Artificial Intelligence

Algorithms and Applications with Python

Chapter 3

Dr. Dominik Jung
dominik.jung42@gmail.com

</> course material: ↗ https://github.com/dominikjung42

Outline

3 Introduction into AI-Programming with Python

- 3.1 Software Tools and Programming Languages in AI
- 3.2 Foundations of Programming with Python
- 3.3 The Anaconda Toolbox for AI Programming
- 3.4 Advanced Programming with Python
- 3.5 AI Related Packages and Concepts in Python

Lectorial 1: Implement Problem-Solving Agents with Python

► **What we will learn:**

- Get an overview of AI software and programming with Python, so that you will be able to build your own agents and AI software components
- Workflow and tools to develop simple scripts and applications with Python
- Discuss advanced concepts of Python programming and AI related packages

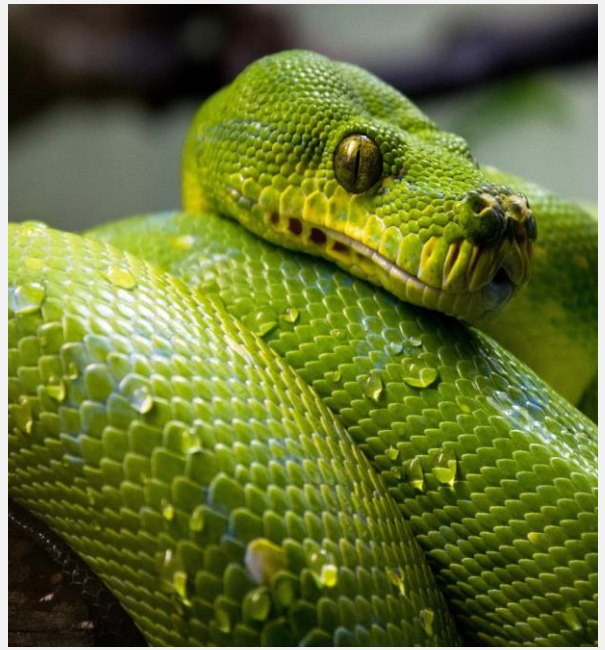


Image source: ↗ Pixabay (2019) / ↗ CC0

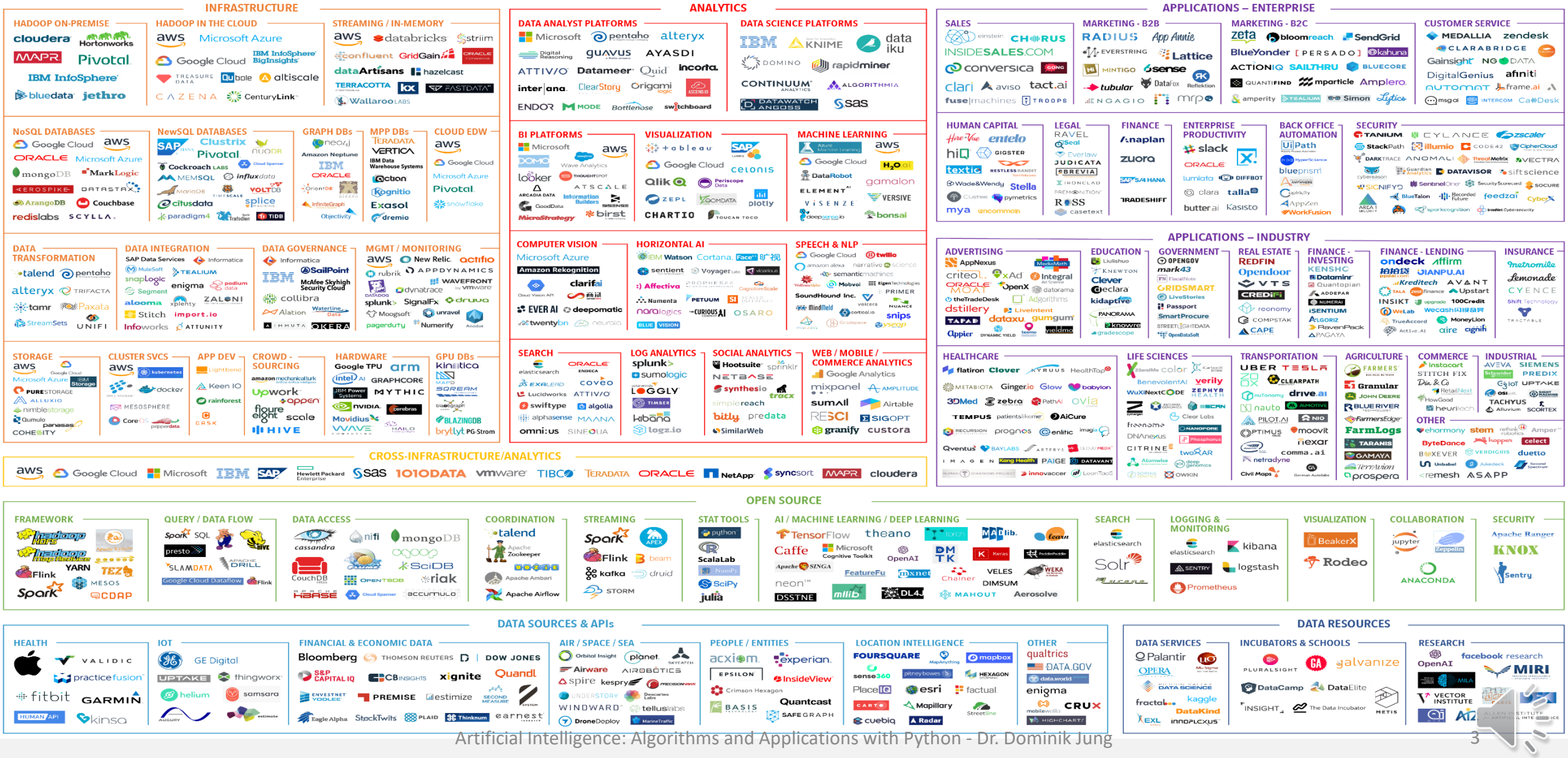
► **Duration:**

- 90 min

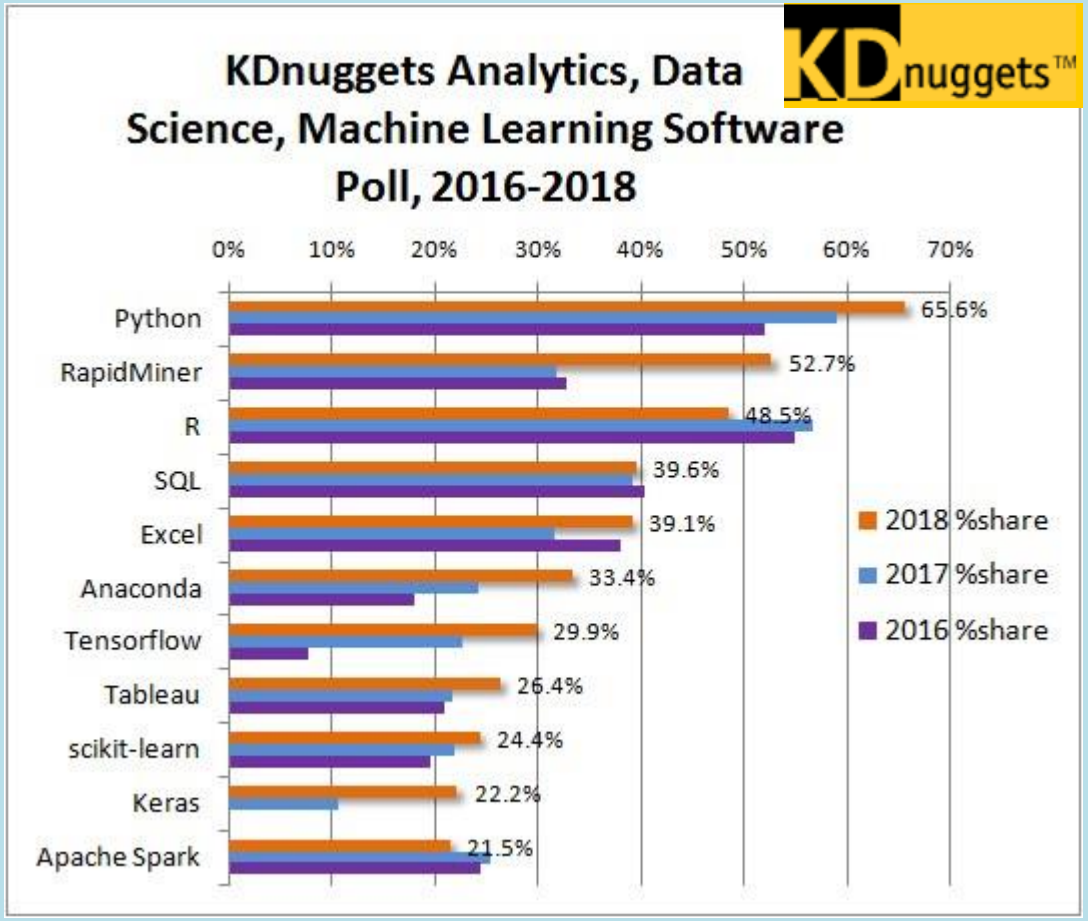
► **Relevant for Exam:**

- 3.1 - 3.5

Landmark Analytics Landscape (today)



3.1 Current Top Tools for AI Specialists



- The landscape of "Data Science Tools" is constantly changing
- Tools are often domain-specific: If you are interested in machine learning you better start with Python, if you are interested in statistics or analytics start with R
- But learn both!
- Tools are also customer-specific: If your customers can not program, you have to build your models with software tools

https://www.kdnuggets.com/2018/05/poll-tools-analytics-data-science-machine-learning-results.html

3.1 Tools of AI Specialists

Software Tools

Software that is designed to be used for a specific use case (e.g. Dashboarding, Data Visualization, Modelling). Most of the software tools in AI have graphical user interfaces (GUI), and only some come with proprietary languages to operate on.

Programming Languages

Language engineered to create a standard form of instructions for a computer. Like human languages they are split into two components, i.e. syntax (form) and semantics (meaning).

The major advantage of using programming languages is that they are usually open source and more flexible because they come with a rich stack of libraries and packages. In contrast, ready-made software tools are usually barely-configurable black boxes – that may however be easier to learn.

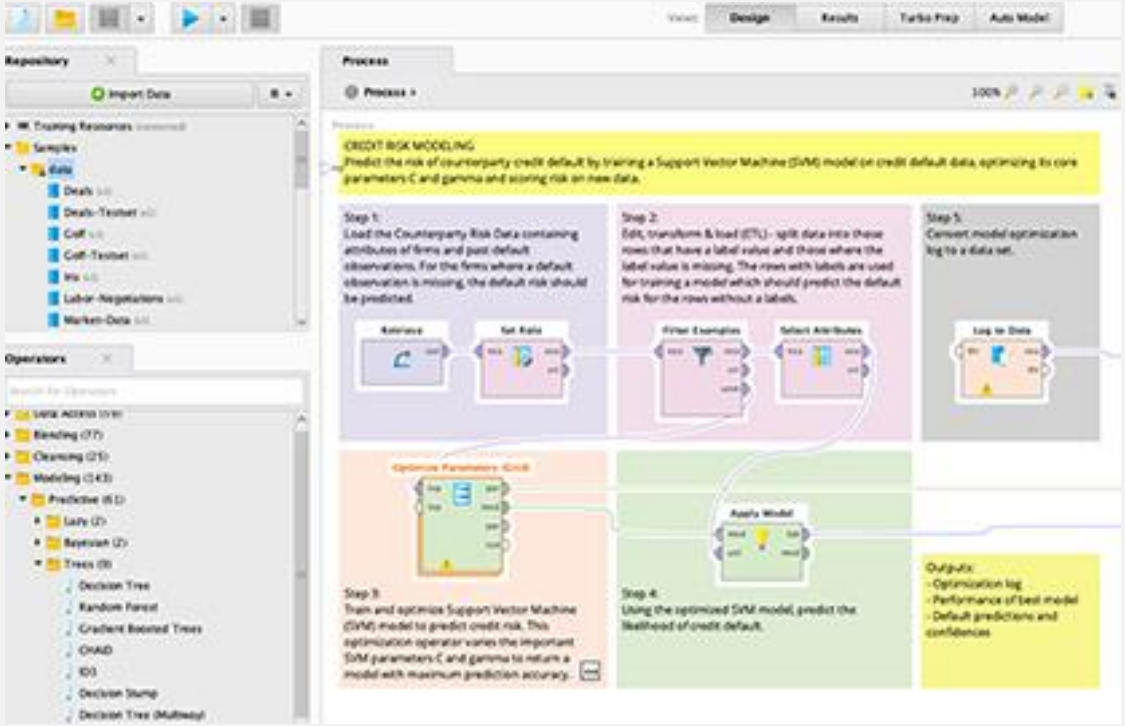
3.1 Machine Learning Tools: RapidMiner

Characteristics

- General purpose machine learning platform with a graphical user interface
- Developed from the AI Unit @TU Dortmund

Application

- Graphical workflow editor that supports all steps of the machine learning process (e.g. data processing, visualization and modelling)
- Many use cases for research, education, training, rapid prototyping, and application development



\+ Easy to use and no programming skills are necessary. Workflow can be communicated easily

\- Not as flexible and powerful as programming languages

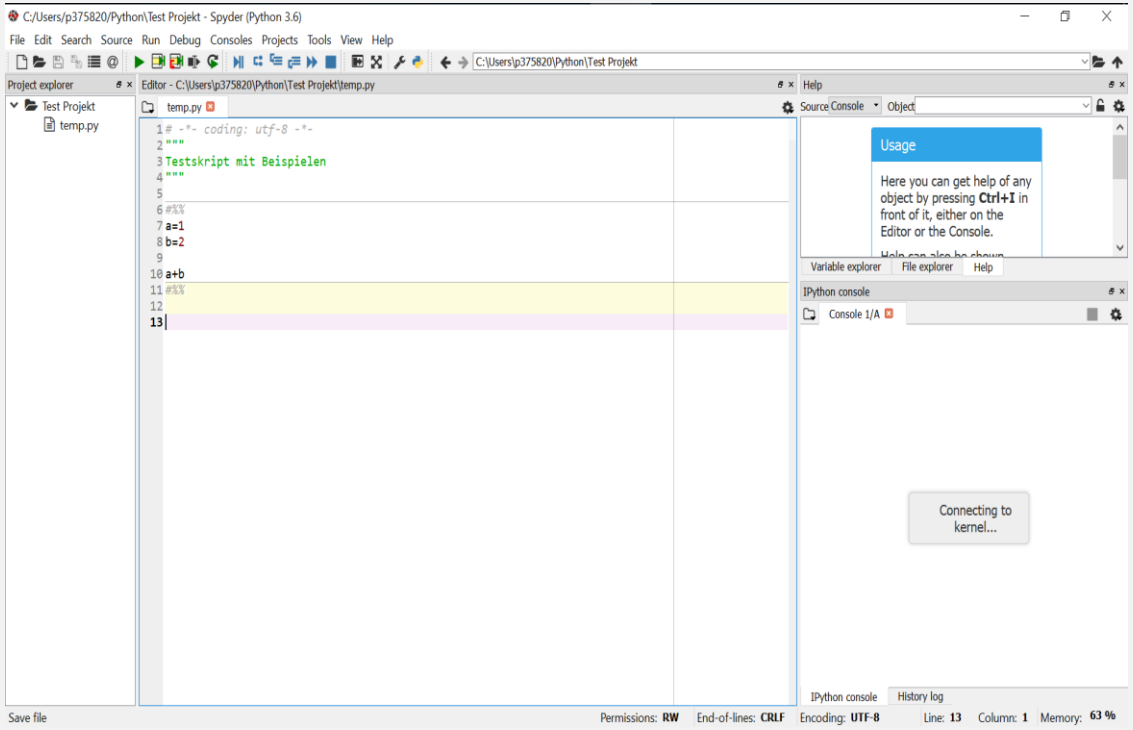
3.1 Programming Languages: Python

Characteristics

- General-purpose interpreted, interactive, object-oriented, and high-level programming language, widely used in various fields, thanks also to its readability
- Open source

Application

- Because it is a general purpose language, the use cases of Python are manifold
- Analytical/Machine Learning Data Science Projects, where Neural Networks are used (e.g. for Natural Language Processing)
- In particularl, Artificial Neural Networks (TensorFlow)



+ Active and large community, with a huge number of great libraries for Deep Learning

- Requires programming skills, but is easy to learn. Sometimes issues arising from version incompatibility

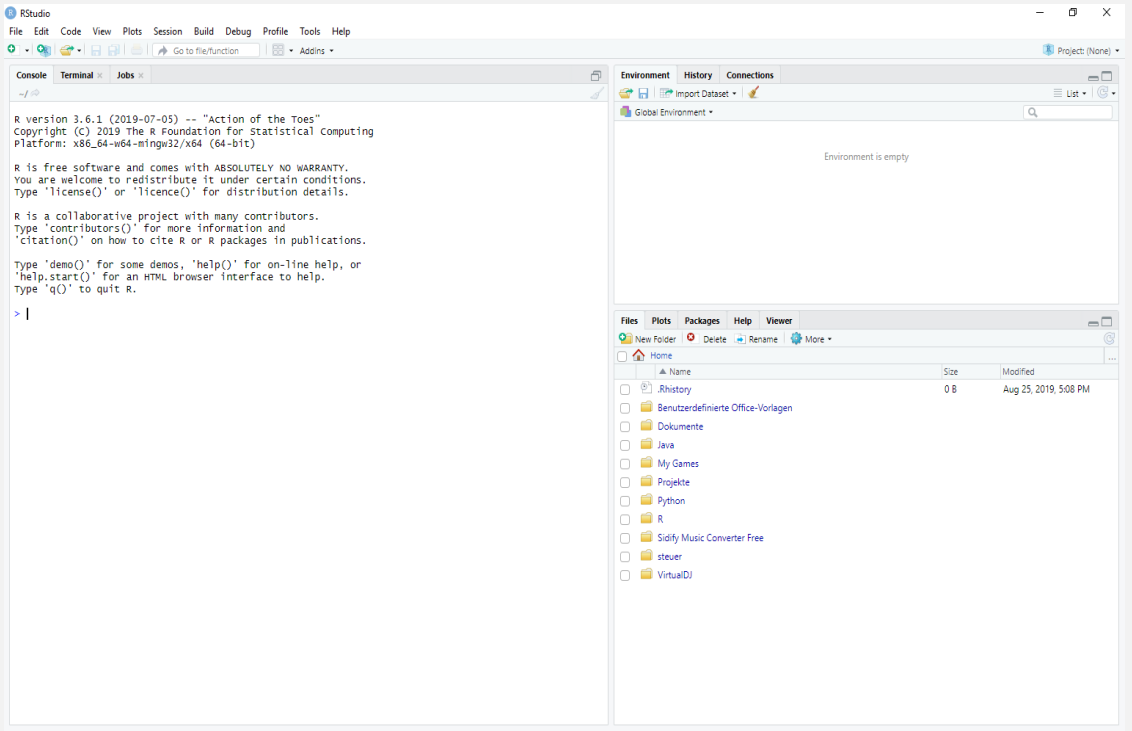
3.1 Programming Languages: R

Characteristics

- R: open source programming language designed for statistical computations
- State-of-the-Art in Analytics
- With R-Studio one of the best IDEs for Statistics

Application

- Popular among the academic and statistical community
- Excellent for Data Science Projects including statistical analysis (for small-to-medium sized amounts of Data) rather than Machine Learning



\+ Large, active community with great libraries for visualization and dashboard design. Besides, language with the most statistical packages

\- Slower and less readable than Python

3.1 Classroom task

Your turn!

Task

Please discuss: in which kind of AI project scenario you would work with software tools and in which you recommend to use an programming language!

Outline

3 Introduction into AI-Programming with Python

- 3.1 Software Tools and Programming Languages in AI
- 3.2 Foundations of Programming with Python
- 3.3 The Anaconda Toolbox for AI Programming
- 3.4 Advanced Programming with Python
- 3.5 AI Related Packages and Concepts in Python

Lectorial 1: Implement Problem-Solving Agents with Python

► **What we will learn:**

- Get an overview of AI software and programming with Python, so that you will be able to build your own agents and AI software components
- Workflow and tools to develop simple scripts and applications with Python
- Discuss advanced concepts of Python programming and AI related packages

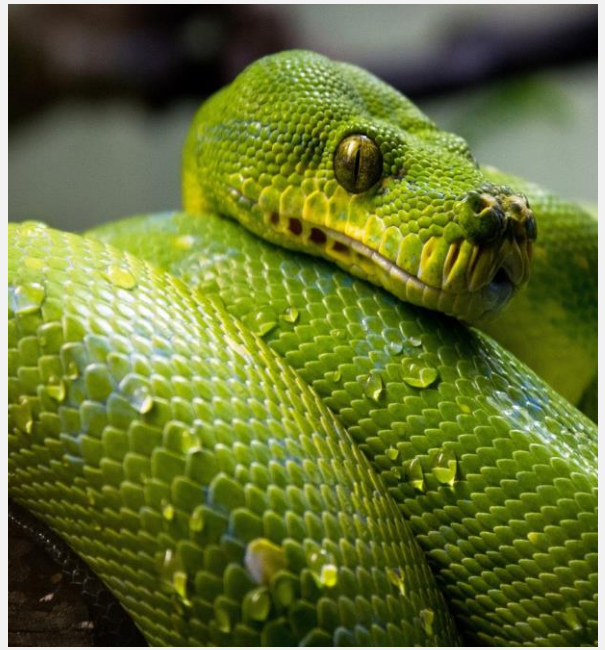


Image source: ↗ Pixabay (2019) / ↗ CC0

► **Duration:**

- 90 min

► **Relevant for Exam:**

- 3.1 - 3.5

3.2 Why do we use Python in this Course?

```
Public class Crawler{
Public static void main(String[] args) throws IOException{
    Print Writer textFile = null;
    try{
        textFile = new PrintWriter(„results.txt“);
        System.out.println(„Enter the URL you with to crawl…“);
        System.out.println(„URL: “);
        String myUrl = new Scanner(System.in).nextLine();

        String response = getContentByUrl(myUrl);

        Matcher matcher = Pattern
           .compile(„href=[\“‘](.[^\“‘)+)[\“‘]“).matcher(response);
        while(matcher.find()){
            String url = matcher.group(1);
            System.out.println(url);
            textFile.println(url);
        }
    finally{
        if(textFile != null){
             textFile.close();
        }
    }
}

Private static String getContentByUrl(String myUrl) throws IOException{
    Url url = new URL(myUrl);
    URLConnection urlConnection = url.openConnection();
…
```

```
If __name__ == ‘__main__‘:
    with open(„results.txt“,“wt“) as textFile:
        print(„Enter the URL you wish to crawl:“)
        myUrl = input(„URL: “)
        for i in re.findall(„href=[\“‘](.[^\“‘)+)[\“‘]“,
        urllib.request.urlopen(myUrl).read().decode(), re.I):
        print(i)
        textFile.write(i+‘\n‘)
```

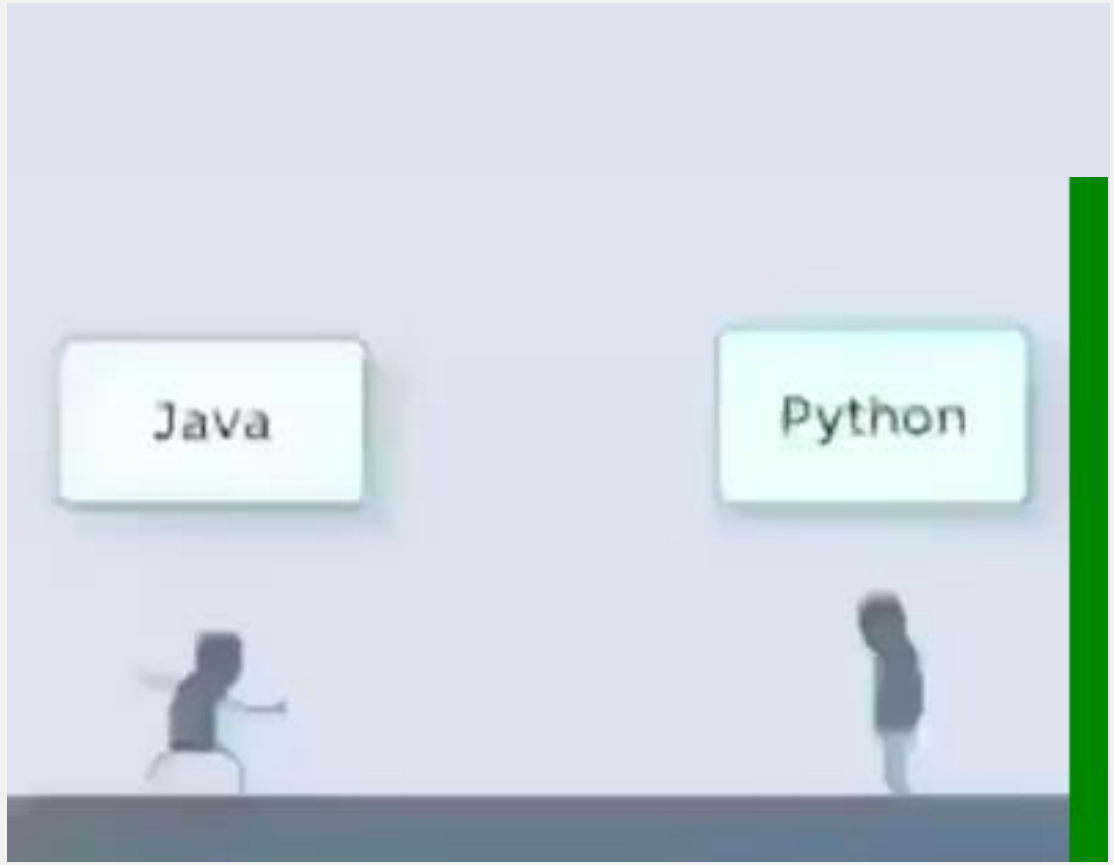
Characteristics:

- Main purpose was to design an easy to learn language with strong focus on code readability
- Multi-purpose programming languages, thus very flexible and applicable to a broad range of cases
- Healthy, active and supportive community that develops state-of-the-art AI packages, e.g.:TensorFlow

Python

- Python has a couple of characteristics that have bolstered its rise in the Data Science community over the past years.
- One of the main aspects is its elegance and readability compared to other languages.

3.2 Why do we use Python in this Course?




```
If __name__ == '__main__':
    with open(„results.txt“,“wt“) as textFile:
        print(„Enter the URL you wish to crawl:“)
        myUrl = input(„URL: “)
        for i in re.findall(„href=[\“'](.[^\“']+)[\“']“,
        urllib.request.urlopen(myUrl).read().decode(), re.I):
        print(i)
        textFile.write(i+'\n')
```

Characteristics:

- Main purpose was to design an easy to learn language with strong focus on code readability
- Multi-purpose programming languages, thus very flexible and applicable to a broad range of cases
- Healthy, active and supportive community that develops state-of-the-art AI packages, e.g.:TensorFlow

Python

- Python has a couple of characteristics that have bolstered its rise in the Data Science community over the past years.
- One of the main aspects is its elegance and readability compared to other languages.

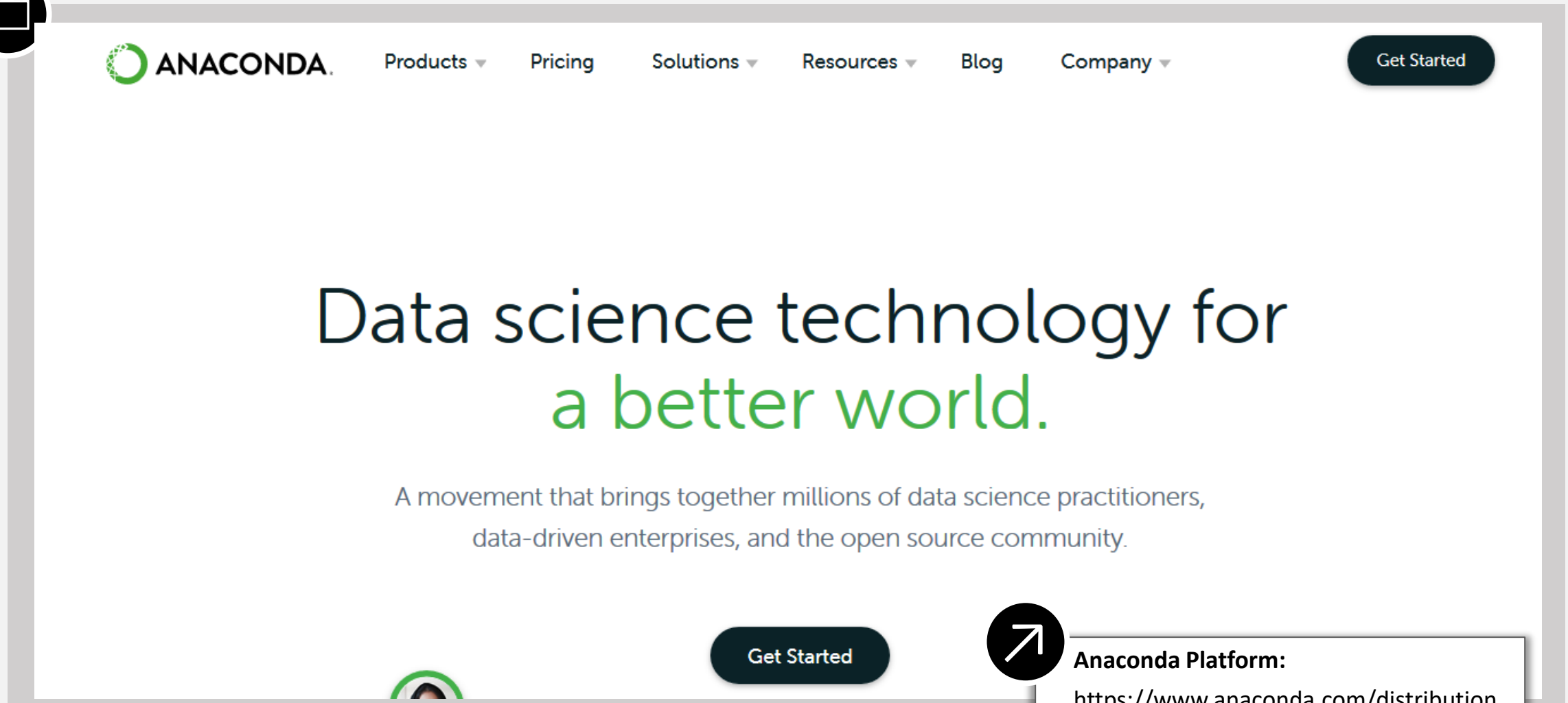
3.2 Python 2 vs. Python 3

- „Breaking changes“ in the standard library functionality

```
print "hello world"
        vs
print("hello world")
```

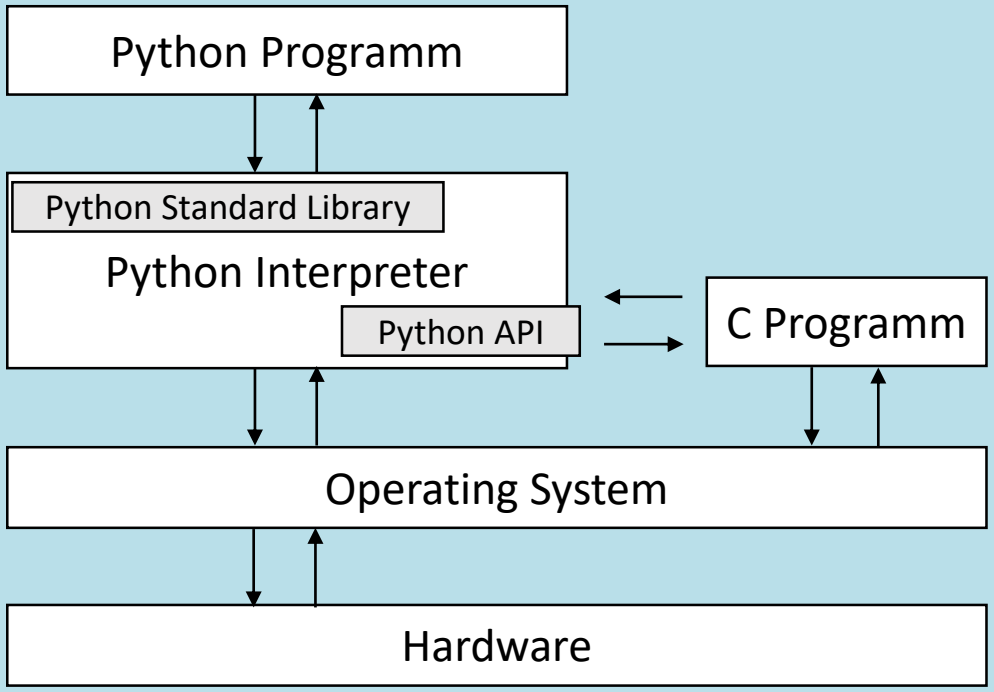
- Modules added to standard library
  - For instance, `asyncio` module for native concurrency support
- Python 2 still dominant in some established libraries
- However, movement towards Python 3

3.2 Install Anaconda



Anaconda Platform:
https://www.anaconda.com/distribution

The Python Interpreter



- The Python interpreter executes your Python program without requiring it previously to have been compiled into a machine language program
- There is a big Standard Library with basic functionalities, the program can rely on
- Python can be extended easily by the Python API

Image adapted from Ernesti, J. & Kaiser, P. (2017)

3.3 Key Concepts of Programming Python

Whitespace Formatting

```
if True:
    print("AI is such a cool lecture")
else:
    print("AI is my favorite lecture")
```

Line indentation

Python does not use braces to indicate code blocks (e.g. for functions or flow control). Blocks are denoted by line indentation instead. The number of spaces in the indentation is variable, but has to be the same within the block.

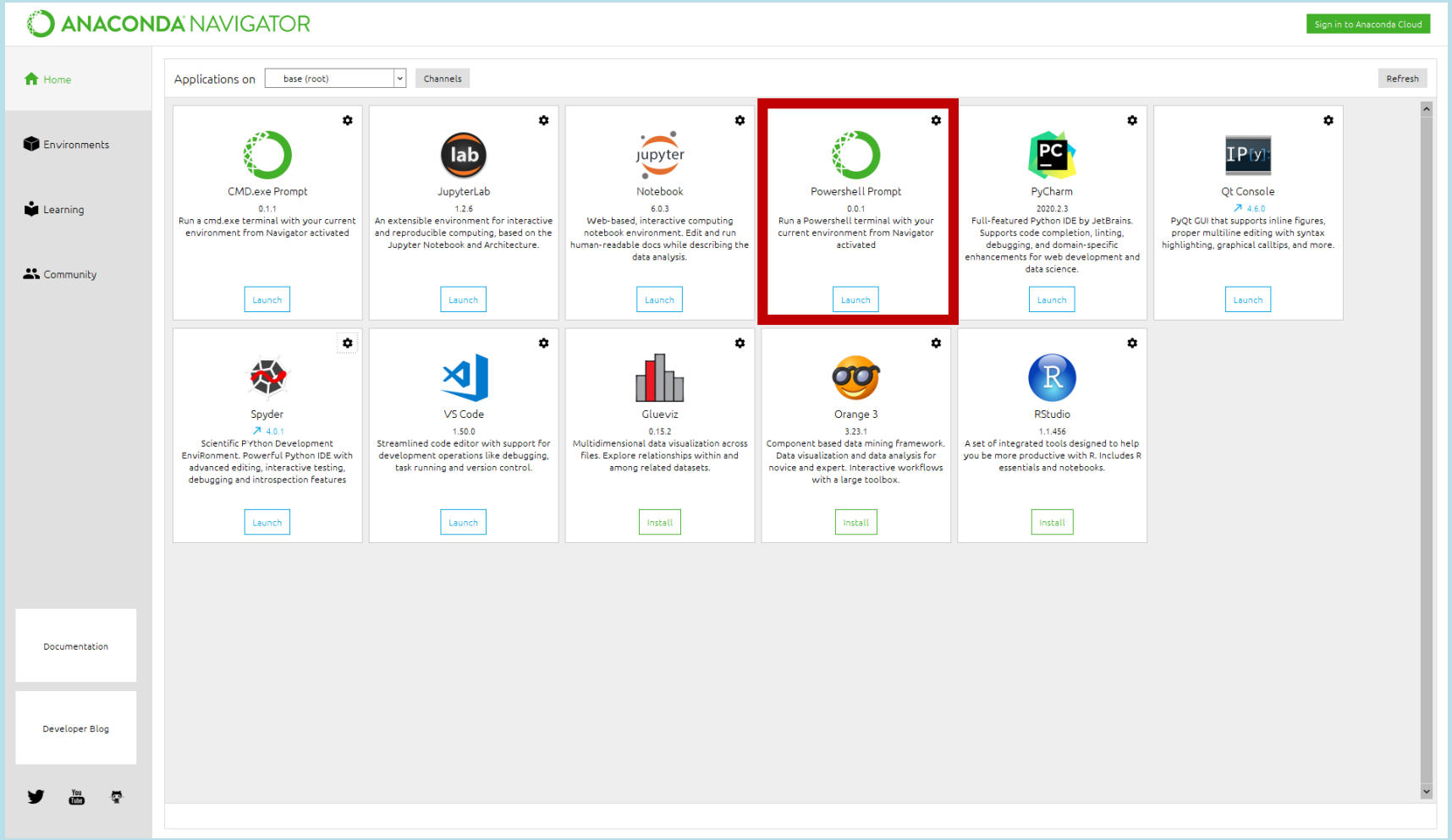
Comments

```
print("Hello Darmstadt!")  # This is a comment

'''
This is a multi-line comment.
Bla bla
'''
```

Multiple statements

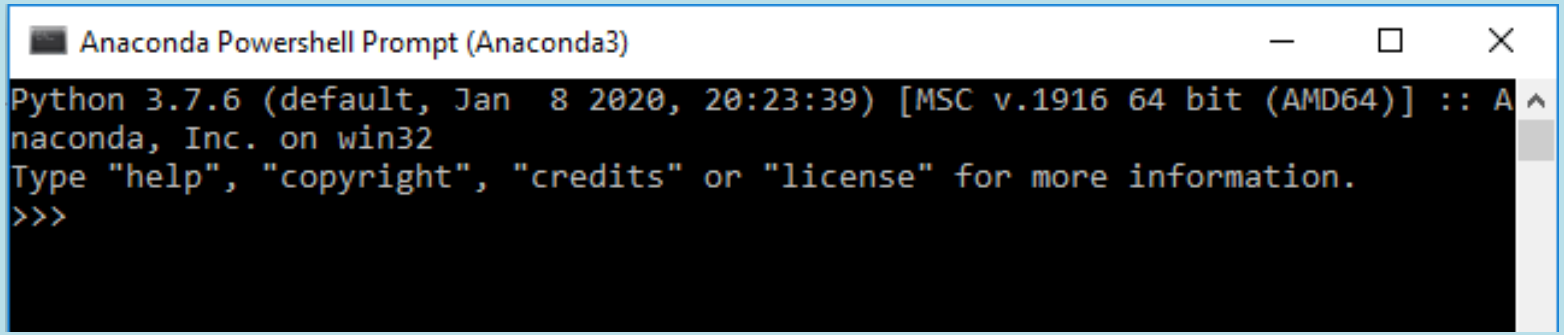
```
print("Hello Darmstadt!");print(" What's up? ")
```

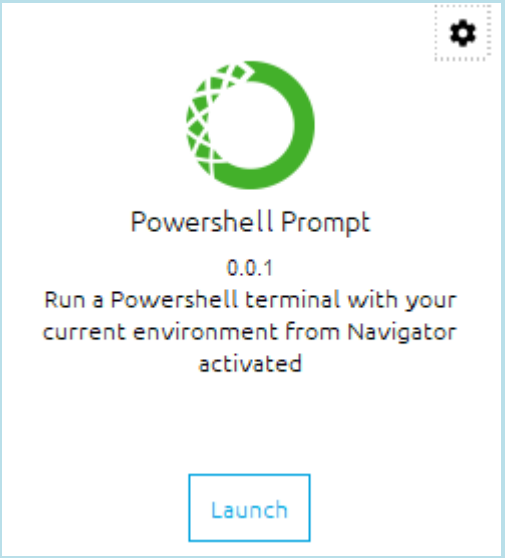
3.2 Start Python Prompt from Anaconda



3.3 First Steps in the Interactive Mode

- To run our first Python command open the Anaconda-Shell and type „Python“

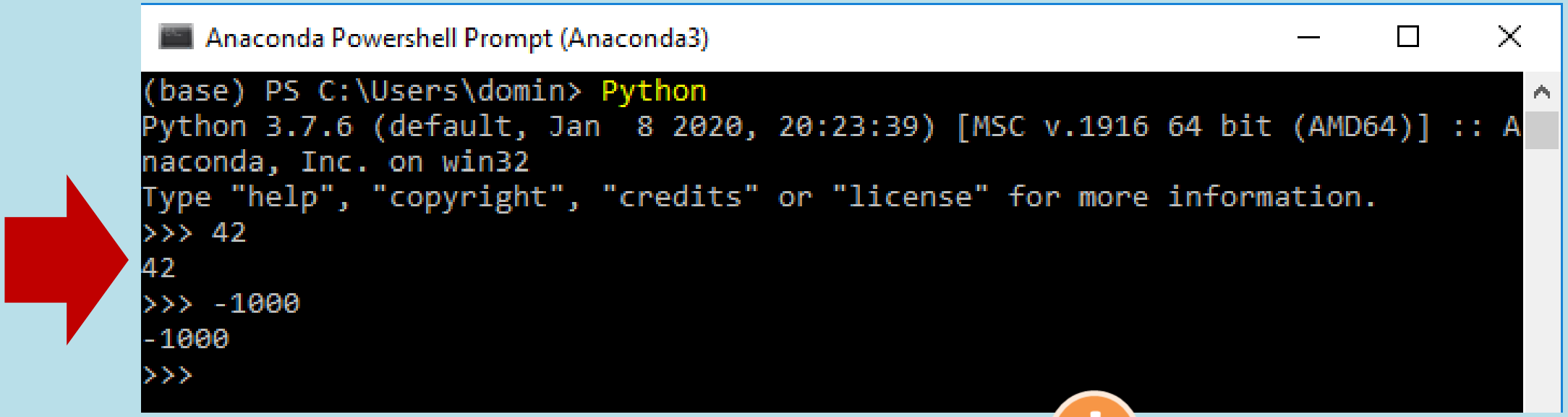




3.2 First Steps in the Interactive Mode

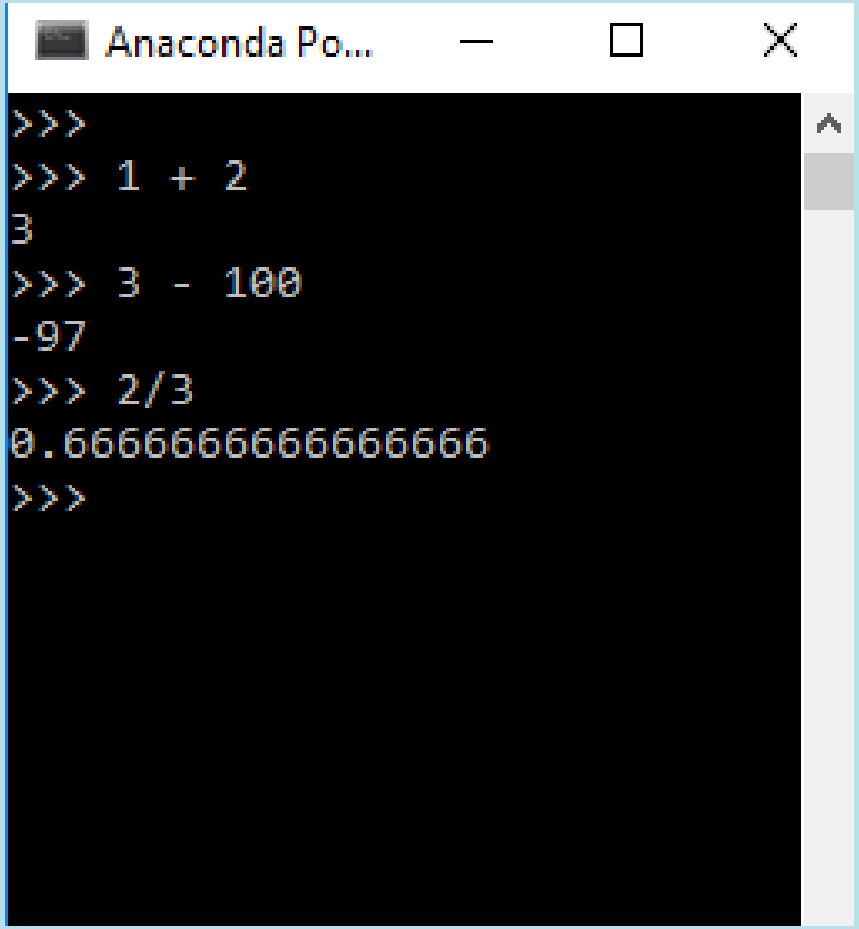
- Let us start by printing the following command in the Python interpreter

```
42
```



The interactive Python mode has a history functionality. Press ↑ or ↓ to re-read your previous code.

3.2 Arithmetic Operations in Python



Arithmetic Operators	
Operator	Command
Addition	`a + b = 3`
Subtraction	`a - b = 3`
Multiplication	`a * b = 3`
Division	`b/a = 1`
Modulus	`b % a = 0`
Exponent	`a ** b`

3.3 Data Types and Structures in Python

Data Structure

A data structure is a collection of data values, the relationships among them, and the functions or operations that can be applied to the data (Wegner & Reilly, pp. 507-512, 2003)

Primitive

Python has four primitive data structures

- `Integer` (42)
- `Float` (2.34567)
- `String` (“Hi world!”)
- `Boolean` (TRUE, FALSE)

To use that primitive data types, Python stores the type of an object with the object (see dynamic typing)

Non-primitive

Python has six non-primitive data structures

- `Array`
- `List`
  a. Linear: `Stacks, Queues`
  b. Non-Linear: `Graphs, Trees`
- `Tuple`
- `Dictionary`
- `Set`
- `File`

They contain the primitive data structures within more complex data structures for special purposes

Adapted from Ernesti, J. & Kaiser, P. (2017)

3.2 Float and Strings

- Float numbers are defined in the following manner

```
>>> 1.2345
1.2345
```

- While you can write strings like this

```
>>> "Hello Python fan"
‘Hello Python fan’

>>> "Hello" + "Python fan"
'HelloPython fan'
```


3.2 Logical Operators in Python

- Python can also handle boolean (or logical operations)

```
>>> 42 < 43
True

>>> 42 != 43
True
```

Comparison Operators	
Operator	Command
NOT	`!x`
AND (elementwise)	`x & y`
AND (1st element)	`x && y`
OR (elementwise)	`x | y`
OR (1st element)	`x || y`
Exclusive OR	`xor(x,y)`

Any idea how to solve the following logical formula with Python:
"`(a - 7) < (b * b + 6.5)`"

3.2 Variables

- In Python you can define variables

```
>>> name = 0.5
>>> var123 = 12
>>> string = "Hallo Welt!"
>>> liste = [1,2,3]
```

- You can access variables later by „calling“ them

```
>>> name
0.5
>>> 2 * name
1.0

>>> else = 4
?
```

Error to run `else = 4`? Some variable names are not allowed in Python, they are reserved for other purposes (e.g. “and”, “if”, “else” etc)

Adapted from Ernesti, J. & Kaiser, P. (2017)

3.2 More Complex Data Structures in Python

Array

```
import array as arr
a = arr.array("I",[3,6,9])
```

List

```
l = []  # empty list
l2 = [4, 8, 15, 16, 23, 42]
```

Dictionary

```
d = { "John" : ["alive", "0.5"],
      "Sansa" : ["alive ", "1"],
      "Eddard" : ["dead ", "1"],
      "Bran" : ["alive", "0"],
      "Arya" : ["missing", "1"]
    }
```

key value

Tuple

```
t = 1,2,3,4,5
```

Set

```
s = set("CAKE&COKE")
```

Files

```
f = open('file_name', 'w')
```

Adapted from Ernesti, J. & Kaiser, P. (2017)

3.2 Lists

- You retrieve values in a data structure by declaring an index inside a square bracket "[]" operator.

```
>>> l = [4, 8, 15, 16, 23, 42]

>>> l[1]
8
```

- You can add new values with append (add new value at the end of the list) and extend (add new list at the end of the list)

```
>>> l.append(1000)
>>> l
[4, 8, 15, 16, 23, 42, 1000]
```

3.2 Dictionaries

- You retrieve values in a data structure by declaring an index inside a square bracket "[]" operator.

```
>>> d = { "John" : ["alive", "0.5"],
>>>       "Arya" : ["missing", "1"]
>>>    }

>>> d["Arya"]
['missing', '1']
```

- And you can add new entries with

```
d["Brandon"] = ["crazy", "1"]
```

3.2 Functions

- From math we know that functions are a kind of assignment rules like:

$$f(x) = x + 2x$$

name, parameters, rule

- In computer science, we use functions to compute an output based on an input, like:

```
>>> max(1,2,3,4,5)
5
```

Functions without return values are termed “procedures”. But compared to other programming values (e.g. C or PASCAL) this makes no difference.

Adapted from Ernesti, J. & Kaiser, P. (2017); Offical Python Documentation (2019): https://docs.python.org

3.2 Writing functions

- In Python you can encapsulate parts of your code into „functions“. You can call them later in your code to reduce redundance

```
def function_name(parameter_1, …, parameter_n):
    command
    …
    command
```

```
def fancy_function(num1, num2):
    result = 42
    num3 = num1 + num2
    return(result-num3)
```

```
>>> variable = fancy_function(1,2)
39
```

Adapted from Ernesti, J. & Kaiser, P. (2017); Offical Python Documentation (2019): https://docs.python.org

3.2 Functions and Methods

- In Python exists many build-in functions you can use

```
>>> max([1,5,2,7,42,3])
42
```

build-in functions

Adapted from Ernesti, J. & Kaiser, P. (2017)

3.2 If else statements

- In Python you can write simple if statements

```
if x == 1:
    print("x is 1")
```

- Or more complex if-else-statements

```
if x == 1:
    print("x is 1")
elif x == 2:
    print("x is 2 or 1")
elif x == 3:
    print("I have no idea about x")
```

Adapted from Ernesti, J. & Kaiser, P. (2017)

3.2 Loops - while

- While statements

```
secret = 42
guess = 123
while guess != secret:
    guess = int(input("Please, give a guess: "))
print("You won!")
```

- You can “break” (leave) loops

```
secret = 42
guess = 123
while guess != secret:
    guess = int(input("Please, give a guess: "))
    if guess == 0:
        break
print("You won!")
```

Adapted from Ernesti, J. & Kaiser, P. (2017)

3.2 Loops - for

- For statements

```
for x in [1,2,3]:
   print(x)
```

What will the print function return?

- If you want to use for as counting loop use range()

```
range(stop)
range(start, stop)
range(start, stop, step)
```

```
for i in range(1, 10, 2):
    print(i)
```

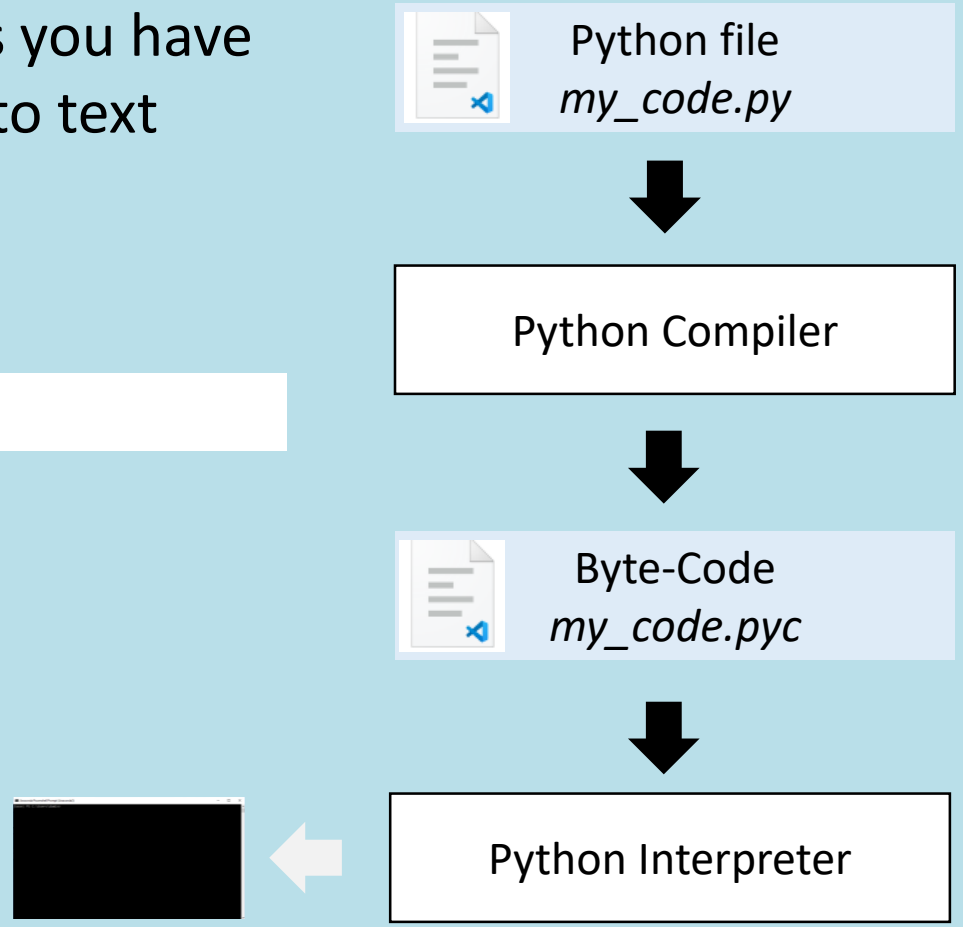
What will the print function return?

Adapted from Ernesti, J. & Kaiser, P. (2017)

3.2 Run Python Files from Console

- If you want to write and run simple scripts you have to write your Python commands (code) into text files and store them as <filename>.py
- Then you can run them from console

```
Python '<PATH TO YOUR FILE>.py'
```



Adapted from Ernesti, J. & Kaiser, P. (2017)

3.2 Python Shebang

- Under a Unix-like operating system such as Linux, Python program files can be made directly executable with the help of a shebang, also called Magic Line. To do this, the first line of the program file must usually be as follows:

```
#!/usr/bin/python
```

- In this case, the operating system is forced to always execute this program file with the Python interpreter. On other operating systems, for example Windows, the shebang line is ignored.

Adapted from Ernesti, J. & Kaiser, P. (2017)

Your turn!

Task

Please open the interactive mode to solve the following tasks.

In Python you can read in user input with `input()`. Save an user input into a new variable age. Than declare a variable `methusalem = 969`. Compare the user's `age` with `methusalem`'s age and `print()` the result.

Please note that you probably have to cast the user input as `int` (with `int()`) and to `string` (with `str()`) to print it.

Too difficult? Do not worry! We come back to this point in the exercise of this course.

3.3 Classroom task

- **Possibility 1**: Run the code from the interactive mode

```
>>> methusalem = 969
>>> your_age = input()
31
>>> print("You are " + str(methusalem - int(your_age)) + " year younger than the oldest man in the world!")
You are 938 year younger than the oldest man in the world!
>>>
```

- **Possibility 2**: Save your code as Python file (*.py) and run it from the console

```
(base) PS C:\Users\domin\Dropbox\Lehre\AI Algorithms and Applications with
Python\Code> Python '.\Lecture 3 -
Methusalem.py'
31
You are 938 year younger than the oldest man in the world!
```


Outline

3 Introduction into AI-Programming with Python

- 3.1 Software Tools and Programming Languages in AI
- 3.2 Foundations of Programming with Python
- 3.3 The Anaconda Toolbox for AI Programming
- 3.4 Advanced Programming with Python
- 3.5 AI Related Packages and Concepts in Python

Lectorial 1: Implement Problem-Solving Agents with Python

► **What we will learn:**

- Get an overview of AI software and programming with Python, so that you will be able to build your own agents and AI software components
- Workflow and tools to develop simple scripts and applications with Python
- Discuss advanced concepts of Python programming and AI related packages

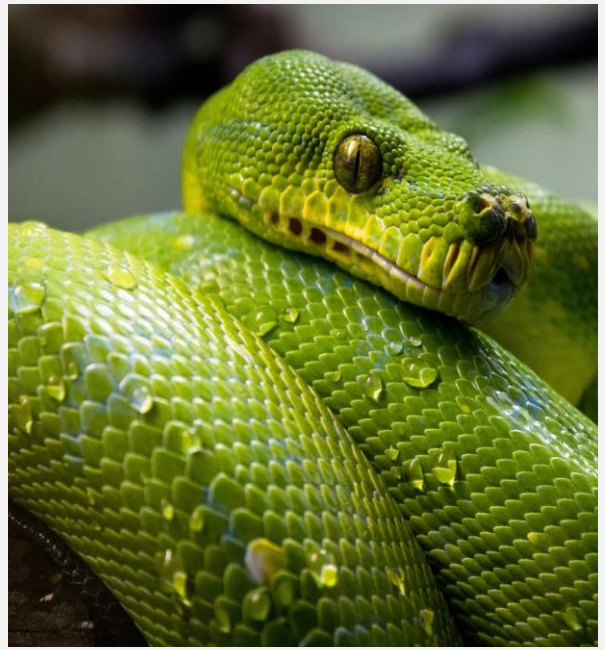


Image source: ↗ Pixabay (2019) / ↗ CC0

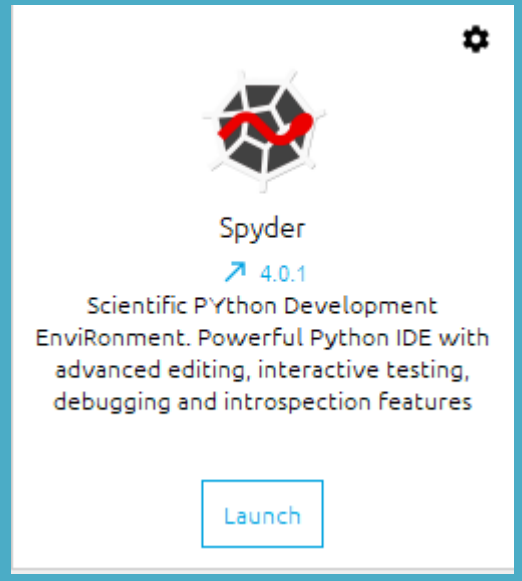
► **Duration:**

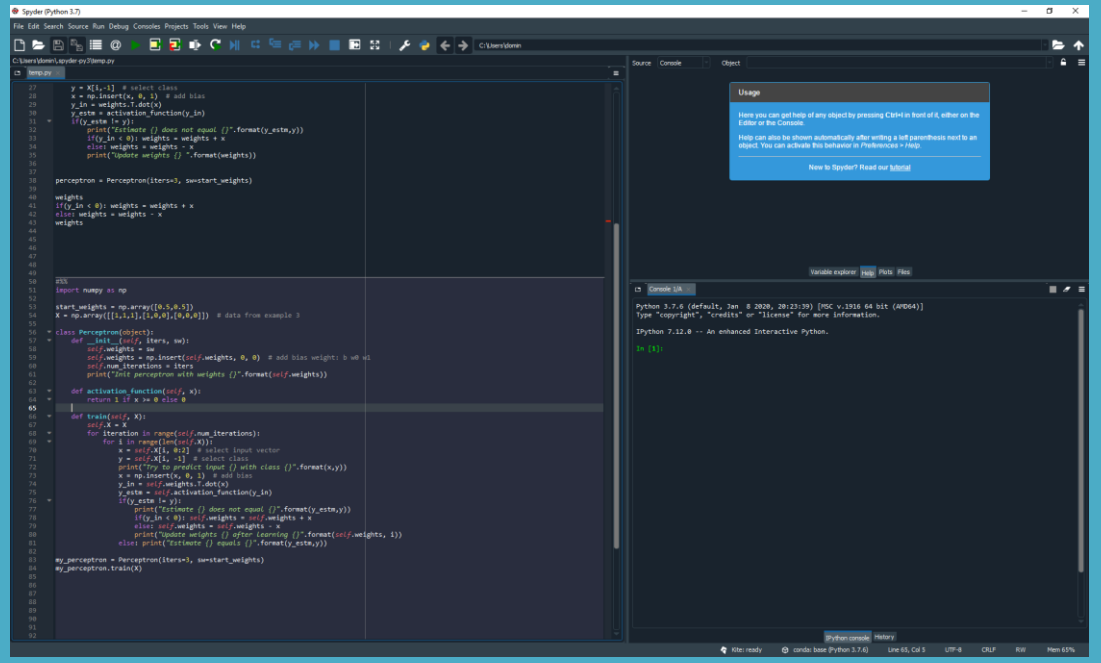
- 90 min

► **Relevant for Exam:**

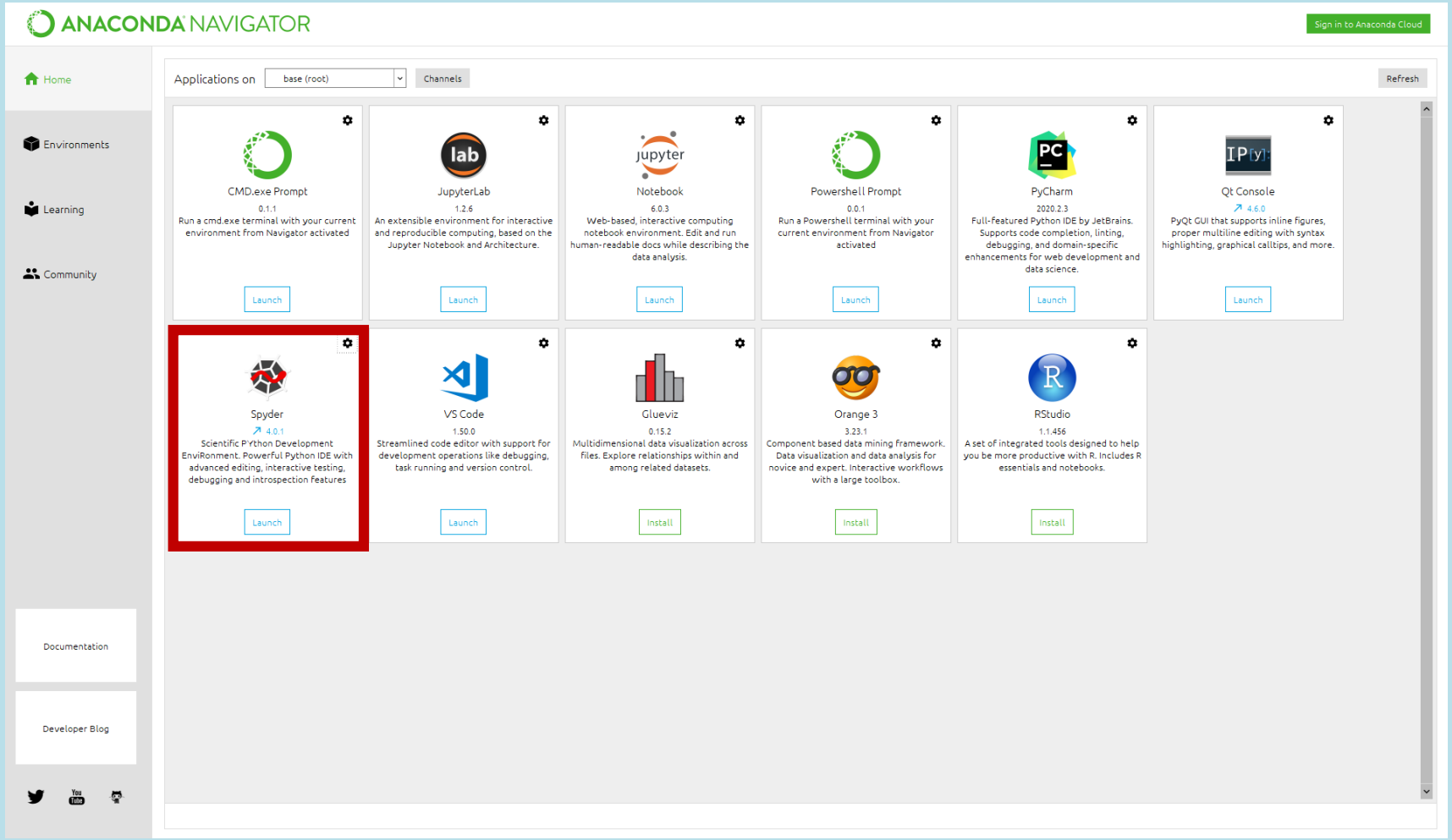
- 3.1 - 3.5

Now, please start the Spyder IDE from your Anaconda!

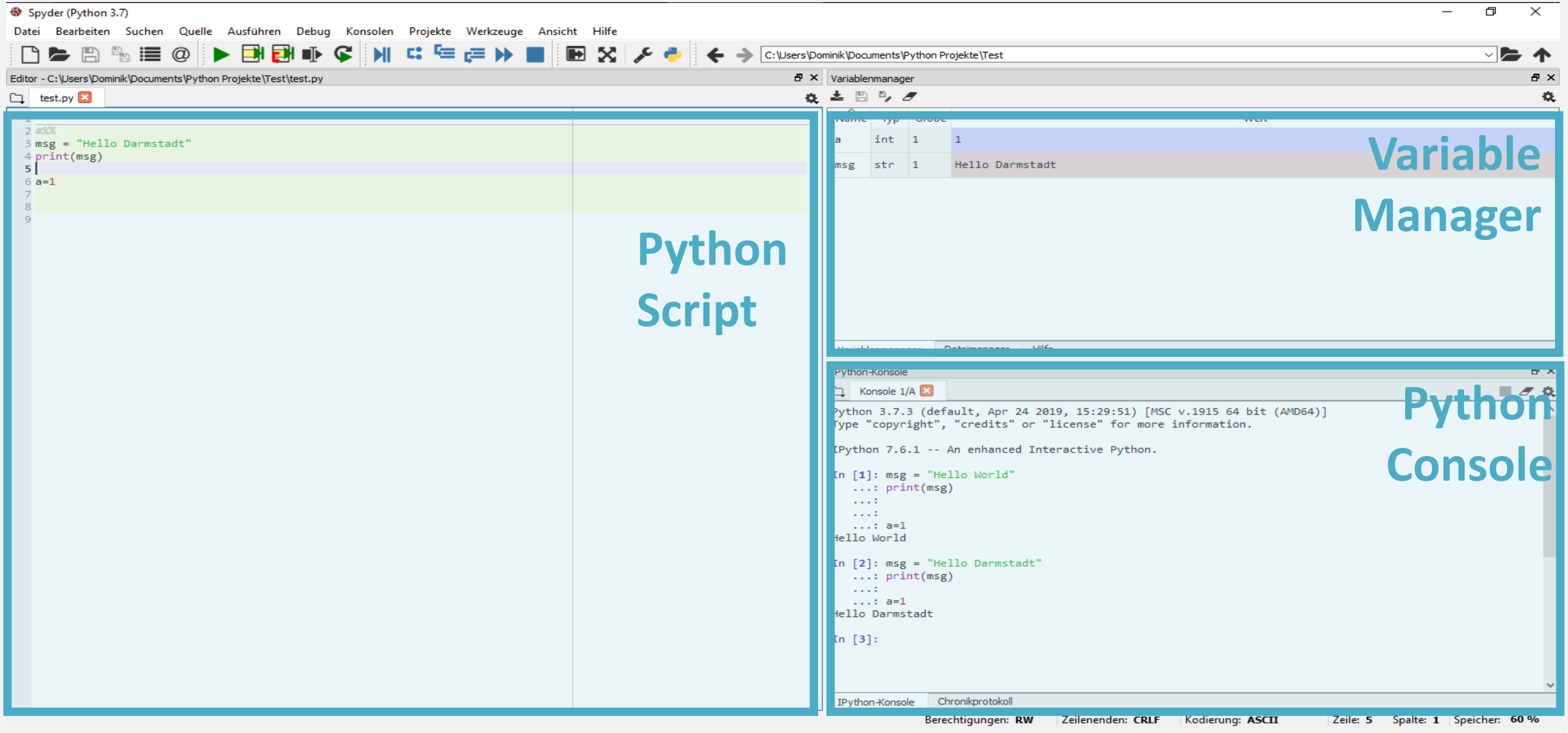




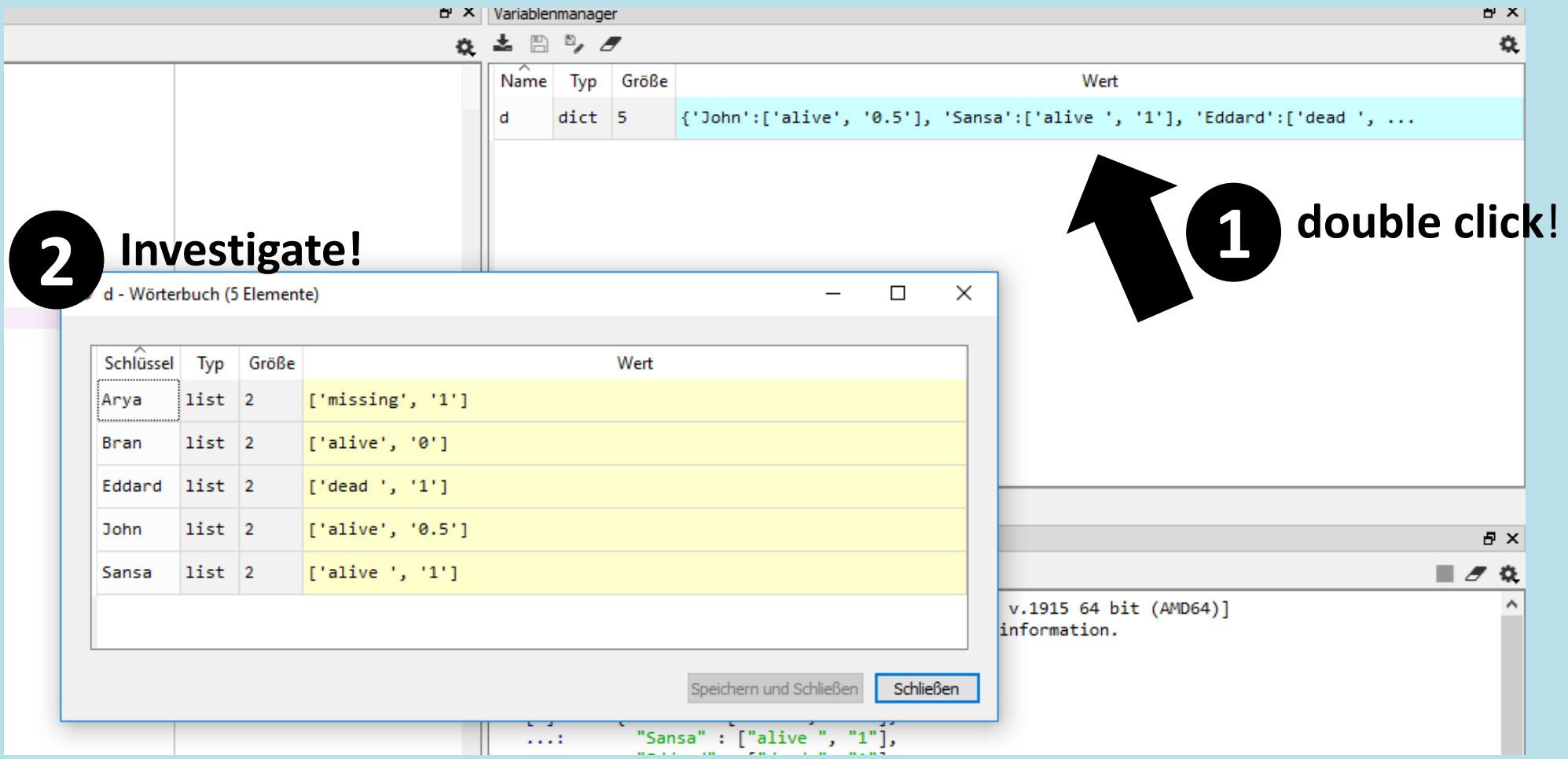
3.3 Start Spyder from Anaconda



3.3 Spyder IDE

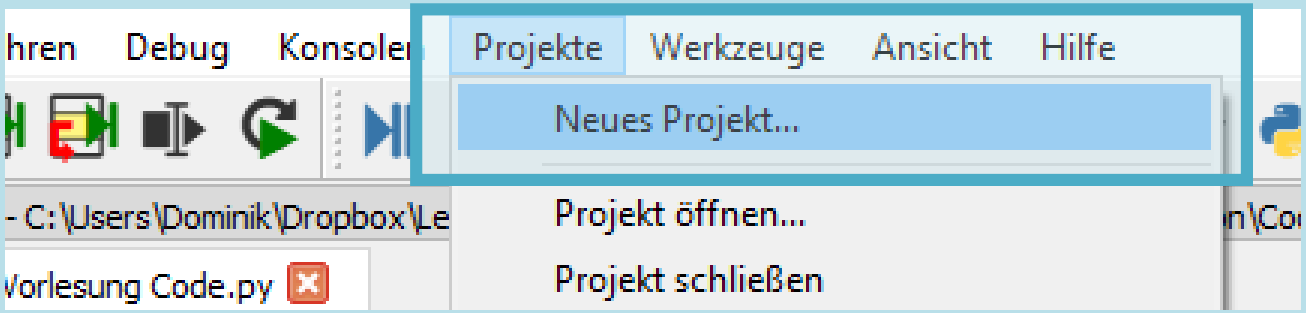


3.3 Investigate Complex Data Structures with the Variable Manager

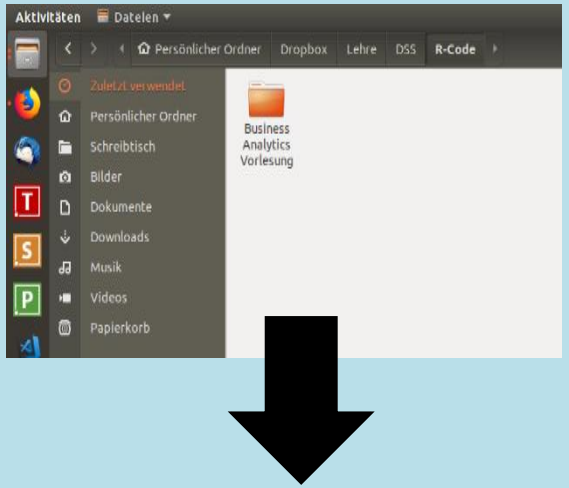


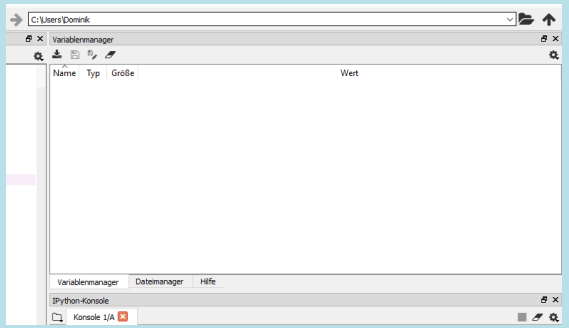
3.3 Projects with Spyder

- Spyder allows you to manage all your files and data with “projects”
- The benefit is that you can throw all your stuff in one folder and have direct access without browsing your files
- Furthermore you can save your current session
- And reload it easily

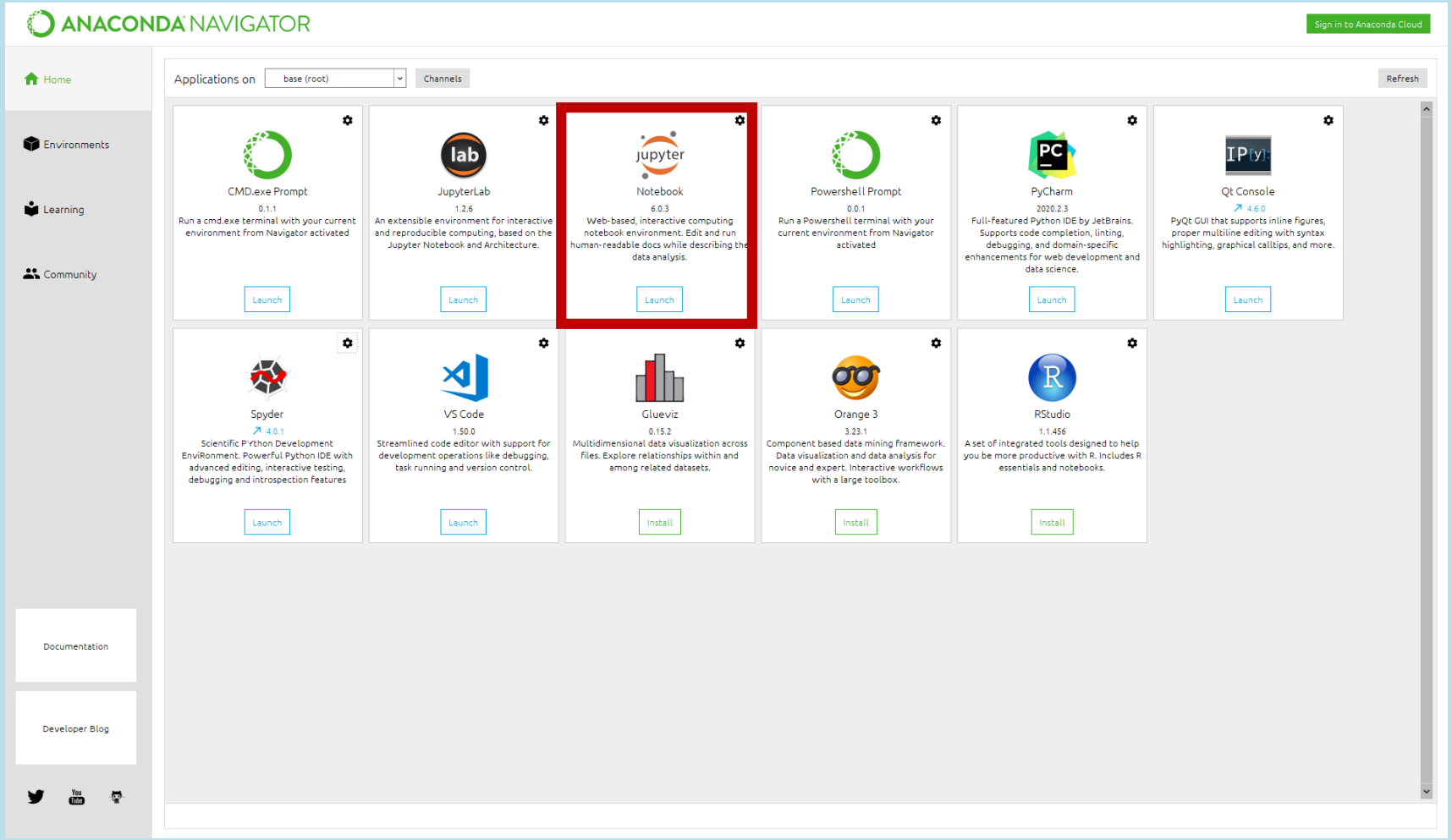


Data Management





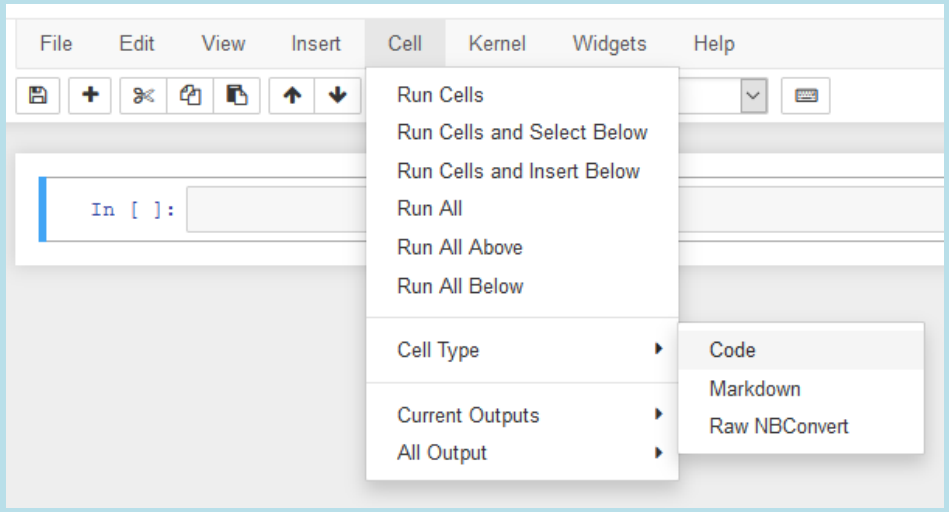
3.3 Start Jupyter from Anaconda



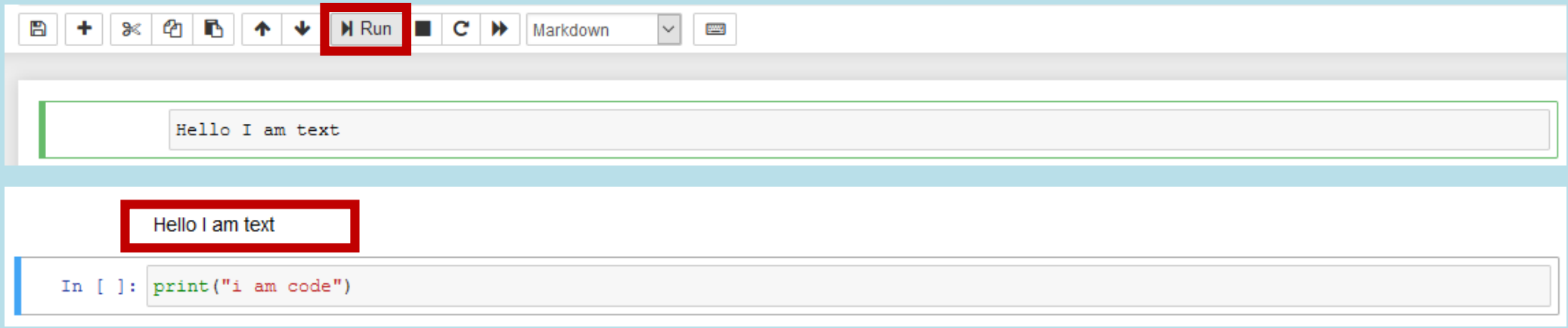
3.3 Notebooks

- Jupyter allows you to build simple code/text documents with your Python code

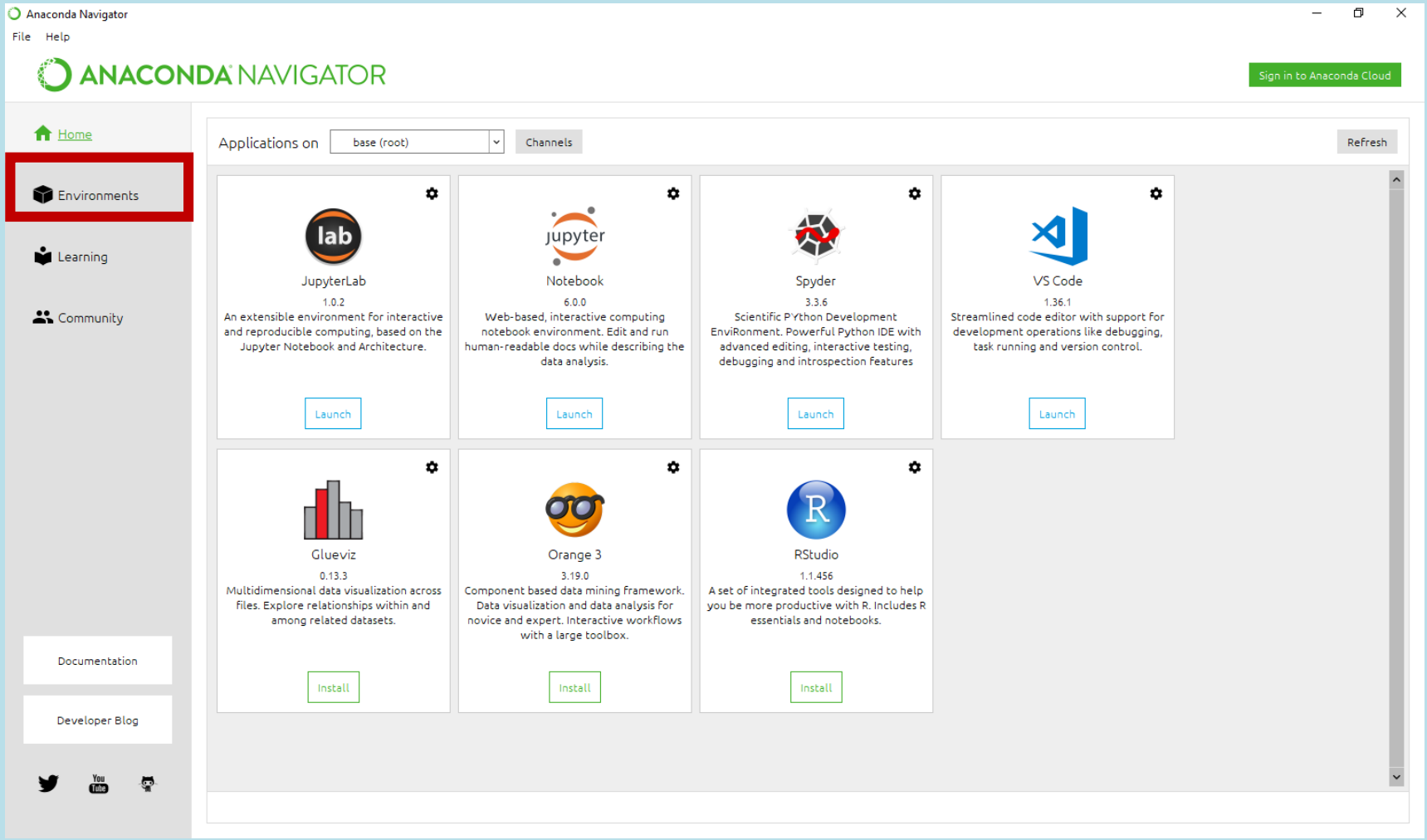
3.3 Notebooks



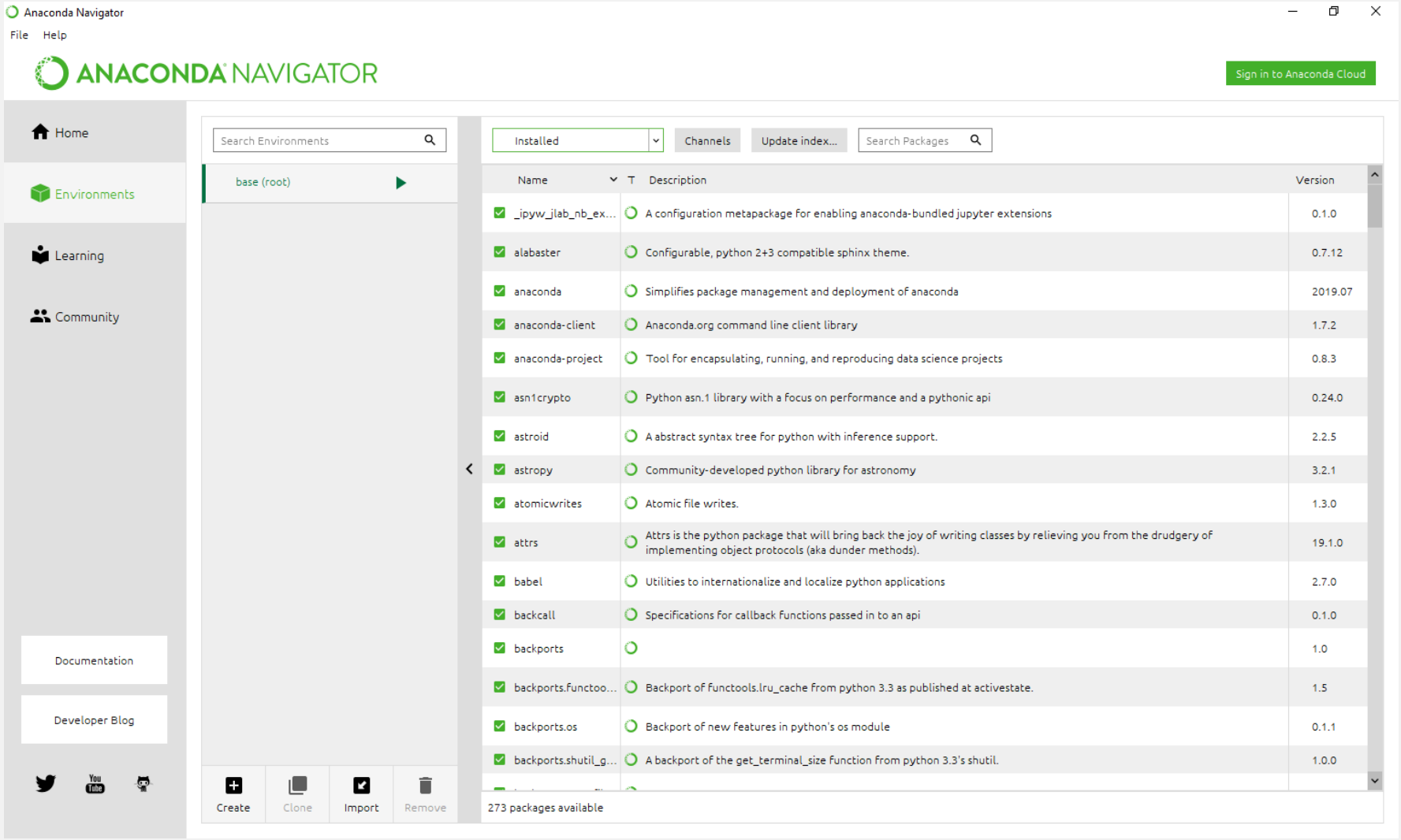
- Jupyter works with different cells. You can mark cells as text (markdown) or code (Python code) cells.
- If you press, run they get activated



3.3 Package Management in Anaconda



3.3 Install and load new Python/R packages



You can manage and organize your environments with the Anaconda Navigator

Use it to install and load new packages

Your turn!

Task

Please explain the difference between Python Development in the Spyder IDE and Jupyter notebooks. Can you name specific use cases when Spyder is a better choice than Jupyter and vice-versa?

Outline

3 Introduction into AI-Programming with Python

- 3.1 Software Tools and Programming Languages in AI
- 3.2 Foundations of Programming with Python
- 3.3 The Anaconda Toolbox for AI Programming
- 3.4 Advanced Programming with Python
- 3.5 AI Related Packages and Concepts in Python

Lectorial 1: Implement Problem-Solving Agents with Python

► **What we will learn:**

- Get an overview of AI software and programming with Python, so that you will be able to build your own agents and AI software components
- Workflow and tools to develop simple scripts and applications with Python
- Discuss advanced concepts of Python programming and AI related packages

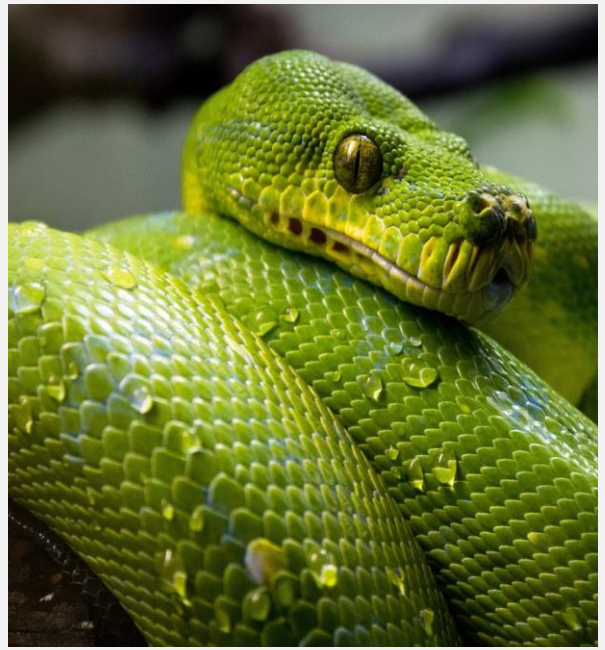


Image source: ↗ Pixabay (2019) / ↗ CC0

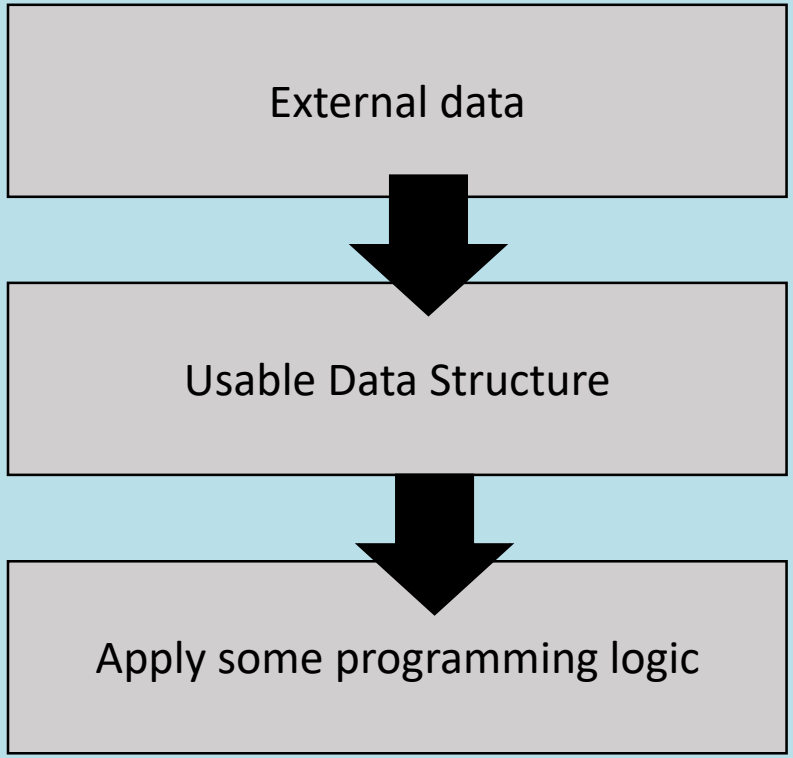
► **Duration:**

- 90 min

► **Relevant for Exam:**

- 3.1 - 3.5

2.4 Advanced Concepts



Example: Guessing Game

```
Please enter a Porsche model: 991
Unknown Porsche model

Please enter a Porsche model: 911
The car has the following VMax:  330 km/h

Please enter a Porsche model:
```

Adapted from Ernesti, J. & Kaiser, P. (2017); Image Source: ↗ Porsche 911 R im Porsche Museum 2018 (2018) by Alexander Migl from ↗ Wikimedia / ↗ CC-BY-SA-4.0 (image not edited)

3.4 Reading Files

- If you want to read a file in your Python code, it must be opened for reading
- For this we use the Built-in Function `open()`. This function returns a so-called file object that we can access:

```
fobj = open("database_extract.txt", "r")
for line in fobj:
    print(line)
fobj.close()
```

We will discuss data imports and handling with Python later in chapter 4!

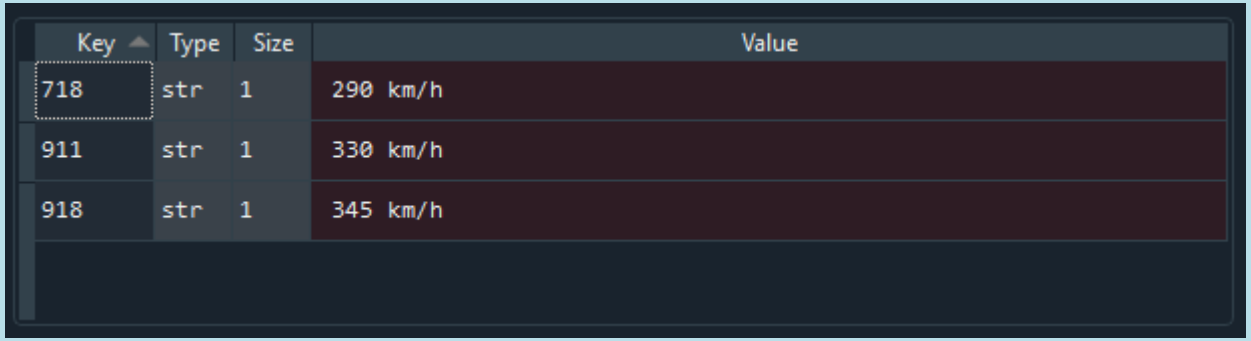
718	290 km/h
911	330 km/h
918	345 km/h

Adapted from Ernesti, J. & Kaiser, P. (2017); Image Source: ↗ Porsche 911 R im Porsche Museum 2018 (2018) by Alexander Migl from ↗ Wikimedia / ↗ CC-BY-SA-4.0 (image not edited)

3.4 Reading Files into Variables I

- Let us know load the file data into a dictionary

```
data = {}
fobj = open("database_extract.txt", "r")
for line in fobj:
    line = line.strip()
    l = line.split(",")
    data[l[0]] = l[1]
fobj.close()
```



Key	Type	Size	Value
718	str	1	290 km/h
911	str	1	330 km/h
918	str	1	345 km/h

Adapted from Ernesti, J. & Kaiser, P. (2017); Offical Python Documentation (2019): https://docs.python.org

3.4 Reading Files into Variables II

- Let us know load the file into a dictionary

```
data = {}
fobj = open("database_extract.txt", "r")
for line in fobj:
    line = line.strip()
    l = line.split(",")
    data[l[0]] = l[1]
fobj.close()

while True:
    car = input("Please enter a Porsche model: ")
    if car in data:
        print("The car has the following VMax:", data[car])
    else:
        print("Unknown Porsche model")
```

```
Please enter a Porsche model: 991
Unknown Porsche model

Please enter a Porsche model: 911
The car has the following VMax:  330 km/h

Please enter a Porsche model:
```

718	290 km/h
911	330 km/h
918	345 km/h

Adapted from Ernesti, J. & Kaiser, P. (2017); Offical Python Documentation (2019): https://docs.python.org

3.4 Write Data into Files

- You can also write data back into files

```
data = {"Taycan" : "260 km/h", "Tesla" : "250 km/h"}

fobj = open("my_file.txt", "w")

for car in data:
    fobj.write("{}, {}\n".format(car, data[car]))
fobj.close()
```

Adapted from Ernesti, J. & Kaiser, P. (2017); Offical Python Documentation (2019): https://docs.python.org

3.4 Read Files Online

- Online files can be accessed directly from Python

```
# Read from online
import urllib3

http = urllib3.PoolManager()

target_url =
"https://raw.githubusercontent.com/dominikjung42/AIAlgorithmsAndApplic
ations/master/Code/database_extract.txt"

Response = http.request("GET", target_url)

data = response.data.decode("utf-8")
```

3.4 Python Runtime Model I

- We have discussed that you can assign variables with `var = value`

```
your_variable = 42
```

- Then you can call the variable

```
>>> your_variable/2
21.0
```

- Alternatively, you can make an instance of the value directly

```
>>> 42
42
```

Adapted from Ernesti, J. & Kaiser, P. (2017); Offical Python Documentation (2019): https://docs.python.org

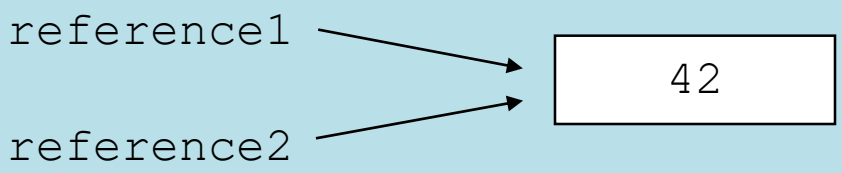
3.4 Python Runtime Model II

- We have discussed that you can assign variables with `var = value`

```
your_variable = 42
```

- Then you can call the variable

```
reference1 = 42
reference2 = reference1
```

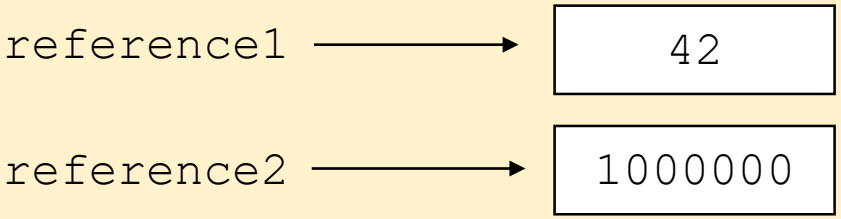


Adapted from Ernesti, J. & Kaiser, P. (2017); Offical Python Documentation (2019): https://docs.python.org

3.4 Python Runtime Model - Example

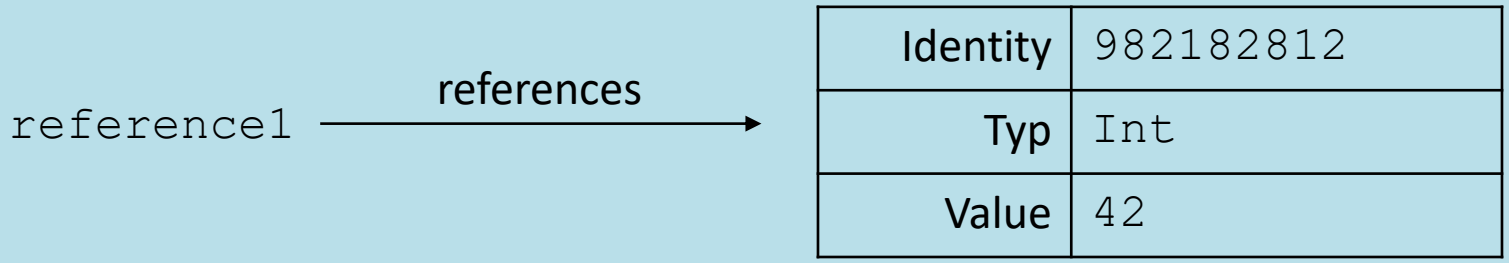
```
>>> reference1 = 42
>>> reference2 = reference1

>>> reference1
42

>>> reference2
42

>>> reference1 = 1000000
>>> reference1
1000000

>>> reference2
42
```

You have to distinguish references as references to instances from the instances themselves. This is a common error in Python programming!



Adapted from Ernesti, J. & Kaiser, P. (2017); Offical Python Documentation (2019): https://docs.python.org

3.4 Structure of Instances in Python



- Every object has an identity, a type and a value
- An object's *identity* never changes once it has been created; you may think of it as the object's address in memory

Adapted from Ernesti, J. & Kaiser, P. (2017); Offical Python Documentation (2019): https://docs.python.org

3.4 Object-oriented Programming I

- Python also supports object-oriented programming (oop):

```
class Porsche:
    # Class attribute
    type = "718"

    # Constructor
    def __init__(self, owner):
        self.owner = owner
```

```
>>> my_car = Porsche("Dominik")
>>> my_car.owner

'Dominik'
```

Adapted from Ernesti, J. & Kaiser, P. (2017); Offical Python Documentation (2019): https://docs.python.org

3.4 Object-oriented Programming II

```
class Porsche:
    # Class attribute
    type = "718"

    # Constructor
    def __init__(self, owner):
        self.owner = owner

    # Instance method
    def speak(self):
        return ("{} says: Hello {}!".format(self.type, self.owner))
```

```
>>> my_car.speak()
'718 says: Hello Dominik!'
```

Adapted from Ernesti, J. & Kaiser, P. (2017); Offical Python Documentation (2019): https://docs.python.org

Your turn!

Task

Please open your Python IDE (e.g. Spyder) and try to solve the following tasks

- Write a program that generates a random number between 1 and 9 `numpy.random.randint(1, 10)` and the user has to guess the number.
- If the guess is wrong he has to guess a further round, otherwise the program returns "Congratulation, you won!"
- You can use the code from chapter *3.4 Reading Files into Variables II* as a start point

Artificial Intelligence

Algorithms and Applications with Python

Chapter 3.5

Dr. Dominik Jung
dominik.jung42@gmail.com

</> course material: ↗ https://github.com/dominikjung42

Outline

3 Introduction into AI-Programming with Python

- 3.1 Software Tools and Programming Languages in AI
- 3.2 Foundations of Programming with Python
- 3.3 The Anaconda Toolbox for AI Programming
- 3.4 Advanced Programming with Python
- 3.5 AI Related Packages and Concepts in Python

Lectorial 1: Implement Problem-Solving Agents with Python

► **What we will learn:**

- Get an overview of AI software and programming with Python, so that you will be able to build your own agents and AI software components
- Workflow and tools to develop simple scripts and applications with Python
- Discuss advanced concepts of Python programming and AI related packages

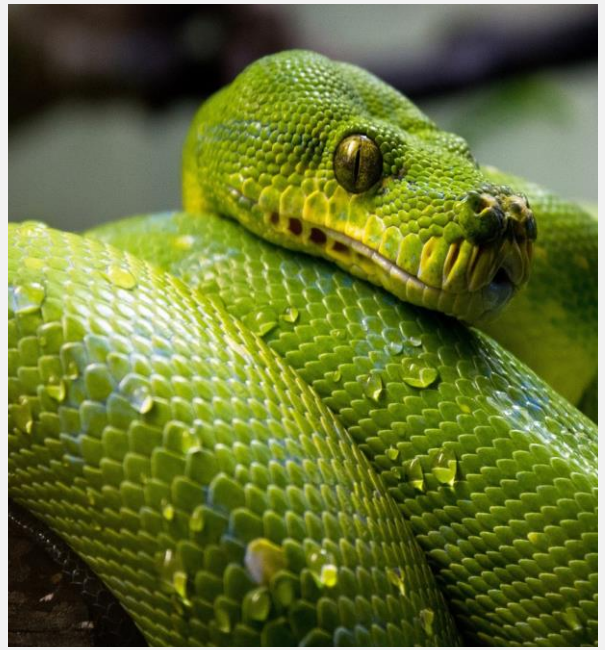


Image source: ↗ Pixabay (2019) / ↗ CC0

► **Duration:**

- 90 min

► **Relevant for Exam:**

- 3.1 - 3.5

3.5 Modules

- We learnt that in Python exists many build-in functions you can use

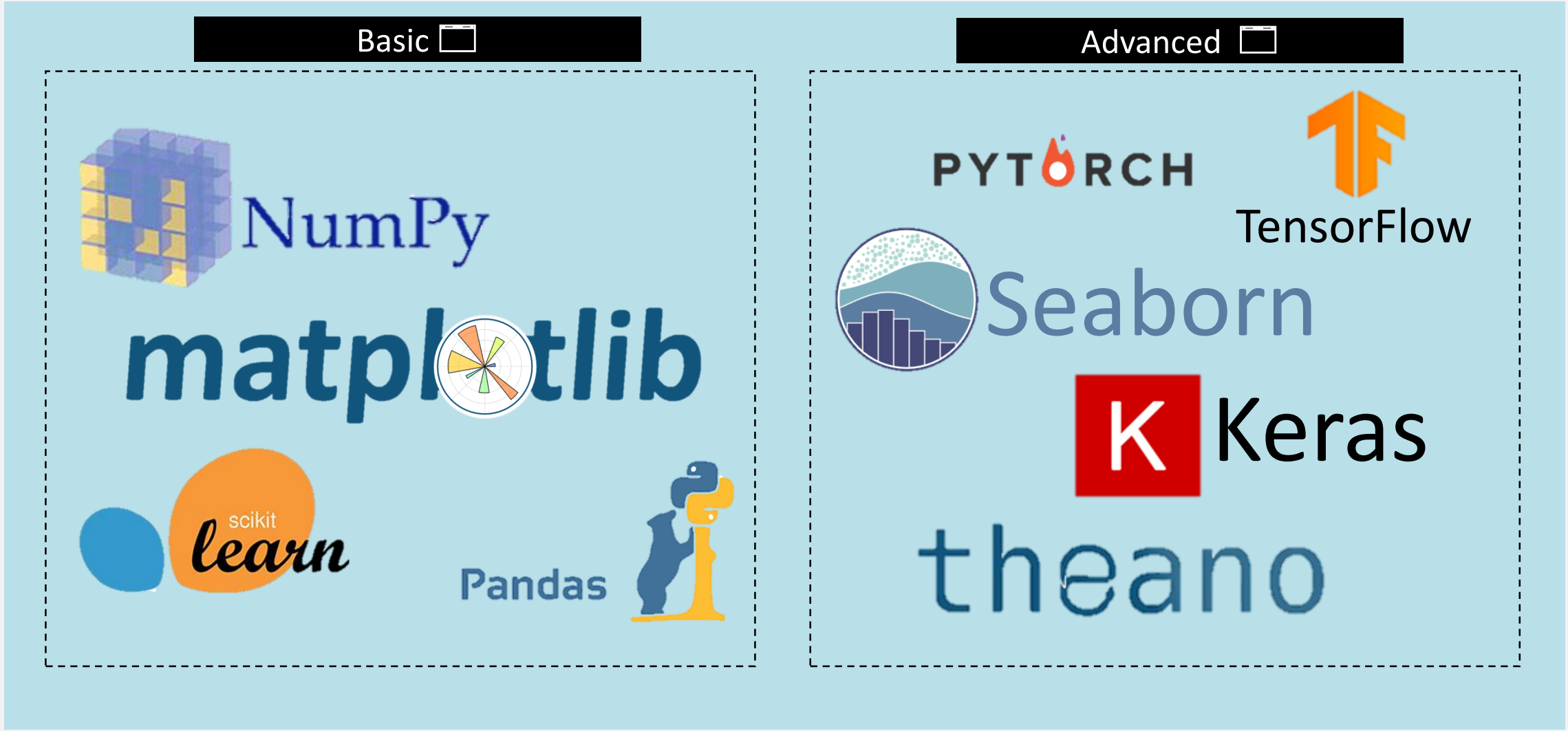
```
>>> max([1,5,2,7,42,3])
42
```

build-in functions

- However, there are many functions organized in external modules (like specific AI functions) we can use in our programm

Adapted from Ernesti, J. & Kaiser, P. (2017)

3.5 Most relevant Packages for AI-Specialists



Images: All images are not my own

3.5 Use these Modules in Python

- You can import existing modules (global modules) with `import`

```
>>> import math
>>> math.sin(math.pi)
1.2246467991473532e-16
```

```
>>> from math import *
>>> sin(math.pi)
1.2246467991473532e-16
```

```
>>> from math import sin, pi
>>> sin(math.pi)
1.2246467991473532e-16
```

Adapted from Ernesti, J. & Kaiser, P. (2017); Offical Python Documentation (2019): https://docs.python.org

3.5 Numpy

```
>>> import numpy
```

The fundamental package for scientific computing with Python

GET STARTED

We will use this package for each kind of mathematical computation in this lecture (see e.g. chapter 9)!

3.5 Basics of Numpy

- In Numpy you will mainly work with homogeneous multidimensional arrays

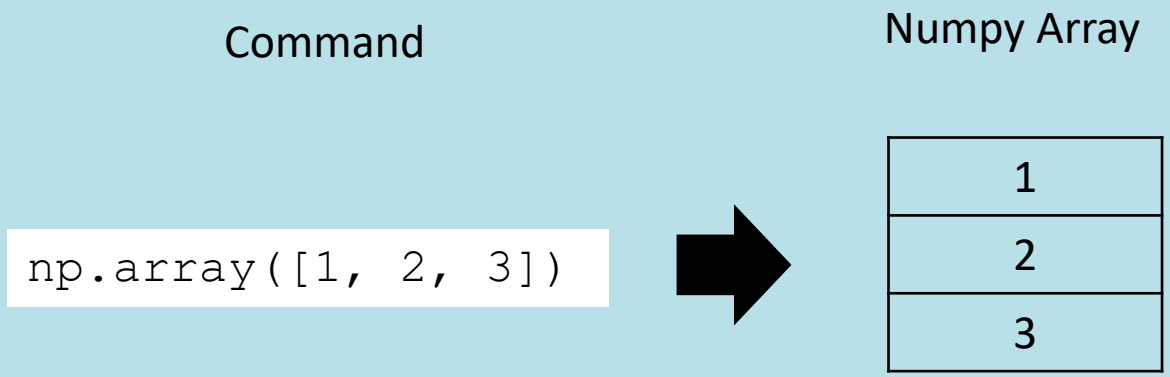
```
[[ 1., 1., 4.],
 [ 0., 3., 2.]]
```

- NumPy gives you an enormous range of fast and efficient ways of creating arrays and manipulating numerical data inside them.
- While a Python list can contain different data types within a single list, all of the elements in a NumPy array should be homogeneous. The mathematical operations that are meant to be performed on arrays would be extremely inefficient if the arrays weren't homogeneous.

3.5 Create Arrays with Numpy

- To create a NumPy array, you can use the function `np.array()`.

```
>>> import numpy as np
>>> a = np.array([1, 2, 3])
```



3.5 Numpy Indexing

```
data = np.array([1, 2, 3])
```

1	1
2	2
3	3

`data[0]`

1

`data[1]`

2

`data[0:2]`

1
2

`data[1:]`

2
3

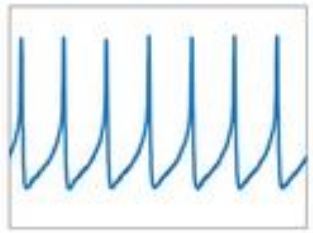
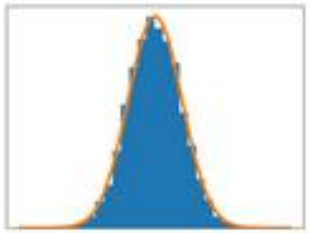
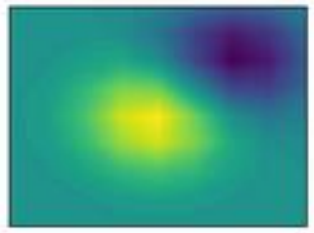
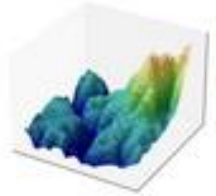
`data[-2:]`

0	1	
1	2	-2
2	3	-1
3		

```
>>> import matplotlib
```

Matplotlib: Visualization with Python

Matplotlib is a comprehensive library for creating static, animated, and interactive visualizations in Python.

Matplotlib makes easy things easy and hard things possible.

We will use this package for each kind of visualization in this lecture (see e.g. chapter 4)!

3.5 About Matplotlib

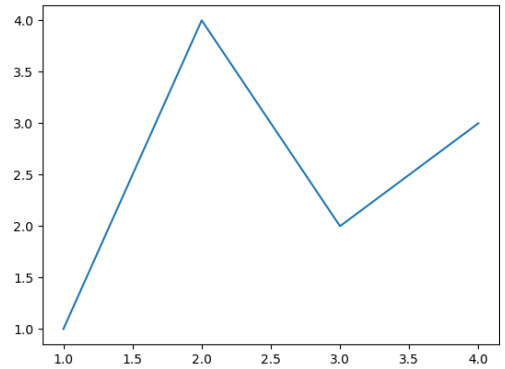
- Matplotlib is a visualization library for Python, and in particular its numerical mathematics extension `Numpy`

- It provides an object-oriented API for embedding plots into applications using general-purpose GUI toolkits like Tkinter, wxPython, Qt, or GTK+.

3.5 Generating Plots

- Matplotlib graphs your data on Figures, each of which can contain one or more Axes

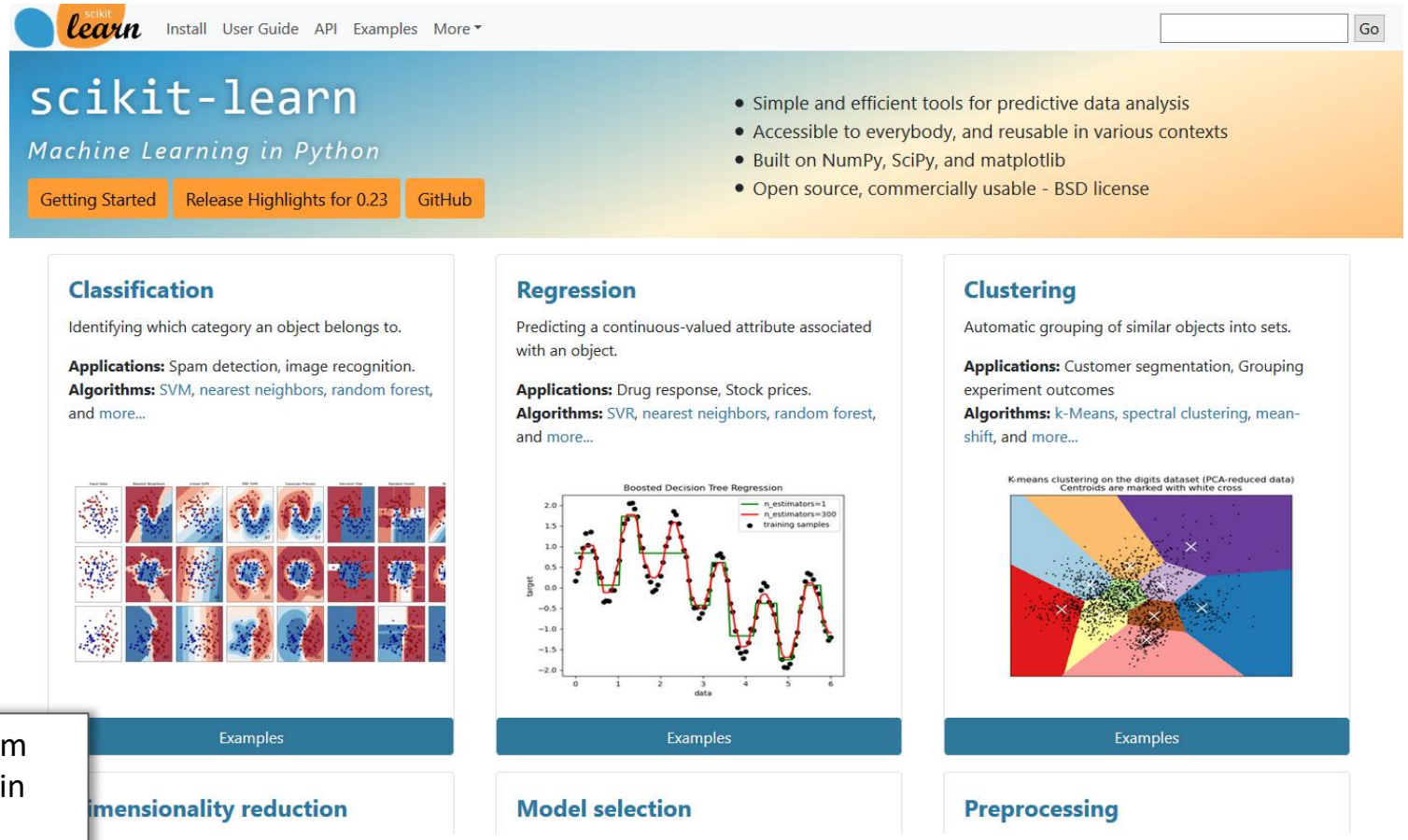
```
import matplotlib.pyplot as plt
import numpy as np

fig, ax = plt.subplots()  # Create a figure containing a single axes.
ax.plot([1, 2, 3, 4], [1, 4, 2, 3])  # Plot some data on the axes.
```



Adapted from matplotlib.org

3.5 Scikit-learn

```
>>> import sklearn
```

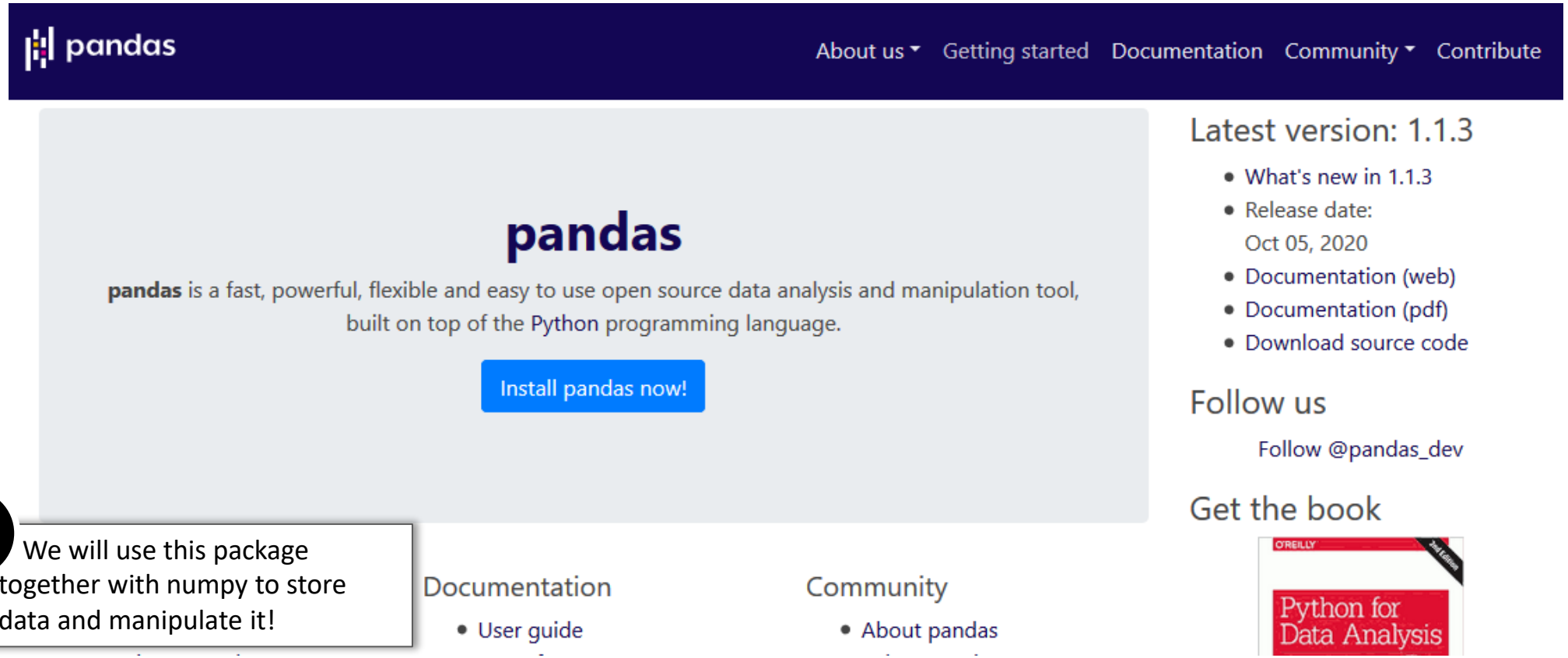


We will use the AI algorithms from this package (see e.g. chapter 5,6) in the following parts of this lecture!

3.5 Pandas

Pandas

```
>>> import pandas
```



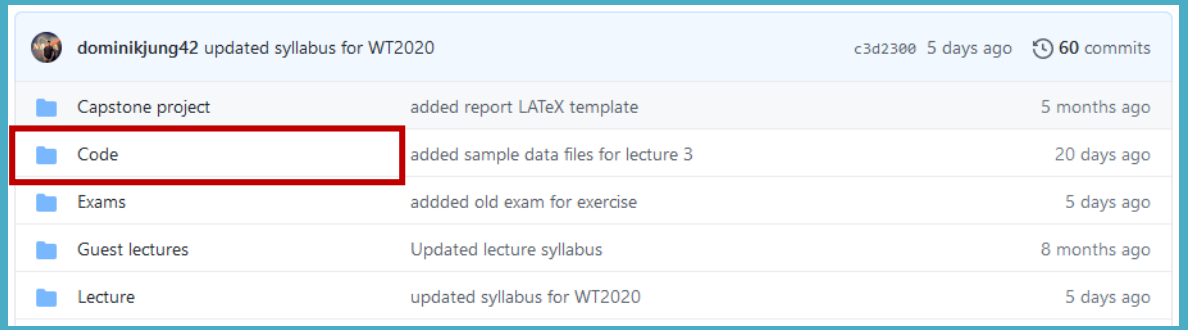
We will use this package together with numpy to store data and manipulate it!

Your turn!

Task

Check out the different packages and make the beginner tutorial of `numpy` and `pandas` to understand the fundamentals and concepts behind these packages. You will need these packages at the beginning of the lecture.

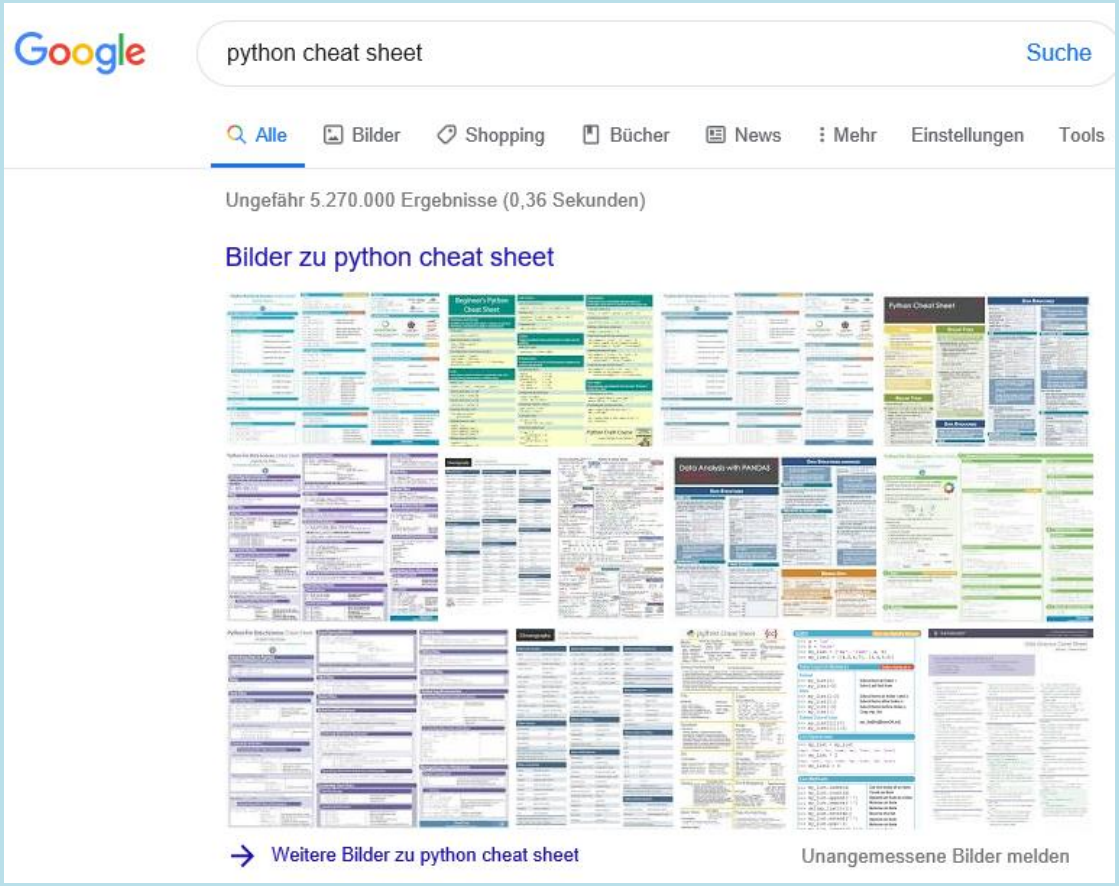
Your first end-to-end project:
Problem-Solving Agents



Feel free to start directly or just continue with the lecture and come back to it later.

Handouts

Python cheatsheet



- Python cheatsheets are very helpful to learn the basics
- They exist for very different topics

Exercises

Workbook Exercises

- *There are no workbook exercises in this unit*

Coding Exercises

- There are many, many Python Beginner books and videos on Youtube. However, the official Python Beginner Tutorial is also not a bad place to start. Make at least this tutorial to deepen the information of this lecture: ↗Python getting started.

You do not have to start the coding exercises directly after each lecture to understand the following lectures. However, they will definitly help you to deepen your understanding of the presented concepts and algorithms.

References

Literature

1. Ernesti, J., & Kaiser, P. (2017). *Python 3: das umfassende Handbuch*. Rheinwerk Verlag.
2. Offical Python Documentation (2020). The Python Language Reference. Online available at https://docs.python.org

News articles

1. Waggoner, P (2018): Advice to Young (and Old) Programmers: A Conversation with Hadley Wickham. Online available at https://www.r-bloggers.com/2018/08/advice-to-young-and-old-programmers-a-conversation-with-hadley-wickham/

References

Images

All images that were not marked other ways are made by myself, or licensed ↗CC0 from ↗Pixabay.

Further reading

- I also can recommend to take a look at the beginner tutorials of the different Python packages (↗ Numpy, ↗ Matplotlib, ↗ Scikit-learn, ↗ Pandas)
- Python has a manifest of ist design principles, which you can find inside the Python interpreter itself by typing `import this`. Check it out!

Glossary

Data Frame	*A two-dimensional data structure, where data is aligned in a tabular fashion in rows and columns.*
IDE	*Integrated Development Environment. Applications that facilitate the development of software or code by giving the user an interface to work with.*
Libraries/Packages	*Collection of pre-written code (functions and methods) to perform certain task.*
Notebooks	*Virtual environment that enables literate programming.*
Programming language	*Language engineered to create a standard form of instructions for a computer. Like human languages they are split into two components, i.e. syntax (form) and semantics (meaning).*
Software Tool	*Software that is designed to be used for a specific use case (e.g. Dashboarding, Data Visualization, Modelling).*

The Zen of Python (Tim Peters)

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!

Offical Python Documentation (2019): https://docs.python.org