Python Programming for Data Science

Week 40, Monday

Object oriented programming

Regular expressions - popquiz

- What does this regular expression match: ([a-z0-9.%+-]+)@([a-z0-9.-]+)\.([a-z]{2,4})
- How many groups are there in the expression
- What values do these groups correspond to?
- How would we match and extract such information from Python?

Objects and Classes

What do we know about objects?

What do we know about objects?

- Every value in Python is an object. (e.g. "hello", 5, [10,20])

What do we know about objects?

- Every value in Python is an object. (e.g. "hello", 5, [10,20])
- An object has *attributes*, accessed using the . operator.

```
l = [1,2,3,4]
l.reverse()   # reverse is a method attribute of list
```

What do we know about objects?

- Every value in Python is an object. (e.g. "hello", 5, [10,20])
- An object has *attributes*, accessed using the . operator.

```
l = [1,2,3,4]
l.reverse()   # reverse is a method attribute of list
```

- In short: An object is "a bundle" of data and functionality.

What is a type?

We've seen several built-in types: int, float, list, ...

What is a type?

We've seen several built-in types: int, float, list, ...

Q: What does it mean that an object has a certain type?

What is a type?

We've seen several built-in types: int, float, list, ...

Q: What does it mean that an object has a certain type?

A: The type defines which attributes the object has.

What is a type?

We've seen several built-in types: int, float, list, ...

Q: What does it mean that an object has a certain type?

A: The type defines which attributes the object has.

A different perspective:

What is a type?

We've seen several built-in types: int, float, list, ...

Q: What does it mean that an object has a certain type?

A: The type defines which attributes the object has.

A different perspective:

- A type corresponds to a template for the creation of objects.

What is a type?

We've seen several built-in types: int, float, list, ...

Q: What does it mean that an object has a certain type?

A: The type defines which attributes the object has.

A different perspective:

- A type corresponds to a template for the creation of objects.
- It specifies which attributes an object should have.

What is a type?

We've seen several built-in types: int, float, list, ...

Q: What does it mean that an object has a certain type?

A: The type defines which attributes the object has.

A different perspective:

- A type corresponds to a template for the creation of objects.
- It specifies which attributes an object should have.
- This template is used whenever you create a new object.

Object-oriented programming: terminology

In OOP, a different terminology is used

Object-oriented programming: terminology

In OOP, a different terminology is used

- A template is called a *class*.

Object-oriented programming: terminology

In OOP, a different terminology is used

- A template is called a *class*.
- An object created from a class is called an *instance* of the class

Object-oriented programming: terminology

In OOP, a different terminology is used

- A template is called a *class*.
- An object created from a class is called an *instance* of the class
- An object is said to *inherit* the attributes of its class

Object-oriented programming: terminology

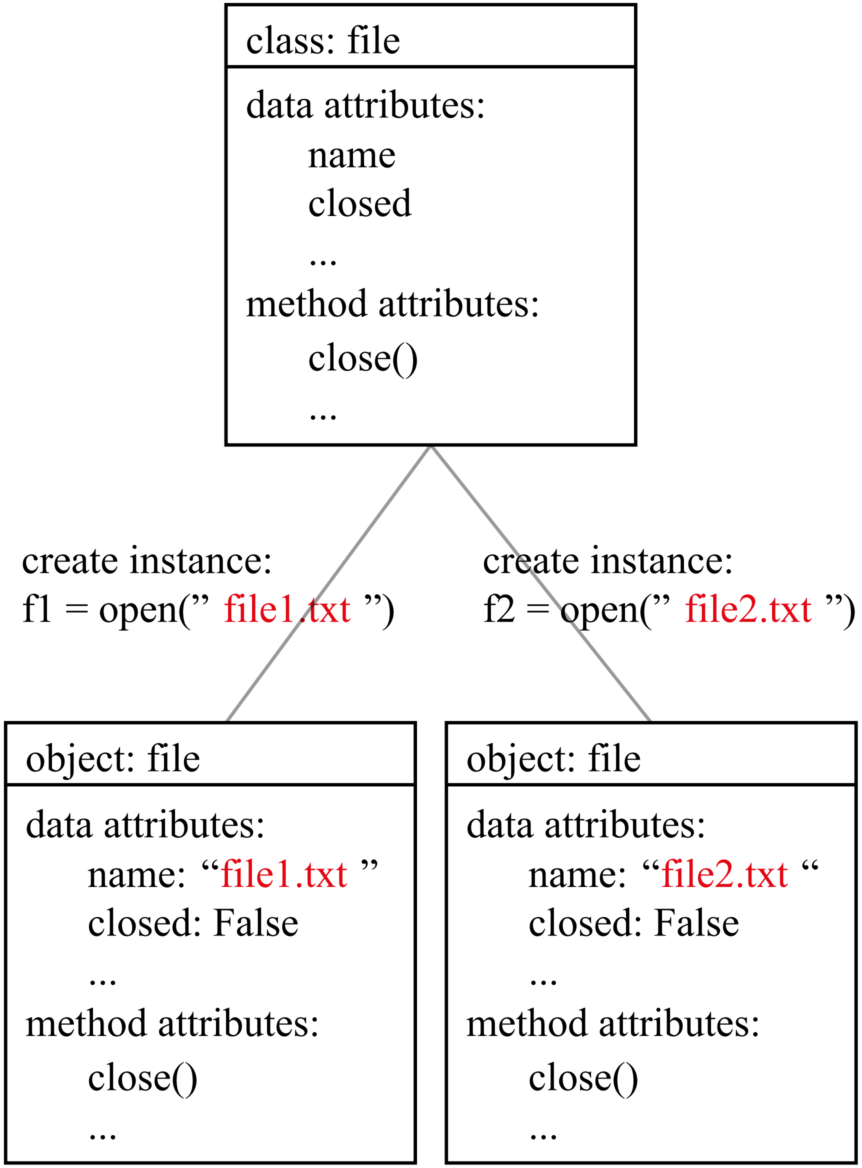
In OOP, a different terminology is used

- A template is called a *class*.
- An object created from a class is called an *instance* of the class
- An object is said to *inherit* the attributes of its class

```
my_list = [1,2,3,4]   # my_list is an instance of the list class
            my_str = "hello"      # my_str is an instance of the str class
```

OOP: creating instances (1)

Example: creating two files:



class: file

data attributes:
name
closed
...
method attributes:
close()
...

create instance:
f1 = open(” file1.txt ”)

create instance:
f2 = open(” file2.txt ”)

object: file

data attributes:
name: “file1.txt ”
closed: False
...
method attributes:
close()
...

object: file

data attributes:
name: “file2.txt “
closed: False
...
method attributes:
close()
...

OOP: creating instances (2)

We have been creating instances all along...

OOP: creating instances (2)

We have been creating instances all along...

```
a = 5            # instance of int
b = "hello"      # instance of str
c = (1,2,3,4)    # instance of tuple
```

OOP: creating instances (2)

We have been creating instances all along...

```
a = 5          # instance of int
b = "hello"    # instance of str
c = (1,2,3,4)  # instance of tuple
```

In the general case, instances are created by calling the class as a function

```
b = str(8)            # construct a string by calling str
c = tuple([1,2,3,4])  # constructing tuple by calling tuple
```

OOP: creating instances (2)

We have been creating instances all along...

```
a = 5            # instance of int
b = "hello"      # instance of str
c = (1,2,3,4)    # instance of tuple
```

In the general case, instances are created by calling the class as a function

```
b = str(8)           # construct a string by calling str
c = tuple([1,2,3,4])   # constructing tuple by calling tuple
```

For the built-in types, this is mainly used for converting between types.

Defining your own types

So far, we have used only the built-in types (int, string, ...)

Defining your own types

So far, we have used only the built-in types (int, string, ...)

Often, it is convenient to define your
own type.

Defining your own types

So far, we have used only the built-in types (int, string, ...)

Often, it is convenient to define your
own type.

Example: DNA sequence.

Defining your own types

So far, we have used only the built-in types (int, string, ...)

Often, it is convenient to define your own type.

Example: DNA sequence. It could be represented as string, but with a new type, we can:

Defining your own types

So far, we have used only the built-in types (int, string, ...)

Often, it is convenient to define your own type.

Example: DNA sequence. It could be represented as string, but with a new type, we can:

- ensure that only nucleotides (A,C,G,T) are allowed

Defining your own types

So far, we have used only the built-in types (int, string, ...)

Often, it is convenient to define your own type.

Example: DNA sequence. It could be represented as string, but with a new type, we can:

- ensure that only nucleotides (A,C,G,T) are allowed
- implement functionality: for instance calculate GC-content

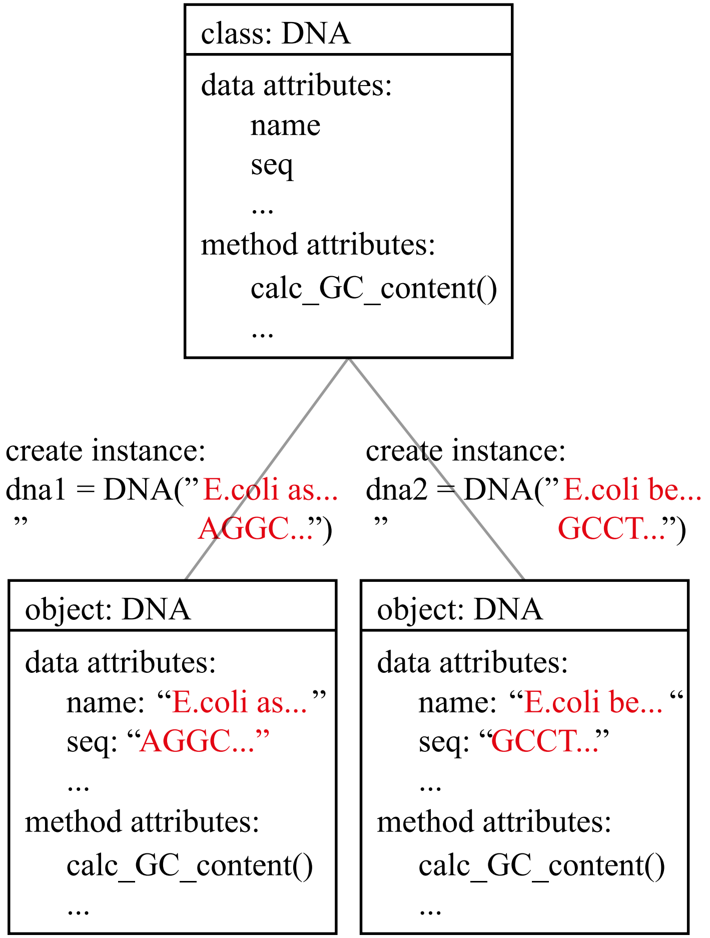
Defining your own types

So far, we have used only the built-in types (int, string, ...)

Often, it is convenient to define your own type.

Example: DNA sequence. It could be represented as string, but with a new type, we can:

- ensure that only nucleotides (A,C,G,T) are allowed
- implement functionality: for instance calculate GC-content



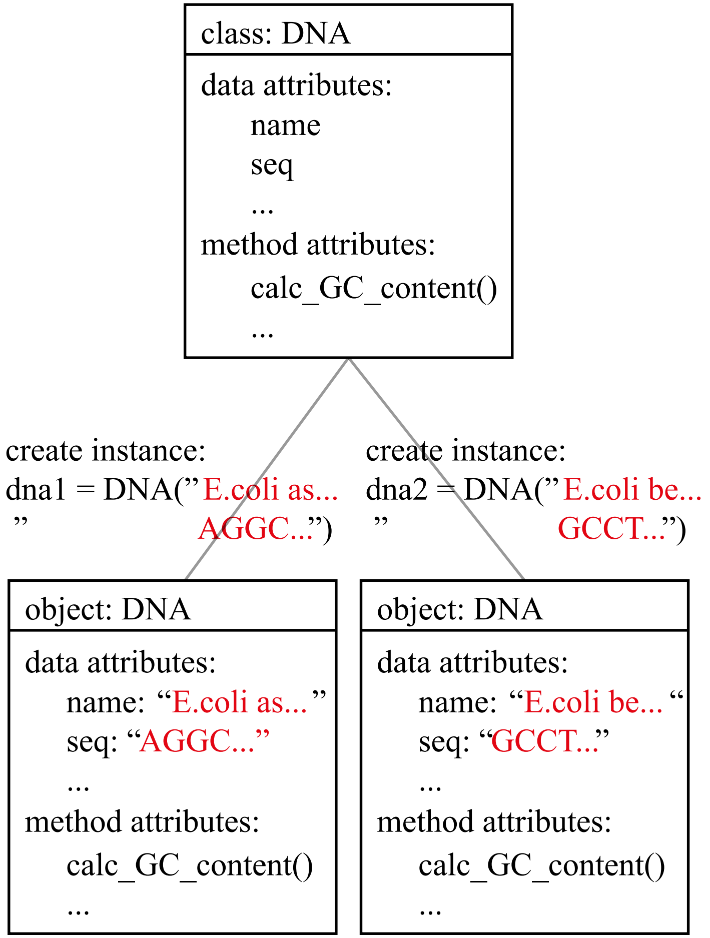
Defining your own types

So far, we have used only the built-in types (int, string, ...)

Often, it is convenient to define your own type.

Example: DNA sequence. It could be represented as string, but with a new type, we can:

- ensure that only nucleotides (A,C,G,T) are allowed
- implement functionality: for instance calculate GC-content



class: DNA

data attributes:
name
seq
...

method attributes:
calc_GC_content()
...

create instance:
dna1 = DNA("E.coli as... ", AGGC...")

create instance:
dna2 = DNA("E.coli be... ", GCCT...")

object: DNA

data attributes:
name: "E.coli as... "
seq: "AGGC..."
...

method attributes:
calc_GC_content()
...

object: DNA

data attributes:
name: "E.coli be... "
seq: "GCCT..."
...

method attributes:
calc_GC_content()
...

How do we define our own data type?

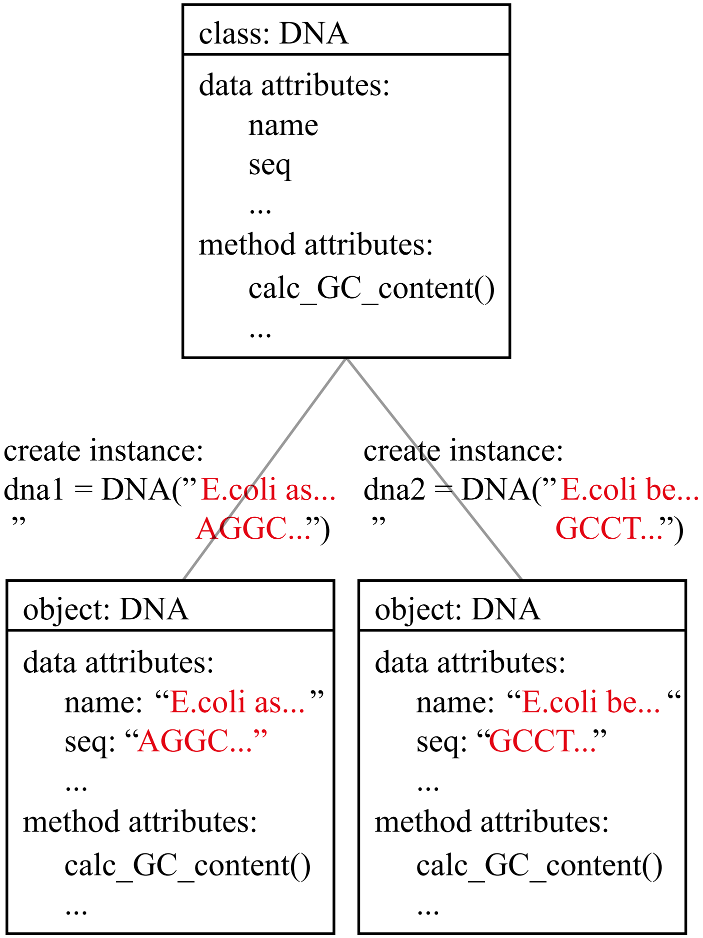
Defining your own types

So far, we have used only the built-in types (int, string, ...)

Often, it is convenient to define your own type.

Example: DNA sequence. It could be represented as string, but with a new type, we can:

- ensure that only nucleotides (A,C,G,T) are allowed
- implement functionality: for instance calculate GC-content



class: DNA

data attributes:
name
seq
...
method attributes:
calc_GC_content()
...

create instance:
dna1 = DNA(”E.coli as... ” AGGC...”)

create instance:
dna2 = DNA(”E.coli be... ” GCCT...”)

object: DNA

data attributes:
name: “E.coli as... ”
seq: “AGGC...”
...
method attributes:
calc_GC_content()
...

object: DNA

data attributes:
name: “E.coli be... “
seq: “GCCT...”
...
method attributes:
calc_GC_content()
...

How do we define our own data type? Write a class.

Writing a class

Defined like functions, but using the class keyword:

```
class class_name:
    class definition
```

Writing a class - simple example

```
class DNA:
    """Class representing sequences of Deoxyribonucleic acid"""
    pass       # Remember, pass is a placeholder - does nothing

dna = DNA()    # dna is an instance of DNA
```

Writing a class - defining methods

A method is defined like a function, but takes `self` as a first argument

Writing a class - defining methods

A method is defined like a function, but takes `self` as a first argument

dna_class.py

```
class DNA:
    """Class representing sequences of Deoxyribonucleic acid"""

    def print_nucleotides(self):
        """print the possible nucleotide letters to screen"""
        print("ACGT")

dna_object = DNA()                  # Create instance of DNA class
dna_object.print_nucleotides()      # Call print_nucleotides method
                                    # in dna_object
```

output

```
ACGT
```

Writing a class - defining methods

A method is defined like a function, but takes `self` as a first argument

dna_class.py

```
class DNA:
    """Class representing sequences of Deoxyribonucleic acid"""

    def print_nucleotides(self):
        """print the possible nucleotide letters to screen"""
        print("ACGT")

dna_object = DNA()                  # Create instance of DNA class
dna_object.print_nucleotides()      # Call print_nucleotides method
                                    # in dna_object
```

output

```
ACGT
```

self points to the current instance. In this case, when we call print_nucleotides(), dna_object will automatically be sent along as `self`.

Writing a class - adding data attributes

Attribute are like variables - but created within `self`.

dna_class.py

```
class DNA:
    """Class representing sequences of Deoxyribonucleic acid"""

    def set_name(self, name):
        """Set name attribute"""
        self.name = name                   # Here we set an attribute

    def get_name(self):
        """Return name attribute"""
        return self.name

dna_object = DNA()                         # Construct object of type DNA
dna_object.set_name("E.coli bla")          # Call the set_name method
print(dna_object.get_name())               # Call the get_name method
print(dna_object.name)                     # Access name attribute directly
```

terminal

```
$ python dna_class.py
E.coli bla
E.coli bla
```

Exercise 1

1. Create an empty class called `MyInfo`.
2. Create a method called `initialize` in the class, which takes three arguments: `firstname`, `lastname`, and `age` and saves these as attributes.
3. Write a method in this class called print that prints out this information as:

```
Name: firstname lastname
Age: age
```

4. Check that you did it correctly, by executing:

```
info = MyInfo()
info.initialize("Barack", "Obama", 58)
info.print()
```

Exercise 1 - solution

1. Create an empty class called `MyInfo`.

```
class MyInfo:
    """Class representing sequences of personal information"""
    pass
```

2. Create a method called `initialize` in the class, which takes three arguments: `firstname`, `lastname`, and `age` and saves these as attributes.

```
class MyInfo:
    """Class representing sequences of personal information"""

    def initialize(self, firstname, lastname, age):
        """Set name and age attributes"""
        self.firstname = firstname
        self.lastname = lastname
        self.age = age
```

Exercise 1 - solution (2)

3. Write a method in this class called print...

```
class MyInfo:
    """Class representing sequences of personal information"""

    ...

    def print(self):
        output = "Name: " + self.firstname + " " + self.lastname + "\nAge: " + str(self.age)
        print(output)
```

4. Check that you did it correctly, by executing:

```
info = MyInfo()
info.initialize("Barack", "Obama", 58)
info.print()
```

Writing a class - initializing

Our example from before

```
class MyInfo:
    pass

info = MyInfo()     # an empty object is pretty useless
```

Writing a class - initializing

Our example from before

```python
class MyInfo:
    pass

info = MyInfo()    # an empty object is pretty useless
```

We would like to initialize our info object like this:

```python
info = MyInfo("Barack", "Obama", 58)
```

Writing a class - initializing

Our example from before

```
class MyInfo:
    pass

info = MyInfo()     # an empty object is pretty useless
```

We would like to initialize our info object like this:

```
info = MyInfo("Barack", "Obama", 58)
```

Q: How do we do this?

Writing a class - initializing

Our example from before

```
class MyInfo:
    pass

info = MyInfo()     # an empty object is pretty useless
```

We would like to initialize our info object like this:

```
info = MyInfo("Barack", "Obama", 58)
```

Q: How do we do this?

A: We define a *constructor* in our class

Writing a class - constructor

A *constructor* is a special method that is called automatically when an object is created

Writing a class - constructor

A *constructor* is a special method that is called automatically when an object is created

In python, you can add a constructor to a class by defining an __init__ method

```
class MyInfo:
    def __init__(self, firstname):
        """Constructor"""
        self.firstname = firstname

info = MyInfo("Barack")
print(info.firstname)
```

Writing a class - constructor

A *constructor* is a special method that is called automatically when an object is created

In python, you can add a constructor to a class by defining an __init__ method

```
class MyInfo:
    def __init__(self, firstname):
        """Constructor"""
        self.firstname = firstname

info = MyInfo("Barack")
print(info.firstname)
```

Note that like all other methods, `__init__` takes `self` as its first argument.

Exercise 2

1. Modify your MyInfo class from before, so that the initialize function becomes a constructor instead.
2. Check that you did it correctly, by executing:

```
info = MyInfo("Barack", "Obama", 58)
info.print()
```

Exercise 2 - solution

1. Modify your MyInfo class from before, so that the initialize function becomes a constructor instead.

```
class MyInfo:
    """Class representing sequences of personal information"""

    def __init__(self, firstname, lastname, age):
        """Set name and age attributes"""
        self.firstname = firstname
        self.lastname = lastname
        self.age = age

     ...
```

Initializing objects: a recap

```
class MyInfo:            # Class definition
    pass                 # this is just a placeholder

info = MyInfo()          # Instantiating our new class
```

Now we have an object. How do we add data to it?

Assigning attributes - 4 strategies

Assigning attributes - 4 strategies

1. Set attribute outside class

```
class MyInfo:
    pass

info = MyInfo()
info.name = "Barack"
```

Assigning attributes - 4 strategies

1. Set attribute outside class

```python
class MyInfo:
    pass

info = MyInfo()
info.name = "Barack"
```

2. Function outside class

```python
class MyInfo:
    pass

def initializer(object, name_arg):
    object.name = name_arg

info = MyInfo()
initializer(info, "Barack")
```

Assigning attributes - 4 strategies

1. Set attribute outside class

```python
class MyInfo:
    pass

info = MyInfo()
info.name = "Barack"
```

2. Function outside class

```python
class MyInfo:
    pass

def initializer(object, name_arg):
    object.name = name_arg

info = MyInfo()
initializer(info, "Barack")
```

3. Initializer method in class

```python
class MyInfo:
    def initialize(self, name_arg):
        self.name = name_arg

info = MyInfo()
info.initialize("Barack")
```

Assigning attributes - 4 strategies

1. Set attribute outside class

```python
class MyInfo:
    pass

info = MyInfo()
info.name = "Barack"
```

2. Function outside class

```python
class MyInfo:
    pass

def initializer(object, name_arg):
    object.name = name_arg

info = MyInfo()
initializer(info, "Barack")
```

3. Initializer method in class

```python
class MyInfo:
    def initialize(self, name_arg):
        self.name = name_arg

info = MyInfo()
info.initialize("Barack")
```

4. Constructor

```python
class MyInfo:
    def __init__(self, name_arg):
        self.name = name_arg

info = MyInfo("Barack")
```

Assigning attributes - 4 strategies

1. Set attribute outside class

```
class MyInfo:
    pass

info = MyInfo()
info.name = "Barack"
```

ATTRIBUTES DEFINED OUTSIDE CLASS

2. Function outside class

```
class MyInfo:
    pass

def initializer(object, name_arg):
    object.name = name_arg

info = MyInfo()
initializer(info, "Barack")
```

3. Initializer method in class

```
class MyInfo:
    def initialize(self, name_arg):
        self.name = name_arg

info = MyInfo()
info.initialize("Barack")
```

4. Constructor

```
class MyInfo:
    def __init__(self, name_arg):
        self.name = name_arg

info = MyInfo("Barack")
```

Assigning attributes - 4 strategies

1. Set attribute outside class

```
class MyInfo:
    pass

info = MyInfo()
info.name = "Barack"
```

ATTRIBUTES DEFINED OUTSIDE CLASS

2. Function outside class

```
class MyInfo:
    pass

def initializer(object, name_arg):
    object.name = name_arg

info = MyInfo()
initializer(info, "Barack")
```

INITIALIZER DEFINED OUTSIDE CLASS

3. Initializer method in class

```
class MyInfo:
    def initialize(self, name_arg):
        self.name = name_arg

info = MyInfo()
info.initialize("Barack")
```

4. Constructor

```
class MyInfo:
    def __init__(self, name_arg):
        self.name = name_arg

info = MyInfo("Barack")
```

Assigning attributes - 4 strategies

1. Set attribute outside class

```
class MyInfo:
    pass

info = MyInfo()
info.name = "Barack"
```

ATTRIBUTES DEFINED OUTSIDE CLASS

2. Function outside class

```
class MyInfo:
    pass

def initializer(object, name_arg):
    object.name = name_arg

info = MyInfo()
initializer(info, "Barack")
```

INITIALIZER DEFINED OUTSIDE CLASS

3. Initializer method in class

```
class MyInfo:
    def initialize(self, name_arg):
        self.name = name_arg

info = MyInfo()
info.initialize("Barack")
```

OK

4. Constructor

```
class MyInfo:
    def __init__(self, name_arg):
        self.name = name_arg

info = MyInfo("Barack")
```

Assigning attributes - 4 strategies

1. Set attribute outside class

```
class MyInfo:
    pass

info = MyInfo()
info.name = "Barack"
```

ATTRIBUTES DEFINED OUTSIDE CLASS

2. Function outside class

```
class MyInfo:
    pass

def initializer(object, name_arg):
    object.name = name_arg

info = MyInfo()
initializer(info, "Barack")
```

INITIALIZER DEFINED OUTSIDE CLASS

3. Initializer method in class

```
class MyInfo:
    def initialize(self, name_arg):
        self.name = name_arg

info = MyInfo()
info.initialize("Barack")
```

OK

4. Constructor

```
class MyInfo:
    def __init__(self, name_arg):
        self.name = name_arg

info = MyInfo("Barack")
```

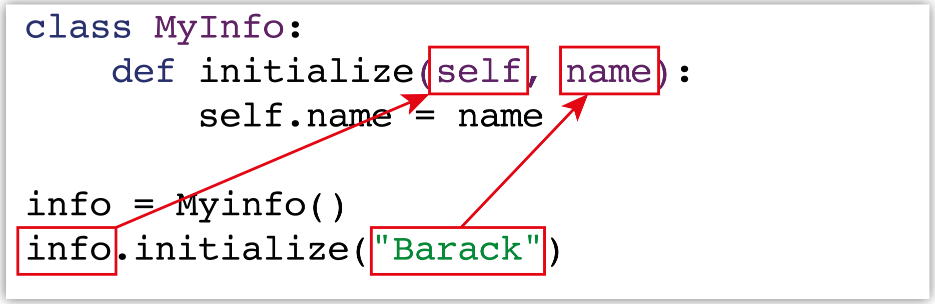
MOST ELEGANT

classes: what does `self` mean?

classes: what does `self` mean?

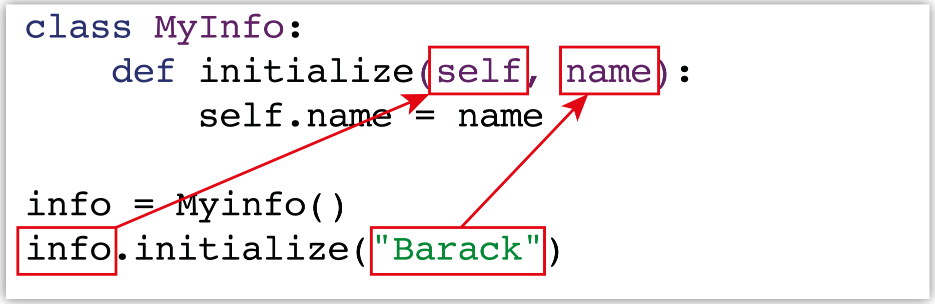
```
class MyInfo:
    def initialize(self, name):
        self.name = name

info = Myinfo()
info.initialize("Barack")
```

classes: what does `self` mean?

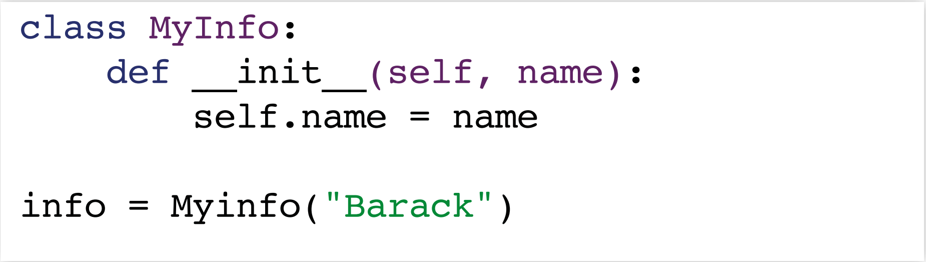


```
class MyInfo:
    def initialize(self, name):
        self.name = name

info = Myinfo()
info.initialize("Barack")
```

classes: what does `self` mean?

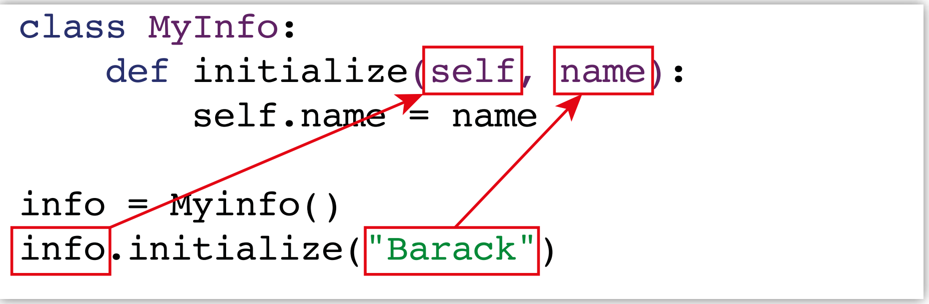


```
class MyInfo:
    def initialize(self, name):
        self.name = name

info = Myinfo()
info.initialize("Barack")
```

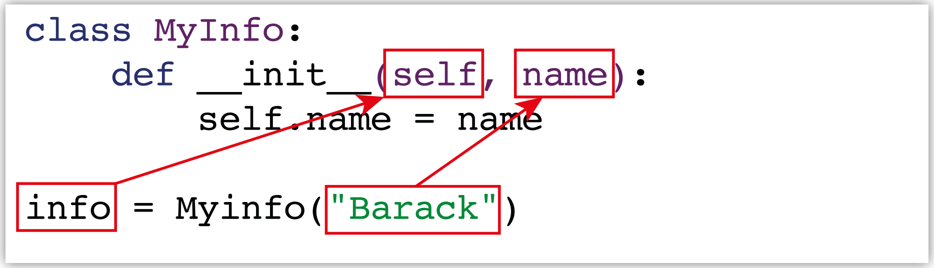


```
class MyInfo:
    def __init__(self, name):
        self.name = name

info = Myinfo("Barack")
```

classes: what does `self` mean?

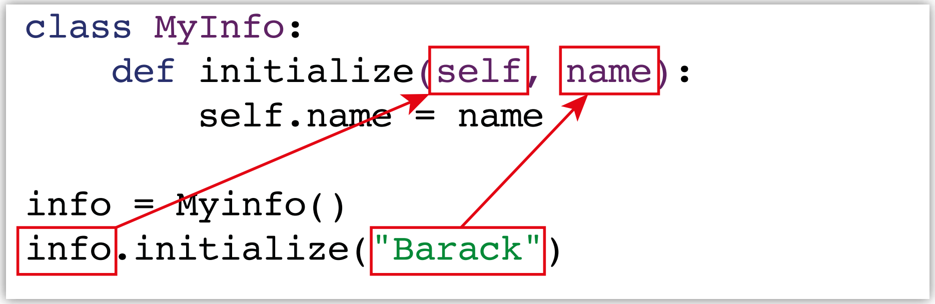


```
class MyInfo:
    def initialize(self, name):
        self.name = name

info = Myinfo()
info.initialize("Barack")
```

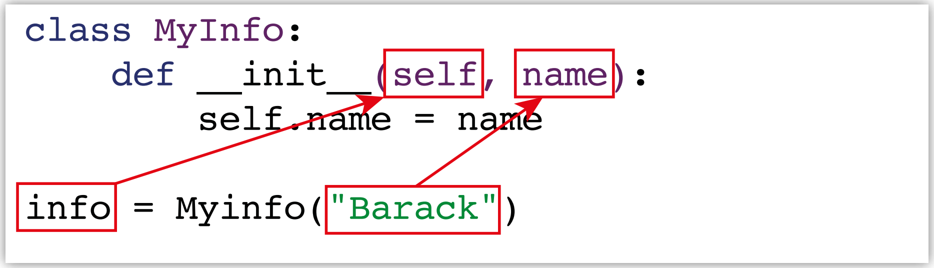


```
class MyInfo:
    def __init__(self, name):
        self.name = name

info = Myinfo("Barack")
```

classes: what does `self` mean?

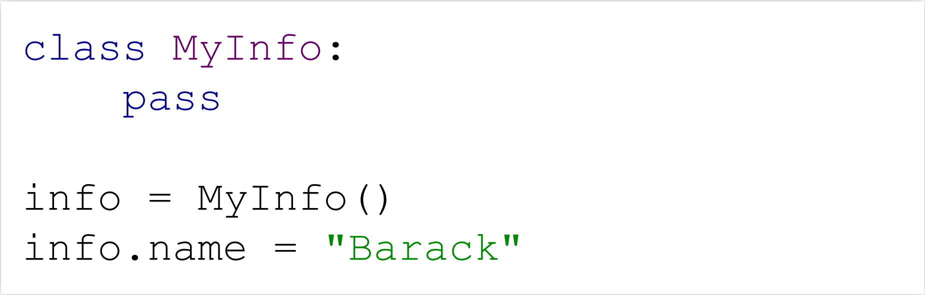


```
class MyInfo:
    def initialize(self, name):
        self.name = name

info = Myinfo()
info.initialize("Barack")
```



```
class MyInfo:
    def __init__(self, name):
        self.name = name

info = Myinfo("Barack")
```

It's really the same as:



```
class MyInfo:
    pass

info = MyInfo()
info.name = "Barack"
```

Object Oriented Programming:
Inheritance

Problem:

How do we write a program that registers and keeps track of information about all types of employees and students at the University.

Different challenges:

Problem:

How do we write a program that registers and keeps track of information about all types of employees and students at the University.

Different challenges:

- Input: fancy graphics or simply from the command line. Let's ignore this part for now.

Problem:

How do we write a program that registers and keeps track of information about all types of employees and students at the University.

Different challenges:

- Input: fancy graphics or simply from the command line. Let's ignore this part for now.
- Main challenge: students and different types of employees have different properties. How do we model this?

Possible strategy

Possible strategy

- We divide people into different categories (types)

Possible strategy

- We divide people into different categories (types)
- Each type of person has its own class.

Possible strategy

- We divide people into different categories (types)
- Each type of person has its own class.
- Whenever we create a new person object, we save it to a list or dictionary.

Student class

Which attributes should it have?

Student class

Which attributes should it have?

- Name

Student class

Which attributes should it have?

- Name
- CPR number

Student class

Which attributes should it have?

- Name
- CPR number
- Address

Student class

Which attributes should it have?

- Name
- CPR number
- Address
- List of passed courses

Student class

Which attributes should it have?

- Name
- CPR number
- Address
- List of passed courses
- Grades

Student class

Which attributes should it have?

- Name
- CPR number
- Address
- List of passed courses
- Grades
- Enrollment date

Student class

Which attributes should it have?

- Name
- CPR number
- Address
- List of passed courses
- Grades
- Enrollment date

```
class Student:
    def __init__(self, name, cpr,
                 address, courses,
                 grades, enrollment):
        self.name = name
        self.cpr = cpr
        self.address = address
        self.courses = courses
        self.grades = grades
        self.enrollment = enrollment

albert = Student("Albert Einstein",
                 "140379-1235",
                 ...)
```

Technical Staff class

Which attributes should it have?

Technical Staff class

Which attributes should it have?

- Name

Technical Staff class

Which attributes should it have?

- Name
- CPR number

Technical Staff class

Which attributes should it have?

- Name
- CPR number
- Address

Technical Staff class

Which attributes should it have?

- Name
- CPR number
- Address
- Office nr

Technical Staff class

Which attributes should it have?

- Name
- CPR number
- Address
- Office nr
- Job description

Technical Staff class

Which attributes should it have?

- Name
- CPR number
- Address
- Office nr
- Job description

```
class Technical:
    def __init__(self, name, cpr,
                 address, office,
                 job_descr):
        self.name = name
        self.cpr = cpr
        self.address = address
        self.office = office
        self.job_descr = job_descr

elvis = Technical("Elvis Presley",
                  "160835-6735",
                  ...
                  "electrician")
```

Researcher class

Which attributes should it have?

Researcher class

Which attributes should it have?

- Name

Researcher class

Which attributes should it have?

- Name
- CPR number

Researcher class

Which attributes should it have?

- Name
- CPR number
- Address

Researcher class

Which attributes should it have?

- Name
- CPR number
- Address
- Office nr

Researcher class

Which attributes should it have?

- Name
- CPR number
- Address
- Office nr
- List of publications

Researcher class

Which attributes should it have?

- Name
- CPR number
- Address
- Office nr
- List of publications
- Research interests

Researcher class

Which attributes should it have?

- Name
- CPR number
- Address
- Office nr
- List of publications
- Research interests

```
class Researcher:
    def __init__(self, name, cpr,
                 address, office,
                 job_descr):
        self.name = name
        self.cpr = cpr
        self.address = address
        self.publications = publications
        self.interests = interests

niels = Researcher("Niels Bohr",
                   "07101885-7459",
                   ...)
```

Our three classes

class Student
name
cpr_number
address
courses
grades
enrollment

class Technical
name
cpr_number
address
office_number
job_descr

class Researcher
name
cpr_number
address
office_number
publications
interests

Our three classes

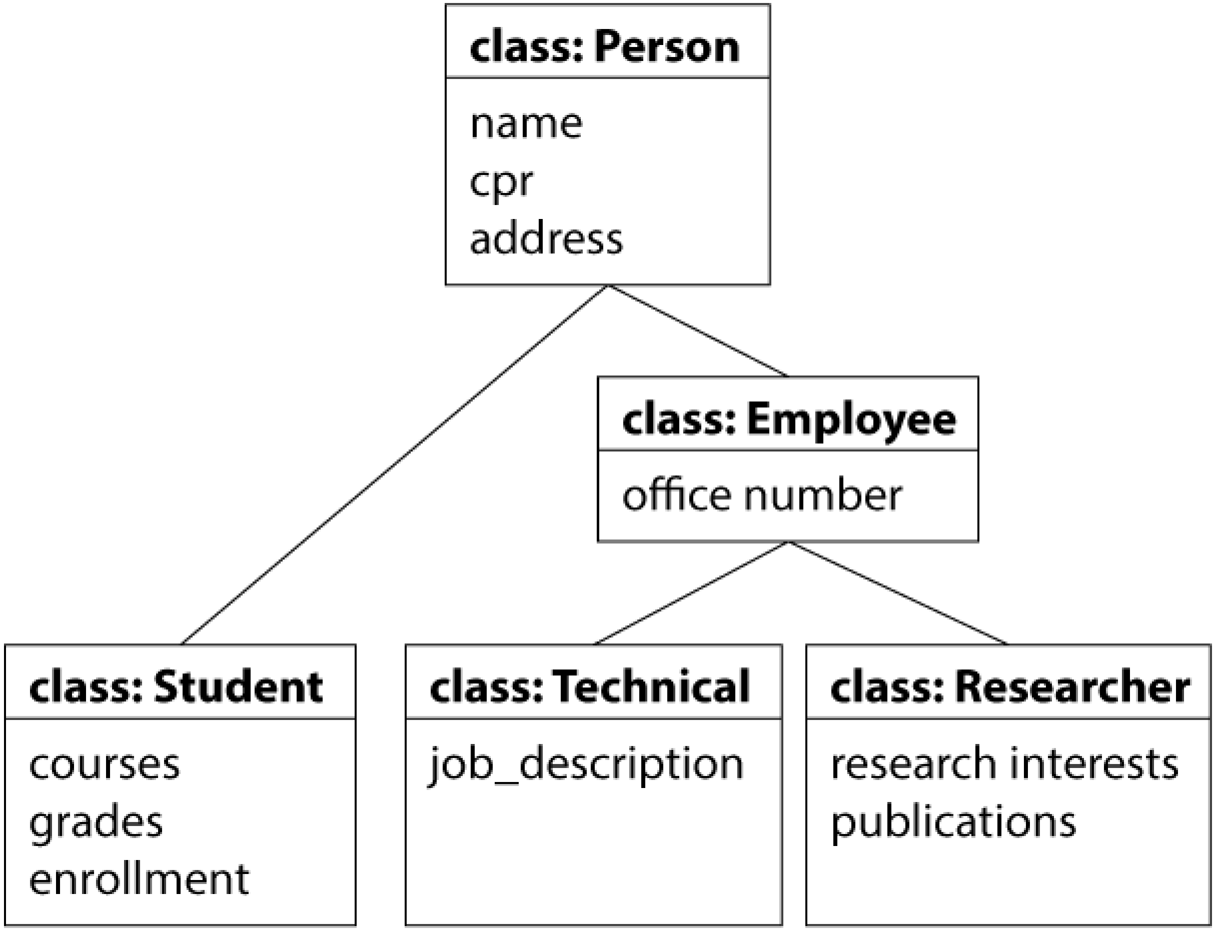
class Student
name
cpr_number
address
courses
grades
enrollment

class Technical
name
cpr_number
address
office_number
job_descr

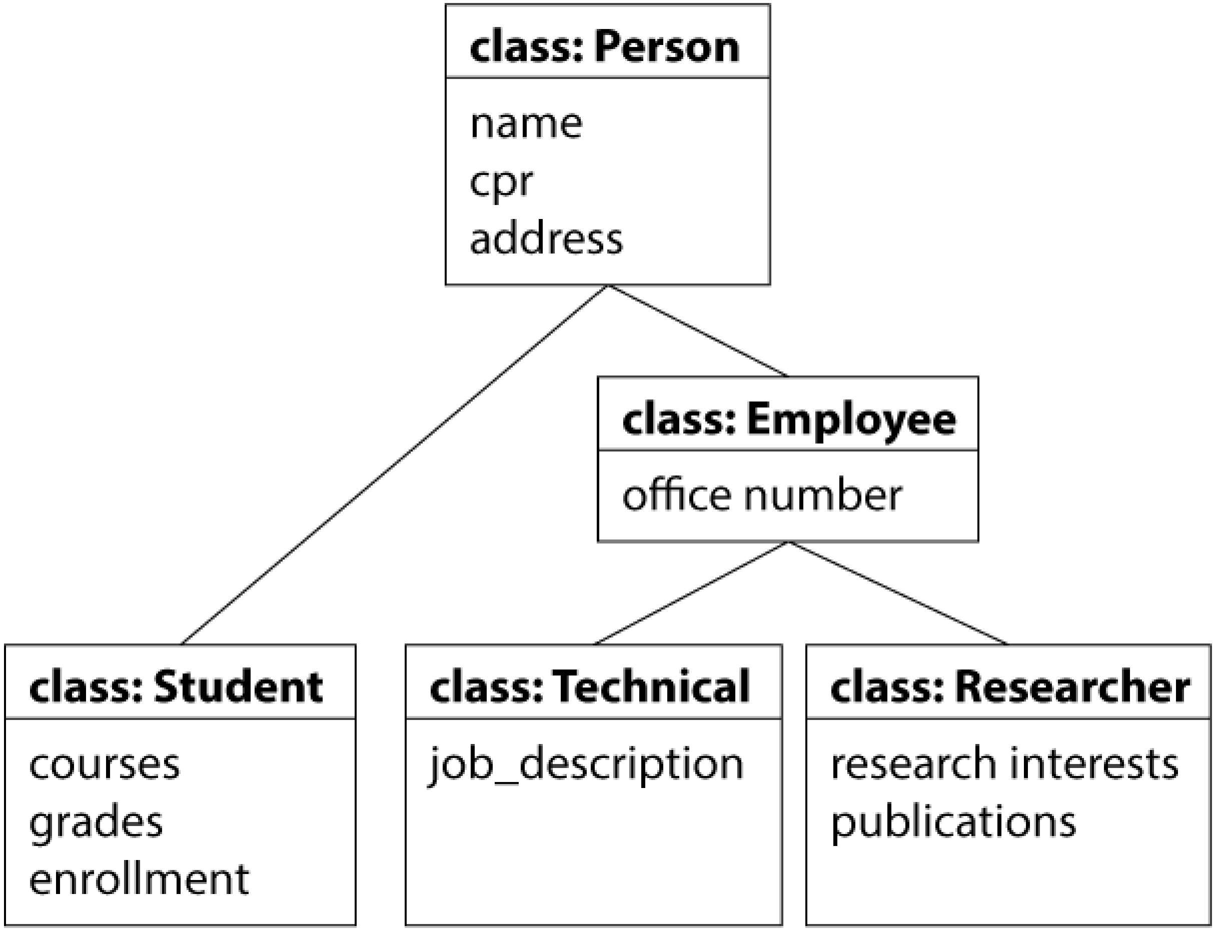
class Researcher
name
cpr_number
address
office_number
publications
interests

There is quite a lot of overlap between our classes.

Consider the data as a hierarchy



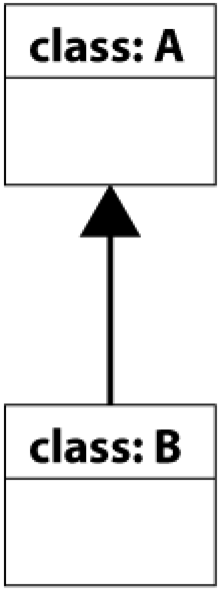
Consider the data as a hierarchy



Idea: classes can *inherit* properties from other classes.

Terminology

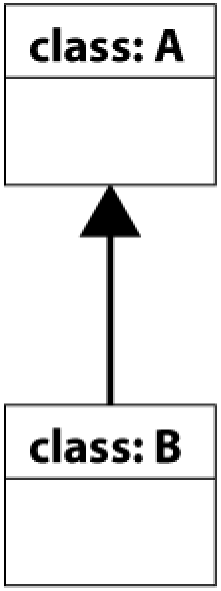
If class B inherits from A



Terminology

If class B inherits from A

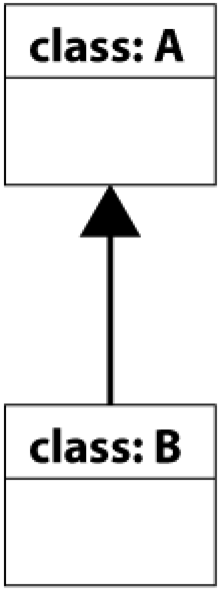
- A is the *base-class* (or *super-class*) of B



Terminology

If class B inherits from A

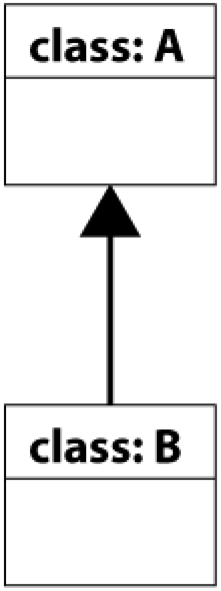
- A is the *base-class* (or *super-class*) of B
- B is a *derived-class*(or *sub-class*) of A



Terminology

If class B inherits from A

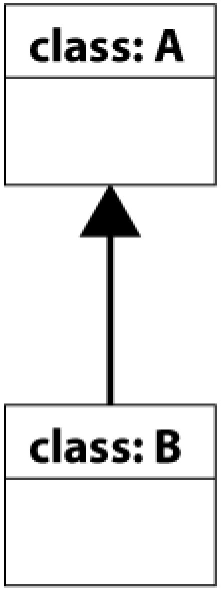
- A is the *base-class* (or *super-class*) of B
- B is a *derived-class*(or *sub-class*) of A
- B is a *specialization* of A



Terminology

If class B inherits from A

- A is the *base-class* (or *super-class*) of B
- B is a *derived-class*(or *sub-class*) of A
- B is a *specialization* of A
- Inheritance is called an *is-a* relationship: "B is an A"



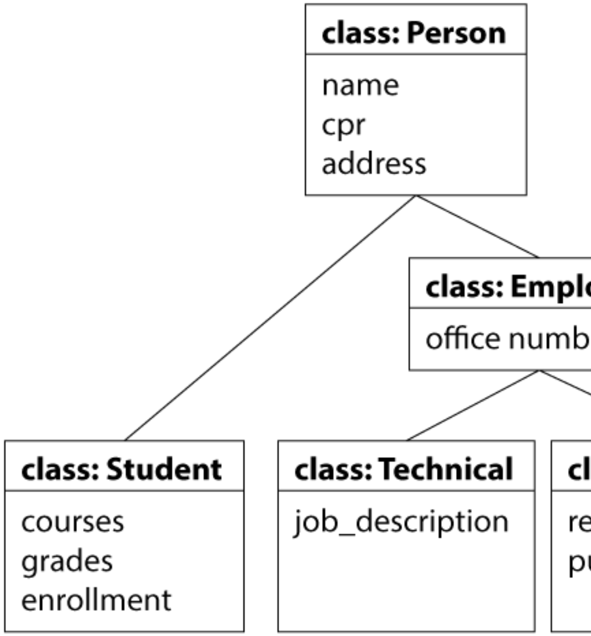
Inheritance in Python

To let a class B inherit from a class A:

```
class B(A):
    class definition
```

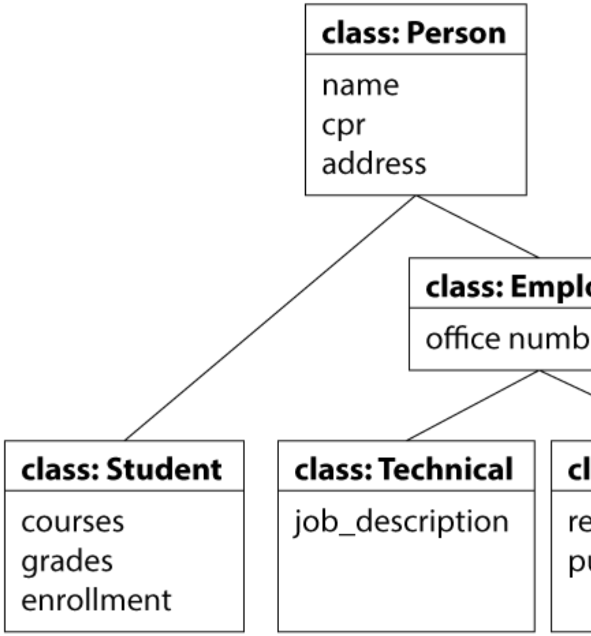
Inheritance in Python - for our data

```
class Person:
    # Attributes: name, cpr, address

class Employee(Person):      # Inherits from Person
    # Attributes: office

class Student(Person):       # Inherits from Person
    # Attributes: courses, grades, enrollment_data

class Technical(Employee):   # Inherits from Employee
    # Attributes: job_description

class Researcher(Employee): # Inherits from Employee
    # Attributes: research_interests, publications
```



Inheritance in Python - for our data

```
class Person:
    # Attributes: name, cpr, address

class Employee(Person):      # Inherits from Person
    # Attributes: office

class Student(Person):       # Inherits from Person
    # Attributes: courses, grades, enrollment_data

class Technical(Employee):   # Inherits from Employee
    # Attributes: job_description

class Researcher(Employee): # Inherits from Employee
    # Attributes: research_interests, publications
```



Note that I omitted the pass statements for brevity

Exercise 3

1. Implement a constructor for the Person class.
2. Implement a constructor for the Student class.

Exercise 3 - Partial solution

```python
class Person:
    def __init__(self, name, cpr, address):
        self.name = name
        self.cpr = cpr
        self.address = address

class Student(Person):
    def __init__(self, name, cpr, address, courses, grades, enrollment):
        self.courses = courses
        self.grades = grades
        self.enrollment = enrollment
        # What about the name, cpr and address parameters?

albert = Student(name="Albert Einstein", cpr="14031879-1235",
                 address="112 Mercer Street, Princeton",
                 courses=["Linear Algebra", "Relativity"],
                 grades=["B","A"], enrollment=1895)
```

Exercise 3 - Partial solution

```
class Person:
    def __init__(self, name, cpr, address):
        self.name = name
        self.cpr = cpr
        self.address = address

class Student(Person):
    def __init__(self, name, cpr, address, courses, grades, enrollment):
        self.courses = courses
        self.grades = grades
        self.enrollment = enrollment
        # What about the name, cpr and address parameters?

albert = Student(name="Albert Einstein", cpr="14031879-1235",
                 address="112 Mercer Street, Princeton",
                 courses=["Linear Algebra", "Relativity"],
                 grades=["B","A"], enrollment=1895)
```

What about the name, cpr, and address parameters in the Student constructor?

Calling the base-class constructor

To pass parameters to the base class, you will need to call the constructor of the base class explicitly

base_class_example.py

```
class A:
    def __init__(self):
        print("Initializing A")

class B(A):     # inherits from A
    def __init__(self):
        A.__init__(self)    # Call constructor of base class
        print("Initializing B")

b = B()
```

terminal

```
$ python base_class_example.py
Initializing A
Initializing B
```

Exercise 4

- Can you solve the previous exercise now?

Exercise 4 - solution

```
class Person:
    def __init__(self, name, cpr, address):
        self.name = name
        self.cpr = cpr
        self.address = address

class Student(Person):
    def __init__(self, name, cpr, address, courses, grades, enrollment):
        Person.__init__(self, name, cpr, address) # Base class constructor
        self.courses = courses
        self.grades = grades
        self.enrollment = enrollment

albert = Student(name="Albert Einstein", cpr="14031879-1235",
                 address="112 Mercer Street, Princeton",
                 courses=["Linear Algebra", "Relativity"],
                 grades=["B","A"], enrollment=1895)
```

Object Oriented Programming: Composition

Modelling a group of students

Modelling a group of students

How do we model a group of students

```
albert = Student(name="Albert Einstein",
                 cpr="14031879-1235",
                 address="112 Mercer Street, Princeton",
                 courses=["Linear Algebra", "Relativity"],
                 grades=["B","A"],
                 enrollment=1895)
niels = Student(name="Niels Bohr",
                cpr="07101885-7459",
                address="Carlsberg Æresbolig, Gamle Carlsbergvej, Valby",
                courses=[],
                grades=[],
                enrollment=1903)
```

Modelling a group of students

How do we model a group of students

```
albert = Student(name="Albert Einstein",
                 cpr="14031879-1235",
                 address="112 Mercer Street, Princeton",
                 courses=["Linear Algebra", "Relativity"],
                 grades=["B","A"],
                 enrollment=1895)
niels = Student(name="Niels Bohr",
                cpr="07101885-7459",
                address="Carlsberg Æresbolig, Gamle Carlsbergvej, Valby",
                courses=[],
                grades=[],
                enrollment=1903)
```

Idea: create a class that has a list of students as attribute

```
dream_team = StudentGroup([albert, niels])
```

Modelling a group of students

How do we model a group of students

```
albert = Student(name="Albert Einstein",
                 cpr="14031879-1235",
                 address="112 Mercer Street, Princeton",
                 courses=["Linear Algebra", "Relativity"],
                 grades=["B","A"],
                 enrollment=1895)
niels = Student(name="Niels Bohr",
                cpr="07101885-7459",
                address="Carlsberg Æresbolig, Gamle Carlsbergvej, Valby",
                courses=[],
                grades=[],
                enrollment=1903)
```

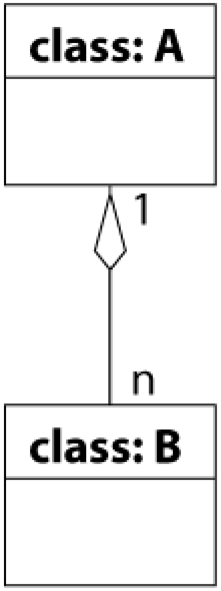
Idea: create a class that has a list of students as attribute

```
dream_team = StudentGroup([albert, niels])
```

An object that contains other objects is called *composition*.

Composition

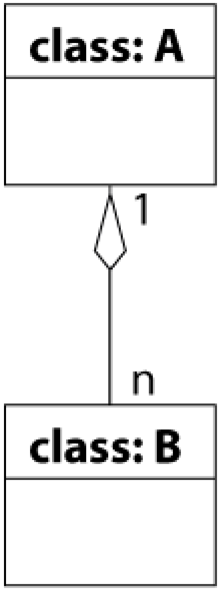
Consider a class A that is a composition of *n* instances of class B:



Composition

Consider a class A that is a composition of *n* instances of class B:

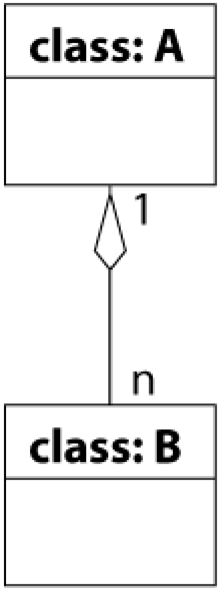
- Objects of type B are components within an object of type A.



Composition

Consider a class A that is a composition of *n* instances of class B:

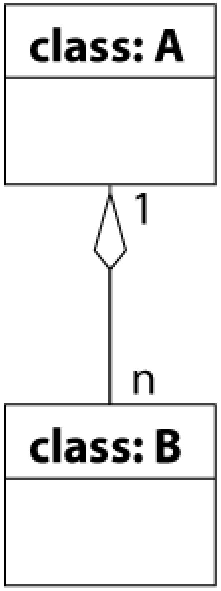
- Objects of type B are components within an object of type A.
- Think of the object A as consisting of smaller parts - each modelled by a separate class B



Composition

Consider a class A that is a composition of *n* instances of class B:

- Objects of type B are components within an object of type A.
- Think of the object A as consisting of smaller parts - each modelled by a separate class B
- Composition is a *has-a* relationship



Example

```python
class StudentGroup:
    def __init__(self, students=[]):
        '''Constructor. The list of students is optional'''
        self.students = students

    def add_student(self, student):
        '''Constructor. Add a new student to the group'''
        self.students.append(student)
```

OOP - composition exercise

- Make sure you have a working Student class (you can copy it from the slides), so you can create Student objects like this

```
albert = Student(name="Albert Einstein",
                 cpr="14031879-1235")       # or with more arguments
```

- Add a get_names() method to the StudentGroup class example on the previous slide. The method should iterate over all students in the group and return a list of their names.

OOP - composition exercise - solution

```
class StudentGroup:

    # ... constructor and add_student() methods omitted

    def get_names(self):
        '''Return list of names of students in group'''
        names = []
        for student in self.students:
            names.append(student.name)
        return names

# Create student objects
albert = Student(name="Albert Einstein", cpr="14031879-1235")
niels = Student(name="Niels Bohr", cpr="07101885-7459")

# Create student group object
group = StudentGroup(students=[albert, niels])

# Call get_names method
print(group.get_names())
```

```
['Albert Einstein', 'Niels Bohr']
```