

CoPL

OBJECT ORIENTED PROGRAMMING

CRASHCOURSE

DAAN WICHMANN

RADBOD UNIVERSITY

02 04 2025

- 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 its **data attributes** and **methods**.

WHAT ARE CLASSES?

Class (definition)

A class is a blueprint for a type of object, defining its **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)
- Separation 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)

- You have already seen a lot of methods

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

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

- 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:

- 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.

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 letter  
from fractions import Fraction
```

THE CONSTRUCTOR

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

```
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 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

```
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}")
```

Accessing a data attribute:

Accessing a data attribute:

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

Accessing a data attribute:

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


Re-assigning an attribute:

Re-assigning an attribute:

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

Re-assigning an attribute:

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

OUR BEAUTIFUL CLASS

```
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}")
```



```
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?

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

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

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

MEMORY MODEL

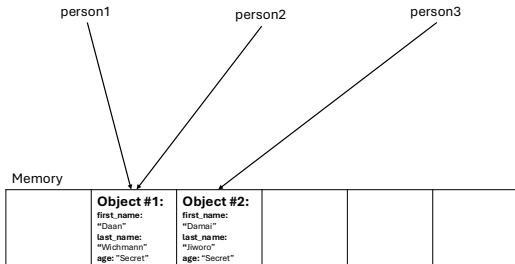


Figure 1: Simplified memory model

SMALL QUIZ

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

SMALL QUIZ

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


SMALL QUIZ

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

SMALL QUIZ

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

SMALL QUIZ

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

ANOTHER ONE

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

ANOTHER ONE

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

ANOTHER ONE

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


ANOTHER ONE

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

Part 1 of 3 done! Break?