DISC 08

OOP

OOP TERMINOLOGY

```python
class Pet:
    happy = True
    def __init__(self, name):
        self.name = name


class Puppy(Pet):
    good_boy = True
    def bark(self):
        if Pet.happy:
            print("woof")
    def break_lamp(self):
        self.good_boy = False
        happy = False
```

OOP TERMINOLOGY

```python
class Pet:
    happy = True
    def __init__(self, name):
        self.name = name
```

```python
class Puppy(Pet):
    good_boy = True
    def bark(self):
        if Pet.happy:
            print("woof")
    def break_lamp(self):
        self.good_boy = False
        happy = False
```

OOP TERMINOLOGY

```
class Pet:
    happy = True
    def __init__(self, name):
        self.name = name
```

class

class attributes

```
class Puppy(Pet):
    good_boy = True
    def bark(self):
        if Pet.happy:
            print("woof")
    def break_lamp(self):
        self.good_boy = False
        happy = False
```

class attributes

class

OOP TERMINOLOGY

```python
class Pet:
    happy = True
    def __init__(self, name):
        self.name = name
```

class

class attributes

instance attributes

```python
class Puppy(Pet):
    good_boy = True
    def bark(self):
        if Pet.happy:
            print("woof")
    def break_lamp(self):
        self.good_boy = False
        happy = False
```

class attributes

class

OOP TERMINOLOGY

```
class Pet:
    happy = True
    def __init__(self, name):
        self.name = name


class Puppy(Pet):
    good_boy = True
    def bark(self):
        if Pet.happy:
            print("woof")
    def break_lamp(self):
        self.good_boy = False
        happy = False
```

class

class attributes

instance attributes

class attributes

methods

class

OOP TERMINOLOGY

```
                                    brian = Puppy("brain")
                                    marvin = Puppy("marv")
class Pet:

    happy = True

    def __init__(self, name):

        self.name = name


class Puppy(Pet):

    good_boy = True

    def bark(self):

        if Pet.happy:

            print("woof")

    def break_lamp(self):

        self.good_boy = False

        happy = False
```

OOP TERMINOLOGY

```python
brian = Puppy("brain")
marvin = Puppy("marv")
```

```python
class Pet:

    happy = True

    def __init__(self, name):
        self.name = name


class Puppy(Pet):

    good_boy = True

    def bark(self):
        if Pet.happy:
            print("woof")

    def break_lamp(self):
        self.good_boy = False
        happy = False
```

brian

OOP TERMINOLOGY

```python
brian = Puppy("brain")
marvin = Puppy("marv")

class Pet:

    happy = True

    def __init__(self, name):
        self.name = name


class Puppy(Pet):

    good_boy = True

    def bark(self):
        if Pet.happy:
            print("woof")

    def break_lamp(self):
        self.good_boy = False
        happy = False
```

brian

no __init__ method defined here
check parent for __init__

OOP TERMINOLOGY

```
brian = Puppy("brain")
marvin = Puppy("marv")
```

```
class Pet:

    happy = True

    def __init__(self, name):
        self.name = name


class Puppy(Pet):

    good_boy = True

    def bark(self):

        if Pet.happy:

            print("woof")

    def break_lamp(self):

        self.good_boy = False

        happy = False
```

brian

name: "brain"

OOP TERMINOLOGY

```python
brian = Puppy("brain")
marvin = Puppy("marv")
```

```python
class Pet:
    happy = True
    def __init__(self, name):
        self.name = name


class Puppy(Pet):
    good_boy = True
    def bark(self):
        if Pet.happy:
            print("woof")
    def break_lamp(self):
        self.good_boy = False
        happy = False
```

brian

name: "brain"

marvin

OOP TERMINOLOGY

```python
brian = Puppy("brain")
marvin = Puppy("marv")

class Pet:

    happy = True

    def __init__(self, name):
        self.name = name


class Puppy(Pet):

    good_boy = True

    def bark(self):
        if Pet.happy:
            print("woof")

    def break_lamp(self):
        self.good_boy = False

        happy = False
```

brian

name: "brain"

no __init__ method defined here
check parent for __init__

marvin

OOP TERMINOLOGY

```
brian = Puppy("brain")
marvin = Puppy("marv")

class Pet:

    happy = True
    def __init__(self, name):
        self.name = name


class Puppy(Pet):
    good_boy = True
    def bark(self):
        if Pet.happy:
            print("woof")
    def break_lamp(self):
        self.good_boy = False
        happy = False
```
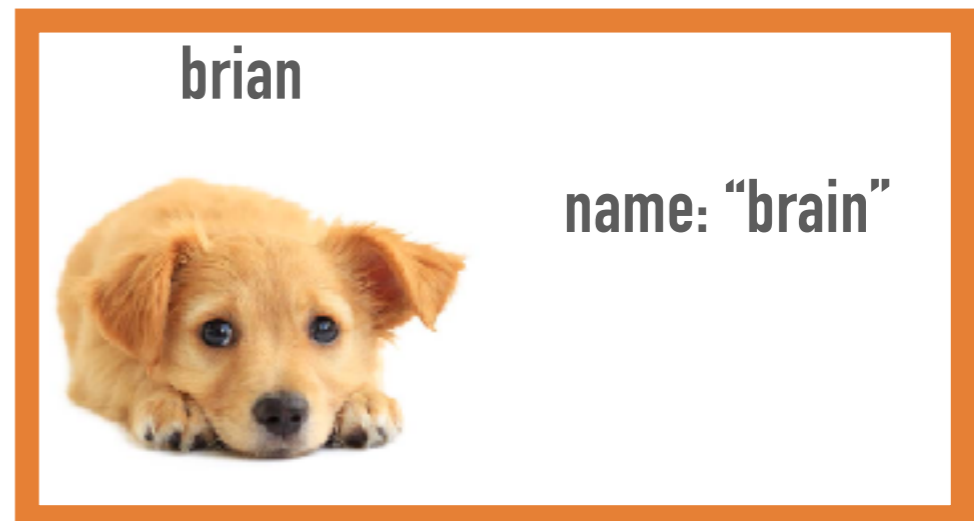
brian

name: "brain"

marvin

name: "marv"

OOP TERMINOLOGY

```python
brian = Puppy("brain")
marvin = Puppy("marv")
```

```python
class Pet:

    happy = True

    def __init__(self, name):

        self.name = name


class Puppy(Pet):

    good_boy = True

    def bark(self):

        if Pet.happy:

            print("woof")

    def break_lamp(self):

        self.good_boy = False

        happy = False
```

objects

brian
name: "brian"

marvin
name: "marv"

HOW TO KEEP TRACK OF CLASSES AND OBJECTS

➤ Creating classes:

  ➤ Go through the class definitions and write class names on the LHS

  ➤ Write class attributes under the corresponding name

  ➤ Write class methods under the corresponding name

➤ Creating objects:

  ➤ Write name of object on the RHS

  ➤ Go to the __init__ of the object's class and write the object's instance attributes under the object name on the RHS

1.1

```python
class Instructor:
    degree = "PhD (Basketball)" # this is a class attribute
    def __init__(self, name):
        self.name = name # this is an instance attribute

    def lecture(self, topic):
        print("Today we're learning about " + topic)

lebron = Instructor("Professor LeBron")
class Student:
    instructor = lebron

    def __init__(self, name, ta):
        self.name = name
        self.understanding = 0
        ta.add_student(self)

    def attend_lecture(self, topic):
        Student.instructor.lecture(topic)
        print(Student.instructor.name + " is awesome!")
        self.understanding += 1

    def visit_office_hours(self, staff):
        staff.assist(self)
        print("Thanks, " + staff.name)

class TeachingAssistant:
    def __init__(self, name):
        self.name = name
        self.students = {}

    def add_student(self, student):
        self.students[student.name] = student

    def assist(self, student):
        student.understanding += 1
```

1.1

Instructor

    degree: "PhD (Basketball)"

    __init__

    lecture

```python
class Instructor:
    degree = "PhD (Basketball)" # this is a class attribute
    def __init__(self, name):
        self.name = name # this is an instance attribute

    def lecture(self, topic):
        print("Today we're learning about " + topic)

lebron = Instructor("Professor LeBron")
class Student:
    instructor = lebron

    def __init__(self, name, ta):
        self.name = name
        self.understanding = 0
        ta.add_student(self)

    def attend_lecture(self, topic):
        Student.instructor.lecture(topic)
        print(Student.instructor.name + " is awesome!")
        self.understanding += 1

    def visit_office_hours(self, staff):
        staff.assist(self)
        print("Thanks, " + staff.name)

class TeachingAssistant:
    def __init__(self, name):
        self.name = name
        self.students = {}

    def add_student(self, student):
        self.students[student.name] = student

    def assist(self, student):
        student.understanding += 1
```

1.1

Instructor

    degree: "PhD (Basketball)"

    ___init___

    lecture

Student

    instructor:

    ___init___

    attend_lecture

    visit_office_hours

```python
class Instructor:
    degree = "PhD (Basketball)" # this is a class attribute
    def __init__(self, name):
        self.name = name # this is an instance attribute

    def lecture(self, topic):
        print("Today we're learning about " + topic)

lebron = Instructor("Professor LeBron")
class Student:
    instructor = lebron

    def __init__(self, name, ta):
        self.name = name
        self.understanding = 0
        ta.add_student(self)

    def attend_lecture(self, topic):
        Student.instructor.lecture(topic)
        print(Student.instructor.name + " is awesome!")
        self.understanding += 1

    def visit_office_hours(self, staff):
        staff.assist(self)
        print("Thanks, " + staff.name)

class TeachingAssistant:
    def __init__(self, name):
        self.name = name
        self.students = {}

    def add_student(self, student):
        self.students[student.name] = student

    def assist(self, student):
        student.understanding += 1
```

1.1

Instructor
   degree: "PhD (Basketball)"
   ___init___
   lecture

Student
   instructor:
      ___init___
   attend_lecture
   visit_office_hours

TeachingAssistant
   ___init___
   add_student
   assist

```python
class Instructor:
    degree = "PhD (Basketball)" # this is a class attribute
    def __init__(self, name):
        self.name = name # this is an instance attribute

    def lecture(self, topic):
        print("Today we're learning about " + topic)

lebron = Instructor("Professor LeBron")
class Student:
    instructor = lebron

    def __init__(self, name, ta):
        self.name = name
        self.understanding = 0
        ta.add_student(self)

    def attend_lecture(self, topic):
        Student.instructor.lecture(topic)
        print(Student.instructor.name + " is awesome!")
        self.understanding += 1

    def visit_office_hours(self, staff):
        staff.assist(self)
        print("Thanks, " + staff.name)

class TeachingAssistant:
    def __init__(self, name):
        self.name = name
        self.students = {}

    def add_student(self, student):
        self.students[student.name] = student

    def assist(self, student):
        student.understanding += 1
```

We missed something!

1.1

Instructor
 degree: "PhD (Basketball)"
 __init__
 lecture

Student
 instructor: lebron
 __init__
 attend_lecture
 visit_office_hours

TeachingAssistant
 __init__
 add_student
 assist

```python
class Instructor:
    degree = "PhD (Basketball)" # this is a class attribute
    def __init__(self, name):
        self.name = name # this is an instance attribute

    def lecture(self, topic):
        print("Today we're learning about " + topic)

lebron = Instructor("Professor LeBron")
class Student:
    instructor = lebron

    def __init__(self, name, ta):
        self.name = name
        self.understanding = 0
        ta.add_student(self)

    def attend_lecture(self, topic):
        Student.instructor.lecture(topic)
        print(Student.instructor.name + " is awesome!")
        self.understanding += 1

    def visit_office_hours(self, staff):
        staff.assist(self)
        print("Thanks, " + staff.name)

class TeachingAssistant:
    def __init__(self, name):
        self.name = name
        self.students = {}

    def add_student(self, student):
        self.students[student.name] = student

    def assist(self, student):
        student.understanding += 1
```

1.1

```
>>> lebron = Instructor("Professor LeBron")
```

1. Create classes

2. Create objects

Classes

OBJECTS

Instructor
- degree: "PhD (Basketball)"
- __init__
- lecture

Student
- instructor: lebron
- __init__
- attend_lecture
- visit_office_hours

TeachingAssistant
- __init__
- add_student
- assist

1.1

```
>>> lebron = Instructor("Professor LeBron")
```

1. Create classes

2. Create objects

Classes

Instructor
    degree: "PhD (Basketball)"
    __init__
    lecture

Student
    instructor: lebron
    __init__
    attend_lecture
    visit_office_hours

TeachingAssistant
    __init__
    add_student
    assist

OBJECTS

lebron
    name: "Professor LeBron"

we call __init__ from Instructor, passing in "Professor LeBron" as name

1.1

The current line will be displayed up here

```
>>> lebron = Instructor("Professor LeBron")
```

Classes

OBJECTS

Instructor
   degree: "PhD (Basketball)"
   __init__
   lecture

lebron
   name: "Professor LeBron"

Student
   instructor: ~~lebron~~
   __init__
   attend_lecture
   visit_office_hours

Inside the Student class, we set instructor to lebron

we call __init__ from Instructor, passing in "Professor LeBron" as name

TeachingAssistant
   __init__
   add_student
   assist

NOW THAT WE SET UP OUR CLASSES AND OBJECTS, WE CAN START EXECUTING THE CODE

1.1

```
>>> steph = TeachingAssistant("Steph")
```

1. Create classes

2. Create objects

Classes

Instructor
   degree: "PhD (Basketball)"
   __init__
   lecture

Student
   instructor:
   __init__
   attend_lecture
   visit_office_hours

TeachingAssistant
   __init__
   add_student
   assist

OBJECTS

lebron
   name: "Professor LeBron"

steph
   name: "Steph"
   students: {}

Go into ___init___ in the
TeachingAssistant class.
We set self.name to name and create
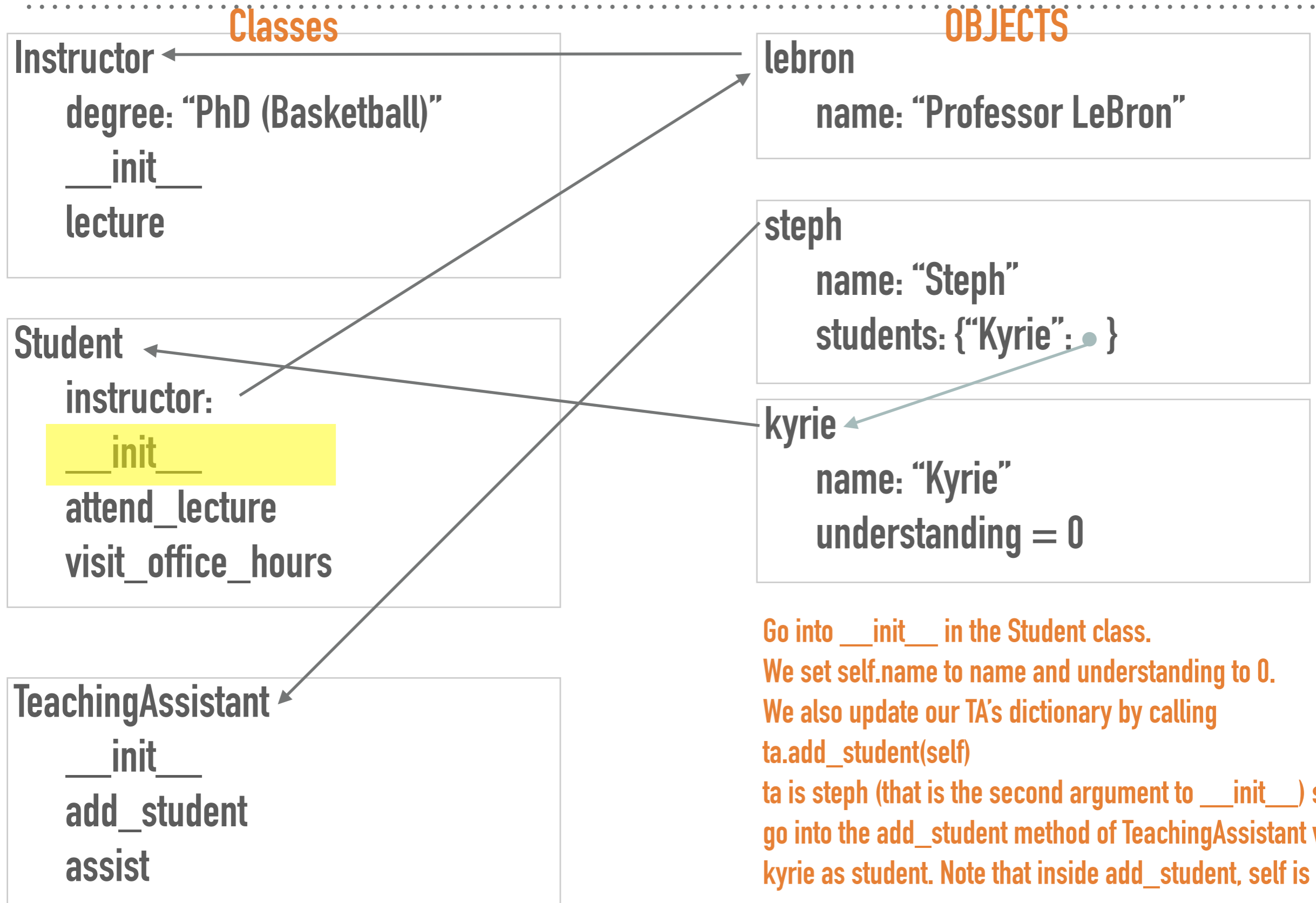a dictionary of students which is
empty at first

1.1

```
>>> kyrie = Student("Kyrie", steph)
```

1. Create classes

2. Create objects

Classes

Instructor
degree: "PhD (Basketball)"
__init__
lecture

Student
instructor:
__init__
attend_lecture
visit_office_hours

TeachingAssistant
__init__
add_student
assist

OBJECTS

lebron
name: "Professor LeBron"

steph
name: "Steph"
students: {"Kyrie": • }

kyrie
name: "Kyrie"
understanding = 0

Go into __init__ in the Student class.
We set self.name to name and understanding to 0.
We also update our TA's dictionary by calling
ta.add_student(self)
ta is steph (that is the second argument to __init__) so we
go into the add_student method of TeachingAssistant with
kyrie as student. Note that inside add_student, self is steph

1.1

```
>>> kyrie.attend_lecture("defense")
```

1. Create classes

2. Create objects

Classes

Instructor
  degree: "PhD (Basketball)"
  __init__
  lecture

Student
  instructor:
  __init__
  ==attend_lecture==
  visit_office_hours

TeachingAssistant
  __init__
  add_student
  assist

OBJECTS

lebron
  name: "Professor LeBron"

steph
  name: "Steph"
  students: {"Kyrie" : • }

kyrie
  name: "Kyrie"
  understanding = 0

The first line in attend_lecture is:

```
Student.instructor.lecture(topic)
```

```
>>> kyrie.attend_lecture("defense")
```

1. Create classes

2. Create objects

Classes

OBJECTS

Instructor
  degree: "PhD (Basketball)"
  __init__
  lecture

lebron
  name: "Professor LeBron"

steph
  name: "Steph"
  students: {"Kyrie" : ● }

Student
  instructor:
  __init__
  <mark>attend_lecture</mark>
  visit_office_hours

kyrie
  name: "Kyrie"
  understanding = 0

TeachingAssistant
  __init__
  add_student
  assist

The first line in attend_lecture is:

```
Student.instructor.lecture(topic)
```

1. Go to the Student class and find the class attribute instructor

1.1
```
>>> kyrie.attend_lecture("defense")
```

1. Create classes

2. Create objects

Classes

Instructor
    degree: "PhD (Basketball)"
    __init__
    lecture

Student
    instructor:
    __init__
    attend_lecture
    visit_office_hours

TeachingAssistant
    __init__
    add_student
    assist

OBJECTS

lebron
    name: "Professor LeBron"

steph
    name: "Steph"
    students: {"Kyrie" : ● }

kyrie
    name: "Kyrie"
    understanding = 0

The first line in attend_lecture is:

```
Student.instructor.lecture(topic)
```

1. Go to the Student class and find the class attribute instructor

```
lebron.lecture(topic)
```

1.1

```
>>> kyrie.attend_lecture("defense")
```

1. Create classes

2. Create objects

Classes

Instructor
    degree: "PhD (Basketball)"
    __init__
    lecture

Student
    instructor:
    __init__
    attend_lecture
    visit_office_hours

TeachingAssistant
    __init__
    add_student
    assist

OBJECTS

lebron
    name: "Professor LeBron"

steph
    name: "Steph"
    students: {"Kyrie" : • }

kyrie
    name: "Kyrie"
    understanding = 0

The first line in attend_lecture is:

```
Student.instructor.lecture(topic)
```

1. Go to the Student class and find the class attribute instructor

```
lebron.lecture(topic)
```

2. Call the method lecture and pass in topic ("defense") as the argument

1.1

```
>>> kyrie.attend_lecture("defense")
```
"Today we're learning about defense"

1. Create classes

2. Create objects

Classes

Instructor
  degree: "PhD (Basketball)"
  __init__
  lecture

Student
  instructor:
  __init__
  attend_lecture
  visit_office_hours

TeachingAssistant
  __init__
  add_student
  assist

OBJECTS

lebron
  name: "Professor LeBron"

steph
  name: "Steph"
  students: {"Kyrie" : ● }

kyrie
  name: "Kyrie"
  understanding = 0

The first line in attend_lecture is:

```
Student.instructor.lecture(topic)
```

1. Go to the Student class and find the class attribute instructor

```
lebron.lecture(topic)
```

2. Call the method lecture and pass in topic ("defense") as the argument

This results in printing "Today we're learning about defense"

1.1

Now we are ready to walk through the lines on page 3
The current line will be displayed up here

```
>>> kyrie.attend_lecture("defense")
```
Today we're learning about defense

1. Create classes

2. Create objects

Classes

Instructor

    degree: "PhD (Basketball)"

    __init__

    lecture

Student

    instructor:

    __init__

    attend_lecture

    visit_office_hours

TeachingAssistant

    __init__

    add_student

    assist

OBJECTS

lebron

    name: "Professor LeBron"

steph

    name: "Steph"

    students: {"Kyrie" : ● }

kyrie

    name: "Kyrie"

    understanding = 0

The second line in attend_lecture is:

```
print(Student.instructor.name + " is awesome!")
```

1.1

```
>>> kyrie.attend_lecture("defense")
```
Today we're learning about defense

1. Create classes

2. Create objects

Classes

OBJECTS

Instructor
degree: "PhD (Basketball)"
__init__
lecture

lebron
name: "Professor LeBron"

steph
name: "Steph"
students: {"Kyrie" : • }

Student
instructor:
__init__
attend_lecture
visit_office_hours

kyrie
name: "Kyrie"
understanding = 0

TeachingAssistant
__init__
add_student
assist

The second line in attend_lecture is:

```
print(Student.instructor.name + " is awesome!")
```

1. Go to the Student class and find the class attribute instructor

1.1

The current line will be displayed up here

```
>>> kyrie.attend_lecture("defense")
```
Today we're learning about defense

1. Create classes

2. Create objects

Classes

OBJECTS

Instructor
  degree: "PhD (Basketball)"
  __init__
  lecture

Student
  instructor:
  __init__
  attend_lecture
  visit_office_hours

TeachingAssistant
  __init__
  add_student
  assist

lebron
  name: "Professor LeBron"

steph
  name: "Steph"
  students: {"Kyrie" : • }

kyrie
  name: "Kyrie"
  understanding = 0

The second line in attend_lecture is:

```
print(Student.instructor.name + " is awesome!")
```

1. Go to the Student class and find the class attribute instructor

```
print(lebron.name + " is awesome!")
```

Now we are ready to walk through the lines on page 3
The current line will be displayed up here

```
>>> kyrie.attend_lecture("defense")
```
Today we're learning about defense

1. Create classes

2. Create objects

Classes

Instructor
  degree: "PhD (Basketball)"
  __init__
  lecture

Student
  <mark>instructor:</mark>
  __init__
  attend_lecture
  visit_office_hours

TeachingAssistant
  __init__
  add_student
  assist

OBJECTS

lebron
  name: "Professor LeBron"

steph
  name: "Steph"
  students: {"Kyrie" : ● }

kyrie
  name: "Kyrie"
  understanding = 0

The second line in attend_lecture is:

```
print(Student.instructor.name + " is awesome!")
```

1. Go to the Student class and find the class attribute instructor

```
print(lebron.name + " is awesome!")
```

2. lebron has the instance attribute name, and its value is "Professor LebBron"

1.1

```
>>> kyrie.attend_lecture("defense")
```

Today we're learning about defense
Professor LeBron is awesome!

1. Create classes

2. Create objects

Classes

Instructor
  degree: "PhD (Basketball)"
  __init__
  lecture

Student
  instructor:
  __init__
  attend_lecture
  visit_office_hours

TeachingAssistant
  __init__
  add_student
  assist

OBJECTS

lebron
  name: "Professor LeBron"

steph
  name: "Steph"
  students: {"Kyrie" : • }

kyrie
  name: "Kyrie"
  understanding = 0

The second line in attend_lecture is:

```
print(Student.instructor.name + " is awesome!")
```

1. Go to the Student class and find the class attribute instructor

```
print(lebron.name + " is awesome!")
```

2. lebron has the instance attribute name, and its value is "Professor LebBron"

```
print("Professor LeBron" + " is awesome!")
```

1. Create classes

2. Create objects

1.1
```
>>> kyrie.attend_lecture("defense")
```
Today we're learning about defense
Professor LeBron is awesome!

Classes

Instructor

   degree: "PhD (Basketball)"

   __init__

   lecture

Student

   instructor:

   __init__

   attend_lecture

   visit_office_hours

TeachingAssistant

   __init__

   add_student

   assist

OBJECTS

lebron

   name: "Professor LeBron"

steph

   name: "Steph"

   students: {"Kyrie" : ● }

kyrie

   name: "Kyrie"

   understanding = 0

The third line in attend_lecture is:

```
self.understanding += 1
```

1.1

```
>>> kyrie.attend_lecture("defense")
```

Today we're learning about defense
Professor LeBron is awesome!

1. Create classes

2. Create objects

Classes

Instructor
   degree: "PhD (Basketball)"
   __init__
   lecture

Student
   instructor:
   __init__
   attend_lecture
   visit_office_hours

TeachingAssistant
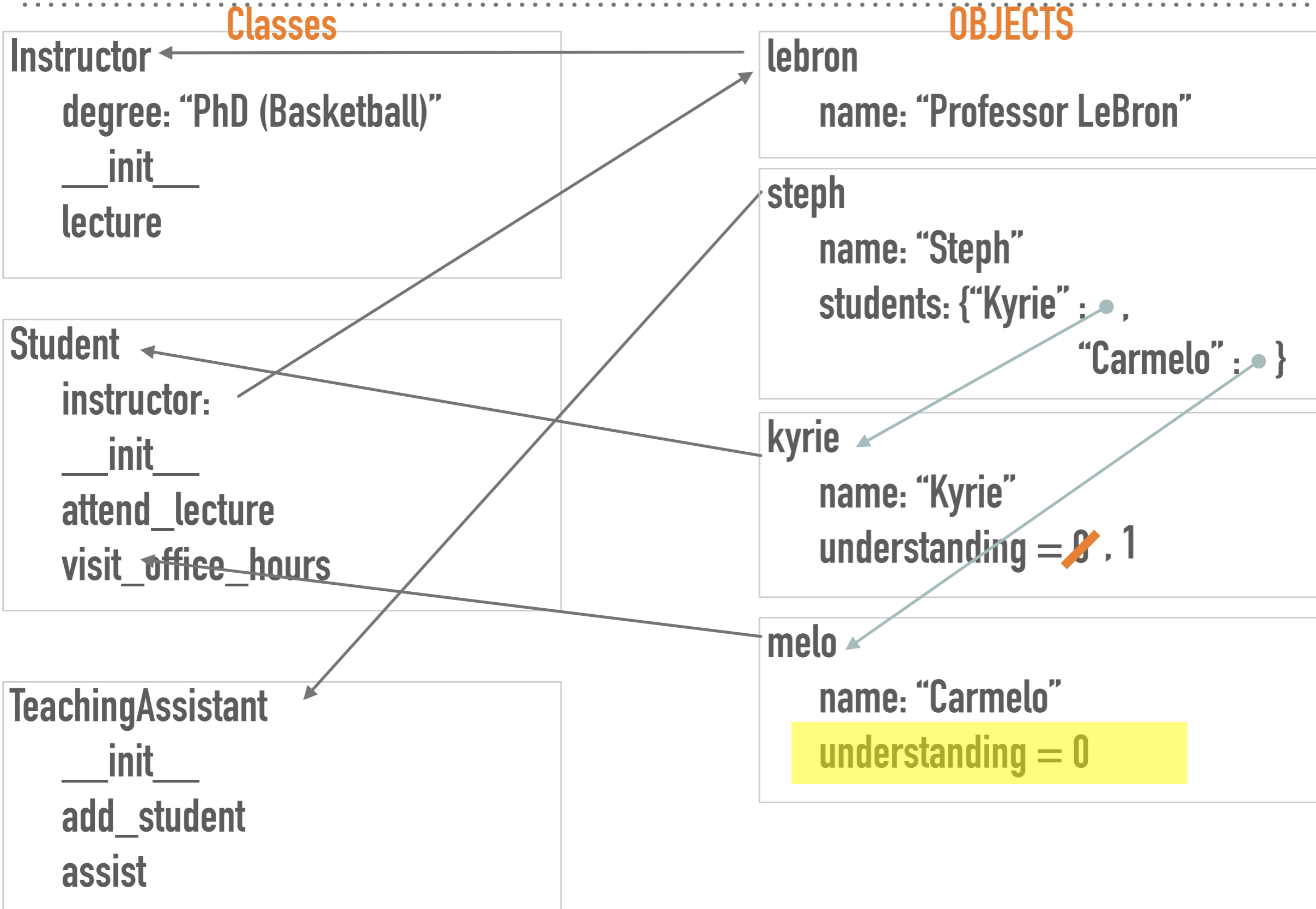   __init__
   add_student
   assist

OBJECTS

lebron
   name: "Professor LeBron"

steph
   name: "Steph"
   students: {"Kyrie" : ● }

kyrie
   name: "Kyrie"
   understanding = 0

The third line in attend_lecture is:

```
self.understanding += 1
```

1. self is kyrie because that is the object we passed in when we did kyrie.attend_lecture("defense")

1.1

```
>>> kyrie.attend_lecture("defense")
```

Today we're learning about defense
Professor LeBron is awesome!

1. Create classes

2. Create objects

Classes

Instructor

   degree: "PhD (Basketball)"

   __init__

   lecture

Student

   instructor:

   __init__

   attend_lecture

   visit_office_hours

TeachingAssistant

   __init__

   add_student

   assist

OBJECTS

lebron

   name: "Professor LeBron"

steph

   name: "Steph"

   students: {"Kyrie" : ● }

kyrie

   name: "Kyrie"

   understanding = ~~0~~ , 1

The third line in attend_lecture is:

```
self.understanding += 1
```

1. self is kyrie because that is the object we passed in when we did kyrie.attend_lecture("defense")

2. increment kyrie's understanding by 1

1.1

```
>>> melo.attend_lecture("championships!")
```

1. Create classes

2. Create objects

Classes

Instructor
  degree: "PhD (Basketball)"
  __init__
  lecture

Student
  instructor:
  __init__
  attend_lecture
  visit_office_hours

TeachingAssistant
  __init__
  add_student
  assist

OBJECTS

lebron
  name: "Professor LeBron"

steph
  name: "Steph"
  students: {"Kyrie" : ● ,
            "Carmelo" : ● }

kyrie
  name: "Kyrie"
  understanding = 0̶ . 1

melo
  name: "Carmelo"
  understanding = 0

1.1

The current line will be displayed up here

```
>>> melo.attend_lecture("championships!")
```
Today we're learning about championships
Professor LeBron is awesome!

1. Create classes

2. Create objects

Classes

Instructor
degree: "PhD (Basketball)"
__init__
lecture

Student
instructor:
__init__
attend_lecture
visit_office_hours

TeachingAssistant
__init__
add_student
assist

OBJECTS

lebron
name: "Professor LeBron"

steph
name: "Steph"
students: {"Kyrie" : •,
                    "Carmelo" : • }
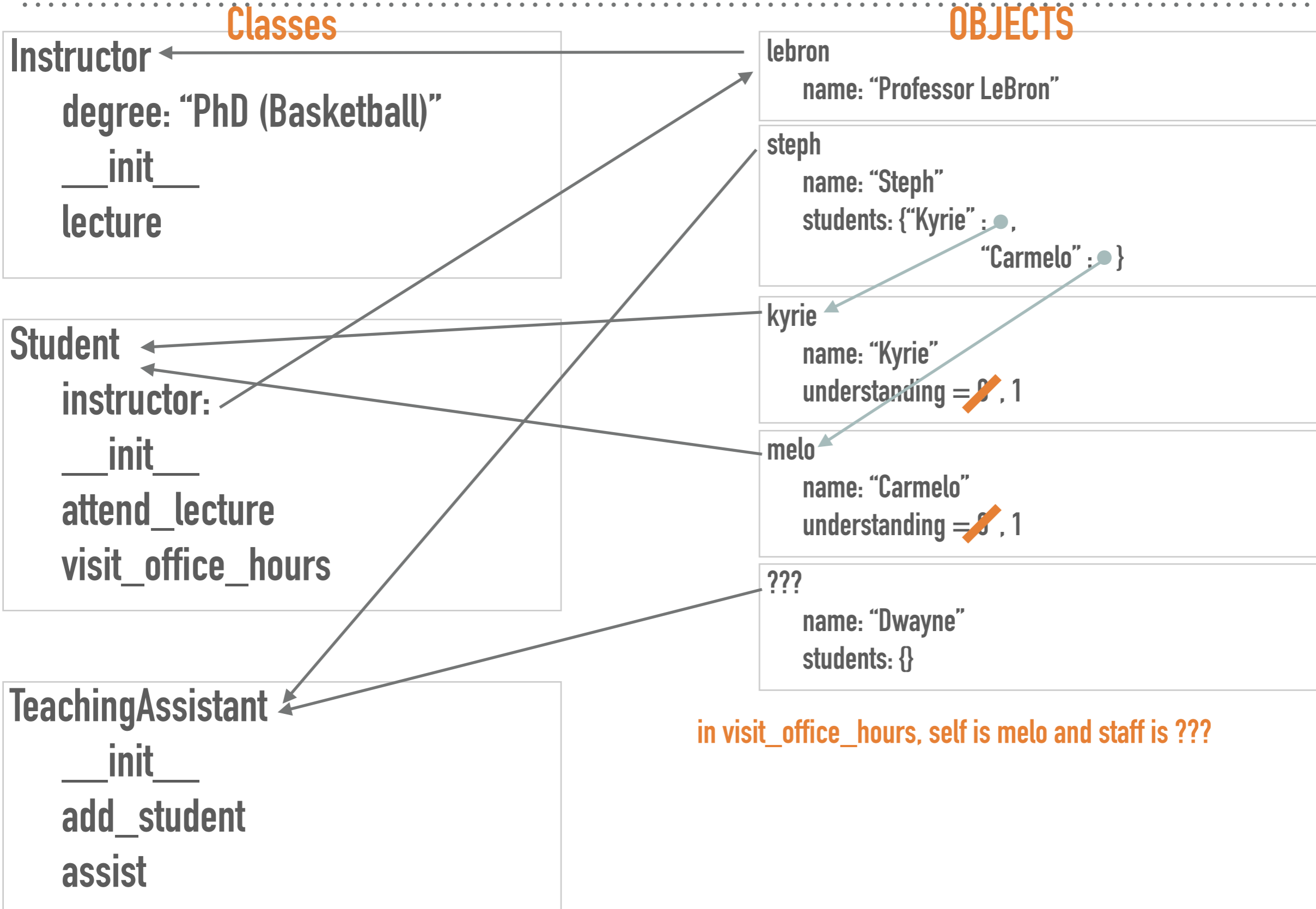
kyrie
name: "Kyrie"
understanding = ~~0~~ . 1

melo
name: "Carmelo"
understanding = ~~0~~ . 1

Repeat almost the same steps from when we called kyrie.attend_lecture('defense') but this time update melo's understanding

1.1

```
>>> melo.visit_office_hours
        (TeachingAssistant("Dwayne"))
```

1. Create classes

2. Create objects

Classes

Instructor

degree: "PhD (Basketball)"

__init__

lecture

Student

instructor:

__init__

attend_lecture

visit_office_hours

TeachingAssistant

__init__

add_student

assist

OBJECTS

lebron
     name: "Professor LeBron"

steph
     name: "Steph"
     students: {"Kyrie" : ●,
                      "Carmelo" : ● }

kyrie
     name: "Kyrie"
     understanding = 0 , 1

melo
     name: "Carmelo"
     understanding = 0 , 1

in visit_office_hours, self is melo and staff is ???

Now we are ready to walk through the lines on page 3
The current line will be displayed up here

```
>>> melo.visit_office_hours
        (TeachingAssistant("Dwayne"))
```

1. Create classes

2. Create objects

Classes

Instructor

degree: "PhD (Basketball)"

__init__

lecture

Student

instructor:

__init__

attend_lecture

visit_office_hours

TeachingAssistant

__init__

add_student

assist

OBJECTS

lebron
    name: "Professor LeBron"

steph
    name: "Steph"
    students: {"Kyrie" : ● ,
                "Carmelo" : ● }

kyrie
    name: "Kyrie"
    understanding = 0, 1

melo
    name: "Carmelo"
    understanding = 0, 1

???
    name: "Dwayne"
    students: {}

in visit_office_hours, self is melo and staff is ???

1.1

```
>>> melo.visit_office_hours
        (TeachingAssistant("Dwayne"))
```

1. Create classes

2. Create objects

Classes

Instructor
　　degree: "PhD (Basketball)"
　　__init__
　　lecture

Student
　　instructor:
　　__init__
　　attend_lecture
　　visit_office_hours

TeachingAssistant
　　__init__
　　add_student
　　assist

OBJECTS

lebron
　　name: "Professor LeBron"

steph
　　name: "Steph"
　　students: {"Kyrie" : ● ,
　　　　　　　　　"Carmelo" : ● }

kyrie
　　name: "Kyrie"
　　understanding = 0̸ , 1

melo
　　name: "Carmelo"
　　understanding = 0̸ , 1

???
　　name: "Dwayne"
　　students: {}

in visit_office_hours, self is melo and staff is ???

then we call staff.assist(self)

Now we are ready to walk through the lines on page 3
The current line will be displayed up here

```
>>> melo.visit_office_hours
         (TeachingAssistant("Dwayne"))
```

1. Create classes

2. Create objects

Classes

OBJECTS

Instructor
degree: "PhD (Basketball)"
__init__
lecture

Student
instructor:
__init__
attend_lecture
visit_office_hours

TeachingAssistant
__init__
add_student
assist

lebron
name: "Professor LeBron"

steph
name: "Steph"
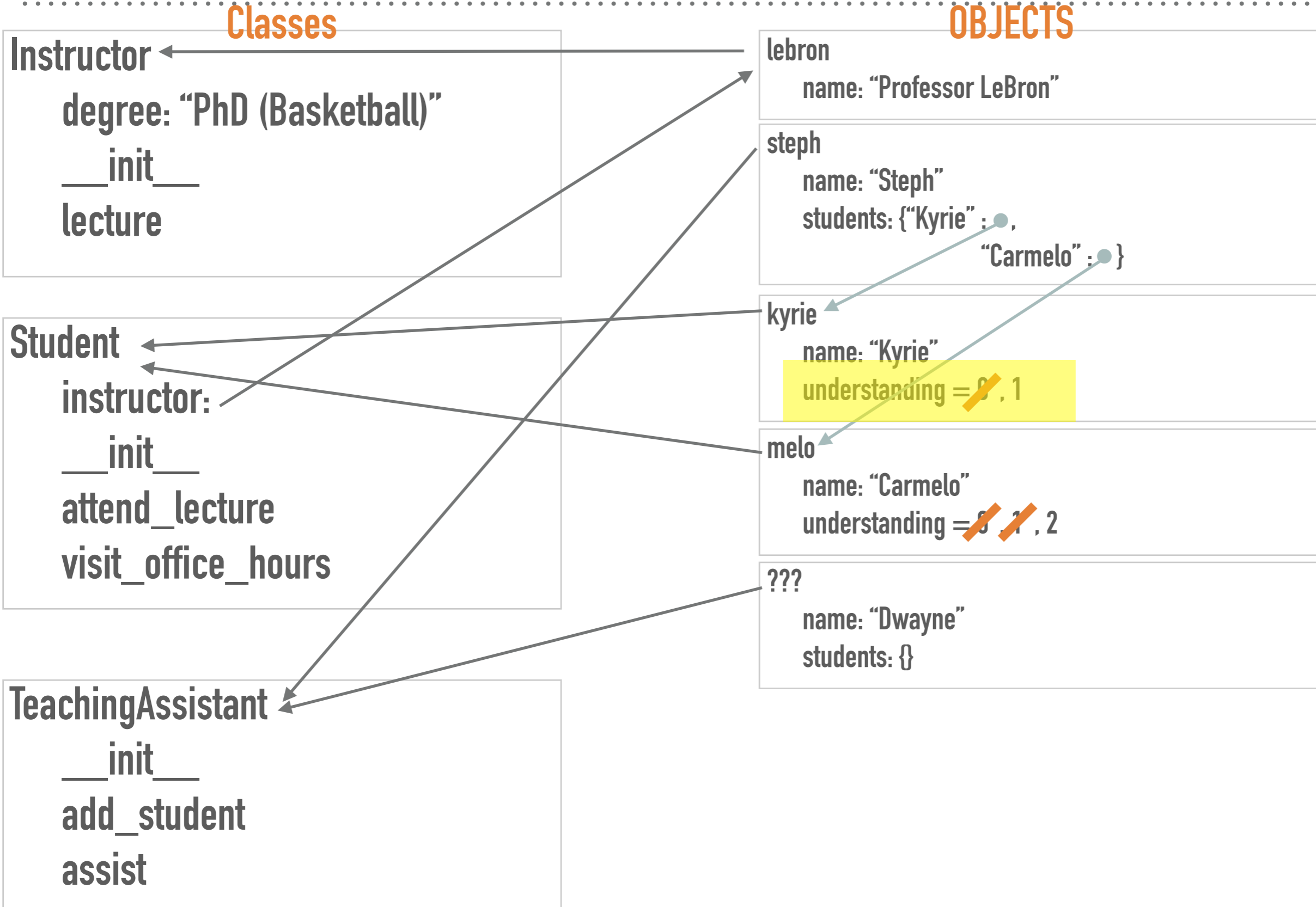students: {"Kyrie" : ● ,
               "Carmelo" : ● }

kyrie
name: "Kyrie"
understanding = 0̶ , 1

melo
name: "Carmelo"
understanding = 0̶ 1̶ , 2

???
name: "Dwayne"
students: {}

in visit_office_hours, self is melo and staff is ???

then we call staff.assist(self)

in assist, we increment melo's understanding by 1

1.1

```
>>> melo.visit_office_hours
        (TeachingAssistant("Dwayne"))
```

Thanks, Dwayne

1. Create classes

2. Create objects

Classes

OBJECTS

Instructor
  degree: "PhD (Basketball)"
  __init__
  lecture

lebron
  name: "Professor LeBron"

steph
  name: "Steph"
  students: {"Kyrie" : ● ,
                    "Carmelo" : ● }

Student
  instructor:
  __init__
  attend_lecture
  <mark>**visit_office_hours**</mark>

kyrie
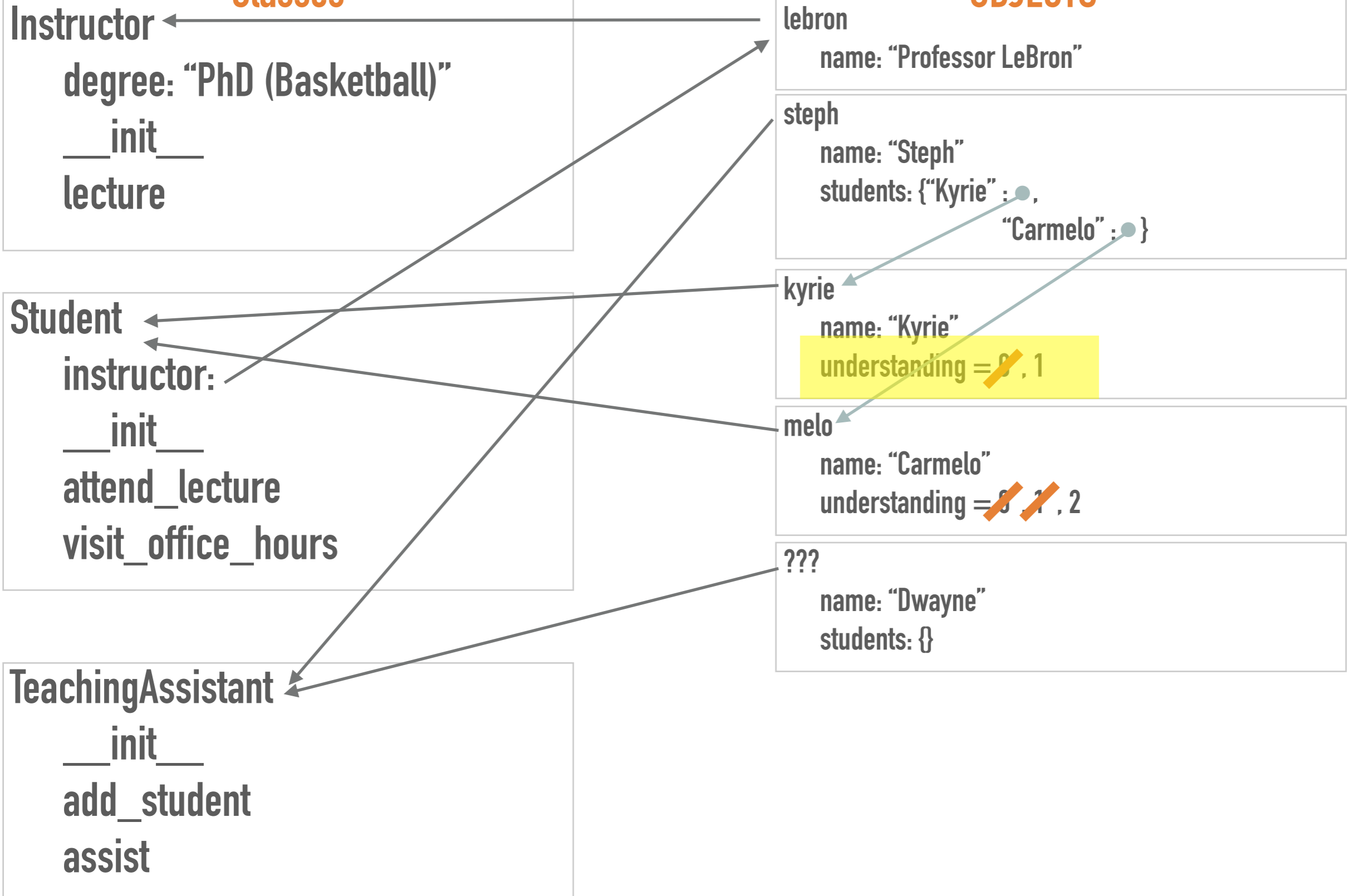 name: "Kyrie"
  understanding = ~~0~~ , 1

melo
  name: "Carmelo"
  <mark>understanding = ~~0~~ ~~1~~ , 2</mark>

???
  name: "Dwayne"
  students: {}

TeachingAssistant
  __init__
  add_student
  <mark>**assist**</mark>

in visit_office_hours, self is melo and staff is ???

then we call staff.assist(self)

in assist, we increment melo's understanding by 1

back in visit_office_hours we have a print statement

Now we are ready to walk through the lines on page 3
The current line will be displayed up here

```
>>> kyrie.understanding
```

1. Create classes

2. Create objects

Classes

OBJECTS

Instructor
   degree: "PhD (Basketball)"
   __init__
   lecture

Student
   instructor:
   __init__
   attend_lecture
   visit_office_hours

TeachingAssistant
   __init__
   add_student
   assist

lebron
   name: "Professor LeBron"

steph
   name: "Steph"
   students: {"Kyrie" : ● ,
                "Carmelo" : ● }

kyrie
   name: "Kyrie"
   understanding = ~~0~~ , 1

melo
   name: "Carmelo"
   understanding = ~~0~~ ~~1~~ , 2

???
   name: "Dwayne"
   students: {}

Now we are ready to walk through the lines on page 3
The current line will be displayed up here
```
>>> kyrie.understanding
    1
```

1. Create classes

2. Create objects

Classes

Instructor
  degree: "PhD (Basketball)"
  __init__
  lecture

Student
  instructor:
  __init__
  attend_lecture
  visit_office_hours

TeachingAssistant
  __init__
  add_student
  assist

OBJECTS

lebron
    name: "Professor LeBron"

steph
    name: "Steph"
    students: {"Kyrie" : ● ,
                "Carmelo" : ● }

kyrie
    name: "Kyrie"
    understanding = 0̶ , 1

melo
    name: "Carmelo"
    understanding = 0̶ 1̶ , 2

???
    name: "Dwayne"
    students: {}

1.1

```
>>> steph.students["Carmelo"].understanding
```

1. Create classes

2. Create objects

Classes

Instructor
degree: "PhD (Basketball)"

__init__

lecture

Student
instructor:

__init__

attend_lecture

visit_office_hours

TeachingAssistant
__init__

add_student

assist

OBJECTS

lebron
   name: "Professor LeBron"

steph
   name: "Steph"
   students: {"Kyrie" : ● ,
                      "Carmelo" : ● }

kyrie
   name: "Kyrie"
   understanding = ~~0~~, 1

melo
   name: "Carmelo"
   understanding = ~~0~~ ~~1~~, 2

???
   name: "Dwayne"
   students: {}

When we have a sequence of words and periods, read from left to right.
1. Find steph
2. steph should have an instance attribute called students
3. students is a dictionary. Look for the key "Carmelo"
4. the value of the key "Carmelo" points at the object melo
5. melo has instance attribute understanding, which is what is displayed

1.1

Now we are ready to walk through the lines on page 3
The current line will be displayed up here

```
>>> steph.students["Carmelo"].understanding
```

1. Create classes

2. Create objects

Classes

Instructor

degree: "PhD (Basketball)"

__init__

lecture

Student

instructor:

__init__

attend_lecture

visit_office_hours

TeachingAssistant

__init__

add_student

assist

OBJECTS

lebron
    name: "Professor LeBron"

steph
    name: "Steph"
    students: {"Kyrie" : ● ,
                        "Carmelo" : ● }

kyrie
    name: "Kyrie"
    understanding = 0̶ , 1

melo
    name: "Carmelo"
    understanding = 0̶ 1̶ , 2

???
    name: "Dwayne"
    students: {}

When we have a sequence of words and periods, read from left to right.
1. Find steph
2. steph should have an instance attribute called students
3. students is a dictionary. Look for the key "Carmelo"
4. the value of the key "Carmelo" points at the object melo
5. melo has instance attribute understanding, which is what is displayed

1.1

Now we are ready to walk through the lines on page 3
The current line will be displayed up here

```
>>> steph.students["Carmelo"].understanding
```

1. Create classes

2. Create objects

Classes

Instructor
  degree: "PhD (Basketball)"
  __init__
  lecture

Student
  instructor:
  __init__
  attend_lecture
  visit_office_hours

TeachingAssistant
  __init__
  add_student
  assist

OBJECTS

lebron
    name: "Professor LeBron"

steph
    name: "Steph"
    students: {"Kyrie" : ● ,
                    "Carmelo" : ● }

kyrie
    name: "Kyrie"
    understanding = 0̶ , 1

melo
    name: "Carmelo"
    understanding = 0̶ 1̶ , 2

???
    name: "Dwayne"
    students: {}

When we have a sequence of words and periods, read from left to right.
1. Find steph
2. steph should have an instance attribute called students
3. students is a dictionary. Look for the key "Carmelo"
4. the value of the key "Carmelo" points at the object melo
5. melo has instance attribute understanding, which is what is displayed

1.1

The current line will be displayed up here

```
>>> steph.students["Carmelo"].understanding
```

1. Create classes

2. Create objects

Classes

Instructor
degree: "PhD (Basketball)"
__init__
lecture

Student
instructor:
__init__
attend_lecture
visit_office_hours

TeachingAssistant
__init__
add_student
assist

OBJECTS

lebron
    name: "Professor LeBron"

steph
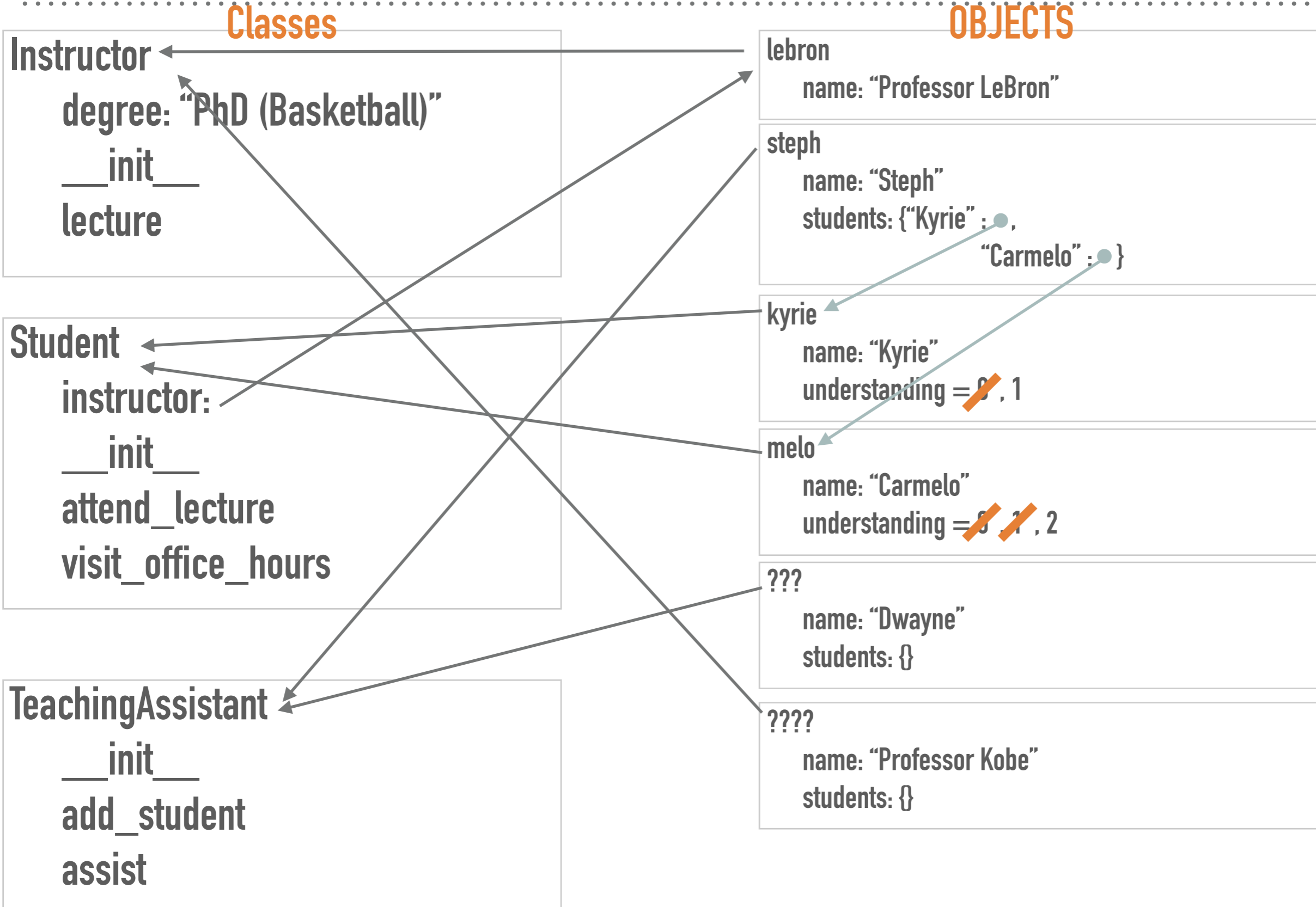   name: "Steph"
    students: {"Kyrie" : ●,
                 "Carmelo" : ● }

kyrie
    name: "Kyrie"
    understanding = ~~0~~, 1

melo
    name: "Carmelo"
    understanding = ~~0~~ ~~1~~, 2

???
    name: "Dwayne"
    students: {}

When we have a sequence of words and periods, read from left to right.
1. Find steph
2. steph should have an instance attribute called students
3. students is a dictionary. Look for the key "Carmelo"
4. the value of the key "Carmelo" points at the object melo
5. melo has instance attribute understanding, which is what is displayed

1.1

```
>>> steph.students["Carmelo"].understanding
2
```

1. Create classes

2. Create objects

Classes

Instructor
  degree: "PhD (Basketball)"
  __init__
  lecture

Student
  instructor:
  __init__
  attend_lecture
  visit_office_hours

TeachingAssistant
  __init__
  add_student
  assist

OBJECTS
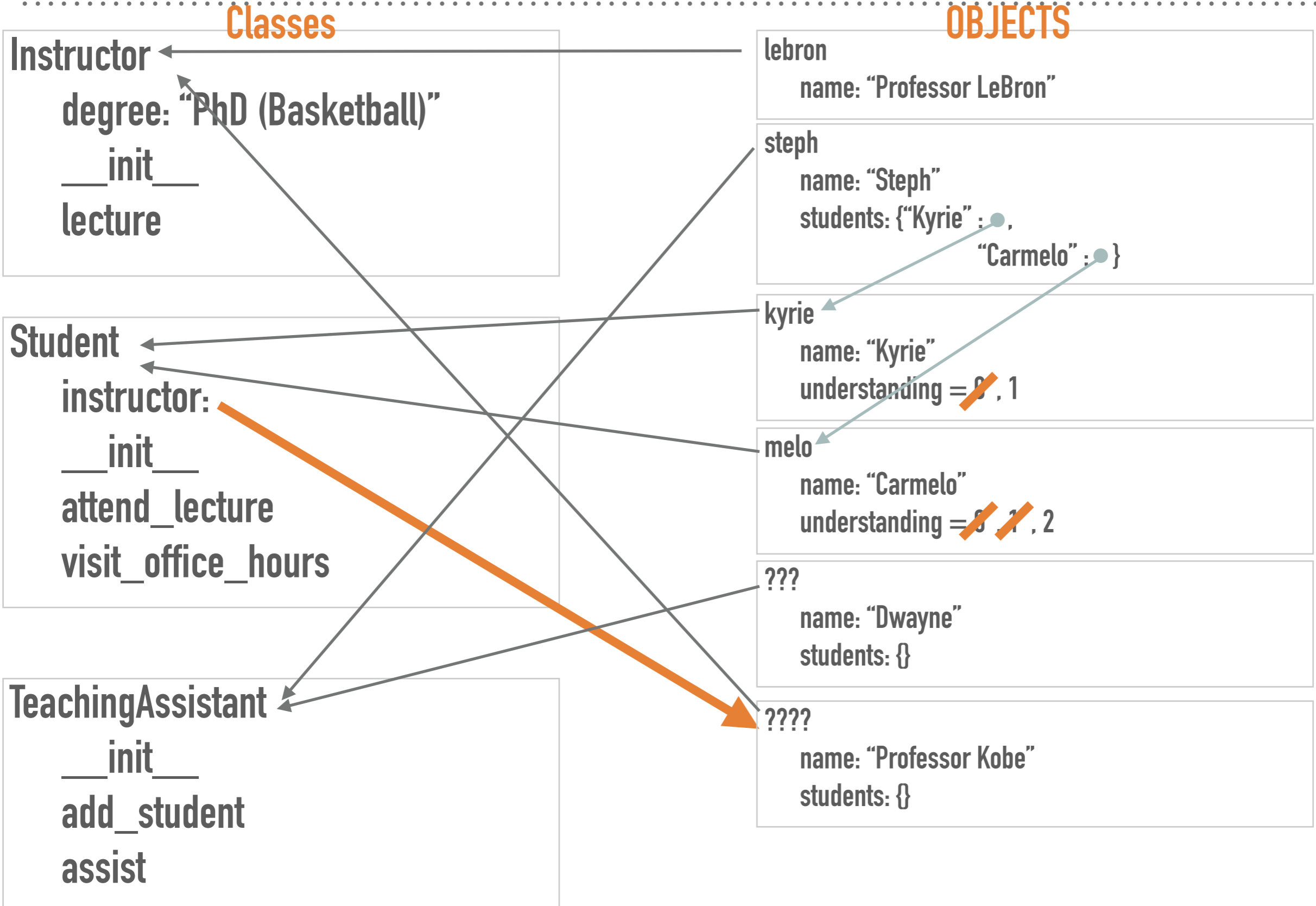
lebron
  name: "Professor LeBron"

steph
  name: "Steph"
  students: {"Kyrie" : ●,
             "Carmelo" : ● }

kyrie
  name: "Kyrie"
  understanding = 0̶, 1

melo
  name: "Carmelo"
  understanding = 0̶ 1̶, 2

???
  name: "Dwayne"
  students: {}

When we have a sequence of words and periods, read from left to right.
1. Find steph
2. steph should have an instance attribute called students
3. students is a dictionary. Look for the key "Carmelo"
4. the value of the key "Carmelo" points at the object melo
5. melo has instance attribute understanding, which is what is displayed

1. Create classes

2. Create objects

1.1

```
>>> Student.instructor =
        Instructor("Professor Kobe")
```

Classes

Instructor

degree: "PhD (Basketball)"

__init__

lecture

Student

instructor:

__init__

attend_lecture

visit_office_hours

TeachingAssistant

__init__

add_student

assist

OBJECTS

lebron
name: "Professor LeBron"

steph
name: "Steph"
students: {"Kyrie" : ●,
          "Carmelo" : ●}
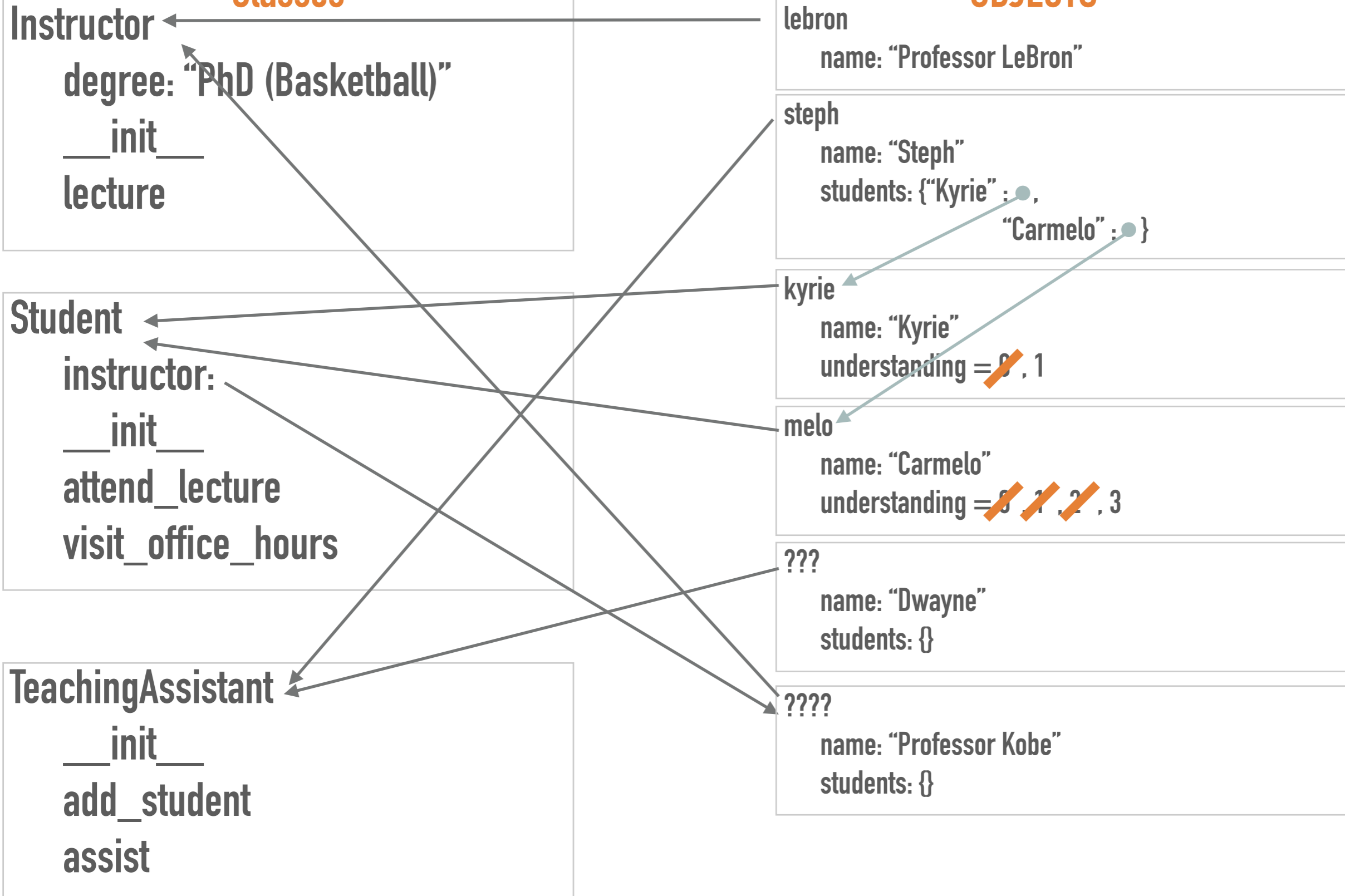
kyrie
name: "Kyrie"
understanding = 0̶, 1

melo
name: "Carmelo"
understanding = 0̶ 1̶, 2

???
name: "Dwayne"
students: {}

1.1

```
>>> Student.instructor =
        Instructor("Professor Kobe")
```

1. Create classes

2. Create objects

Classes

Instructor

degree: "PhD (Basketball)"

__init__

lecture

Student

instructor:

__init__

attend_lecture

visit_office_hours

TeachingAssistant

__init__

add_student

assist

OBJECTS

lebron

name: "Professor LeBron"

steph

name: "Steph"

students: {"Kyrie" : ●,

"Carmelo" : ● }

kyrie

name: "Kyrie"

understanding = 0̸ , 1

melo

name: "Carmelo"

understanding = 0̸ , 1̸ , 2

???

name: "Dwayne"

students: {}

????

name: "Professor Kobe"

students: {}

Now we are ready to walk through the lines on page 3
The current line will be displayed up here

```
>>> Student.instructor =
        Instructor("Professor Kobe")
```

1. Create classes

2. Create objects

Classes

Instructor
   degree: "PhD (Basketball)"
   __init__
   lecture

Student
   instructor:
   __init__
   attend_lecture
   visit_office_hours

TeachingAssistant
   __init__
   add_student
   assist

OBJECTS

lebron
   name: "Professor LeBron"

steph
   name: "Steph"
   students: {"Kyrie" : •,
                  "Carmelo" : • }

kyrie
   name: "Kyrie"
   understanding = 0̶, 1

melo
   name: "Carmelo"
   understanding = 0̶ 1̶, 2

???
   name: "Dwayne"
   students: {}

????
   name: "Professor Kobe"
   students: {}

1.1

The current line will be displayed up here

```
>>> Student.attend_lecture(melo,
                          "game winners")
```

Today we're learning about game winners
Professor Kobe is awesome!

1. Create classes

2. Create objects

Classes

Instructor
  degree: "PhD (Basketball)"
  __init__
  lecture

Student
  instructor:
  __init__
  attend_lecture
  visit_office_hours

TeachingAssistant
  __init__
  add_student
  assist

OBJECTS

lebron
  name: "Professor LeBron"

steph
  name: "Steph"
  students: {"Kyrie" : ●,
             "Carmelo" : ●}

kyrie
  name: "Kyrie"
  understanding = ~~0~~, 1

melo
  name: "Carmelo"
  understanding = ~~0~~, ~~1~~, ~~2~~, 3

???
  name: "Dwayne"
  students: {}

????
  name: "Professor Kobe"
  students: {}
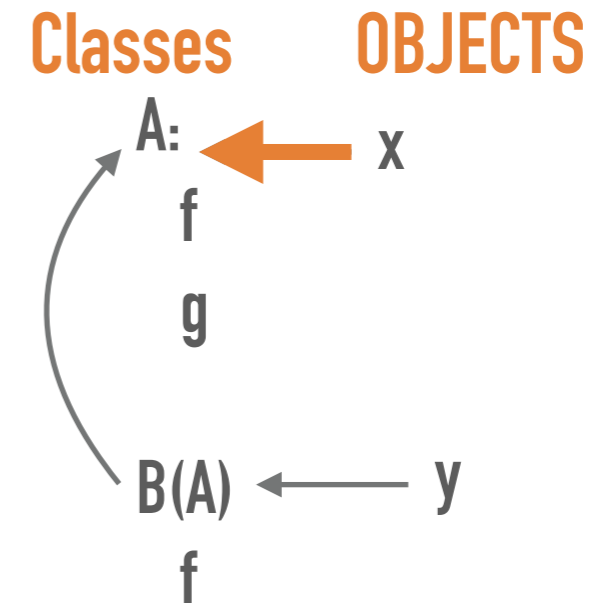
```
class A:
    def f(self):
        return 2
    def g(self, obj, x):
        if x == 0:
            return A.f(obj)
        return obj.f() + self.g(self, x = 1)
class B(A):
    def f(self):
        return 4
```

2.1 #3

```
class A:
    def f(self):
        return 2
    def g(self, obj, x):
        if x == 0:
            return A.f(obj)
        return obj.f() + self.g(self, x = 1)
class B(A):
    def f(self):
        return 4


>>> x.f()
```
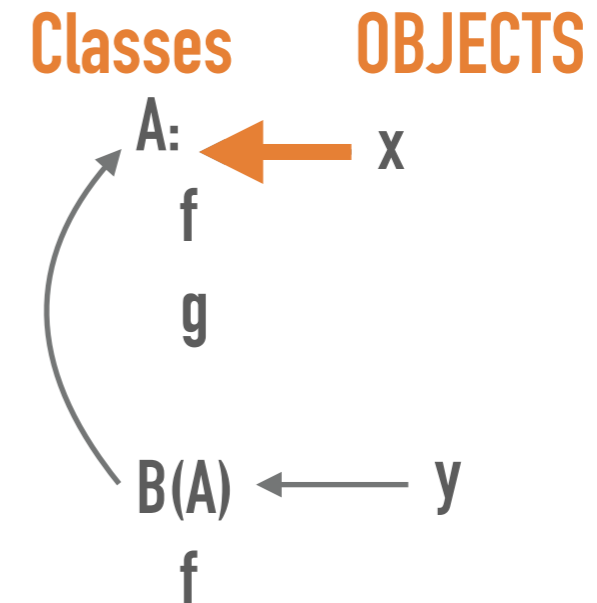
Classes   OBJECTS

A: ← x

f

g

B(A) ← y
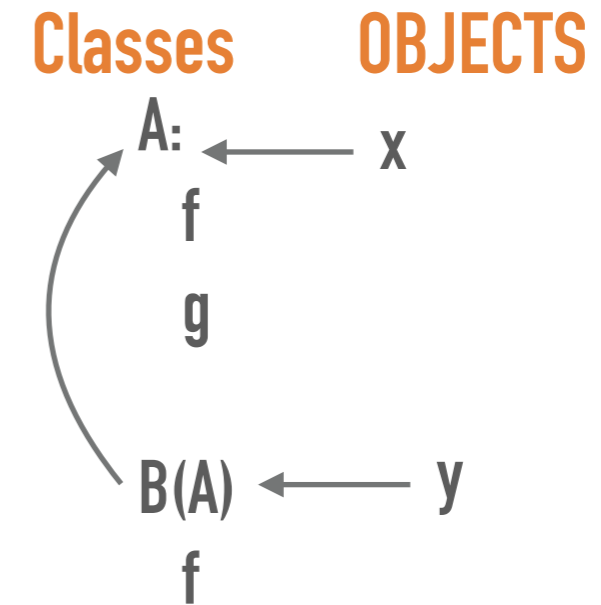
f

2.1 #3

```
class A:
    def f(self):
        return 2
    def g(self, obj, x):
        if x == 0:
            return A.f(obj)
        return obj.f() + self.g(self, x = 1)
class B(A):
    def f(self):
        return 4
```

Classes   OBJECTS

A: ← x
 f

 g

B(A) ← y
 f

```
>>> x.f()
```

1. check what type of object x is

2.1 #3

```
class A:
    def f(self):
        return 2
    def g(self, obj, x):
        if x == 0:
            return A.f(obj)
        return obj.f() + self.g(self, x = 1)
class B(A):
    def f(self):
        return 4
```

Classes   OBJECTS

A: ⬅ x
 f

 g

B(A) ⬅ y
 f

```
>>> x.f()
```

1. check what type of object x is

x points to A in our diagram

1. Create classes
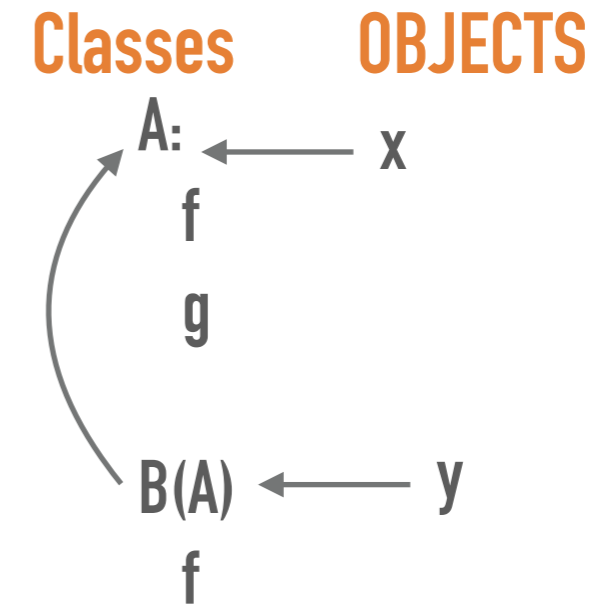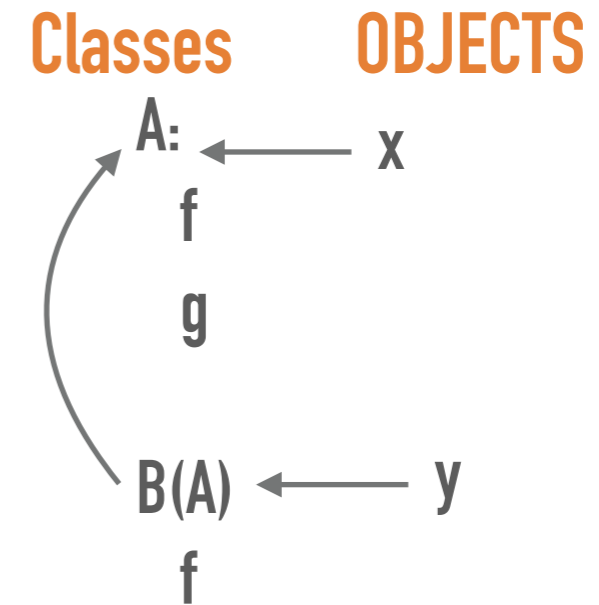
2. Create objects

```
class A:
    def f(self):
        return 2
    def g(self, obj, x):
        if x == 0:
            return A.f(obj)
        return obj.f() + self.g(self, x = 1)
class B(A):
    def f(self):
        return 4
```

```
>>> x.f()
```

1. check what type of object x is

x points to A in our diagram

2. execute the method f in the class A, passing in x as self

Classes   OBJECTS

A: ← x
f
g

B(A) ← y
f

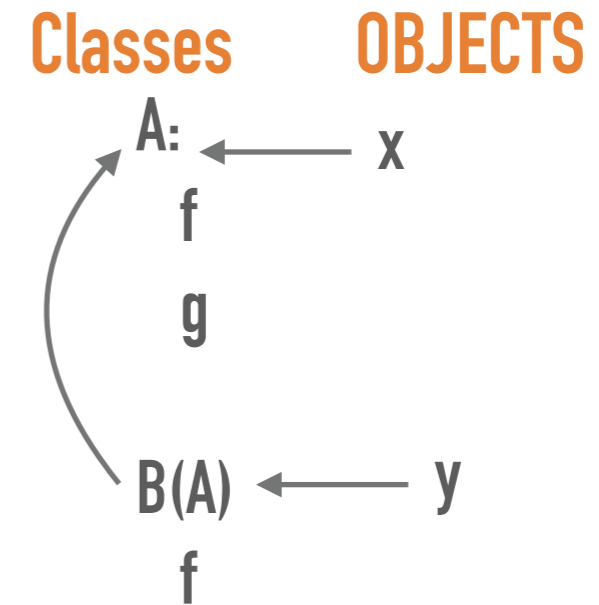# 2.1 #3

```
class A:
    def f(self):
        return 2
    def g(self, obj, x):
        if x == 0:
            return A.f(obj)
        return obj.f() + self.g(self, x = 1)
class B(A):
    def f(self):
        return 4
```

Classes   **OBJECTS**

A: ⬅ **x**
f

g

B(A) ← **y**
f

```
>>> x.f()
```

1. check what type of object x is

x points to A in our diagram

2. execute the method f in the class A, passing in x as self

return 2

2.1 #3

```
class A:
    def f(self):
        return 2
    def g(self, obj, x):
        if x == 0:
            return A.f(obj)
        return obj.f() + self.g(self, x = 1)
class B(A):
    def f(self):
        return 4



>>> x.f()

2
```

Classes   **OBJECTS**

A: ← x

f

g

B(A) ← y

f

2.1 #3
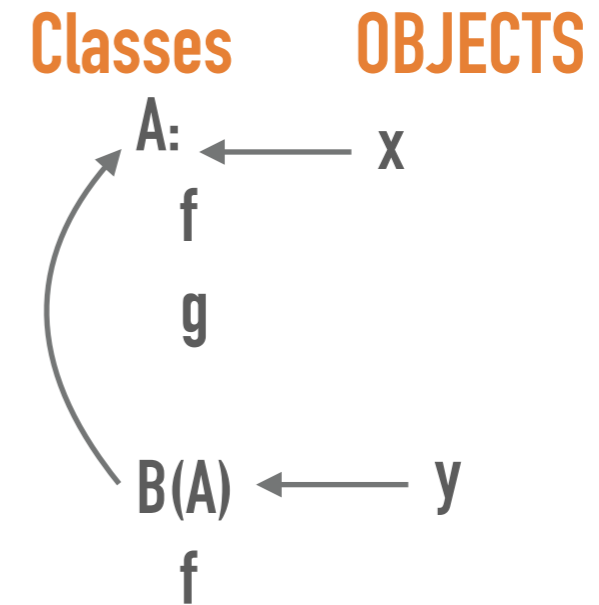
```
class A:
    def f(self):
        return 2
    def g(self, obj, x):
        if x == 0:
            return A.f(obj)
        return obj.f() + self.g(self, x = 1)
class B(A):
    def f(self):
        return 4
```

```
>>> x.f()
2

>>> B.f()
```

Classes   OBJECTS

A: ←——— x

f

g

B(A) ←——— y

f

2.1 #3

```
class A:
    def f(self):
        return 2
    def g(self, obj, x):
        if x == 0:
            return A.f(obj)
        return obj.f() + self.g(self, x = 1)
class B(A):
    def f(self):
        return 4
```

```
>>> x.f()
2

>>> B.f()
```

1. B is a class — we need an object to pass in as self

Classes **OBJECTS**

A: ← x

f

g

B(A) ← y

f

2.1 #3

```
class A:
    def f(self):
        return 2
    def g(self, obj, x):
        if x == 0:
            return A.f(obj)
        return obj.f() + self.g(self, x = 1)
class B(A):
    def f(self):
        return 4
```
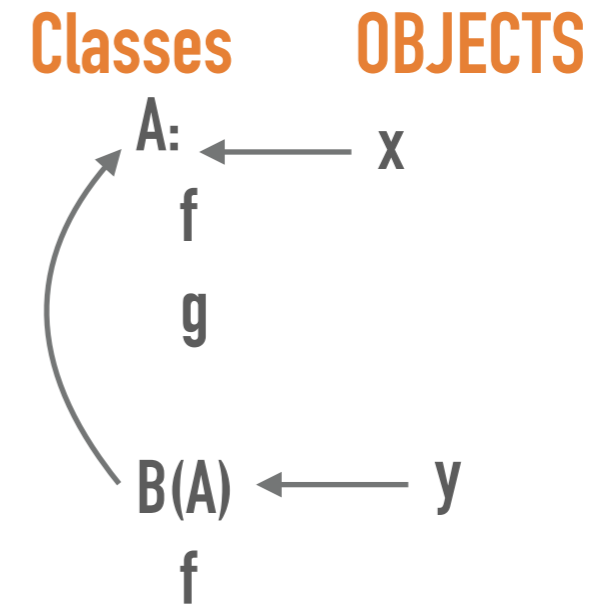
```
>>> x.f()
2

>>> B.f()
```

Classes　　**OBJECTS**

A: ← x
  f
  g

B(A) ← y
  f

1. B is a class — we need an object to pass in as self

since we didn't pass anything in as self, this code will error

2.1 #3

```
class A:
    def f(self):
        return 2
    def g(self, obj, x):
        if x == 0:
            return A.f(obj)
        return obj.f() + self.g(self, x - 1)
class B(A):
    def f(self):
        return 4
```
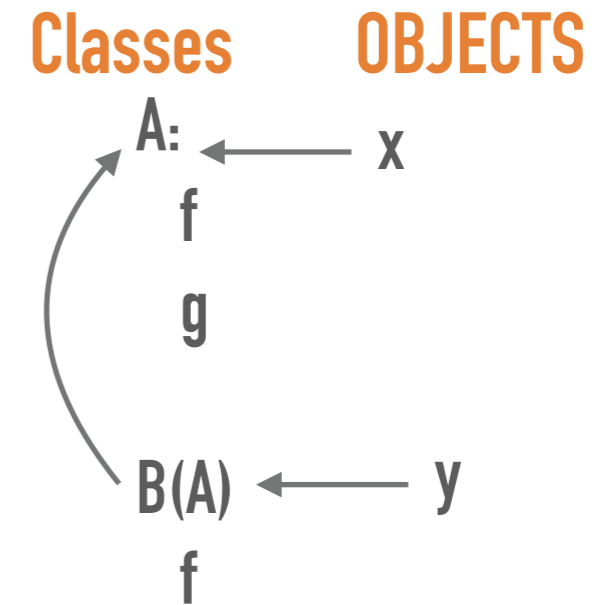
```
>>> x.f()

2

>>> B.f()

Error
```

Classes   **OBJECTS**

A: ← x

f

g

B(A) ← y

f

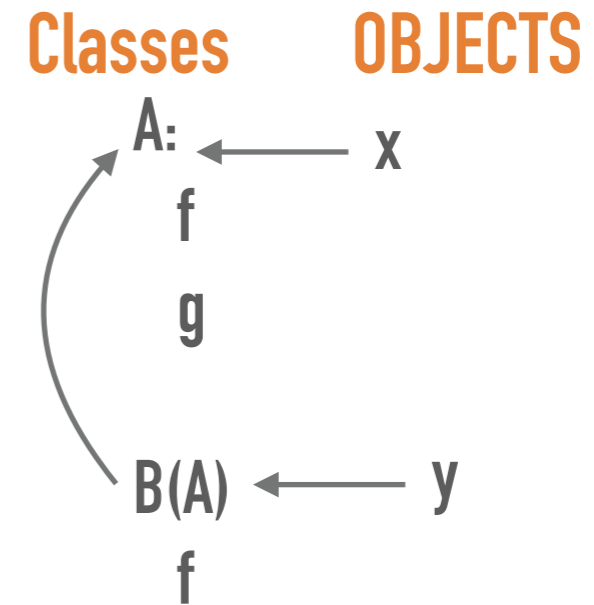# 2.1 #3

```
class A:
    def f(self):
        return 2
    def g(self, obj, x):
        if x == 0:
            return A.f(obj)
        return obj.f() + self.g(self, x - 1)
class B(A):
    def f(self):
        return 4
```

```
>>> x.f()
2

>>> B.f()

Error

>>> x.g(x, 1)
```

Classes **OBJECTS**

A: ← x

f

g

B(A) ← y

f

2.1 #3

```
class A:
    def f(self):
        return 2
    def g(self, obj, x):
        if x == 0:
            return A.f(obj)
        return obj.f() + self.g(self, x - 1)
class B(A):
    def f(self):
        return 4
```

Classes   OBJECTS

A: ← x
f
g

B(A) ← y
f

1. what does this code mean in english?
what is self? what method are we calling?
what are we passing in as arguments?

```
>>> x.f()
2

>>> B.f()

Error

>>> x.g(x, 1)
```

2.1 #3

```
class A:
    def f(self):
        return 2
    def g(self, obj, x):
        if x == 0:
            return A.f(obj)
        return obj.f() + self.g(self, x - 1)
class B(A):
    def f(self):
        return 4
```

Classes   OBJECTS

A: ← x
  f
  g

B(A) ← y
  f

```
>>> x.f()
2

>>> B.f()

Error

>>> x.g(x, 1)
```

1. what does this code mean in english?
what is self? what method are we calling?
what are we passing in as arguments?

self is x because its in front of the dot
we are calling the method g in the class A since x is an object whose type is A
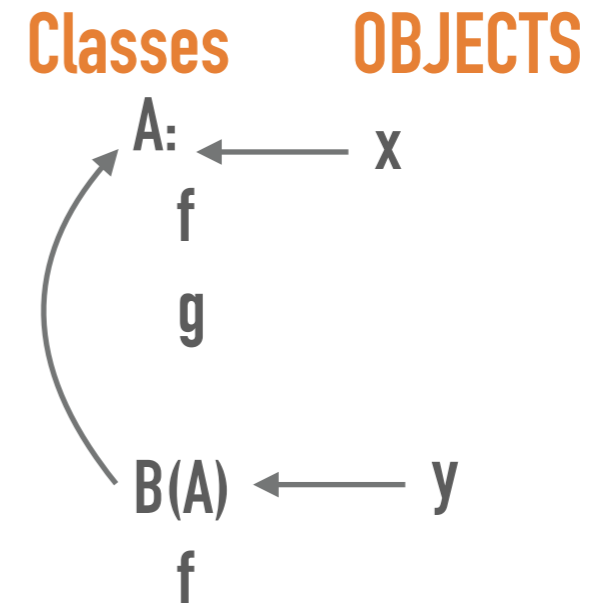g takes in 2 arguments: obj is x and x is 1

2.1 #3

```
class A:
    def f(self):
        return 2
    def g(self, obj, x):
        if x == 0:
            return A.f(obj)
        return obj.f() + self.g(self, x - 1)
class B(A):
    def f(self):
        return 4
```

Classes   **OBJECTS**

A: ← x
 f

 g

B(A) ← y
 f

```
>>> x.f()
2

>>> B.f()

Error

>>> x.g(x, 1)
```

1. what does this code mean in english?
what is self? what method are we calling?
what are we passing in as arguments?
self is x because its in front of the dot
we are calling the method g in the class A since x is an object whose type is A
g takes in 2 arguments: obj is x and x is 1

2. now we are ready to execute code! is x == 0 true?

2.1 #3

```
class A:
    def f(self):
        return 2
    def g(self, obj, x):
        if x == 0:
            return A.f(obj)
        return obj.f() + self.g(self, x – 1)
class B(A):
    def f(self):
        return 4
```

Classes   OBJECTS

A: ← x
  f
  g

B(A) ← y
  f

```
>>> x.f()
2

>>> B.f()
Error

>>> x.g(x, 1)
```

1. what does this code mean in english?
what is self? what method are we calling?
what are we passing in as arguments?

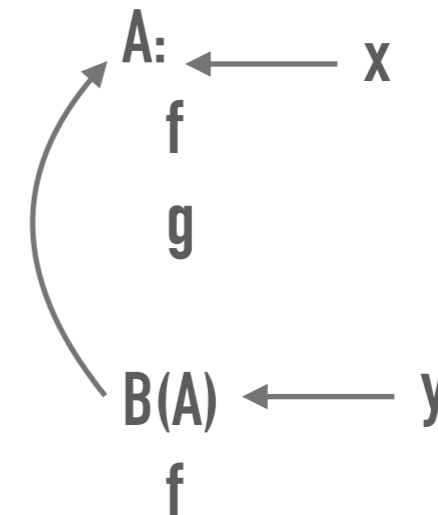self is x because its in front of the dot
we are calling the method g in the class A since x is an object whose type is A
g takes in 2 arguments: obj is x and x is 1

2. now we are ready to execute code! is x == 0 true?
x is 1, so we need to execute the second return statement.

1. Create classes

2. Create objects

```
class A:
    def f(self):
        return 2
    def g(self, obj, x):
        if x == 0:
            return A.f(obj)
        return obj.f() + self.g(self, x - 1)
class B(A):
    def f(self):
        return 4
```

Classes   OBJECTS

A: ← x
  f
  g

B(A) ← y
  f

```
>>> x.f()
2

>>> B.f()

Error

>>> x.g(x, 1)
```

1. what does this code mean in english?
what is self? what method are we calling?
what are we passing in as arguments?
self is x because its in front of the dot
we are calling the method g in the class A since x is an object whose type is A
g takes in 2 arguments: obj is x and x is 1

2. now we are ready to execute code! is x == 0 true?
x is 1, so we need to execute the second return statement.

3. what does obj.f() return?

2.1 #3

```
class A:
    def f(self):
        return 2
    def g(self, obj, x):
        if x == 0:
            return A.f(obj)
        return obj.f() + self.g(self, x - 1)
class B(A):
    def f(self):
        return 4
```

Classes   OBJECTS

A: ← x
   f
   g

B(A) ← y
   f

```
>>> x.f()
2
>>> B.f()
Error
>>> x.g(x, 1)
```

1. what does this code mean in english?
what is self? what method are we calling?
what are we passing in as arguments?
   self is x because its in front of the dot
   we are calling the method g in the class A since x is an object whose type is A
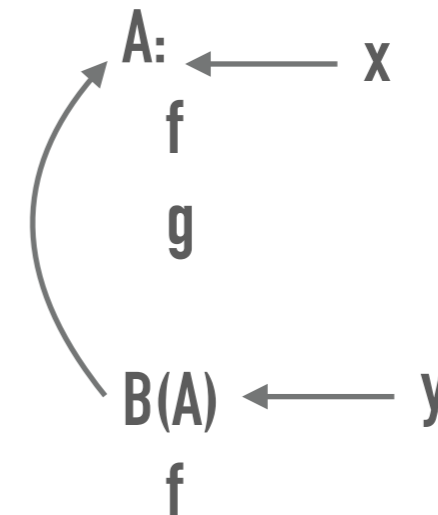   g takes in 2 arguments: obj is x and x is 1

2. now we are ready to execute code! is x == 0 true?
   x is 1, so we need to execute the second return statement.

3. what does obj.f() return?
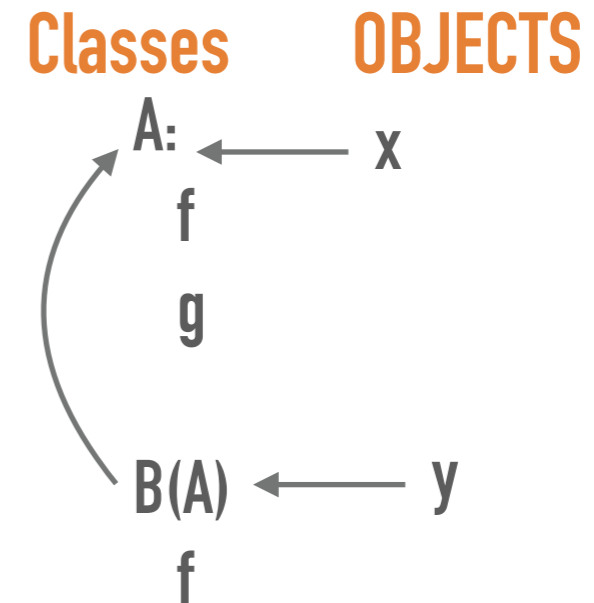   obj is x! we are calling the method f in class A where self is x. this just returns 2

2.1 #3

```
class A:
    def f(self):
        return 2
    def g(self, obj, x):
        if x == 0:
            return A.f(obj)
        return obj.f() + self.g(self, x – 1)
class B(A):
    def f(self):
        return 4
```

Classes   **OBJECTS**

A: ← x
  f
  g

B(A) ← y
  f

```
>>> x.f()
2
>>> B.f()
Error
>>> x.g(x, 1)
```

1. what does this code mean in english?
what is self? what method are we calling?
what are we passing in as arguments?

  self is x because its in front of the dot
  we are calling the method g in the class A since x is an object whose type is A
  g takes in 2 arguments: obj is x and x is 1

2. now we are ready to execute code! is x == 0 true?
  x is 1, so we need to execute the second return statement.

3. what does obj.f() return?
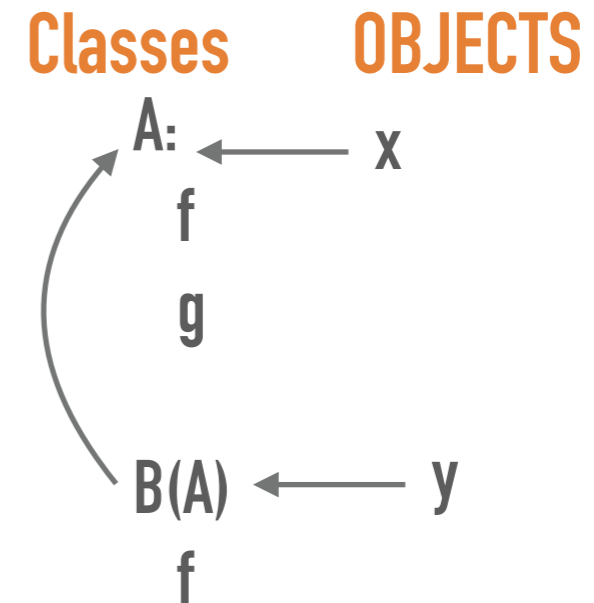 obj is x! we are calling the method f in class A where self is x. this just returns 2

4. what does self.g(self, x –1) return?

2.1 #3

1. Create classes

2. Create objects

```
class A:
    def f(self):
        return 2
    def g(self, obj, x):
        if x == 0:
            return A.f(obj)
        return obj.f() + self.g(self, x – 1)
class B(A):
    def f(self):
        return 4
```

Classes   OBJECTS

A: ← x
  f
  g

B(A) ← y
  f

```
>>> x.f()
2

>>> B.f()

Error

>>> x.g(x, 1)
```

1. what does this code mean in english?
what is self? what method are we calling?
what are we passing in as arguments?
self is x because its in front of the dot
we are calling the method g in the class A since x is an object whose type is A
g takes in 2 arguments: obj is x and x is 1

2. now we are ready to execute code! is x == 0 true?
x is 1, so we need to execute the second return statement.

3. what does obj.f() return?
obj is x! we are calling the method f in class A where self is x. this just returns 2

4. what does self.g(self, x –1) return?
self is x! we are calling the method g in class A where self is x and x is now 0.
this will cause us to go into the first if statement.
now we need to execute A.f(obj) where obj is x. we are calling method f in class A where self is x.
this returns 2.

2.1 #3

```
class A:
    def f(self):
        return 2
    def g(self, obj, x):
        if x == 0:
            return A.f(obj)
        return obj.f() + self.g(self, x – 1)
class B(A):
    def f(self):
        return 4
```
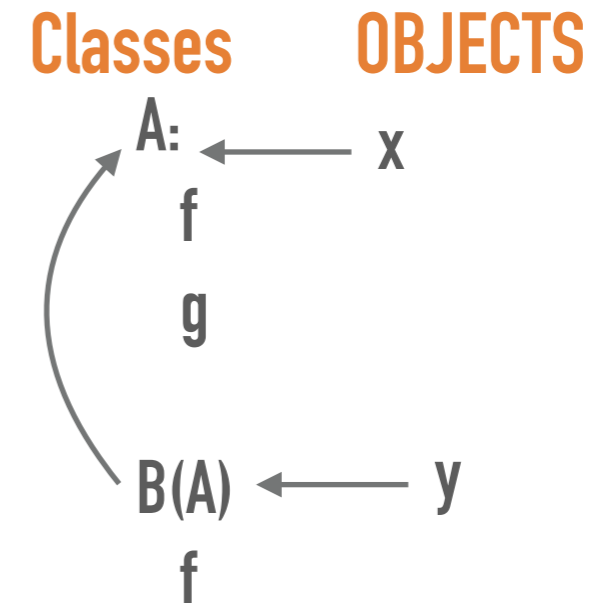
Classes   OBJECTS

A: ⟵ x
  f
  g

B(A) ⟵ y
  f

```
>>> x.f()
2
>>> B.f()
Error
>>> x.g(x, 1)
```

1. what does this code mean in english?
what is self? what method are we calling?
what are we passing in as arguments?
    self is x because its in front of the dot
    we are calling the method g in the class A since x is an object whose type is A
    g takes in 2 arguments: obj is x and x is 1

2. now we are ready to execute code! is x == 0 true?
    x is 1, so we need to execute the second return statement.

3. what does obj.f() return?
    obj is x! we are calling the method f in class A where self is x. this just returns 2

4. what does self.g(self, x –1) return?
    self is x! we are calling the method g in class A where self is x and x is now 0.
    this will cause us to go into the first if statement.
    now we need to execute A.f(obj) where obj is x. we are calling method f in class A where self is x.
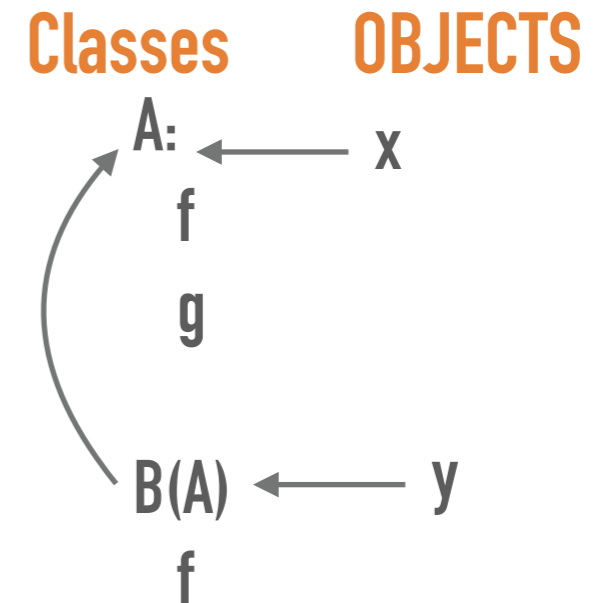   this returns 2.

5. combine the results! what is the final return value?

2.1 #3

```
class A:
    def f(self):
        return 2
    def g(self, obj, x):
        if x == 0:
            return A.f(obj)
        return obj.f() + self.g(self, x - 1)
class B(A):
    def f(self):
        return 4
```

Classes   OBJECTS

A: ⟵ x
  f
  g

B(A) ⟵ y
  f

```
>>> x.f()
2
>>> B.f()
Error
>>> x.g(x, 1)
4
```

1. what does this code mean in english?
what is self? what method are we calling?
what are we passing in as arguments?

self is x because its in front of the dot
we are calling the method g in the class A since x is an object whose type is A
g takes in 2 arguments: obj is x and x is 1

2. now we are ready to execute code! is x == 0 true?
x is 1, so we need to execute the second return statement.

3. what does obj.f() return?
obj is x! we are calling the method f in class A where self is x. this just returns 2

4. what does self.g(self, x –1) return?
self is x! we are calling the method g in class A where self is x and x is now 0.
this will cause us to go into the first if statement.
now we need to execute A.f(obj) where obj is x. we are calling method f in class A where self is x.
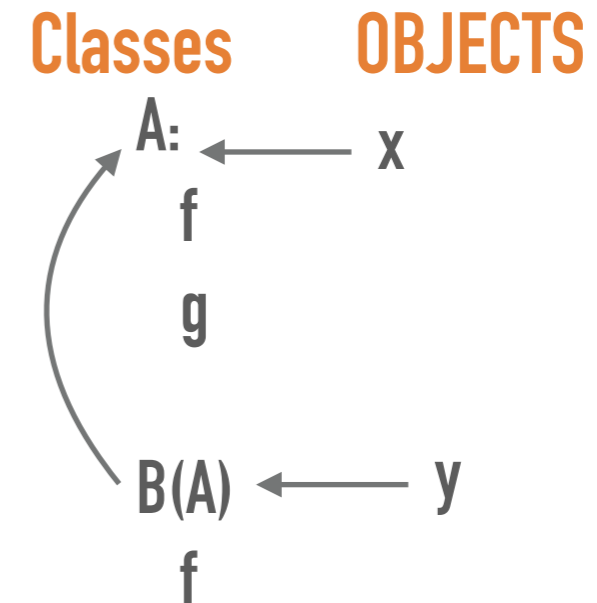this returns 2.

5. combine the results! what is the final return value? 4

1. Create classes

2. Create objects

```
class A:
    def f(self):
        return 2
    def g(self, obj, x):
        if x == 0:
            return A.f(obj)
        return obj.f() + self.g(self, x - 1)
class B(A):
    def f(self):
        return 4
```

Classes   OBJECTS

A: ⟵ x
  f
  g

B(A) ⟵ y
  f

```
>>> x.f()
2

>>> B.f()

Error

>>> x.g(x, 1)

4

>>> y.g(x, 2)
```

1. what does this code mean in english? what is self? what method are we calling? what are we passing in as arguments?

self is y because its in front of the dot
we are calling the method g in the class A since y is an object whose type is B but there is no method g in class B and class A inherits from class A
g takes in 2 arguments: obj is x and x is 2

2. now we are ready to execute code! is x == 0 true?
x is 2, so we need to execute the second return statement.

3. what does obj.f() return?
obj is x! we are calling the method f in class A where self is x. this just returns 2

4. what method are we calling when we execute the line self.g(self, x - 1)? what arguments are we passing in?
self is y! we are calling the method g in class B. But since class B does not have method g, we look at its parent. So we call g in class A where self is y and x is 1

5. is x == 0 true?
We passed in 1 as x, so this statement is false.

6.