

INF1-IP

Lecture 7 – Objects 1: Classes and instances

Rayo Verweij

12/10/2026



> Methods

Methods are functions that are associated with a particular object.

We have already seen a couple of methods that work on strings:

```
up = "hello".upper()  
clean = "  with some spaces  ".strip()
```

Which methods are available depends on the object's **type**.

 nonsense = 10.upper() *# Throws AttributeError*

> Classes and instances

In Python, every piece of data is some sort of **object**.

Each object has a **type** (e.g. `str`, `int`, `list`).

The type of an object determines:

1. The **values** it can hold
2. The **methods** that are available

The specification of a type is called a **class**.

Objects that follow this specification are **instances** of the class.

For example,

```
food = "potato"
```

The **variable** *food* refers to an **object** of **type** *str* that holds the **value** *"potato"*.

This means that *str* **methods** such as *.upper()* are available for use.

In other words, *"potato"* is an **instance** of the *str* **class**.

> Creating classes

Create classes using the `class` keyword.

By convention, class names are capitalised.

```
class Course:  
    name = "Intro to Programming"
```

```
ip = Course()  
print(ip.name)
```

→ *Intro to Programming*

> Class instantiation

Inside classes we can define data and methods.

The **constructor** method `__init__()` runs when the class is created.

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

ip = Course("Intro to Programming", 1)
sepp = Course("Software Engineering and Professional Practice", 2)

print(ip.name)      # Intro to Programming
print(sepp.year)    # 2
```

Note: `__init__()` uses **2** underscores on each side!

> Class vs instance attributes

Class attributes are shared between all instances of a class.

Instance attributes are local to a particular instance.

```
class Course:
    school = "Informatics"

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

ip = Course("Intro to Programming", 1)
sepp = Course("Software Engineering and Professional Practice", 2)

print(ip.school)          # Informatics
```

> Methods

In addition to `__init__()`, classes can have any number of methods. Each method receives the calling object as its first argument, *self*. *self* can also be used to refer to instance attributes.

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

    def code(self, year):
        return self.name[0] + self.name[1] + str(year)

ip = Course("Intro to Programming")
first_letter = ip.code(2026)

print(first_letter)      # In2026
```

{ Try it yourself #1

Write a Parrot class.

Give it:

- Instance attributes name, age, and colour
- Class attributes `animal_class` (bird) and `can_fly` (true)
- A `speak` method that takes a sentence and returns a string of the format *<parrot's name> says: <sentence>!*
- A `fly` method that checks whether `can_fly` is True and if so, prints a message that the parrot is flying

Then create your own parrot and make it speak.

{ Example solution

```
class Parrot:
    animal_class = "bird"
    can_fly = True

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

    def speak(self, sentence):
        return f"{self.name} says: {sentence}!"

    def fly(self):
        if self.can_fly:
            print(f"{self.name} is flying!")
        else:
            print(f"{self.name} is grounded...")

horatio = Parrot("Horatio", 33, "green")
print(horatio.speak("Good-night, sweet prince"))
```



Common mistakes with writing methods

Forgetting *self*.

A name without *self* is just a variable local to the method!

Forgetting parentheses.

When calling a function or method, even if it does not have input, you need to write the parentheses.

Forgetting double underscores.

`__init__` takes two!

> Changing instance attributes

Methods may change the attributes of an object.

```
class Parrot:
    can_fly = True

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

    def birthday(self):
        self.age += 1

horatio = Parrot("Horatio", 33)
horatio.birthday()
print(horatio.age)      # 34
```

> Changing class attributes

What would happen if you try to update a class attribute?

```
class Parrot:
    can_fly = True

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

    def ground(self):
        self.can_fly = False

horatio = Parrot("Horatio", 33)
bob = Parrot("Bob", 1)

horatio.ground()
print(bob.can_fly)      # ???
```

> Changing class attributes

Rather than updating the class attribute, we created an instance attribute *with the same name* that was specific to `horatio`.

If you want to update a class attribute, you have to do it using the class name directly:

```
Parrot.can_fly = False
```

When objects have class and instance attributes with the same name, the instance attributes always take precedence.

> Deleting attributes

How do we then get rid of instance attributes, so we can revert to the default behaviour?

The `del` keyword allows us to delete names.

```
del horatio.can_fly
```

Now the instance attribute has been removed and `horatio.can_fly` once again checks the class attribute.

`del` works for variables as well, though has less use there.

> Describing objects

By default, calling `print()` or `str()` on an object just reports basic info:

```
print(horatio)           # <__main__.Parrot object at 0x00000197FCC13230>
```

We can **define our own string representation** with `__str__()`:

```
class Parrot:
    ...

    def __str__(self):    # __str__ needs to return a string
        return f"A parrot named {self.name}"

horatio = Parrot("Horatio", 33, "green")
print(horatio)           # A parrot named Horatio
```

{ Try it yourself #2 (extra time edition)

Write a Car class.

Give it:

- Instance attributes `model`, `age`, and `max_speed`
- Class attribute `wheels` (4)
- A string representation of your choice

Create two cars and print descriptions of each.

Update one car to be a limousine with 8 wheels instead and print it.

Then, revert it back to a normal car with 4 wheels and print again.

> Summary

In this lecture, we have...

- Understood the relationship between a class and its instances
- Defined classes and set initial values with `__init__`
- Read and changed instance attributes through methods
- Chosen between class attributes and instance attributes
- Deleted attributes and variables using `del`
- Used `__str__` to give an object a readable description

> Good luck on Friday!

Make sure to **work through the worksheets** and **bring your laptop!**

If you need help...

- Ask questions on EdStem!
- Drop in to InfPALS!
- Check out the extra practice material on OpenCourse!
- Work with your friends!
- Talk to your Student Adviser or the Advice Place!