

Data Science and AI for Medicine Training School

TRAINING: Python Basics

SPEAKER:

GEFÖRDERT VOM



Bundesministerium
für Forschung, Technologie
und Raumfahrt



SACHSEN Diese Maßnahme wird gefördert durch die Bundesregierung aufgrund eines Beschlusses des Deutschen Bundestages. Diese Maßnahme wird mitfinanziert durch Steuermittel auf der Grundlage des von den Abgeordneten des Sächsischen Landtags beschlossenen Haushaltes.



Come2Data
Kompetenzzentrum für
interdisziplinäre Datenwissenschaften

Data Science and AI for Medicine Training School
Training: Python Basics

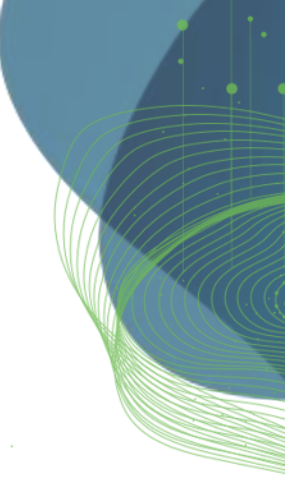
Slide 1

ScaDS.AI
DRESDEN LEIPZIG



AGENDA

- Programming Concepts
- Python – Terms and Definitions
- Execution of Python Code
- Python – Built-in types
- Python – Conditions
- Python – Loops



Programming Concepts

What is programming?

- Use of programming language to implement software requirements as a computer program
- Computer program is converted into machine code for execution (compiled or interpreted)

What is a programming language?

- Tool for formulating algorithms and data structures
- Formal language with syntax and semantics

Algorithm

- Consists of instructions to solve a problem
- Instructions consist of permitted patterns

Data Structure

- Object to store and organize data in memory

Syntax

- Formal set of rules for the use of instructions
- “Grammar” of a programming language

Semantics

- Actual meaning of the instructions

What is Python?

Universal high-level programming language, also often used for scripting

- Released in 1994, recent stable version is 3.13
- Goals: simplicity, clarity, extensibility
 - Few reserved keywords, reduced syntax
 - Extensive standard library, e.g., file handling, math, text processing, ...
 - Easy integration of additional packages / libraries
- Open Source, portable on multiple platforms
- Extensively used in data science, data analysis, artificial intelligence
- Easy management and use of additional packages and extensions
 - Built-in package manager “pip” with [Python package index PyPI](https://pypi.org/)
 - Python distributions shipping Python + alternative package manager (e.g., “conda”) + virtual environments + preinstalled packages) – e.g., [Miniconda](https://docs.continuum.io/miniconda/), [Anaconda](https://www.anaconda.com/)



(TM) Trademark of the PSF
<https://www.python.org>

What is Python?

Terms and definitions

Variable

Container for storing assigned data in memory, using a name for reference

Object

Complex structure which bundles data and methods to operate on the data

Method

Block of code tied to an object, usable via dot-Operator ("method is mine")

Everything in Python
is an object

```
1  x = 5
2
3  class Human:
4      def __init__(self, name):
5          self.name = name
6
7      def say_hello(self):
8          return "Hello, I'm " + self.name
9
10 myself = Human("Matthias")
11
12 print(myself.say_hello())
```

Assignment

Variable name

Value of specific type

Definition of custom object

Instantiation and assignment to variable

Use of object's method

What is Python?

Terms and definitions

Function

Independent block of reusable code for a specific task
("function is free")

Module

File containing Python code which can be imported into other Python code

Comment

Lines in code not interpreted by Python, used for documentation, starting with #

Import module for additional functionality

Comment, not interpreted

```
1  import math
2
3  # My custom function
4  def greet(name):
5      |   return "Hello, " + name
6
7  print(greet("Matthias"))
8  print("Pi = " + str(math.pi))
```

Definition of custom function

Use of custom function

Use of built-in function

Use of built-in function

Use method from imported module

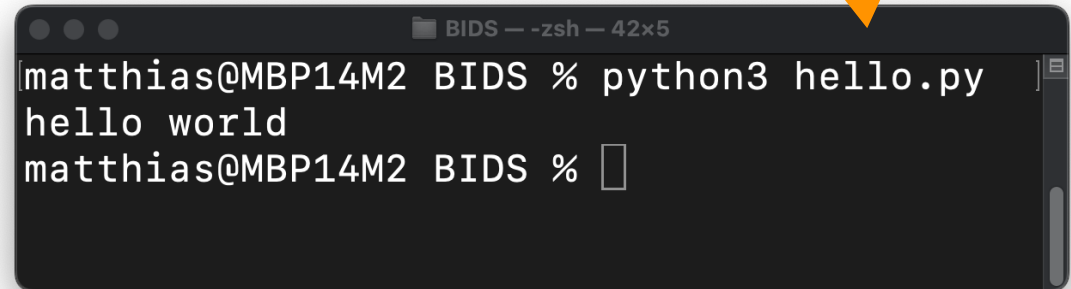
Execution of Python code

Execution via Python file

- Save code in file with file extension “.py”
- Execute file with installed Python

hello.py

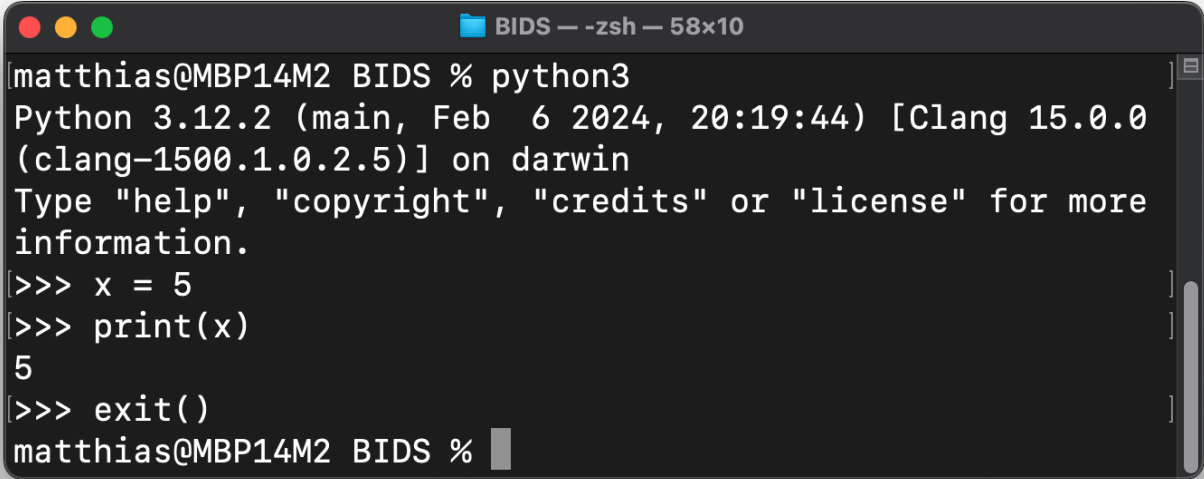
```
1 print("hello world")
```



A terminal window titled "BIDS - zsh - 42x5" showing the command `python3 hello.py` being executed. The output is `hello world`. The prompt is `matthias@MBP14M2 BIDS %`.

Interactive execution in terminal

- Start interactive Python session
- Enter and execute instructions line by line

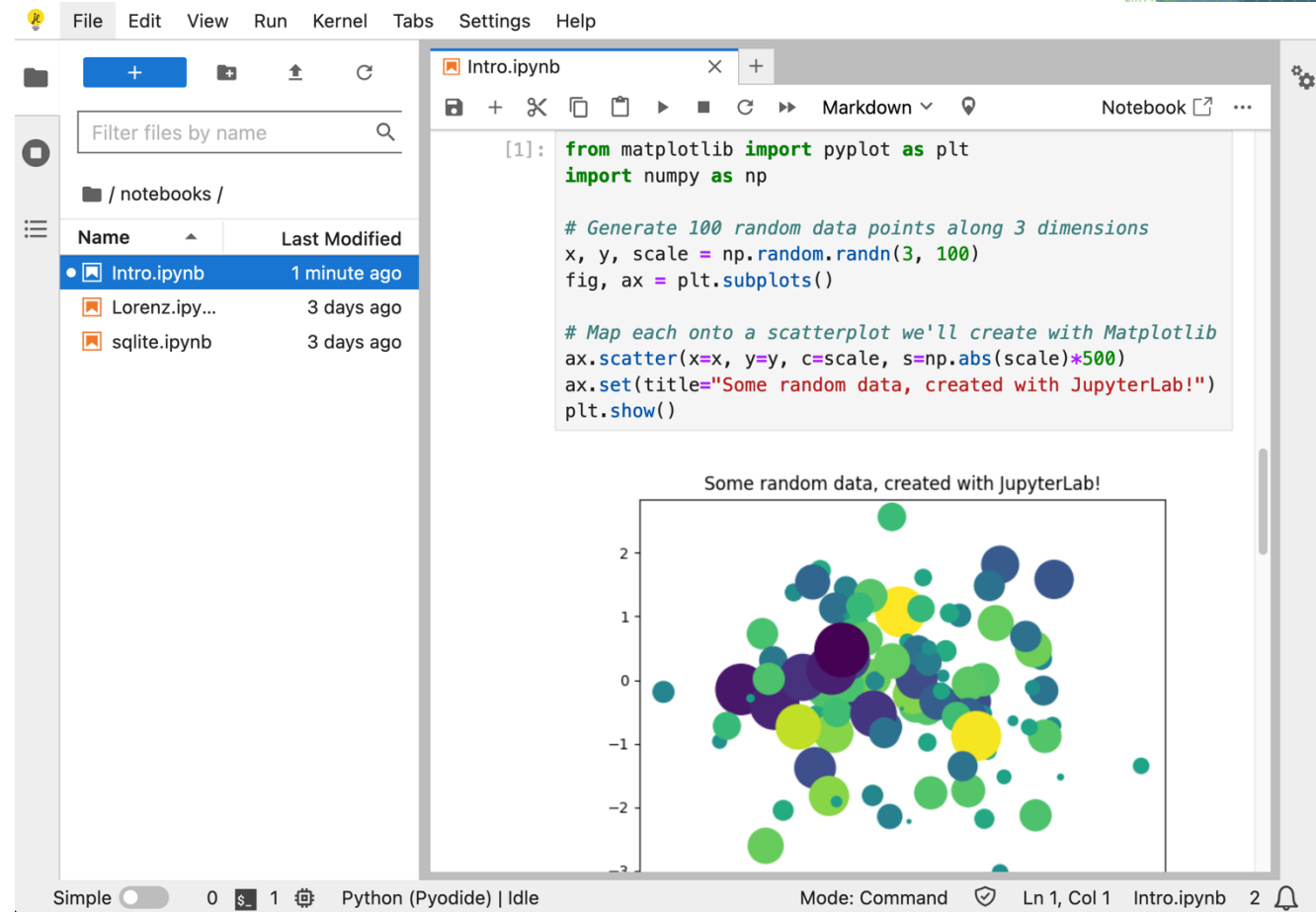


A terminal window titled "BIDS - zsh - 58x10" showing an interactive Python session. The command `python3` is entered, followed by the Python version and environment information. The user enters `>>> x = 5`, `>>> print(x)`, and `>>> exit()`. The output is `5`. The prompt is `matthias@MBP14M2 BIDS %`.

Execution of Python code

Interactive execution in Jupyter Notebook

- Web-based interface with cells for
 - Executable Python code
 - Rich text for documentation
 - Rich output for text, images, plots
- **Jupyter Lab** with
 - Jupyter notebook
 - File browser
 - Terminal access
 - Plugins



<https://jupyter.org/try-jupyter/lab/index.html>

Built-in types

Truth and Boolean

Truth value and Boolean

- Objects can be tested for a truth value
- Truth values can be used in conditions
- Represented by Booleans: `True` (1) and `False` (0)
- There are default truth values for objects, e.g., number zero or empty strings are considered `False`

Boolean operators and comparisons

- Used to evaluate a truth value
- Operators are `and`, `or`, `not`
- Comparisons are, e.g., `<` (strictly less), `==` (equal), `>=` (greater than or equal), `!=` (not equal)

```
1 # Boolean operators
2 print(True and False)
3 print(True or False)
4 print(not True)
Executed at 2024.05.05 09:52:32 in 4ms
```

✓ False
True
False

```
1 # Comparisons
2 print(True == False)
3 print(True != False)
4 print(True > False) # But why?
Executed at 2024.05.05 09:52:32 in 2ms
```

✓ False
True
True

```
1 # Math with Boolean
2 print(int(True), int(False))
3 print(True + True)
Executed at 2024.05.05 09:52:32 in 1ms
```

1 0
2

Built-in types

Numeric types

Numeric types

- Integers (`int`)
- Floating point numbers (`float`)
- Complex numbers (`complex`)

Supported operations

- Mathematical operators, e.g., `+`, `-`, `/`
- Comparisons
- Mathematical functions

```
1 # Numerical types
2 print(type(5))
3 print(type(1.5))
4 print(type(2j))
```

Executed at 2024.05.05 10:16:36 in 4ms

✓

```
<class 'int'>
<class 'float'>
<class 'complex'>
```

```
1 # Operators
2 print(5 + 5)
3 print(5 * 5)
4 print(5 / 5)
```

Executed at 2024.05.05 10:16:36 in 1ms

✓

```
10
25
1.0
```

```
1 # Comparisons
2 print(5 > 1)
3 print(5 == 1)
```

Executed at 2024.05.05 10:16:36 in 1ms

True
False

```
1 # Mathematical functions
2 print(abs(-5))
3 print(pow(5, 2))
4 print(round(4.5))
```

Executed at 2024.05.05 10:16:36 in 1ms

✓

```
5
25
4
```

Some may behave unexpected!

Built-in types

Sequence types

Some basics on sequences

- Data structures to store and manipulate multiple values
- Values can be of homogeneous or heterogeneous type
- Sequences are either mutable (values can be changed “in place”) or immutable
- Values can be accessed by an index on the sequence, starting at 0

```
1 my_list = ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I']  
Executed at 2024.05.05 10:42:01 in 3ms
```

Index	0	1	2	4	5	6	7	8	9
Values	A	B	C	D	E	F	G	H	I

Built-in types

Sequence types

Lists

- *Mutable*, construction via brackets []
- Homogenous or heterogenous values

```
1 # Define a list
2 my_list = ['A', 'B', 'C', 'D', 'E', 'F']
3 print(type(my_list))
4 print(my_list)
```

Executed at 2024.05.05 10:57:22 in 5ms

```
<class 'list'>
['A', 'B', 'C', 'D', 'E', 'F']
```

```
1 # Access separate elements
2 # Called "indexing"
3 print(my_list[0])
4 print(my_list[-1])
```

Executed at 2024.05.05 10:57:22 in 2ms

```
A
F
```

Use negative index to start at the last element

Get elements from index 1 to 2

```
# Access subsets of elements
2 # Called "slicing"
3 print(my_list[1:3])
4 print(my_list[:4])
5 print(my_list[2:])
6 print(my_list[2::2])
```

Executed at 2024.05.05 10:59:07 in 3ms

Get all elements up to index 3

Get all elements starting at index 2

Get every second element, start at index 2

```
['B', 'C']
['A', 'B', 'C', 'D']
['C', 'D', 'E', 'F']
['C', 'E']
```

```
1 # Built-in methods
2 my_list.reverse()
3 print(my_list)
4 my_list.sort()
5 print(my_list)
```

Executed at 2024.05.05 11:13:18 in 3ms

Call built-in method to reverse the list

```
['F', 'E', 'D', 'C', 'B', 'A']
['A', 'B', 'C', 'D', 'E', 'F']
```

Built-in types

Sequence types

Tuples

- *Immutable*, construction via parentheses ()
- Homogenous or heterogenous values
- Indexing and slicing works like for lists

Ranges

- *Immutable*, construction via `range ()`
- Homogenous numerical values
- Indexing and slicing works like for lists

```
1 # Define a tuple
2 my_tuple = ('A', 1)
3 print(type(my_tuple))
4 print(my_tuple)
5 print(my_tuple[0])
6 # Immutable!
7 my_tuple[0] = 'B'
```

Executed at 2024.05.05 12:24:07 in 32ms

✓ <class 'tuple'>
('A', 1)
A

> Traceback...
TypeError: 'tuple' object does not support item assignment

```
1 # Define a range
2 my_range = range(10)
3 print(type(my_range))
4 print(my_range)
5 print(my_range[-1])
6 # Convert to list
7 print(list(my_range))
```

Executed at 2024.05.05 12:24:53 in 4ms

✓ <class 'range'>
range(0, 10)
9
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

Built-in types

Sequence types

Text sequence - string

- *Immutable*, construction via quotes `"", ''`
- Values of type Unicode codepoints
- Indexing and slicing works like for lists

```
1 # Define a string
2 my_string = 'Hello World!'
3 print(type(my_string))
4 print(my_string)
```

Executed at 2024.05.05 12:30:56 in 3ms

```
<class 'str'>
Hello World!
```

```
1 # Indexing and slicing
2 print(my_string[0])
3 print(my_string[6:])
```

Executed at 2024.05.05 12:31:02 in 3ms

```
H
World!
```

```
1 # Built-in methods
2 print(my_string.upper())
3 print(my_string.split(' '))
```

Executed at 2024.05.05 12:32:39 in 4ms

```
HELLO WORLD!
['Hello', 'World!']
```

Convert all letters to
uppercase

Split the string at
whitespace and
return a list of
resulting strings

Built-in types

Sequence types

Further operations on sequences

- Sequences can be concatenated (append them) with + operator
- Sequences can be tested for their content with in

```
1 # Concatenate strings
2 hello = 'Hello'
3 world = 'World'
4 full = hello + ' ' + world
5 print(type(full))
6 print(full)
```

Executed at 2024.05.05 12:49:04 in 4ms

<class 'str'>

Hello World

```
1 # Concatenate lists
2 list_1 = [1, 2, 3]
3 list_2 = ['A', 'B', 'C']
4 list_3 = list_1 + list_2
5 print(type(list_3))
6 print(list_3)
```

Executed at 2024.05.05 12:49:14 in 3ms

<class 'list'>

[1, 2, 3, 'A', 'B', 'C']

```
1 # Test for specific values
2 print('A' in list_3)
3 print(10 in list_3)
```

Executed at 2024.05.05 12:50:50 in 3ms

True

False

:

Built-in types

Dictionaries

[Mapping types](#) or dictionaries (dicts)

- *Mutable*, construction via braces { }
- Provide a mapping from key → value, i.e. a list of key → value pairs
- Indexing and slicing works NOT like for lists

```
1 # Define a dict
2 german_english_dict = {
3     'Vorlesung': 'Lecture',
4     'Gleichung': 'Equation'
5 }
6 print(type(german_english_dict))
7 print(german_english_dict)
```

Executed at 2024.05.05 13:12:17 in 3ms

```
<class 'dict'>
{'Vorlesung': 'Lecture', 'Gleichung': 'Equation'}
```

```
1 # Access values by their keys
2 print(german_english_dict['Vorlesung'])
3 print(german_english_dict['Unknown'])
```

Executed at 2024.05.05 13:13:26 in 13ms

Lecture

> Traceback...

KeyError: 'Unknown'

```
1 # Modify the values
2 german_english_dict['Vorlesung'] = 'Course'
3 # Add new entries
4 german_english_dict['Eintrag'] = 'Entry'
5 print(german_english_dict)
```

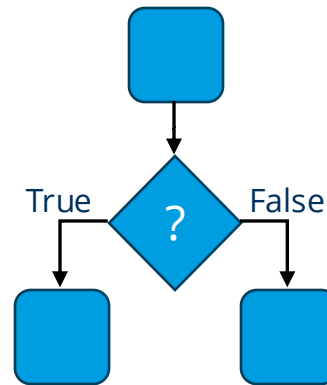
Executed at 2024.05.05 13:15:09 in 2ms

```
{'Vorlesung': 'Course', 'Gleichung': 'Equation',
 'Eintrag': 'Entry'}
```

Conditions

Conditional statements

- Used as control flow tool, e.g., to check
 - if pre-requisites are met
 - if data has the right format or value
 - if there are any errors
- The `if` statement is used to
 - Evaluate a Truth value for given expressions, e.g., with Boolean operators of comparisons
 - Executes subsequent code if the Truth value evaluates to `True`
- The `else` statement can be used to execute code if the given expressions evaluate to `False`



```
1  # Preceding code
2  # Defines and works on my_list
3
4  # Check condition
5  if 'Z' in my_list:
6      # Do this if the condition is True
7      print('Z is in my_list!')
8  else:
9      # Do this if the condition is False
10     print('Z is not in my_list!')
11
12 # Subsequent code
    Executed at 2024.05.05 13:41:29 in 3ms
```

Z is not in my_list!

:

Loops

Loop statements

- Used as control flow tool for repeated execution of code
- Different kinds of loop statements
 - for: iterates over elements of a sequence (e.g. list), or iterable objects in general
 - while: repeats subsequent code as long an expression is `True`
- Both can be controlled in more detail using
 - break to terminate the loop
 - continue to skip the current iteration

```
1 my_list = ['A', 'B', 'C', 'D', 'E', 'F']
2 # Use for to iterate over my_list
3 for i in my_list:
4     # Skip iterations for letters between B and E
5     if 'B' < i < 'E':
6         continue
7     print(i)
```

Executed at 2024.05.05 14:25:27 in 3ms

✓ A
B
E
F

```
1 i = 10
2 # Use while to decrement number till 0
3 while i >= 0:
4     i = i - 1
5     # Stop loop if number hits criteria
6     if i % 5 == 0:
7         break
8     print(i)
```

Executed at 2024.05.05 14:31:33 in 3ms

✓ 9
8
7
6



Any questions or remarks?

Let's practice – Python Basics in Jupyter Lab