

Introduction to Programming with Python

Dr. Anatol Wegner

Chair of Machine Learning for Complex Networks
Center for Artificial Intelligence and Data Science (CAIDAS)
Julius-Maximilians-Universität Würzburg
Würzburg, Germany

anatol.wegner@uni-wuerzburg.de

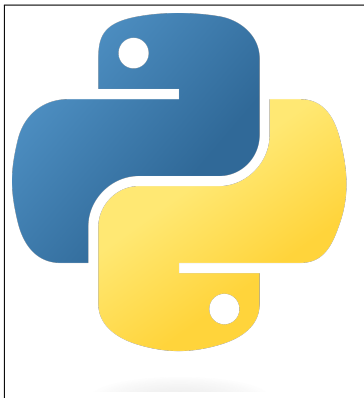
Lecture 04
Interacting with modules and files

November 22, 2024



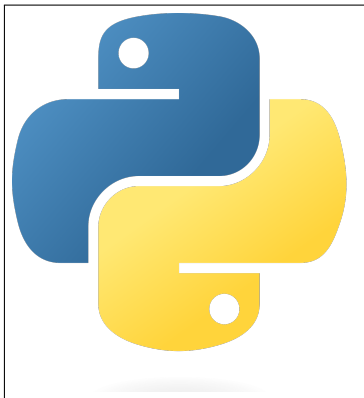
Recap

- ▶ We introduced **functions** in Python
- ▶ We introduced **classes** in Python and learned how to define them
- ▶ and class **attributes** and **methods**



Recap

- ▶ We introduced **functions** in Python
- ▶ We introduced **classes** in Python and learned how to define them
- ▶ and class **attributes** and **methods**



Today

- ▶ Modules
- ▶ Interacting with files

What Are Modules?

Definition:

- ▶ Modules are reusable pieces of Python code.
- ▶ They can be built-in or user-defined.
- ▶ Allow for organized and efficient code reuse.

Examples:

- ▶ Built-in: `math`, `os`, `random`.
- ▶ User-defined: A Python file you create with functions and classes.

Using Built-in Modules

How to Import a Module:

- ▶ `import module_name`
- ▶ `from module_name import specific_item`
- ▶ `import module_name as alias`

Examples:

```
import math
print(math.sqrt(16))
```

```
from random import randint
print(randint(1, 10))
```

```
import os as operating_system
print(operating_system.getcwd())
```

Creating a User-Defined Module

Steps to Create a Module:

1. Create a Python file (e.g., `mymodule.py`).
2. Define functions, variables, or classes in it.
3. Import the file into your script.

Example: `mymodule.py`

```
def greet(name):  
    return f"Hello, {name}!"
```

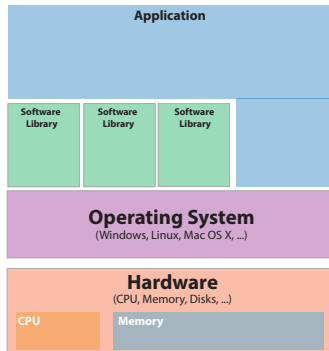
`main.py`

```
import mymodule
```

```
print(mymodule.greet("Alice"))
```

Libraries and APIs

- ▶ **software libraries** contain code that can be reused by other programs
- ▶ most languages provide libraries that **facilitate common tasks**
 - ▶ File access, OS functions
 - ▶ Network communication
 - ▶ Complex mathematical operations
 - ▶ Machine learning
 - ▶ ...



Libraries and APIs

- ▶ **software libraries** contain code that can be reused by other programs
- ▶ most languages provide libraries that **facilitate common tasks**
 - ▶ File access, OS functions
 - ▶ Network communication
 - ▶ Complex mathematical operations
 - ▶ Machine learning
 - ▶ ...
- ▶ **application programming interface (API)** enables programs to use provided functions



some Python libraries

Libraries and APIs

- ▶ **software libraries** contain code that can be reused by other programs
- ▶ most languages provide libraries that **facilitate common tasks**
 - ▶ File access, OS functions
 - ▶ Network communication
 - ▶ Complex mathematical operations
 - ▶ Machine learning
 - ▶ ...
- ▶ **application programming interface (API)** enables programs to use provided functions
- ▶ example: **NumPy** is a Python library for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions.



some Python libraries

Installing Third-Party Libraries

What Are Third-Party Libraries?

- ▶ Libraries are collections of modules designed for specific tasks.
- ▶ Examples:
 - ▶ `numpy` for numerical computations.
 - ▶ `matplotlib` for data visualization.
 - ▶ `requests` for web requests.

Installing Libraries with conda:

- ▶ Command: `conda install library_name.`
- ▶ Example: `conda install numpy.`

Installing Libraries from PyPI:

- ▶ Use `pip` within an active conda environment.
- ▶ Command: `pip install library_name.`
- ▶ Example: `pip install requests.`

Viewing Installed Libraries:

```
conda list
```

Creating and Managing Environments

Why Use Environments?

- ▶ Isolate dependencies for different projects.
- ▶ Avoid version conflicts between modules.

Basic conda Environment Commands:

- ▶ To create a new environment: `conda create -name myenv`
- ▶ Activate the environment: `conda activate myenv`
- ▶ Deactivate the environment: `conda deactivate`
- ▶ Delete an environment: `conda remove -name myenv -all`

Creating and Managing Environments

Why Use Environments?

- ▶ Isolate dependencies for different projects.
- ▶ Avoid version conflicts between modules.

Basic conda Environment Commands:

- ▶ To create a new environment: `conda create -name myenv`
- ▶ Activate the environment: `conda activate myenv`
- ▶ Deactivate the environment: `conda deactivate`
- ▶ Delete an environment: `conda remove -name myenv -all`

Best Practices:

- ▶ Use separate environments for each project.

Creating and Managing Environments

Why Use Environments?

- ▶ Isolate dependencies for different projects.
- ▶ Avoid version conflicts between modules.

Basic conda Environment Commands:

- ▶ To create a new environment: `conda create -name myenv`
- ▶ Activate the environment: `conda activate myenv`
- ▶ Deactivate the environment: `conda deactivate`
- ▶ Delete an environment: `conda remove -name myenv -all`

Best Practices:

- ▶ Use separate environments for each project.

Practice Session 1

- ▶ Modules in Python

https://gitlab2.informatik.uni-wuerzburg.de/ml4nets_notebooks/2024_wise_infhaf_notebooks/-/blob/main/PythonIntroNotebooks/Lecture_04.ipynb

File Handling Basics

Key Concepts:

- ▶ Files are opened with the `open()` function.
- ▶ Modes:
 - ▶ `'r'`: Read mode (default).
 - ▶ `'w'`: Write mode (overwrites existing content).
 - ▶ `'a'`: Append mode.
 - ▶ `'b'`: Binary mode.
- ▶ Use the `with` statement to handle files safely.

Reading Text Files

Basic File Reading Methods:

- ▶ `file.read()`: Reads the entire file.
- ▶ `file.readline()`: Reads one line at a time.
- ▶ `file.readlines()`: Reads all lines into a list.

Example:

```
with open('example.txt', 'r') as file:  
    content = file.read()  
    print(content)
```

```
with open('example.txt', 'r') as file:  
    for line in file:  
        print(line.strip())
```

Writing to Text Files

Writing Methods:

- ▶ `file.write(string)`: Writes a string to the file.
- ▶ `file.writelines(list)`: Writes a list of strings to the file.

Example: Writing Data to a File

```
with open('output.txt', 'w') as file:  
    file.write('Hello, World!\n')
```

```
data = ['Line 1\n', 'Line 2\n', 'Line 3\n']  
with open('output.txt', 'a') as file:  
    file.writelines(data)
```


Working with File Paths

Using the `os` Module:

- ▶ Check if a file exists: `os.path.exists(filepath)`.
- ▶ Get the current directory: `os.getcwd()`.
- ▶ Create directories: `os.makedirs()`.

Example:

```
import os

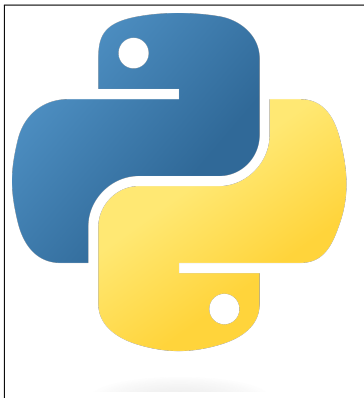
if os.path.exists('example.txt'):
    print('File exists!')

print(f'Current directory: {os.getcwd()}')

os.makedirs('new_folder', exist_ok=True)
```

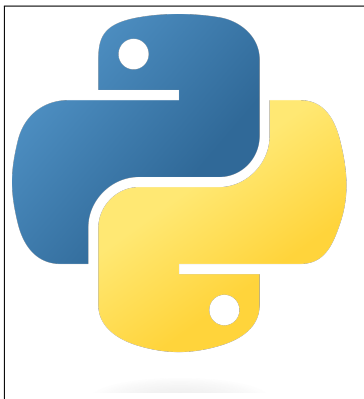
In summary

- ▶ we introduced **modules** in Python
- ▶ we learned how to import modules
- ▶ In-built modules (math, random, os ...)
- ▶ Creating our own modules
- ▶ Installing 3rd party libraries
- ▶ Creating Python **environments**
- ▶ Opening files in 'r', 'w', 'a' and 'b'
- ▶ Reading files and lines
- ▶ Writing and appending to files



In summary

- ▶ we introduced **modules** in Python
- ▶ we learned how to import modules
- ▶ In-built modules (math, random, os ...)
- ▶ Creating our own modules
- ▶ Installing 3rd party libraries
- ▶ Creating Python **environments**
- ▶ Opening files in 'r', 'w', 'a' and 'b'
- ▶ Reading files and lines
- ▶ Writing and appending to files



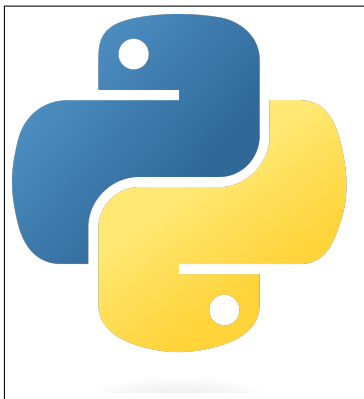
Practice Session 2

- ▶ File handling

https://gitlab2.informatik.uni-wuerzburg.de/ml4nets_notebooks/2024_wise_infhaf_notebooks/-/blob/main/PythonIntroNotebooks/Lecture_04.ipynb

In summary

- ▶ we introduced **modules** in Python
- ▶ we learned how to import modules
- ▶ In-built modules (math, random, os ...)
- ▶ Creating our own modules
- ▶ Installing 3rd party libraries
- ▶ Creating Python **environments**
- ▶ Opening files in 'r', 'w', 'a' and 'b'
- ▶ Reading files and lines
- ▶ Writing and appending to files



Exercise Session

- ▶ Modules and File handling

https://gitlab2.informatik.uni-wuerzburg.de/ml4nets_notebooks/2024_wise_infhaf_notebooks/-/blob/main/PythonIntroNotebooks/Exercise_L04.ipynb

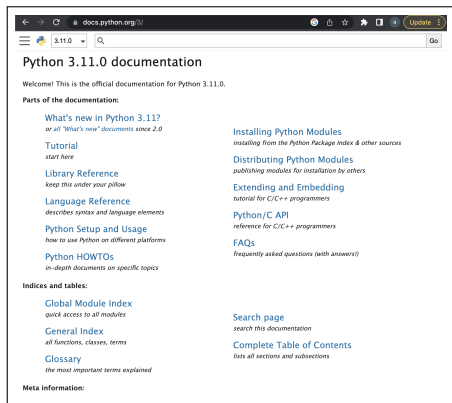
Self-study questions

1. What is a Python module and how is it used in Python?
2. How do you import a module in Python? Give examples of different import methods.
3. Name some commonly used built-in Python modules and explain their functionalities (e.g., 'os', 'math', 'sys').
4. How do you access the functions and classes inside a module once it is imported?
5. What is the 'os' module and how can it be used for file and directory management?
6. What is file handling in Python? How do you read and write files using Python?
7. What are the different file modes in Python ("r", "w", "a", "b", etc.)? Describe each one.
8. How do you append data to an existing file without overwriting its contents?
9. What is a Python virtual environment, and why is it useful?

Literature

reading list

- ▶ F Kaefer, P Kaefer: **Introduction to Python Programming for Business and Social Science Applications**, SAGE Publications, 2020
- ▶ **Official Python documentation**
<https://docs.python.org/>
- ▶ **Python tutorial:**
<https://docs.python.org/3/tutorial/>



The screenshot shows the Python 3.11.0 documentation website. The browser address bar displays 'docs.python.org/'. The page title is 'Python 3.11.0 documentation'. Below the title, there is a welcome message: 'Welcome! This is the official documentation for Python 3.11.0.' The page is organized into sections: 'Parts of the documentation:' which includes links for 'What's new in Python 3.11?', 'Tutorial', 'Library Reference', 'Language Reference', 'Python Setup and Usage', and 'Python HOWTOs'; 'Indices and tables:' which includes 'Global Module Index', 'General Index', and 'Glossary'; and 'Meta information:'. On the right side, there are additional links for 'Installing Python Modules', 'Distributing Python Modules', 'Extending and Embedding', 'Python/C API', 'FAQs', 'Search page', and 'Complete Table of Contents'.