Courses (/student) / Web Development 1 (Wien) (/student/course/5727275479728128) / 13) Object-oriented programming (OOP)

13) Object-oriented programming (OOP)

Agenda

- Data structures
- What is OOP?
- Model and object
- Inheritance

What we've learned so far

So far we've learned about:

a) Primitive data types

A data type that can hold **one** piece of data.

Example:

```
some_name = "Nina"  # string
age = 25  # integer
human = True  # boolean
```

b) Lists and dictionaries

Data structures, that can hold **many** pieces of data.

Example:

```
us_cities_list = ["Boston", "New York", "Los Angeles", "San Francisco"]

emails_dict = {"john.smith@yahoo.com": "John Smith", "janedoe@gmail.com": "Jane Doe", "fluffy-p
```

A new, revolutionary data structure

But today will learn about a new data structure, much more powerful than the ones we learned about so far.

This data structure is called: **an object.** The invention of this data structure was so **revolutionary**, that it changed the way we write programs. We call this way of writing code: **object-oriented programming** (OOP).

> *The OOP has been widely used since the 80s, but it's still the main programming paradigm.*

We've all used OOP before...

... we just didn't know it was called OOP. :)

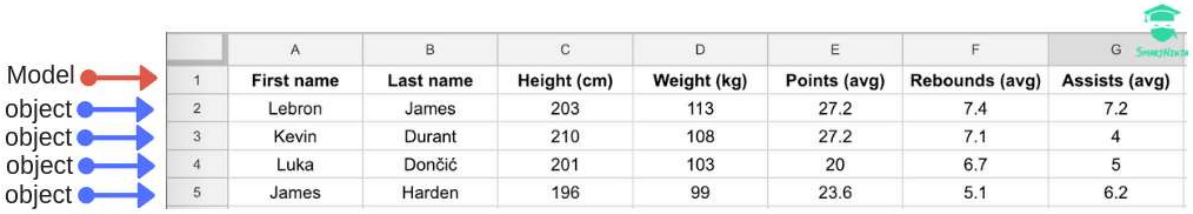
For example, when you create a table in Excel, you're using the OOP paradigm.

Let's say we want to create a list of **basketball players** in our Excel table. How would we start?

1) First we'd create a table header, that would include these fields:

- First name
- Last name
- Height (in centimeters)
- Weight (in kilograms)
- Average number of points
- Average number of rebounds
- Average number of assists

2) Then we'd starting adding data in the table, each player in their own row.



	A	B	C	D	E	F	G
1	First name	Last name	Height (cm)	Weight (kg)	Points (avg)	Rebounds (avg)	Assists (avg)
2	Lebron	James	203	113	27.2	7.4	7.2
3	Kevin	Durant	210	108	27.2	7.1	4
4	Luka	Dončić	201	103	20	6.7	5
5	James	Harden	196	99	23.6	5.1	6.2

This gives us the two most important **concepts** in OOP:

- Model, and
- object

The table **header** is a **model** and each of the **following rows** is an **object.** The data of each object is structured based on the model.

Let's write this example with Python!

Model

First let's write the Python code for our model. The keyword that we'll use here is **class** (consider *class* and *model* as synonyms):

```
class BasketballPlayer():
    def __init__(self, first_name, last_name, height_cm, weight_kg, points, rebounds, assists):
        self.first_name = first_name
        self.last_name = last_name
        self.height_cm = height_cm
        self.weight_kg = weight_kg
        self.points = points
        self.rebounds = rebounds
        self.assists = assists
```

Okay, this code looks a bit weird... What does it mean?

The first line defines the name of our model (or class). We gave it a name `BasketballPlayer`.

The second line is a function that **initializes** a new object. We'll call this function when we'll want to **create a new object**.

Let's see how this looks like:

```
lebron = BasketballPlayer(first_name="Lebron", last_name="James", height_cm=203, weight_kg=113,

kev_dur = BasketballPlayer(first_name="Kevin", last_name="Durant", height_cm=210, weight_kg=108
```

As you can see, we didn't have to call the `__init__` function (or method) by its name. This is an in-built method that is automatically called when we create a new object.

Let's print some data from our new objects:

```
print(lebron.first_name)
print(lebron.height_cm)

print(kev_dur.last_name)
print(kev_dur.rebounds)

# list of players
bball_players = [lebron, kev_dur]

for player in bball_players:
    print(player.last_name + ", " + player.first_name)
```

As you can see, we can store many data in an object and easily access this data.

How is this different from a dictionary?

We could easily do that also with a dictionary!

That's true. We could create a dictionary like this:

```
lebron_dict = {"first_name": "Lebron", "last_name": "James", "height_cm": 203, "weight_kg": 113
```

and access its data like this:

```
print(lebron_dict["first_name"])
print(lebron_dict["height_cm"])
```

BUT! Objects can do **much more** than just store many pieces of data.

Objects can also have its own **functions** which can manipulate the data. We call these functions: **methods**.

Methods

An ability to create **custom functions** (methods) inside objects is what makes objects **different** and more **powerful** than dictionaries.

Let's see how this looks like.

For example, let's say we want to have a method that would **automatically convert kilograms into pounds** (lbs). Let's add this method into our class:

```
class BasketballPlayer():
    def __init__(self, first_name, last_name, height_cm, weight_kg, points, rebounds, assists):
        self.first_name = first_name
        self.last_name = last_name
        self.height_cm = height_cm
        self.weight_kg = weight_kg
        self.points = points
        self.rebounds = rebounds
        self.assists = assists

    def weight_to_lbs(self):
        pounds = self.weight_kg * 2.20462262
        return pounds
```

We've created a method called `weight_to_lbs()` . One kilogram equals 2.20462262 lbs, that's why we multiplied the weight in kg with this number.

Let's try this out:

```
print(lebron.weight_to_lbs())
print(kev_dur.weight_to_lbs())
```

Voila! Our method easily converted kilograms into pounds. Yaaay!

> *Make sure you add parenthesis at the end of a function call:* `weight_to_lbs()`*.*
>
> *The* `self` *keyword simply means* ***this exact object*** *whose data we are manipulating right now.*

But methods are not the only thing that makes OOP special. There's much more that just that, but for let's learn about one last thing about objects: **inheritance**.

Inheritance

Let's say we now want to have a **list of football players** (or **soccer** players, if you're from the USA).

We want our class to accept the following data:

- First name
- Last name
- Height
- Weight
- Goals
- Yellow cards
- Red cards

FootballPlayer class

The FootballPlayer class would look something like this:

```
class FootballPlayer():
    def __init__(self, first_name, last_name, height_cm, weight_kg, goals, yellow_cards, red_ca
        self.first_name = first_name
        self.last_name = last_name
        self.height_cm = height_cm
        self.weight_kg = weight_kg
        self.goals = goals
        self.yellow_cards = yellow_cards
        self.red_cards = red_cards
```

Let's create a couple of objects now:

```
ronaldo = FootballPlayer(first_name="Cristiano", last_name="Ronaldo", height_cm=184, weight_kg=

messi = FootballPlayer(first_name="Lionel", last_name="Messi", height_cm=170, weight_kg=67, goa

print(ronaldo.first_name)
print(ronaldo.goals)
print(messi.first_name)
print(messi.goals)
```

As we can see, some types of the data about football players are the same as for basketball players: first name, last name, height and weight.

Which means we should follow the DRY approach.

DRY

DRY means **don't repeat yourself**. It's a good coding practice that reduces the amount of code you have to write and it makes your code better maintainable. Instead of repeating your code, try to reuse it.

How can we reuse code in OOP?

By using **inheritance**. In OOP classes can **inherit** code from **other** classes.

Let's create a new model called Player:

```
class Player():
    def __init__(self, first_name, last_name, height_cm, weight_kg):
        self.first_name = first_name
        self.last_name = last_name
        self.height_cm = height_cm
        self.weight_kg = weight_kg

    def weight_to_lbs(self):
        pounds = self.weight_kg * 2.20462262
        return pounds
```

As you can see, we moved all the code that `BasketballPlayer` and `FootballPlayer` can share in the `Player` class.

Now let's adapt our `BasketballPlayer` and `FootballPlayer` models so they inherit the code from the `Player` model:

```
class BasketballPlayer(Player):
    def __init__(self, first_name, last_name, height_cm, weight_kg, points, rebounds, assists):
        super().__init__(first_name=first_name, last_name=last_name, height_cm=height_cm, weigh
        self.points = points
        self.rebounds = rebounds
        self.assists = assists


class FootballPlayer(Player):
    def __init__(self, first_name, last_name, height_cm, weight_kg, goals, yellow_cards, red_ca
        super().__init__(first_name=first_name, last_name=last_name, height_cm=height_cm, weigh
        self.goals = goals
        self.yellow_cards = yellow_cards
        self.red_cards = red_cards
```

Do you think this approach will work? Not sure? Let's try it out!

```
kev_dur = BasketballPlayer(first_name="Kevin", last_name="Durant", height_cm=210, weight_kg=108

print(kev_dur.last_name)
print(kev_dur.rebounds)
print(kev_dur.weight_to_lbs())

messi = FootballPlayer(first_name="Lionel", last_name="Messi", height_cm=170, weight_kg=67, goa

print(messi.first_name)
print(messi.goals)
print(messi.weight_to_lbs())
```

It works! The `BasketballPlayer` and `FootballPlayer` classes successfully inherited attributes and a method from the `Player` class.

Class and object

The relationship between a class and an object is similar to a rubber **ink stamp** and the result of using it on a paper.

The **ink stamp** is like a **class**. And each time you **stamp it** on a paper you **create a new object**.

To sum up

Objects are **similar** to lists and dictionaries in a sense that they can hold **multiple pieces** of data.

But objects/classes can be **more useful** than dictionaries, because they can have its own **functions** (methods) and can **inherit** methods from other classes.

Q&A

Any question?

If there's enough time after Q&A, students can start working on their homework.

Homework 13.1

Let's make our program for basketball and football players a bit more useful. Add an option that a user can enter data (via `input()`) and at the end of the program the data gets saved in a text file (using `json` library).

> ***Hint:*** *you can easily convert object into a dictionary via the in-built* `__dict__` *method. Try this in your program:* `lebron.__dict__` *(note that there are 2 underscores on each side).*

Use the knowledge that you gained while building the Guess the secret number game.

Homework 13.2: Model for "Guess the secret number" results

Design your "Guess the secret number" game in a way that every result will be stored as an object.

Create a model called Result, that takes the following data:

- score
- player_name
- date

Then save the objects via json into a `results.txt` file (use the `__dict__` method, like in the previous homework).