CoPL
Object Oriented Programming

Crashcourse

Daan Wichmann

Radboud University

02 04 2025

Before we start

- Lot's of topics to cover, limited time per topic
- Assuming basic Python knowledge from Programming 1 & Programming 2.

Objects & Methods

What are objects?

What are objects?

Objects

Objects are Python's abstraction for data. All data in a Python program is represented by objects or by relations between objects.

What are objects?

What are objects?

The following are objects in Python:

What are objects?

The following are objects in Python:

- Integers (e.g. 127)

What are objects?

The following are objects in Python:

- Integers (e.g. 127)
- Strings (e.g. "CognAC")

What are objects?

The following are objects in Python:

- Integers (e.g. 127)
- Strings (e.g. "CognAC")
- Lists (e.g. ["Damai", "Daan", "Ella"])

What are objects?

The following are objects in Python:

- Integers (e.g. 127)
- Strings (e.g. "CognAC")
- Lists (e.g. ["Damai", "Daan", "Ella"])
- **Every value in Python!**

What are methods?

What are methods?

Methods

Methods are functions that belong to a certain **type of object**.

What are methods?

Methods

Methods are functions that belong to a certain **type of object**.

- You call methods on an object.

What are methods?

Methods

Methods are functions that belong to a certain **type of object**.

- You call methods on an object.
- Methods that belong to a certain type of object, can not be called on another type of object

What are methods?

You have already seen a lot of methods!

What are methods?

You have already seen a lot of methods!

```
my_list = [1, 2, 3]
```

What are methods?

You have already seen a lot of methods!

```
my_list = [1, 2, 3]
my_list.append(4)
```

What are methods?

You have already seen a lot of methods!

```
my_list = [1, 2, 3]
my_list.append(4)
print(my_list)
>>> [1, 2, 3, 4]
```

What are methods?

You have already seen a lot of methods!

```
my_list = [1, 2, 3]
my_list.append(4)
print(my_list)
>>> [1, 2, 3, 4]
```

append is a **method** of lists.

What are methods? Another example.

What are methods? Another example.

```
my_string = "daan"   # Note: all lowercase
```

What are methods? Another example.

```
my_string = "daan"  # Note: all lowercase
my_capitalized_string = my_string.capitalize()
```

What are methods? Another example.

```
my_string = "daan"  # Note: all lowercase
my_capitalized_string = my_string.capitalize()
print(my_capitalized_string)
>>> "Daan"
```

What are methods? Another example.

```
my_string = "daan"  # Note: all lowercase
my_capitalized_string = my_string.capitalize()
print(my_capitalized_string)
>>> "Daan"
```

capitalize is a **method** of strings.

What are methods?

What are methods?

```
my_list = [1, 2, 3]
```

What are methods?

```
my_list = [1, 2, 3]
my_list.capitalize()
```

What are methods?

```
my_list = [1, 2, 3]
my_list.capitalize()
>>> AttributeError
```

What are methods?

```
my_list = [1, 2, 3]
my_list.capitalize()
>>> AttributeError
```

Why? *'capitalize' is not a method of 'list'*

Classes

What are classes?

What are classes?

Class (definition)

A class is a blueprint for a type of object, defining it's **data attributes** and **methods**.

What are classes?

Class (definition)

A class is a blueprint for a type of object, defining it's **data attributes** and **methods**.

Terminology

We call an object from a certain class, an **instance** of that class. E.g. "OOP" is an instance of the class str (string).

Why do we use classes?

Why do we use classes?

Why do we use classes?

- Python only gives us a limited amount of types, integers, floats, strings, etcetera.

Why do we use classes?

- Python only gives us a limited amount of types, integers, floats, strings, etcetera.
- With classes we can extend this and model anything we want!

Why do we use classes?

- Python only gives us a limited amount of types, integers, floats, strings, etcetera.
- With classes we can extend this and model anything we want!
  - Like students, bank accounts, courses, shopping cart.

Why do we use classes?

- Python only gives us a limited amount of types, integers, floats, strings, etcetera.
- With classes we can extend this and model anything we want!
  - Like students, bank accounts, courses, shopping cart.
- Encapsulation (later)

Why do we use classes?

- Python only gives us a limited amount of types, integers, floats, strings, etcetera.
- With classes we can extend this and model anything we want!
  - Like students, bank accounts, courses, shopping cart.
- Encapsulation (later)
- Seperation of Concerns (later)

What do classes consist of?

What do classes consist of?

- Methods (behaviour)

What do classes consist of?

- Methods (behaviour)
- Data Attributes (properties/fields)

What do classes consist of?

What do classes consist of?

Data Attributes:

What do classes consist of?

Data Attributes:

- Variables that belong to a single object

What do classes consist of?

Data Attributes:

- Variables that belong to a single object
- For example a person can have the following data variables (i.e. data attributes) which we would like to store:

What do classes consist of?

Data Attributes:

- Variables that belong to a single object
- For example a person can have the following data variables (i.e. data attributes) which we would like to store:
  - first name (string)
  - last name (string)
  - age (int)

What do classes consist of?

Data Attributes:

- Variables that belong to a single object
- For example a person can have the following data variables (i.e. data attributes) which we would like to store:
  - first name (string)
  - last name (string)
  - age (int)
- These variables can be of any type! (Strings, integers, lists, and also objects of other classes)

Methods

- You have already seen a lot of methods

Methods

- You have already seen a lot of methods
- For example, for a person we could have:

Methods

- You have already seen a lot of methods
- For example, for a person we could have:
  - Speak
  - Dance
  - Study

Methods

- You have already seen a lot of methods
- For example, for a person we could have:
  - Speak
  - Dance
  - Study
- Every class needs one special method:

Methods

- You have already seen a lot of methods
- For example, for a person we could have:
  - Speak
  - Dance
  - Study
- Every class needs one special method:
  - **The constructor**

What is a constructor?

What is a constructor?

Constructor

In Python, a **constructor** is a special method called when an object is created. It initializes all data attributes.

The constructor

To create objects of a certain class:

The constructor

To create objects of a certain class:

```
my_string = "CognAC"  # Class str
my_int = 18  # Class int
my_list = [18, 2003]  # Class list
```

The constructor

To create objects of a certain class:

```
my_string = "CognAC"  # Class str
my_int = 18  # Class int
my_list = [18, 2003]  # Class list
```

But what if we have a new type of object of a new class, for example a Fraction class

The constructor

To create objects of a certain class:

```
my_string = "CognAC"  # Class str
my_int = 18  # Class int
my_list = [18, 2003]  # Class list
```

But what if we have a new type of object of a new class, for example a Fraction class *We use the constructor!*

The constructor

```
# We import the class from the module 'fractions'
# Classes always start with a capital lette
from fractions import Fraction
```

The constructor

```python
# We import the class from the module 'fractions'
# Classes always start with a capital lette
from fractions import Fraction
# We call the constructor by 'calling the class',
# this creates a new object for us!
# The constructor takes two arguments:
# numerator and denominator (in this case 8 and 10)
my_fraction = Fraction(8, 10)  # Fraction of 8/10th
```

Making our own class!

Making our own class!

- We now have all ingredients to create our own classes!

Making our own class!

- We now have all ingredients to create our own classes!
- Let's say we want to model a person in python

Making our own class!

- We now have all ingredients to create our own classes!
- Let's say we want to model a person in python
- We want it to have a *first name*, *last name*, *age* (data attributes)

Making our own class!

- We now have all ingredients to create our own classes!
- Let's say we want to model a person in python
- We want it to have a *first name*, *last name*, *age* (data attributes)
- And we want a person to 'speak' i.e. print some text (methods)

Making our own class!

- We now have all ingredients to create our own classes!
- Let's say we want to model a person in python
- We want it to have a *first name*, *last name*, *age* (data attributes)
- And we want a person to 'speak' i.e. print some text (methods)
- We create our own class **Person**

Creating our Person class

```
class Person():
```

Creating our Person class

```
class Person():

    # The constructor
    def __init__(self, first_name, last_name, age):
        self.first_name = first_name  # Data attribute
        self.last_name = last_name  # Data attribute
        self.age = age  # Data attribute
```

Creating our Person class

```python
class Person():

    # The constructor
    def __init__(self, first_name, last_name, age):
        self.first_name = first_name  # Data attribute
        self.last_name = last_name  # Data attribute
        self.age = age  # Data attribute

    # Method
    def speak(self):
        print(f"Hey, my name is {self.first_name}")
```

The 'self' parameter

The 'self' parameter

- The first parameter in a constructor definition is always named **self**.

The 'self' parameter

- The first parameter in a constructor definition is always named **self**.
- This refers to the object itself, and is necessary for declaring any data attributes attached to the object.

The 'self' parameter

- The first parameter in a constructor definition is always named **self**.
- This refers to the object itself, and is necessary for declaring any data attributes attached to the object.
- It is automatically filled in by the interpreter with the object it is called on.

Creating our Person class

```python
class Person():

    # The constructor
    def __init__(self, first_name, last_name, age):
        self.first_name = first_name  # Data attribute
        self.last_name = last_name  # Data attribute
        self.age = age  # Data attribute

    # Method
    def speak(self):
        print(f"Hey, my name is {self.first_name}")
```

Data Attributes

Accessing a data attribute:

Data Attributes

Accessing a data attribute:

```
person = Person("Daan", "Wichmann", "Secret")
```

Data Attributes

Accessing a data attribute:

```
person = Person("Daan", "Wichmann", "Secret")
print(person.first_name)
>>> "Daan"
```

Data Attributes

Re-assigning an attribute:

Data Attributes

Re-assigning an attribute:

```
person = Person("Daan", "Wichmann", "Secret")
```

Data Attributes

Re-assigning an attribute:

```
person = Person("Daan", "Wichmann", "Secret")
person.first_name = "Damai"
print(person.first_name)
>>> "Damai"
```

Our beautiful class

```python
class Person():

    # The constructor
    def __init__(self, first_name, last_name, age):
        self.first_name = first_name  # Attribute
        self.last_name = last_name  # Attribute
        self.age = age  # Attribute

    # Method
    def speak(self):
        print(f"Hey, my name is {self.first_name}")
```

Calling our method

Calling our method

```
person = Person("Daan", "Wichmann", "Secret")
```

Calling our method

```
person = Person("Daan", "Wichmann", "Secret")
person.speak()
>>> Hey, my name is Daan
# Note that we don't fulfill the 'self'
# parameter, but the interpreter
# does it automatically for us.
```

Why?

Why do we want to use classes?

- If we want to to represent something that is not already in Python
- To make collaboration easier
- If we want to 'group together' functionality

Objects & References

What do our variables store exactly?

References

```
person1 = Person("Daan", "Wichmann", "Secret")
```

References

```
person1 = Person("Daan", "Wichmann", "Secret")
person2 = person1
```

References

```
person1 = Person("Daan", "Wichmann", "Secret")
person2 = person1
person3 = Person("Damai", "Jiworo", "Secret")
```

Memory model

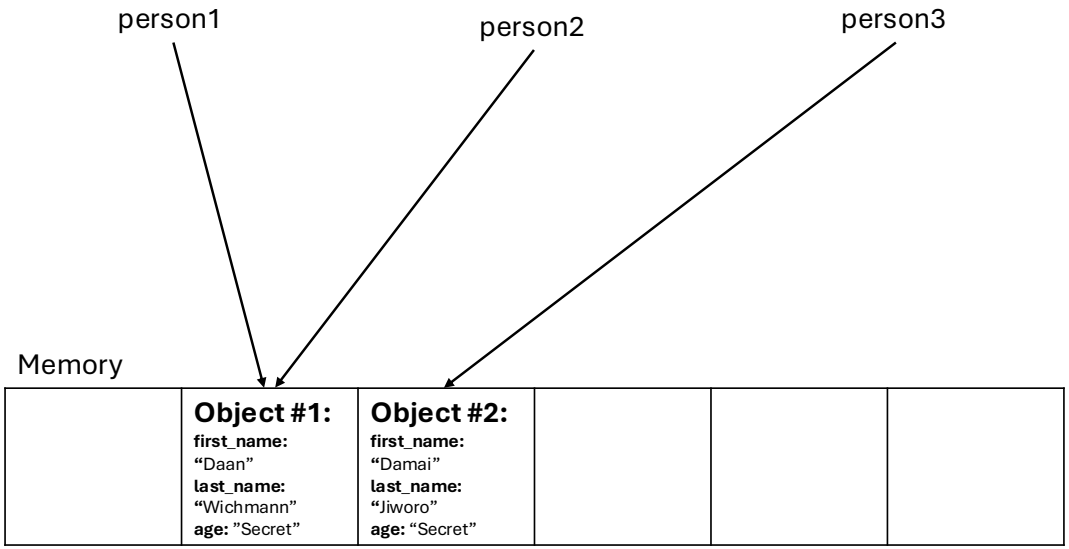

Figure 1: Simplified memory model

Small quiz

```
person1 = Person("Damai", "Jiworo", "Secret")
```

Small quiz

```
person1 = Person("Damai", "Jiworo", "Secret")
person2 = person1
```

Small quiz

```
person1 = Person("Damai", "Jiworo", "Secret")
person2 = person1
person3 = Person("Daan", "Wichmann", "Secret")
```

Small quiz

```
person1 = Person("Damai", "Jiworo", "Secret")
person2 = person1
person3 = Person("Daan", "Wichmann", "Secret")
person2.first_name = "Emma"
```

Small quiz

```
person1 = Person("Damai", "Jiworo", "Secret")
person2 = person1
person3 = Person("Daan", "Wichmann", "Secret")
person2.first_name = "Emma"
print(person1.first_name)  # What is the output?
```

Small quiz

```
person1 = Person("Damai", "Jiworo", "Secret")
person2 = person1
person3 = Person("Daan", "Wichmann", "Secret")
person2.first_name = "Emma"
print(person1.first_name)  # What is the output?
>>> "Emma"
```

Another one

```
person1 = Person("Damai", "Jiworo", "Secret")
```

Another one

```
person1 = Person("Damai", "Jiworo", "Secret")
person2 = Person("Damai", "Jiworo", "Secret")
```

Another one

```
person1 = Person("Damai", "Jiworo", "Secret")
person2 = Person("Damai", "Jiworo", "Secret")
person2.first_name = "Emma"
```

Another one

```
person1 = Person("Damai", "Jiworo", "Secret")
person2 = Person("Damai", "Jiworo", "Secret")
person2.first_name = "Emma"
print(person1.first_name)  # What is the output?
```

Another one

```
person1 = Person("Damai", "Jiworo", "Secret")
person2 = Person("Damai", "Jiworo", "Secret")
person2.first_name = "Emma"
print(person1.first_name)  # What is the output?
>>> "Damai"
```

Part 1 of 3 done! Break?