Week 3:
Advanced Python

1. **Resources**
2. **Object-Oriented Programming**
3. **Iterators**
4. **Exceptions**
5. **Comprehensions**
6. **Outlook**

Resources

- Official Tutorial on Classes: **https://docs.python.org/3.7/tutorial/classes.html**
- Official Docs on Data Model**: https://docs.python.org/3.7/reference/datamodel.html**
- Official Docs on Exceptions: **https://docs.python.org/3.7/library/exceptions.html**
- Official Tutorial on (List-) Comprehensions:

  https://docs.python.org/3.7/tutorial/datastructures.html#list-comprehensions
- Video Tutorial on Python OOP: **https://www.youtube.com/watch?v=JeznW_7DlB0**
- Basic Programming in Python 2019 Slides on OOP

Object-Oriented Programming (OOP)

Everything is an object in Python.

However, **making your own custom classes and objects** is completely optional.

Objects have attributes that can be accessed with the syntax **object.attribute**.

Ways in which you have already used OOP:

- list.append()
- dict.items()
- len(list) → calls list.__len__() internally

Dunder Methods

Double **under**score methods or **magic methods** start with "__" and control internal functions.

- **len(x)** calls **x.__len__()**
- **str(x)** calls **x.__str__()**
- **a + b** calls **a.__add__(b)**
- **a * b** calls **a.__mul__(b)**
- **a < b** calls **a.__lt__(b)**
- **list(x)** calls **list.__init__(x)**

→ Find many more on **https://docs.python.org/3.7/reference/datamodel.html**

Custom Classes

```
class Person:

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

    def __str__(self):
        return "I am " + self.name + "!"


class Wizard(Person):          # class Wizard inherits from class Person

    def __init__(self, name, allegiance="fellowship"):

        super().__init__(name) # call the inherited constructor of parent

        self.allegiance = allegiance


    def cast_spell(self):
        print("Fabulous fireworks ensue ...")


gandalf = Wizard("Gandalf")

print(gandalf)
gandalf.cast_spell()
```

```
I am Gandalf!
Fabulous fireworks ensue ...
```

Iterators

```
class MyRange:

    def __init__(self, start, end):

        self.start = start
        self.end = end

    def __iter__(self):

        return self

    def __next__(self):

        if self.start < self.end:

            self.start += 1
            return self.start - 1

        else:

            raise StopIteration()


for item in MyRange(1, 7):
    print(item)
```

Exceptions

You can also catch all kinds of exceptions with help of a `try` ... `except` block:

```
my_dict = {
    "valid_key": "solved!"
}

try:

    print(my_dict["invalid_key"])

except KeyError:

    print("That key was not correct!")
```

```
That key was not correct!
```

Comprehensions

Comprehensions are a special kind of `for` -loop over a data structure that is usually contained in one line of code only.

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

new_list = []

for old_item in old_list:

    new_item = old_item + 8
    new_list.append(new_item)

print(new_list)
```

```
[9, 10, 11]
```

List Comprehension:

```
old_list = [1, 2, 3]
new_list = [item + 8 for item in old_list]
print(new_list)
```

```
[9, 10, 11]
```

Outlook

Next Steps:

1. **Open the Jupyter Notebook (**advanced_python_notebook.ipynb)
2. **Accept the homework assignment** (link will be in StudIP announcement)
3. **Complete the homework until Sunday at midnight** (2021-05-03 00:00:00+02:00)

Next Week: Matrix manipulations and more with NumPy

Enjoy the week!