

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

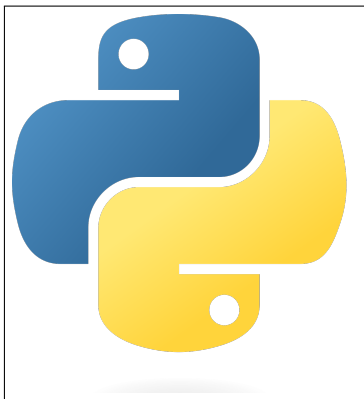
Lecture 03
Functions and Classes

November 15, 2024



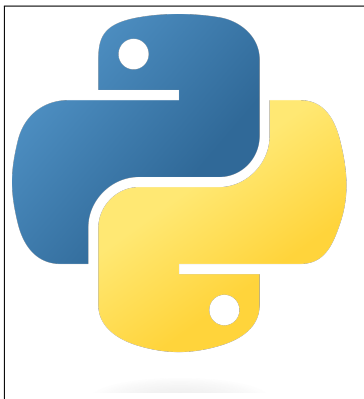
Recap

- ▶ we learned about **basic variable types and operations**
- ▶ we learned about basic Python objects such as **arrays, dictionaries and sets**



Recap

- ▶ we learned about **basic variable types and operations**
- ▶ we learned about basic Python objects such as **arrays, dictionaries and sets**
- ▶ we introduced **if, elif and else** statements
- ▶ we learned how to use **for** and **while** loops



Today

- ▶ Functions and Classes in Python
- ▶ Exercise session

Functions

- ▶ A function is a reusable block of code that performs a specific task.
- ▶ Functions help with modularity and code reusability:
 - ▶ accept input variables/parameters
 - ▶ return values
 - ▶ modify input variables

```
def Hello(name):  
    print('Hello', name)
```

a function in Python

Functions

- ▶ A function is a reusable block of code that performs a specific task.
- ▶ Functions help with modularity and code reusability:
 - ▶ accept input variables/parameters
 - ▶ return values
 - ▶ modify input variables
- ▶ functions can be nested

```
def Hello(name):  
    print('Hello',name)
```

a function in Python

```
def add(a,b):  
    return a+b
```

another function in Python

Defining and Calling Functions

- ▶ Define a function using the **def** keyword.
 - ▶ the body/content of the function is determined by indentation.
 - ▶ use **return** to specify the output of the function.
- ▶ Call the function by using its name and providing arguments.

Example

```
def greet(name):  
    return "Hello, " + name  
print(greet("Alice"))  
# Output: Hello, Alice
```

Scope of Variables

- ▶ Variables created inside a function are **local** to that function.
- ▶ Global variables are accessible throughout the entire script.
- ▶ Best practice: avoid modifying global variables inside functions.

Example

```
def add(x, y):  
    result = x + y  
    return result  
print(add(2, 3))  
# Output: 5
```

Object oriented programming

- ▶ Python is an object oriented programming language.
 - ▶ arrays, dictionaries, sets are Python objects
 - ▶ each type of object has their own properties and methods
- ▶ In Python we can define new classes/object types with their own:
 - ▶ properties
 - ▶ methods/functions

```
class Person:  
    def __init__(self, name, age):  
        self.name = name  
        self.age = age  
        self.alive=True  
    def birthday(self): #a function  
        print('Happy Birthday!',self.name)  
        self.age+=1
```

a class in Python

What is a Class?

- ▶ A class is a blueprint for creating objects (instances).
- ▶ Objects represent entities with specific attributes and behaviors.

```
class Person:  
    def __init__(self, name, age):  
        self.name = name  
        self.age = age  
        self.alive=True  
    def birthday(self): #a function  
        print('Happy Birthday!',self.name)  
        self.age+=1
```

a class in Python

What is a Class?

- ▶ A class is a blueprint for creating objects (instances).
- ▶ Objects represent entities with specific attributes and behaviors.
- ▶ Syntax:
 - ▶ in Python new classes are defined using the **class** command.
 - ▶ **`__init__()`**: defines initialization routine.
 - ▶ **Attributes** are variables within a class, unique to each instance.
 - ▶ **Methods** are functions defined inside a class.

```
class Person:  
    def __init__(self, name, age):  
        self.name = name  
        self.age = age  
        self.alive=True  
    def birthday(self): #a function  
        print('Happy Birthday!',self.name)  
        self.age+=1
```

a class in Python

What is a Class?

- ▶ A class is a blueprint for creating objects (instances).
- ▶ Objects represent entities with specific attributes and behaviors.
- ▶ Syntax:
 - ▶ in Python new classes are defined using the **class** command.
 - ▶ **`__init__()`**: defines initialization routine.
 - ▶ **Attributes** are variables within a class, unique to each instance.
 - ▶ **Methods** are functions defined inside a class.

```
class Person:  
    def __init__(self, name, age):  
        self.name = name  
        self.age = age  
        self.alive=True  
    def birthday(self): #a function  
        print('Happy Birthday!',self.name)  
        self.age+=1
```

a class in Python

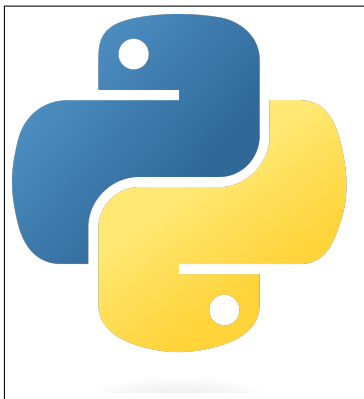
Practice Session

- ▶ Functions & Classes

https://gitlab2.informatik.uni-wuerzburg.de/ml4nets_notebooks/2024_wise_infhaf_notebooks/-/blob/main/PythonIntroNotebooks/Lecture_03.ipynb

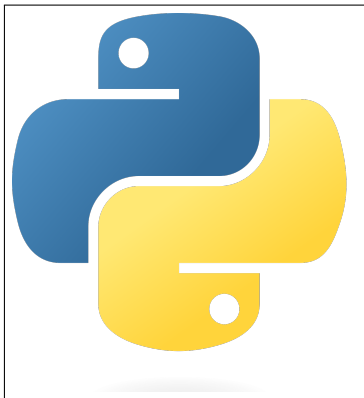
In summary

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



In summary

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



Exercise Session

https://gitlab2.informatik.uni-wuerzburg.de/ml4nets_notebooks/2024_wise_infhaf_notebooks/-/blob/main/PythonIntroNotebooks/Exercise_L03.ipynb

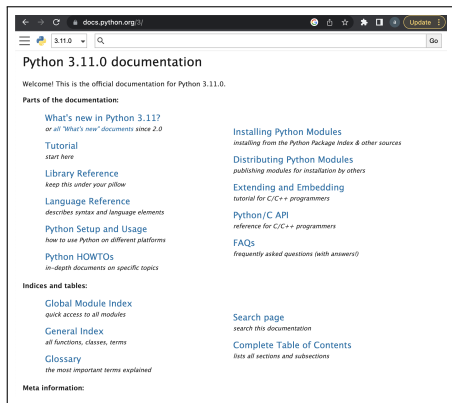
Self-study questions

1. How are functions defined in Python?
2. What does it mean for a variable to be local?
3. How are classes defined in Python?
4. Can you explain what the `__init()` method is for?
5. What is the difference between an attribute and a method?

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?' (with a sub-link 'or all "What's new" documents since 2.0'), 'Tutorial' (with a sub-link 'start here'), 'Library Reference' (with a sub-link 'keep this under your pillow'), 'Language Reference' (with a sub-link 'describes syntax and language elements'), 'Python Setup and Usage' (with a sub-link 'how to use Python on different platforms'), and 'Python HOWTOs' (with a sub-link 'in-depth documents on specific topics'). To the right, there are links for 'Installing Python Modules' (with a sub-link 'installing from the Python Package Index & other sources'), 'Distributing Python Modules' (with a sub-link 'publishing modules for installation by others'), 'Extending and Embedding' (with a sub-link 'tutorial for C/C++ programmers'), 'Python/C API' (with a sub-link 'reference for C/C++ programmers'), and 'FAQs' (with a sub-link 'frequently asked questions (with answers!)'). Below this, there is an 'Indices and tables:' section with links for 'Global Module Index' (with a sub-link 'quick access to all modules'), 'General Index' (with a sub-link 'all functions, classes, terms'), and 'Glossary' (with a sub-link 'the most important terms explained'). To the right of this section are links for 'Search page' (with a sub-link 'search this documentation') and 'Complete Table of Contents' (with a sub-link 'lists all sections and subsections'). At the bottom, there is a 'Meta information:' section.