_sides = n
        self.value = 1

    def roll(self):
        self.value = randint(1, self.num_sides)

    def getValue(self):
        return self.value
```

attribute values get **initialized** here

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

nicely indented

temp.py

```python
# Import random module
from random import randint

# Define Die Class
class Die:
    def __init__(self, n):
        self.num_sides = n
        self.value = 1

    def roll(self):
        self.value = randint(1, self.num_sides)

    def getValue(self):
        return self.value
```

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

temp.py

```python
# Import random module
from random import randint

# Define Die Class
class Die:
    def __init__(self, n):
        self.num_sides = n
        self.value = 1

    def roll(self):
        self.value = randint(1, self.num_sides)

    def getValue(self):
        return self.value
```

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

temp.py

```
# Import random module
from random import randint

# Define Die Class
class Die:
    def __init__(self, n):
        self.num_sides = n
        self.value = 1

    def roll(self):
        self.value = randint(1, self.num_sides)

    def getValue(self):
        return self.value
```

when attached to a variable, **`self`** makes the variable a "**member**" of the **`object`** (i.e. the **`object`** owns it)

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

temp.py

```python
# Import random module
from random import randint

# Define Die Class
class Die:
    def __init__(self, n):
        self.num_sides = n
        self.value = 1

    def roll(self):
        self.value = randint(1, self.num_sides)

    def getValue(self):
        return self.value
```

every method in a class **automatically** gets passed a reference to the `object` as its first parameter

Again, this happens **automatically**

*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

we don't put the **`self`** reference into any of the method calls

But the effect is **really cool**

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

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

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

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

Ln: 20 Col: 0

15-minute exercise: Music Library

Suppose we want to use our music library to make playlists.

Define Playlist class. Make sure it has methods to ADD, REMOVE, PRINT, and SEARCH songs

What did you come up with?