

Artificial Intelligence

Algorithms and Applications with Python

Chapter 3



Dr. Dominik Jung
dominik.jung42@gmail.com



This lecture is aimed at two complementary audiences:

- Intermediate information systems / computer science students who want to get a general understanding of artificial intelligence (AI), understand how AI works, and learn new strategies for solving diverse AI problems.
- Students from other domains who are planning to use AI methods (e.g. machine learning) in their future and want to understand why it works the way it does.



If you are familiar with Python you can skip this chapter and continue in chapter 4.

3 Introduction into AI-Programming with Python

3.1 The Anaconda Toolbox for AI Programming

3.2 Foundations of Programming with Python

3.3 AI Programming with 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

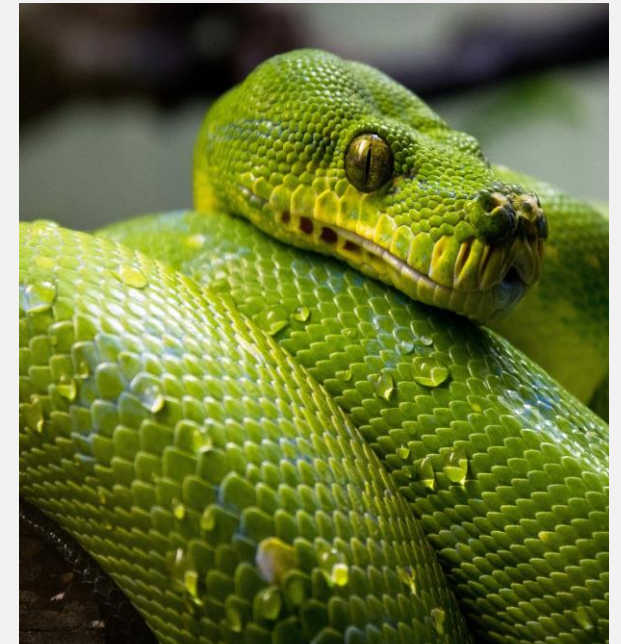


Image source: ↗ [Pixabay](#) (2019) / ↗ [CCO](#)

► Duration:

- 90 min

► Relevant for Exam:

- 3.1 - 3.5

3.1 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 wish 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’)
```



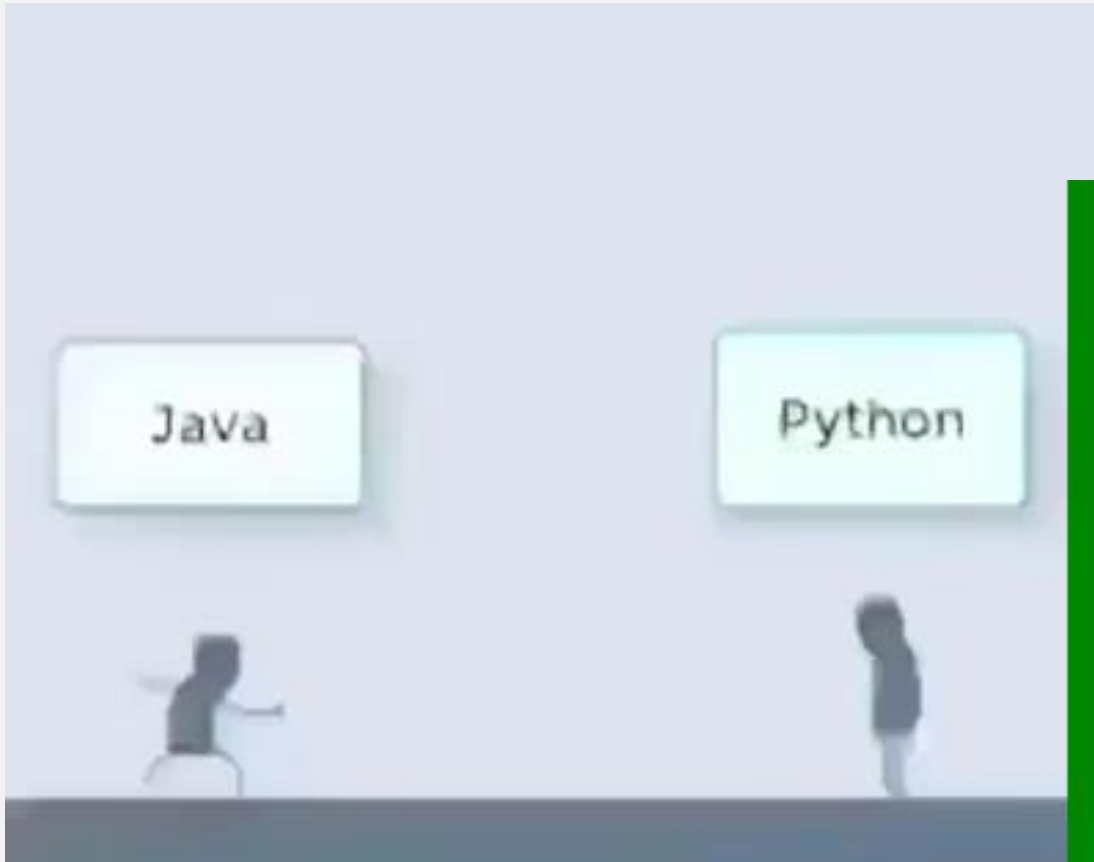
Characteristics:

- 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.

3.1 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.

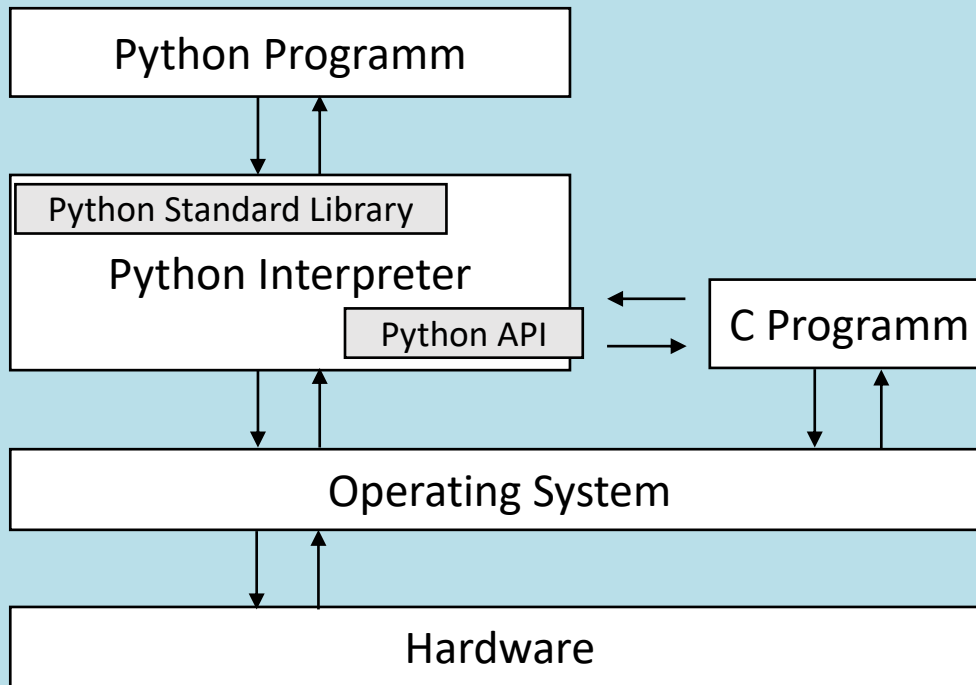
3.1 Python 2 vs. Python 3

- „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

3.1 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 that the program can rely on
- Python can be extended easily by the Python API

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

3.1 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

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? ")
```



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.

3.1 Install Anaconda



Products ▼

Pricing

Solutions ▼

Resources ▼

Blog

Company ▼

Get Started

Data science technology for a better world.

A movement that brings together millions of data science practitioners,
data-driven enterprises, and the open source community.

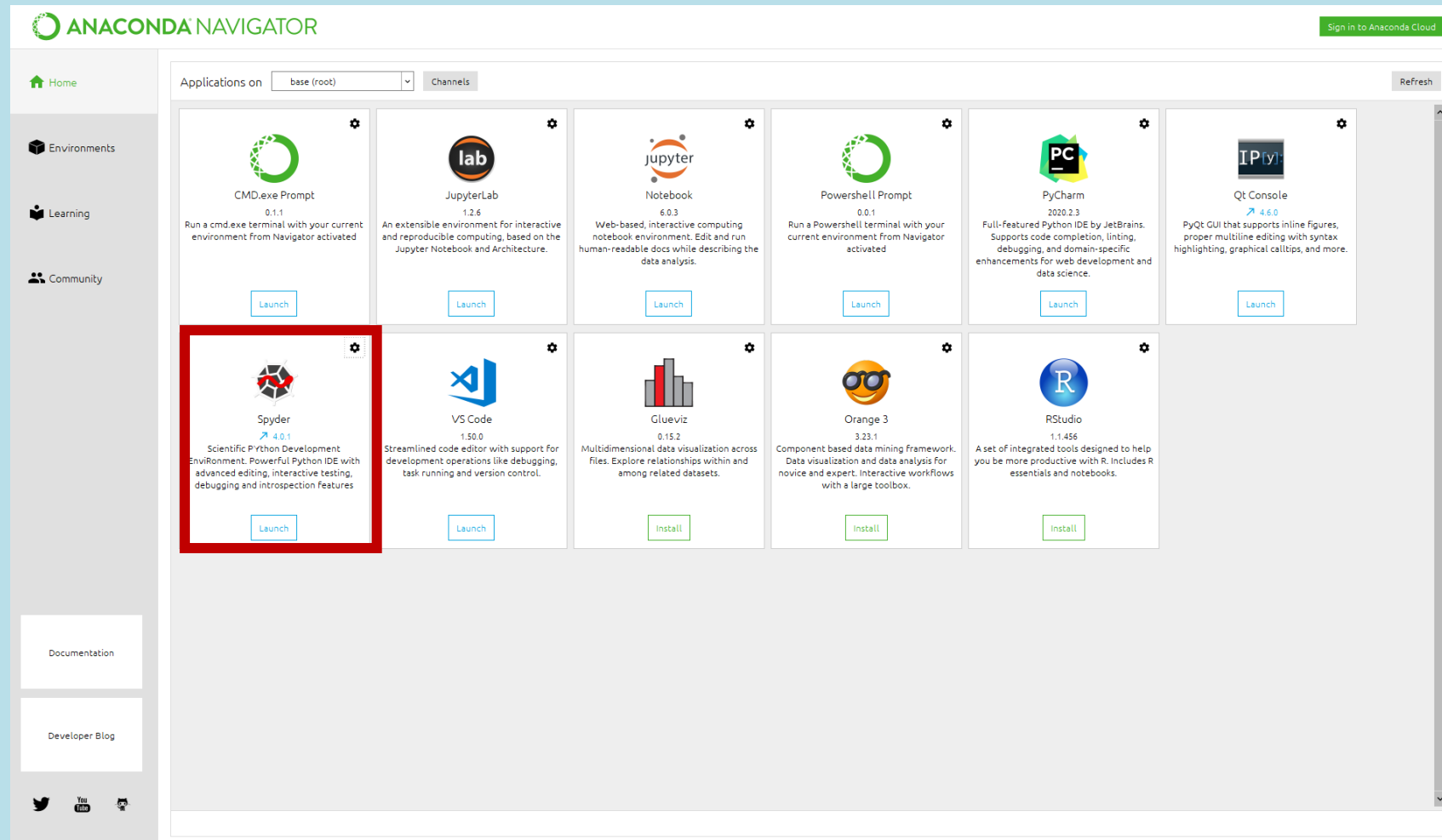
Get Started



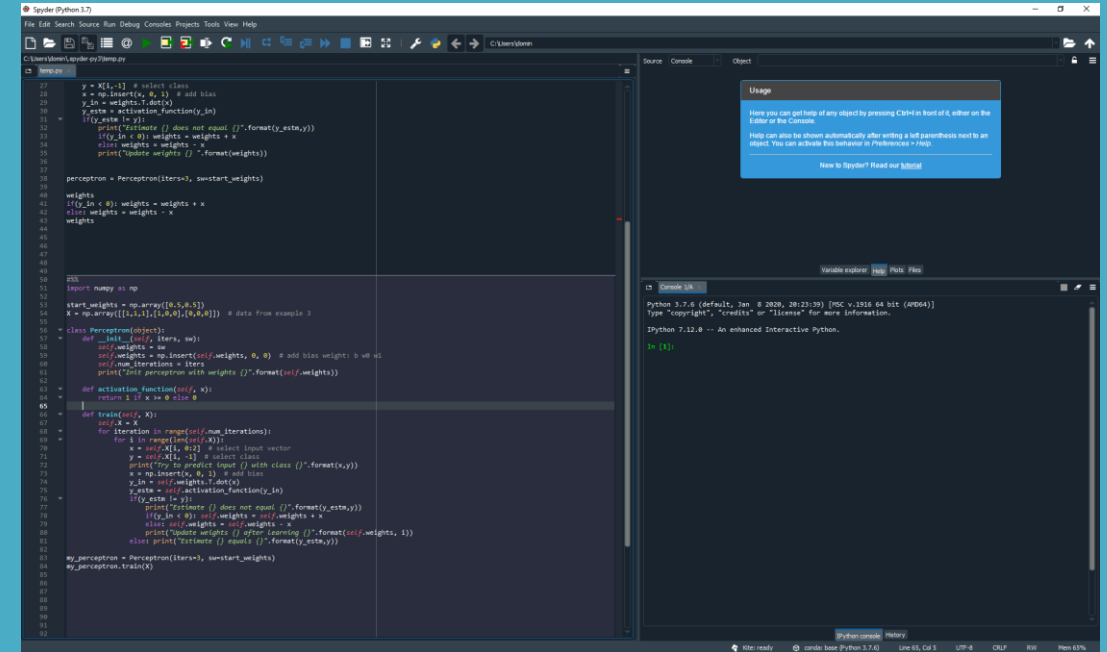
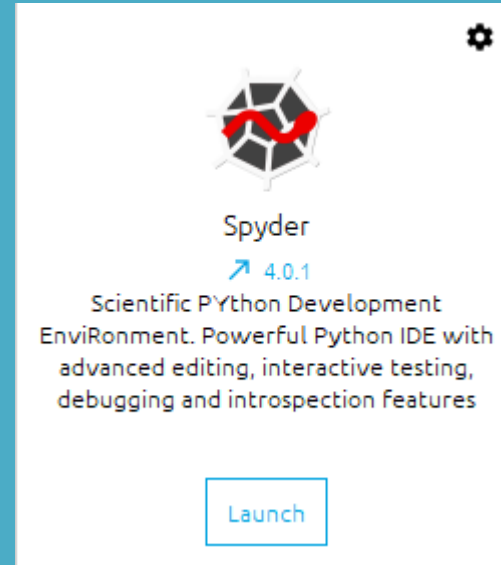
Anaconda Platform:

<https://www.anaconda.com/distribution>

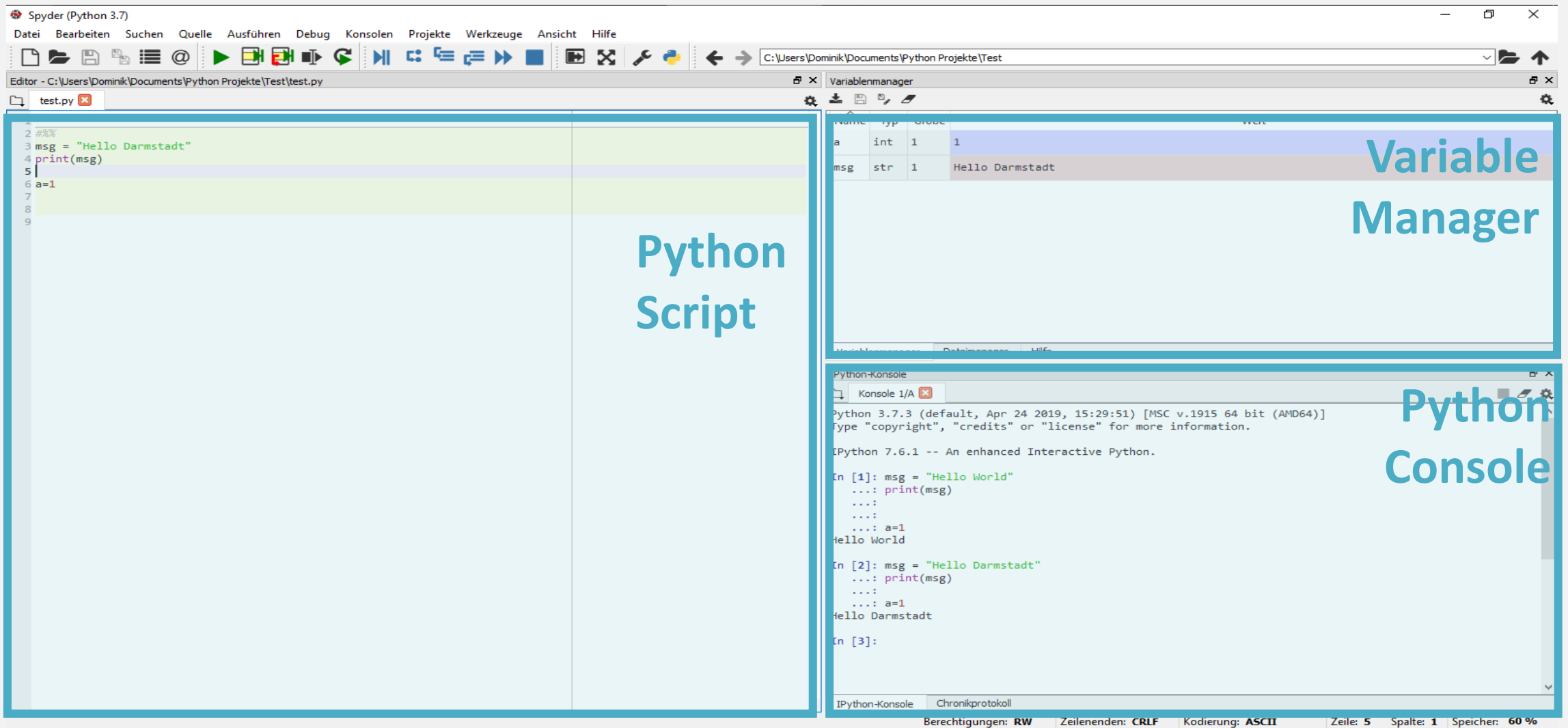
3.1 Lets Take a Quick Look at Programming with the Spyder IDE in Anaconda



Now, please start the Spyder IDE from your Anaconda!

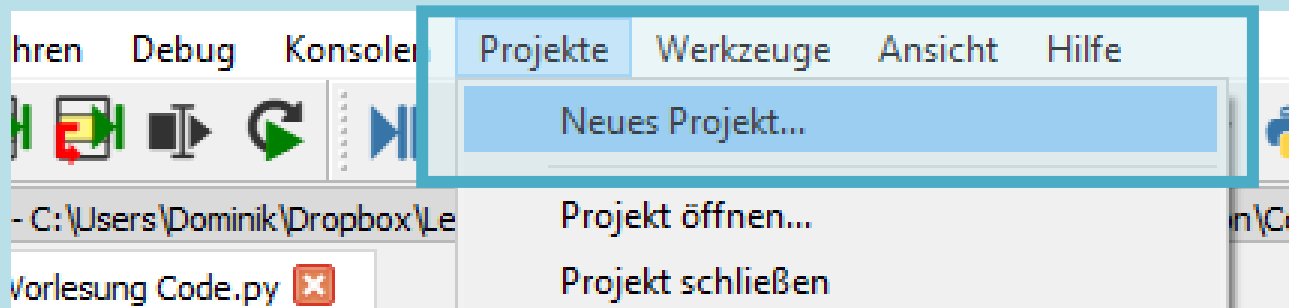


3.1 Spyder IDE

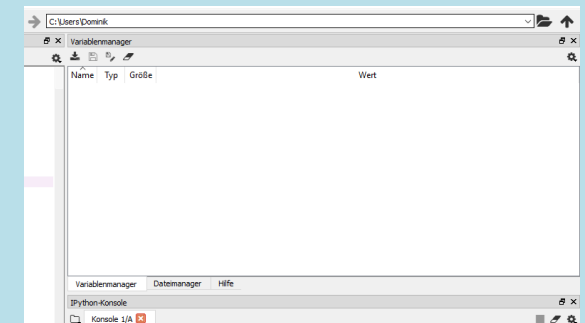
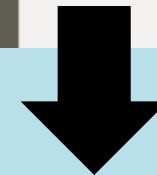
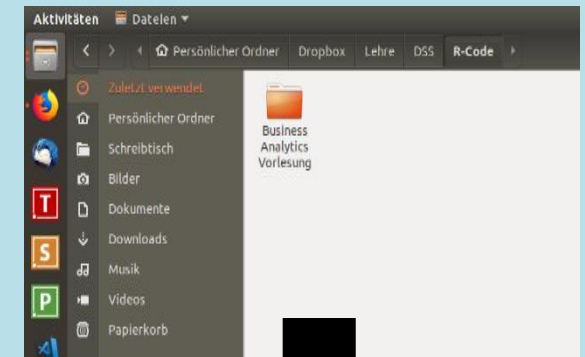


3.1 Projects with Spyder

- 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

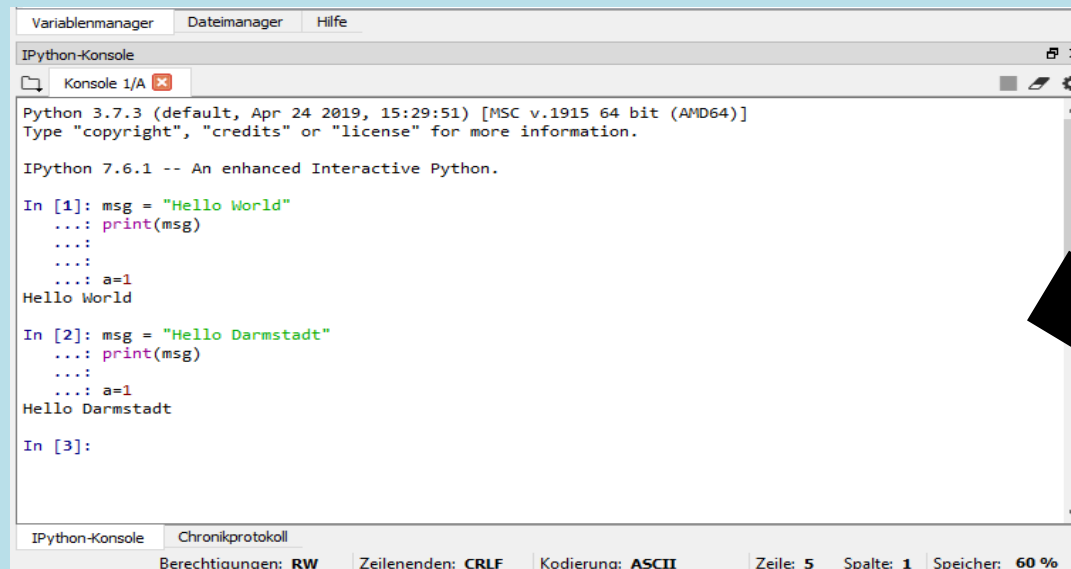


Data Management



3.1 First Steps in the Interactive Mode

- To run Python commands you can use the console from the Spyder IDE or open the Anaconda-Shell and type „Python“



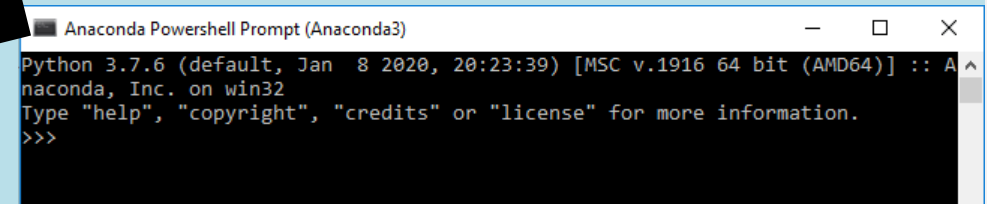
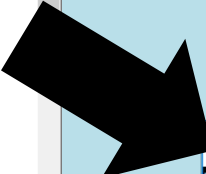
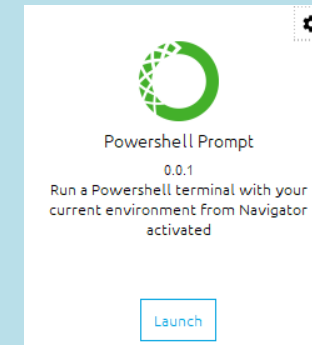
```
Python 3.7.3 (default, Apr 24 2019, 15:29:51) [MSC v.1915 64 bit (AMD64)]
Type "copyright", "credits" or "license" for more information.

IPython 7.6.1 -- An enhanced Interactive Python.

In [1]: msg = "Hello World"
...: print(msg)
...:
...:
...: a=1
Hello World

In [2]: msg = "Hello Darmstadt"
...: print(msg)
...:
...:
...: a=1
Hello Darmstadt

In [3]:
```

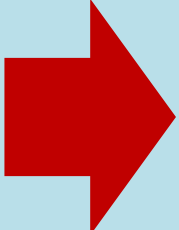


```
Python 3.7.6 (default, Jan 8 2020, 20:23:39) [MSC v.1916 64 bit (AMD64)] :: A
naconda, Inc. on win32
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

3.1 First Steps in the Interactive Mode

- Let us start by printing the following command in the Python interpreter (in the Spyder IDE or in the Python prompt directly)

42



```
Anaconda Powershell Prompt (Anaconda3)
(base) PS C:\Users\domin> Python
Python 3.7.6 (default, Jan  8 2020, 20:23:39) [MSC v.1916 64 bit (AMD64)] :: A
naconda, Inc. on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> 42
42
>>> -1000
-1000
>>>
```



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

3.1 Arithmetic Operations in Python

```
Anaconda Po...  —  □  ✕  
>>>  
>>> 1 + 2  
3  
>>> 3 - 100  
-97  
>>> 2/3  
0.6666666666666666  
>>>
```

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	$a ** b$

3.1 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
```

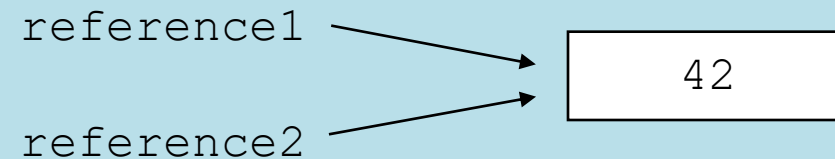
3.1 Python Runtime Model II

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

```
your_variable = 42
```

- Then you can call the variable

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



3.1 Python Runtime Model - Example

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

>>> reference1
42

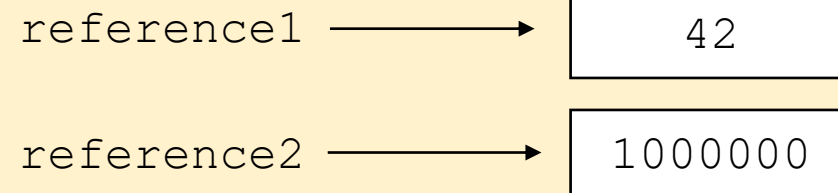
>>> reference2
42

>>> reference1 = 1000000
>>> reference1
1000000

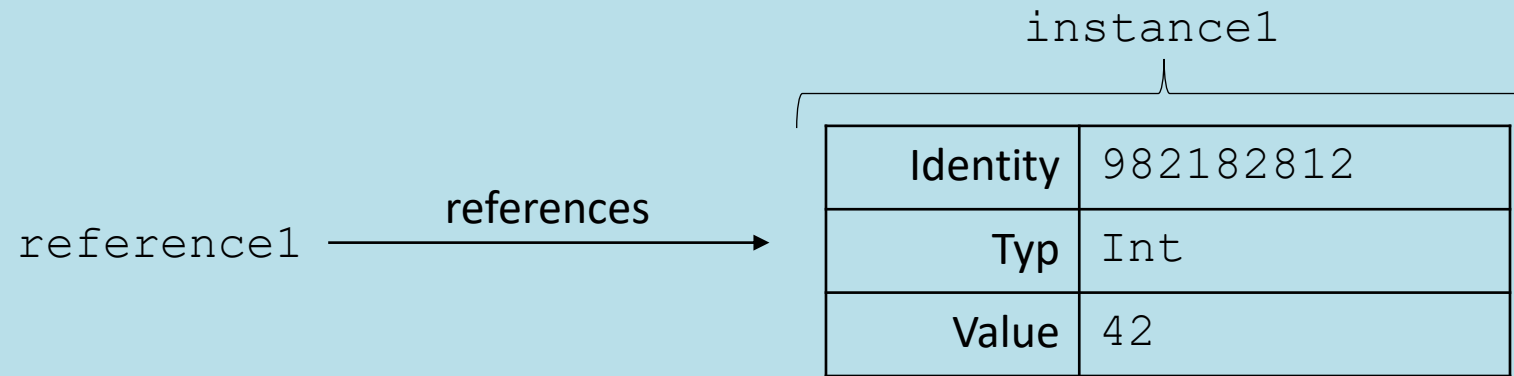
>>> reference2
42
```



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



3.1 Structure of Instances in Python

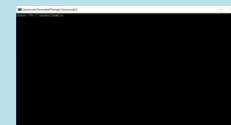
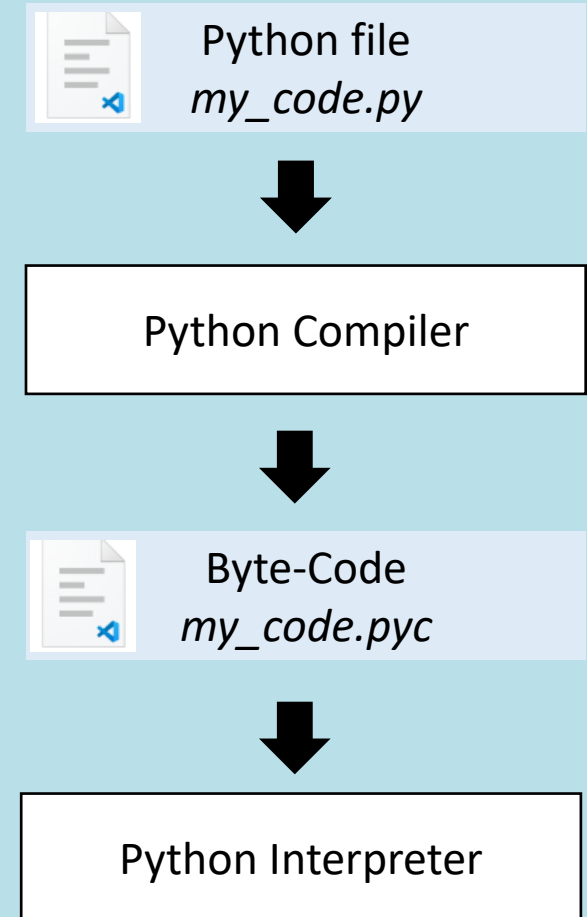


- Every object has an identity, a type and a value
- An object's *identity* never changes once it has been created; you may think of it as the object's address in memory

3.1 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'
```



3.1 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.

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`. Then 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.

3.1 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!
```

3.1 Classroom Task

The image shows a screenshot of a Python IDE with a dark theme. The top menu bar includes File, Edit, Search, Source, Run, Debug, Consoles, Projects, Tools, View, and Help. Below the menu is a toolbar with various icons. A blue arrow points to the 'Run' icon (a green play button), which is labeled with a large blue circle containing the number '1' and the text 'double click!'. The main editor window displays a Python script named 'Lecture 3 - Methusalem.py'. The script contains a docstring and a simple calculation. The console window at the bottom right shows the output of the script, with a blue arrow pointing to it from the 'Run' icon. The console output includes the prompt 'In [4]:', the command 'runfile(...)', the input '31', and the output 'You are 938 year younger than the oldest man in the world!'. The console also shows 'In [5]:' at the bottom.

```
1  # -*- coding: utf-8 -*-
2
3  """
4  Lecture 3 - Introduction into AI Programming with Python
5  @author: Dominik Jung (dominik.jung42@gmail.com)
6  """
7
8  methusalem = 969
9  your_age = input()
10 print("You are " + str(methusalem - int(your_age)) + " year younger than the oldest man in the world!")
11
```

1 double click!

**2 enter your age in the console!
Press [ENTER] to confirm**

In [4]: runfile('C:/Users/domin/Dropbox/Lehre/AI Algorithms and Methusalem.py', wdir='C:/Users/domin/Dropbox/Lehre/AI Algorithms and Methusalem.py')

31
You are 938 year younger than the oldest man in the world!

In [5]:

3 Introduction into AI-Programming with Python

3.1 The Anaconda Toolbox for AI Programming

3.2 Foundations of Programming with Python

3.3 AI Programming with 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

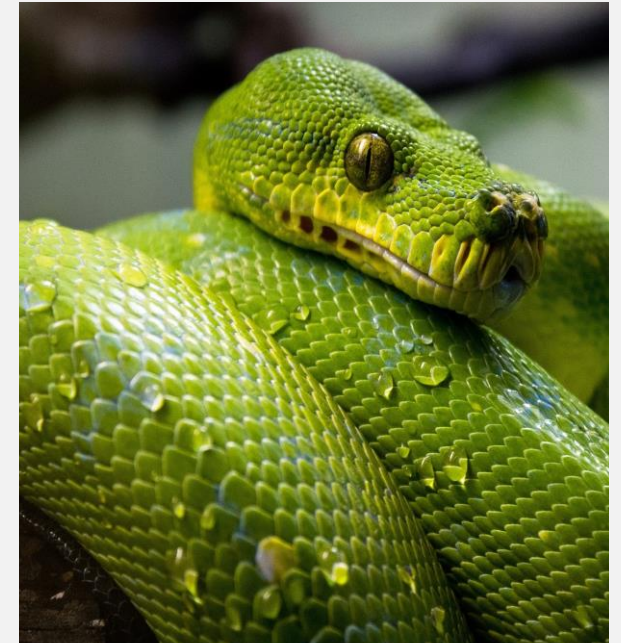


Image source: ↗ [Pixabay](#) (2019) / ↗ [CCO](#)

► Duration:

- 90 min

► Relevant for Exam:

- 3.1 - 3.3

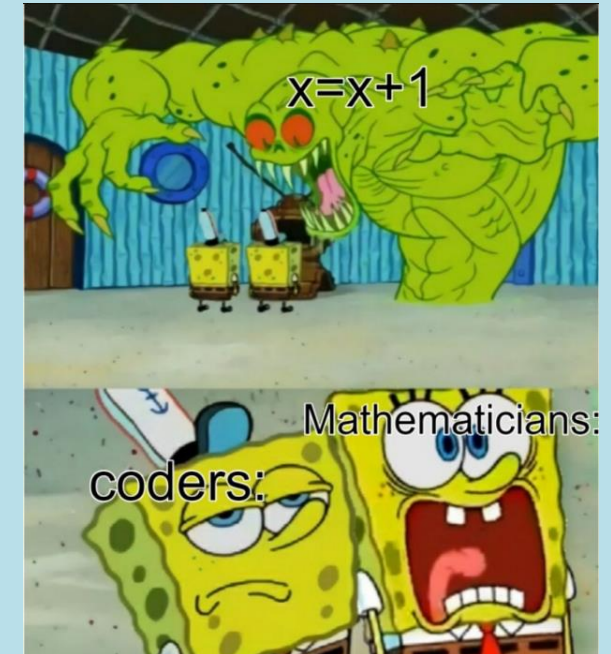
3.2 Recap: Variables

- In Python you can define variables

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

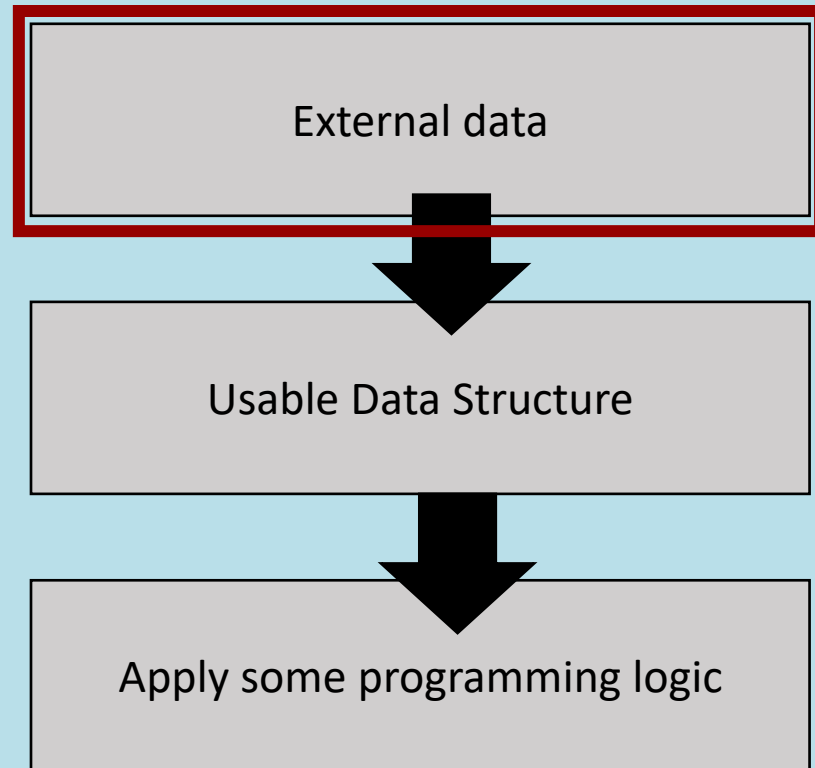
- You can access variables later by „calling“ them

```
>>> 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)

3.2 Typical AI Programming Workflow (High-Level)



Example: Guessing Game

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

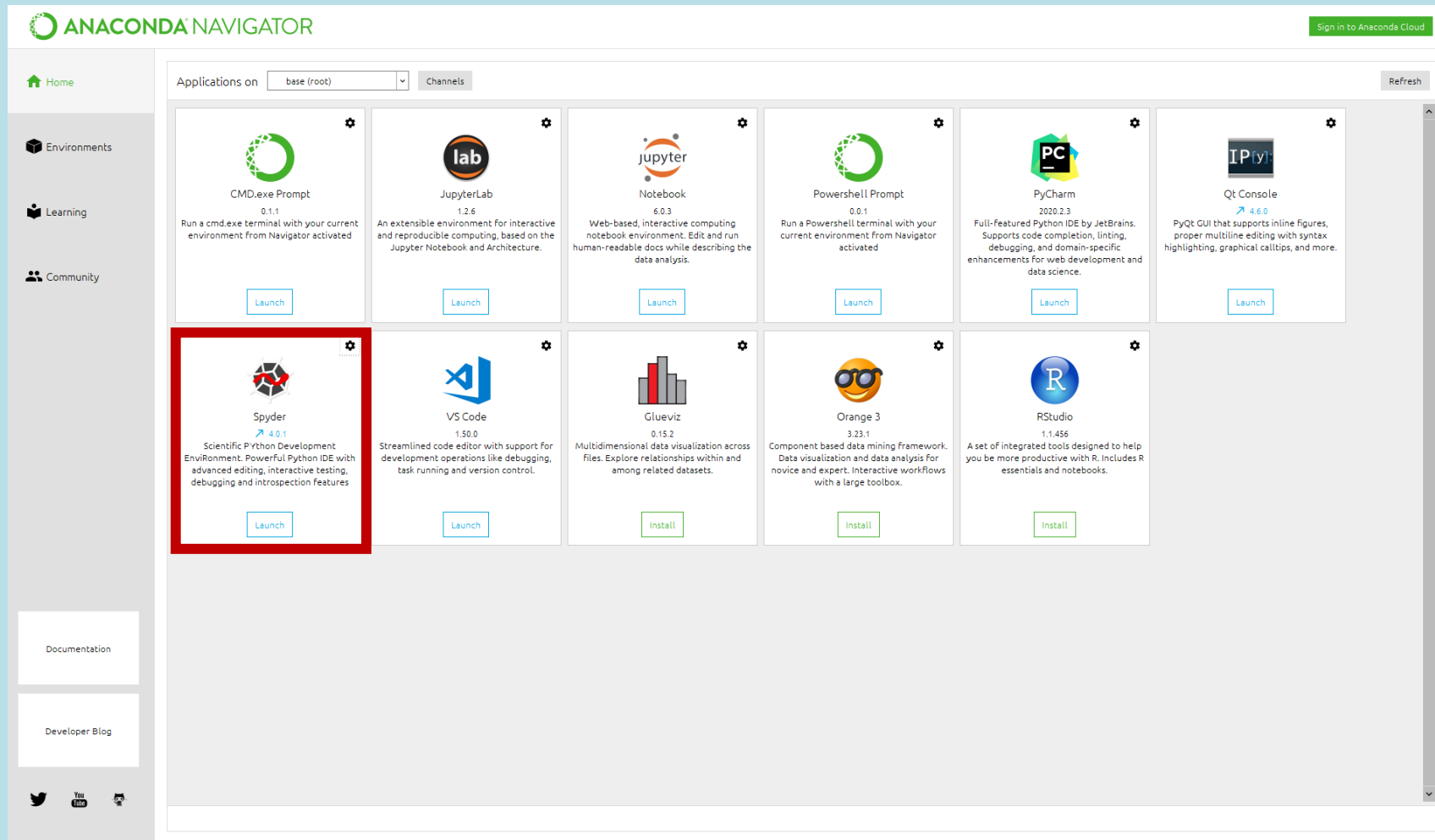
Please enter a Porsche model: 911
The car has the following VMax: 330 km/h

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)

3.2 Start Spyder from Anaconda



3.2 Reading Files

- If you want to read a file in your Python code, it must be open 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

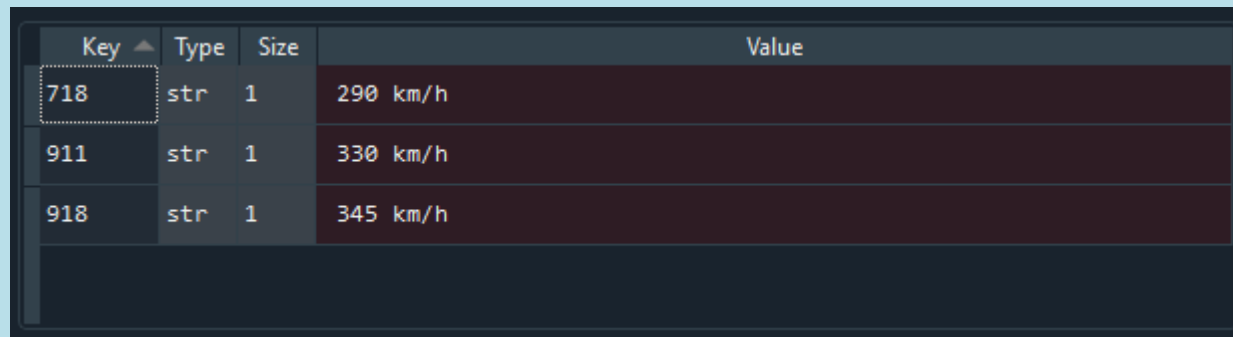


Adapted from Ernesti, J. & Kaiser, P. (2017); Image Source: ↗ [Porsche 911 R im Porsche Museum 2018](#) (2018) by [Alexander Migl](#) from ↗ Wikimedia / ↗ [CC-BY-SA-4.0](#) (image not edited)

3.2 Reading Files into Variables I

- Let us now 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()
```



A screenshot of a Jupyter Notebook interface showing a table representation of a dictionary. The table has four columns: 'Key', 'Type', 'Size', and 'Value'. The 'Key' column contains car models: '718', '911', and '918'. The 'Type' column shows 'str' for all entries. The 'Size' column shows '1' for all entries. The 'Value' column shows the corresponding speed values: '290 km/h', '330 km/h', and '345 km/h'.

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); Official Python Documentation (2019): <https://docs.python.org>

3.2 Reading Files into Variables II

- Let us now 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()
```

```
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

3.2 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()
```

3.2 Read Files Online

- Online files can be accessed directly from Python

```
# Read from online
import urllib3

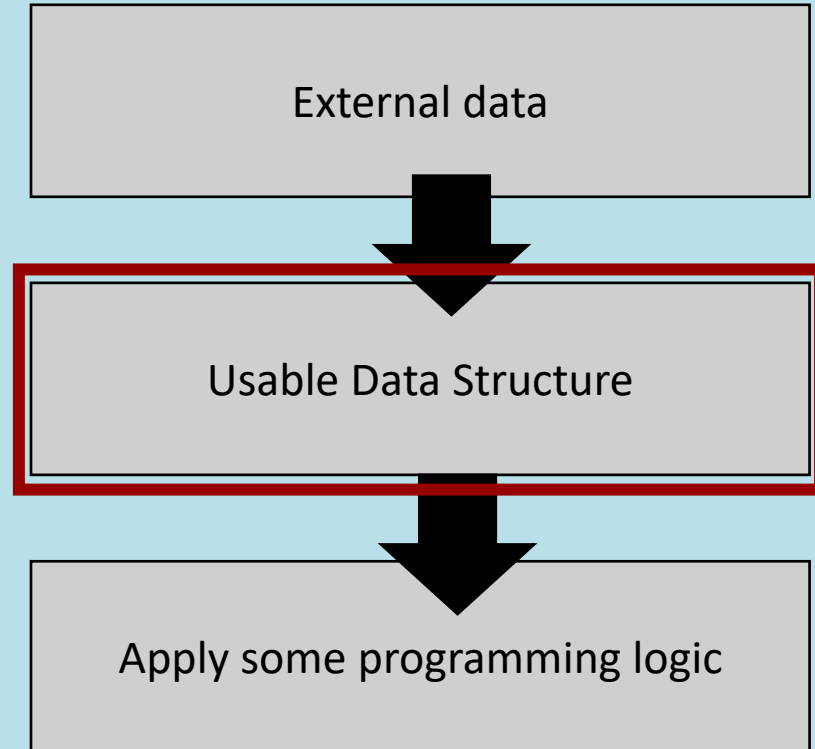
http = urllib3.PoolManager()

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

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

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

3.2 Typical AI Programming Workflow (High-Level)



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)

3.2 Data Types and Structures in Python

D

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
```

- And 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 (aka logical) operations

```
>>> 42 < 43
```

```
True
```

```
>>> 42 != 43
```

```
True
```

```
>>> is_cool = True
```

```
>>> is_cool
```

```
True
```

Comparison Operators

Operator	Command
NOT	! x
AND (elementwise)	x & y
AND (1 st element)	x && y
OR (elementwise)	x y
OR (1 st 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 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 Investigate Complex Data Structures with the Variable Manager

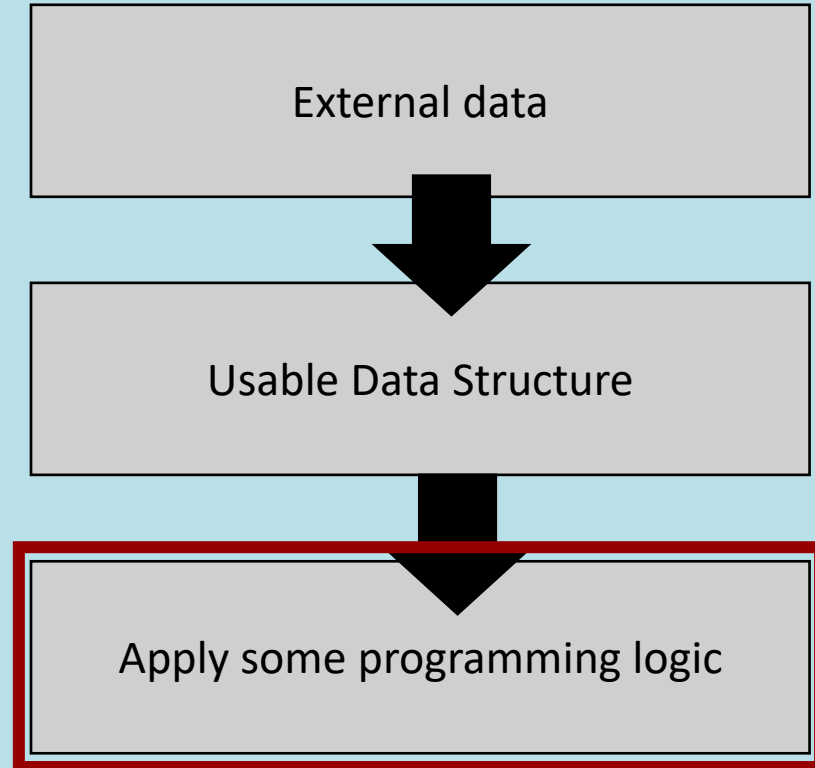
2 Investigate!

1 double click!

Schlüssel	Typ	Größe	Wert
Arya	list	2	['missing', '1']
Bran	list	2	['alive', '0']
Eddard	list	2	['dead ', '1']
John	list	2	['alive', '0.5']
Sansa	list	2	['alive ', '1']

Speichern und Schließen Schließen

3.2 Typical AI Programming Workflow (High-Level)



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)

3.2 Functions

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

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

Diagram illustrating the components of the function definition $f(x) = x + 2x$:

- f : name
- x : parameters
- $= x + 2x$: 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.

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 redundancy

```
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
```

3.2 yield and return

- Python provides generator functions as convenient way to build iterators:

```
def firstn(n):  
    num = 0  
    while num < n:  
        yield num  
        num += 1
```

```
top5 = firstn(5)
```

- The previous generator yields its items instead of returning a list

```
>>>next(top5)  
0  
>>>next(top5)  
1
```

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")
```

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!")
```

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?

3.2 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'
```

3.2 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!'
```

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

3 Introduction into AI-Programming with Python

3.1 The Anaconda Toolbox for AI Programming

3.2 Foundations of Programming with Python

3.3 AI Programming with 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

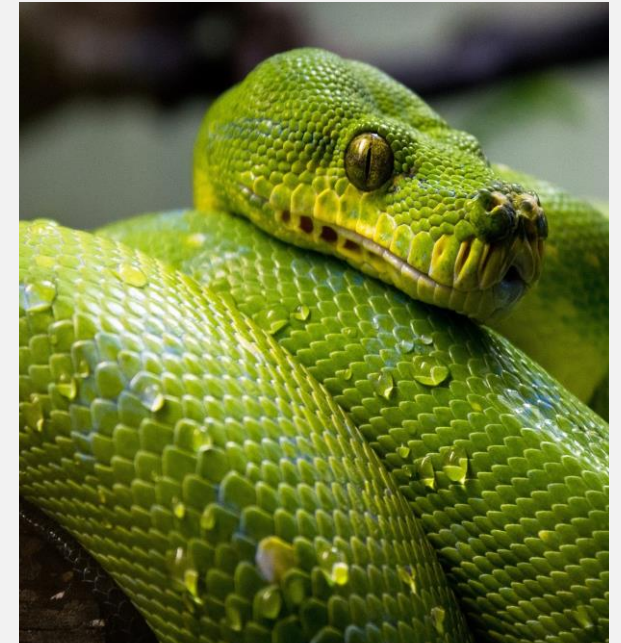


Image source: ↗ [Pixabay](#) (2019) / ↗ [CCO](#)

► Duration:

- 90 min

► Relevant for Exam:

- 3.1 - 3.3

3.3 Modules

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

```
>>> max([1, 5, 2, 7, 42, 3]) ← build-in functions  
42
```

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

3.3 Modules Import

- Import modules

```
import module  
module.spec_func()
```

- Import specific function from modul

```
from module import spec_func  
spec_func()
```

- Import modules with specific namespaces

```
import module as mo  
mo.spec_func()
```

- Import every function from modul

```
from module import *  
spec_func()
```

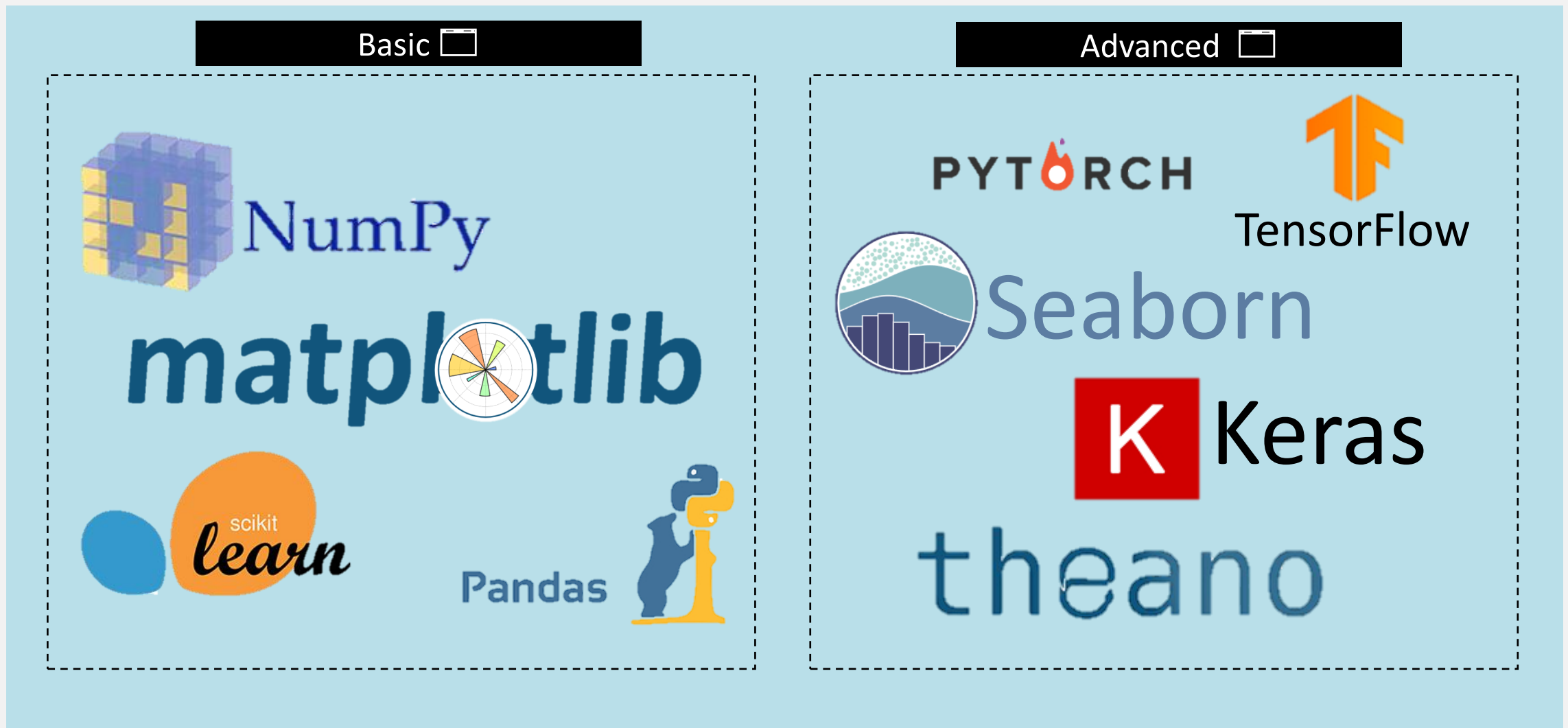


Problem: some modul names are very long and you import functions you probably do not need



Do this only if you know what you are doing!

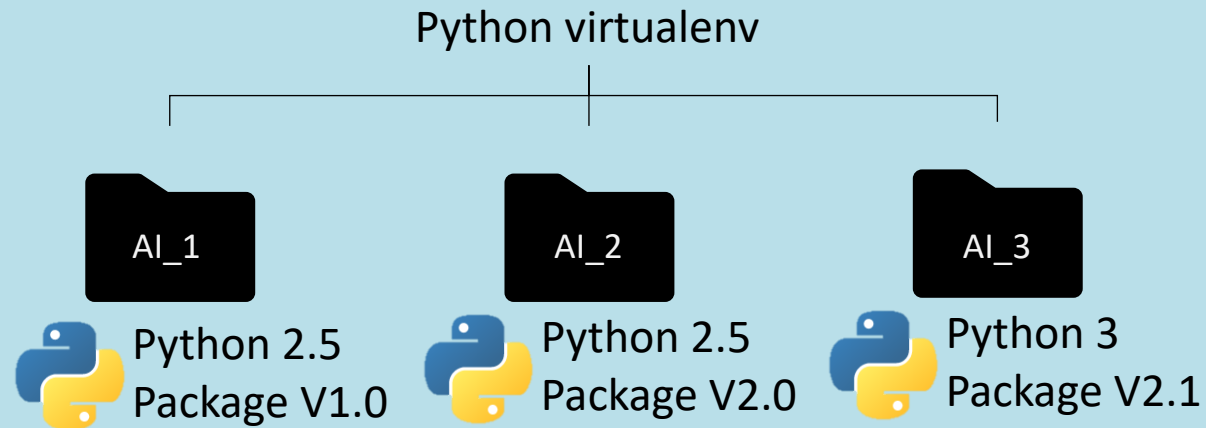
3.3 Most relevant Packages for AI-Specialists



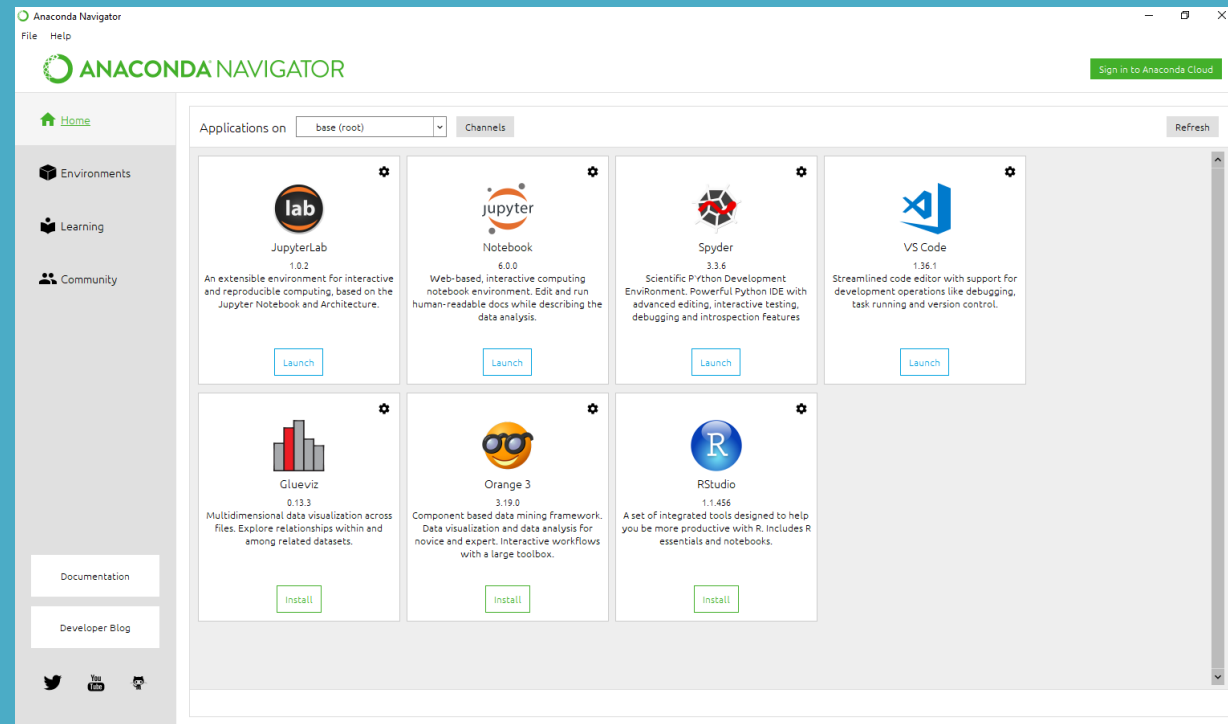
Images: From the respective websites

3.3 Management of Different Packages and Versions?

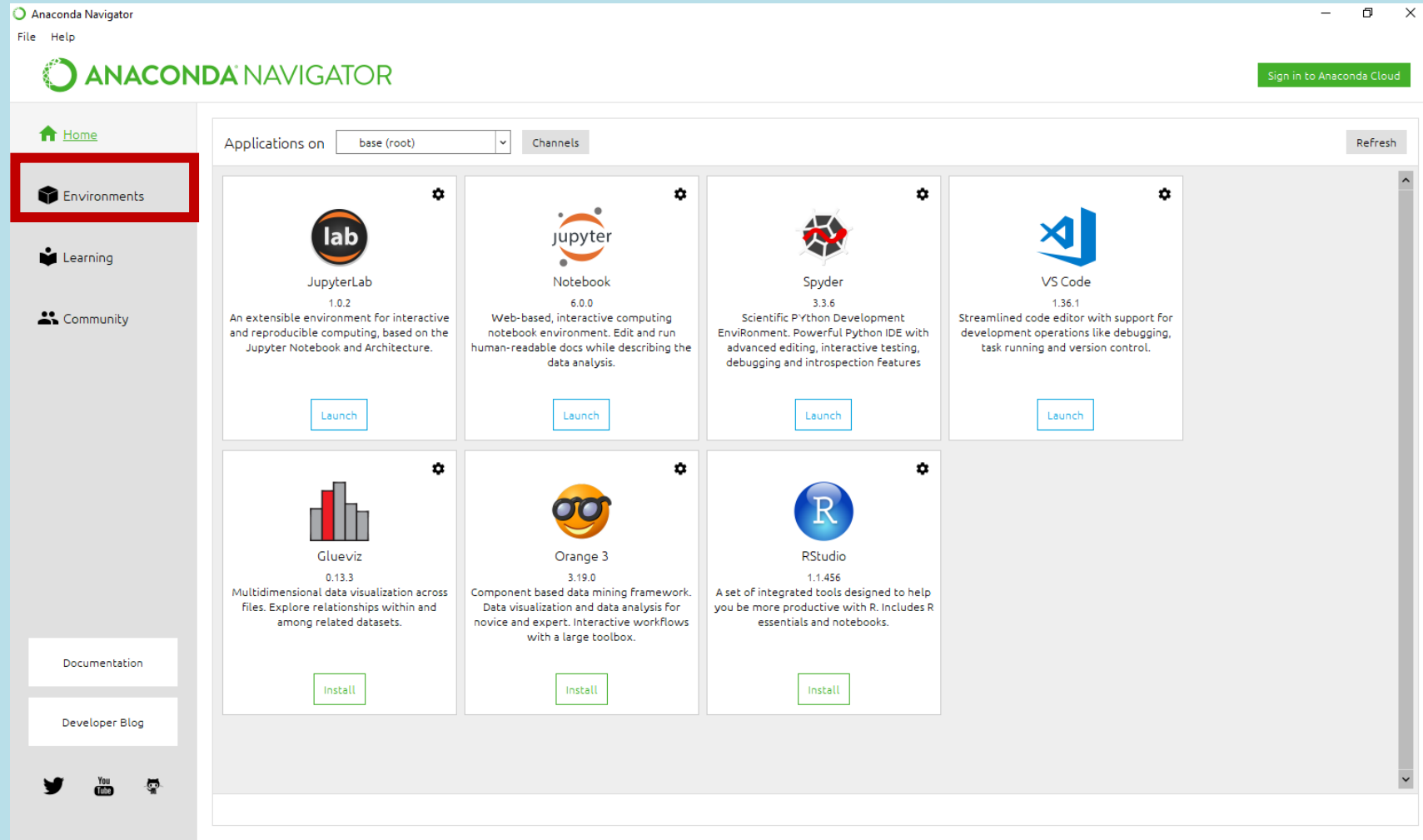
- Python virtual environments allow to create an isolated environment for Python projects so that each of your projects has its own individual dependencies, regardless of what dependencies every other project has.



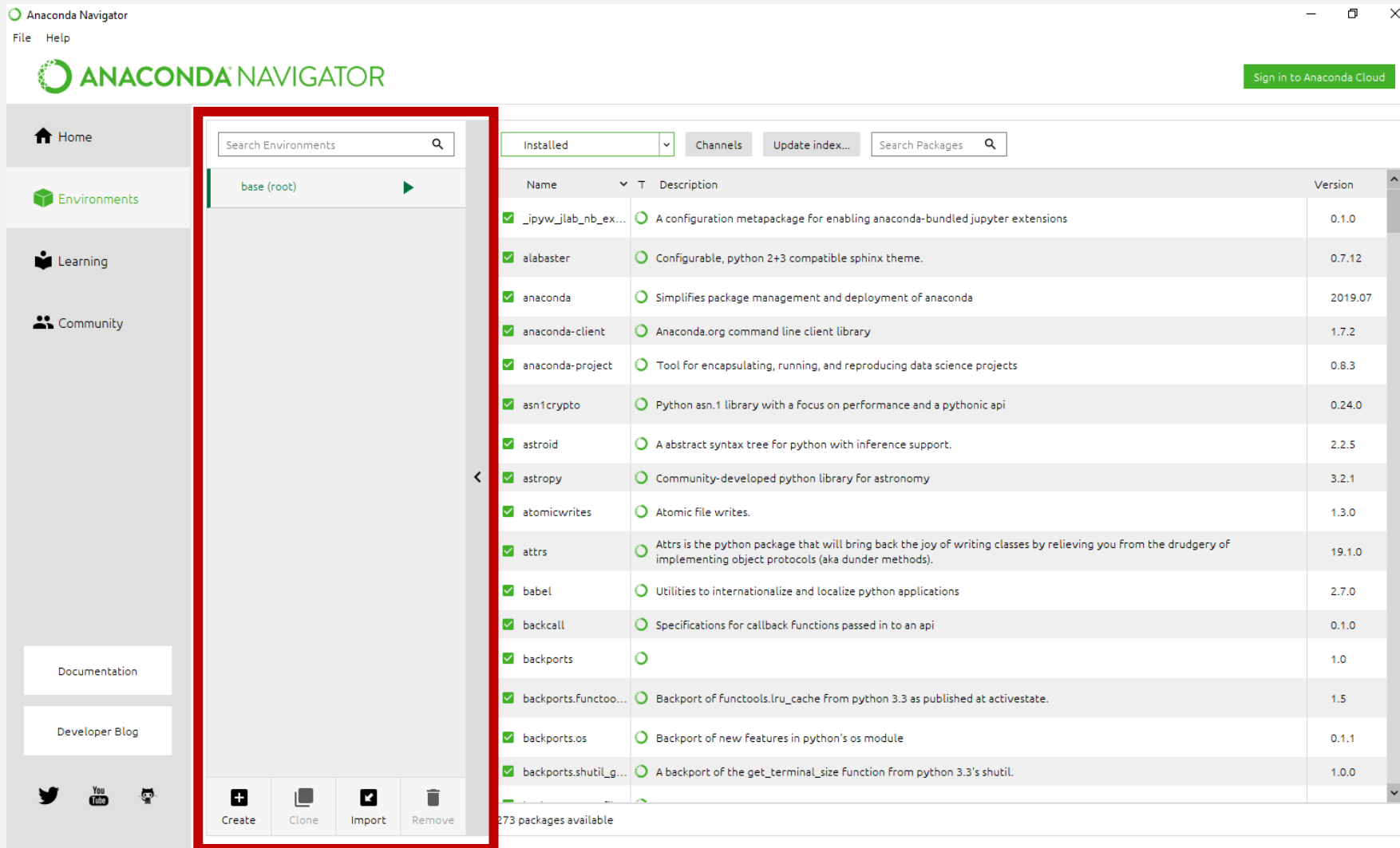
Now, please start Anaconda!



3.3 Package Management in Anaconda



3.3 Install and Load New Packages



The screenshot displays the Anaconda Navigator application window. The left sidebar contains navigation links for Home, Environments, Learning, and Community. The main area is divided into two panes. The left pane, titled 'base (root)', is highlighted with a red border and contains a search bar and a list of environments. The right pane shows a table of installed packages. The table has columns for Name, Description, and Version. The packages listed include _ipyw_jlab_nb_ex..., alabaster, anaconda, anaconda-client, anaconda-project, asn1crypto, astroid, astropy, atomicwrites, attrs, babel, backcall, backports, backports.functoo..., backports.os, and backports.shutil_g... The bottom of the right pane indicates that 273 packages are available.

Name	Description	Version
✓ _ipyw_jlab_nb_ex...	A configuration metapackage for enabling anaconda-bundled jupyter extensions	0.1.0
✓ alabaster	Configurable, python 2+3 compatible sphinx theme.	0.7.12
✓ anaconda	Simplifies package management and deployment of anaconda	2019.07
✓ anaconda-client	Anaconda.org command line client library	1.7.2
✓ anaconda-project	Tool for encapsulating, running, and reproducing data science projects	0.8.3
✓ asn1crypto	Python asn.1 library with a focus on performance and a pythonic api	0.24.0
✓ astroid	A abstract syntax tree for python with inference support.	2.2.5
✓ astropy	Community-developed python library for astronomy	3.2.1
✓ atomicwrites	Atomic file writes.	1.3.0
✓ attrs	Attrs is the python package that will bring back the joy of writing classes by relieving you from the drudgery of implementing object protocols (aka dunder methods).	19.1.0
✓ babel	Utilities to internationalize and localize python applications	2.7.0
✓ backcall	Specifications for callback Functions passed in to an api	0.1.0
✓ backports		1.0
✓ backports.functoo...	Backport of functools.lru_cache from python 3.3 as published at activestate.	1.5
✓ backports.os	Backport of new Features in python's os module	0.1.1
✓ backports.shutil_g...	A backport of the get_terminal_size function from python 3.3's shutil.	1.0.0

- You can manage and organize your environments with the Anaconda Navigator

3.3 Install Required Packages for this Course

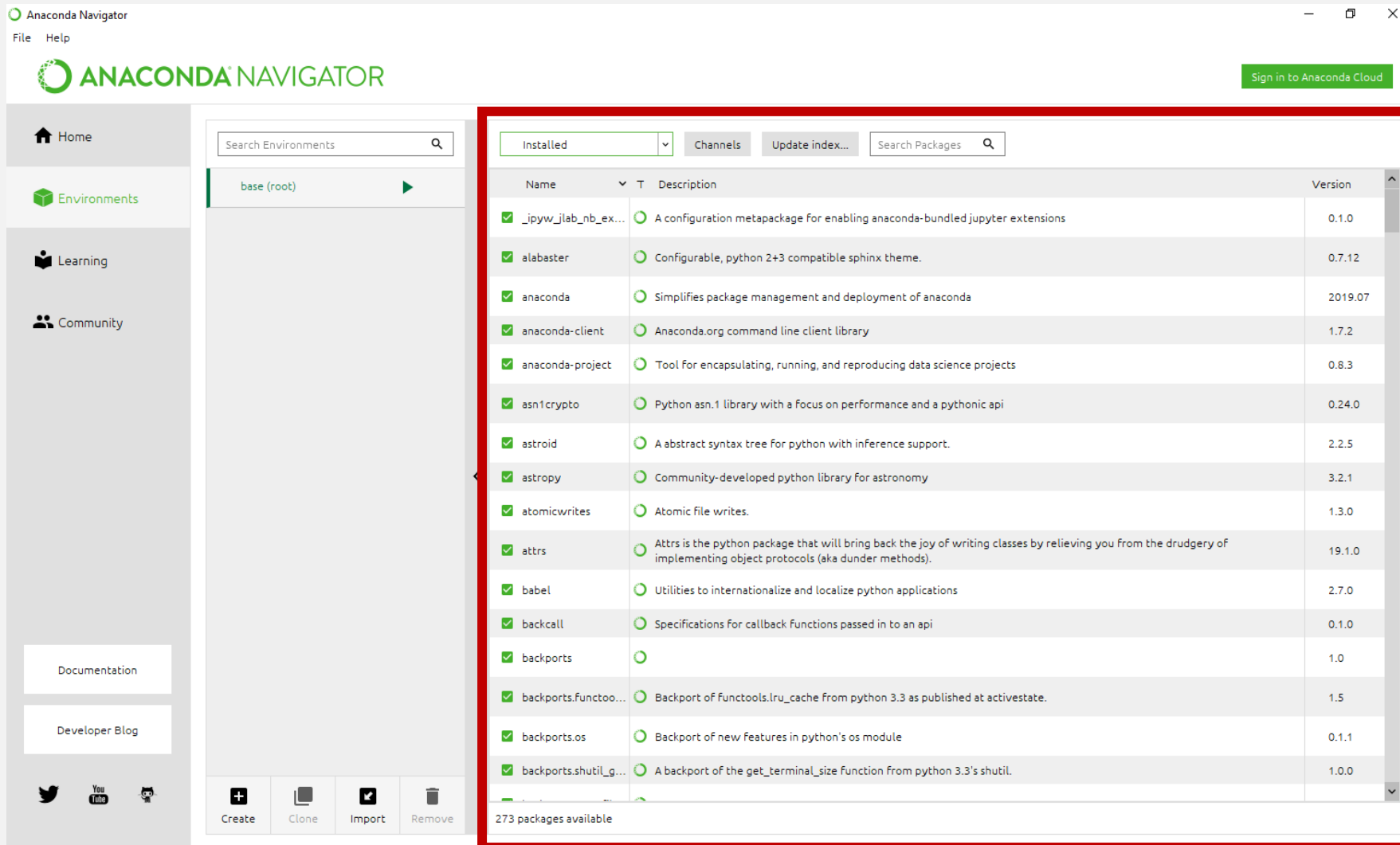
- Please activate your environment (if you have one)

```
>>> conda activate <environmentname>
```

- And then read in the requirements.txt to automatically install all required packages for this course

```
>>> pip install -r packages.txt
```

3.3 Install and Load New Packages



The screenshot shows the Anaconda Navigator application window. The left sidebar contains navigation links for Home, Environments, Learning, and Community. The main panel displays the 'base (root)' environment. A red box highlights the 'Installed' tab, which shows a list of installed packages. The table below lists the packages and their versions.

Name	Description	Version
✓ _ipyw_jlab_nb_ex...	A configuration metapackage for enabling anaconda-bundled jupyter extensions	0.1.0
✓ alabaster	Configurable, python 2+3 compatible sphinx theme.	0.7.12
✓ anaconda	Simplifies package management and deployment of anaconda	2019.07
✓ anaconda-client	Anaconda.org command line client library	1.7.2
✓ anaconda-project	Tool for encapsulating, running, and reproducing data science projects	0.8.3
✓ asn1crypto	Python asn.1 library with a focus on performance and a pythonic api	0.24.0
✓ astroid	A abstract syntax tree for python with inference support.	2.2.5
✓ astropy	Community-developed python library for astronomy	3.2.1
✓ atomicwrites	Atomic file writes.	1.3.0
✓ attrs	Attrs is the python package that will bring back the joy of writing classes by relieving you from the drudgery of implementing object protocols (aka dunder methods).	19.1.0
✓ babel	Utilities to internationalize and localize python applications	2.7.0
✓ backcall	Specifications for callback Functions passed in to an api	0.1.0
✓ backports		1.0
✓ backports.functoo...	Backport of functools.lru_cache from python 3.3 as published at activestate.	1.5
✓ backports.os	Backport of new Features in python's os module	0.1.1
✓ backports.shutil_g...	A backport of the get_terminal_size function from python 3.3's shutil.	1.0.0

273 packages available

- Use it to install and load new packages

```
>>> import numpy
```

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)!

3.3 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.

3.3 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])
```

Command

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



NumPy Array

1
2
3

3.3 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

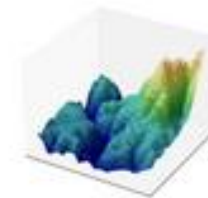
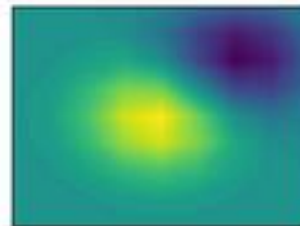
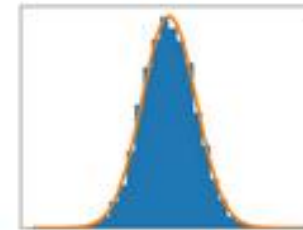
```
data[-2:]
```

0	1	-2
1	2	-1
2	3	
3		

```
>>> import matplotlib
```

Matplotlib: Visualization with Python

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



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

Matplotlib makes easy things easy and hard things possible.

3.3 About Matplotlib

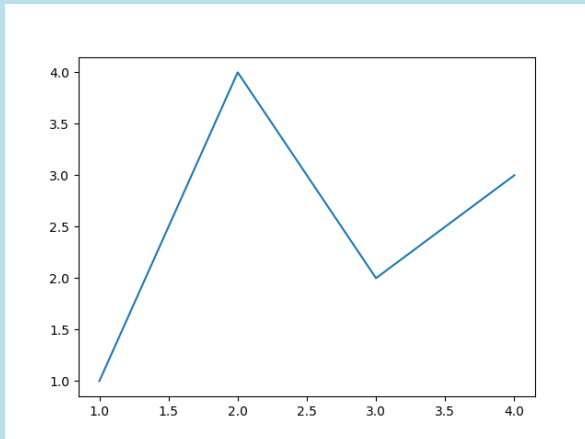
- Matplotlib is a visualization library for Python, and in particular its numerical mathematics extension `Numpy`
- It provides an object-oriented API for embedding plots into applications using general-purpose GUI toolkits like Tkinter, wxPython, Qt, or GTK+.

3.3 Generating Plots

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

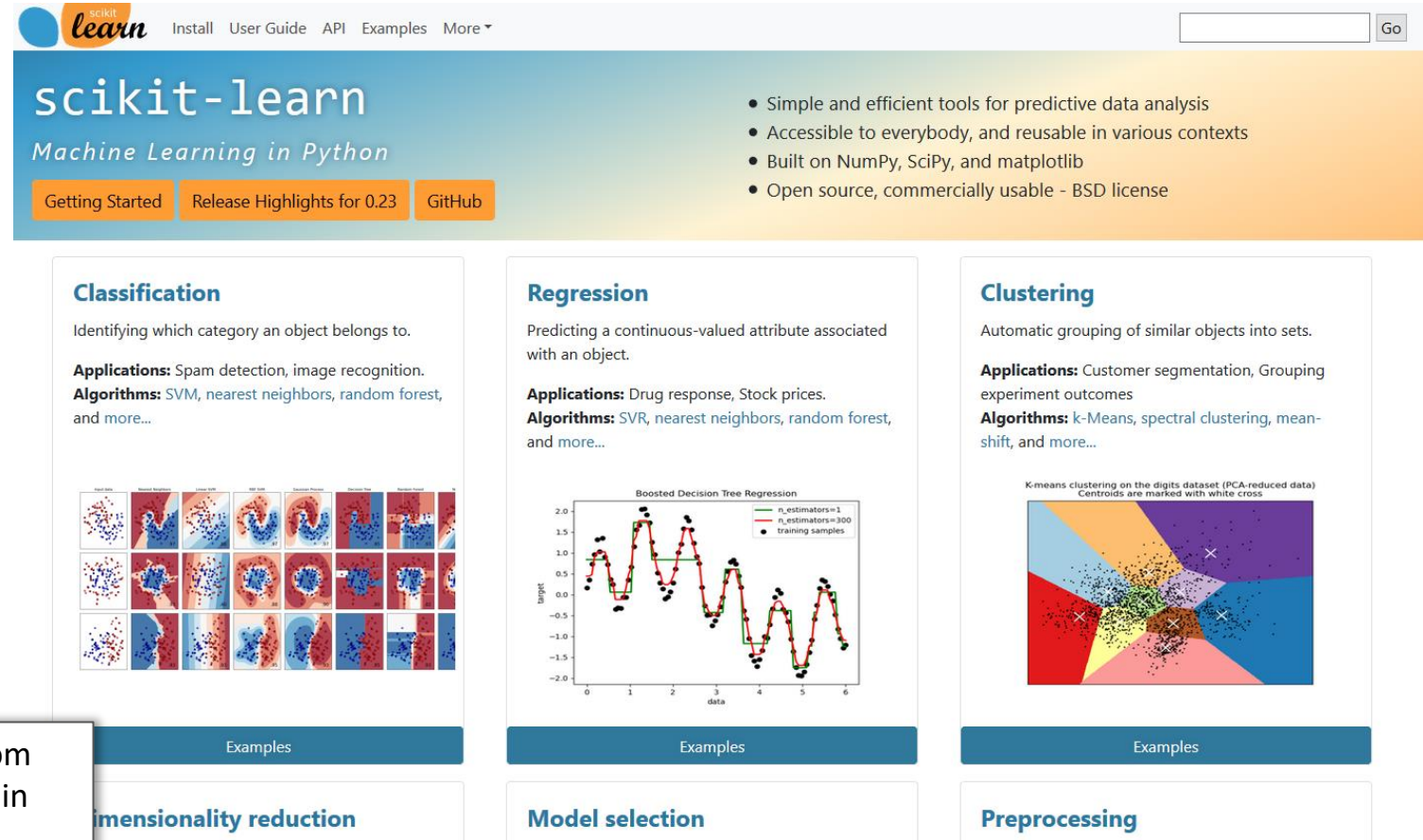
```
import matplotlib.pyplot as plt  
import numpy as np
```

```
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

```
>>> import sklearn
```



The screenshot shows the Scikit-learn website. At the top, there's a navigation bar with links: Install, User Guide, API, Examples, and More. The main header features the 'scikit-learn' logo and the tagline 'Machine Learning in Python'. Below this, there are three buttons: 'Getting Started', 'Release Highlights for 0.23', and 'GitHub'. To the right, a list of features is displayed: 'Simple and efficient tools for predictive data analysis', 'Accessible to everybody, and reusable in various contexts', 'Built on NumPy, SciPy, and matplotlib', and 'Open source, commercially usable - BSD license'.

The main content area is divided into three columns, each representing a machine learning task:

- Classification**: Identifying which category an object belongs to. Applications include Spam detection and image recognition. Algorithms include SVM, nearest neighbors, random forest, and more. Below the text is a 3x6 grid of small images showing various handwritten digits and patterns.
- Regression**: Predicting a continuous-valued attribute associated with an object. Applications include Drug response and Stock prices. Algorithms include SVR, nearest neighbors, random forest, and more. Below the text is a line plot titled 'Boosted Decision Tree Regression' showing 'target' vs 'data' with training samples and two different model fits.
- Clustering**: Automatic grouping of similar objects into sets. Applications include Customer segmentation and Grouping experiment outcomes. Algorithms include k-Means, spectral clustering, mean-shift, and more. Below the text is a scatter plot titled 'K-means clustering on the digits dataset (PCA-reduced data)' showing data points grouped into clusters with centroids marked by white crosses.

At the bottom of each column, there are buttons labeled 'Examples', 'Dimensionality reduction', 'Model selection', and 'Preprocessing'.



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

```
>>> import pandas
```



[About us](#) ▾ [Getting started](#) [Documentation](#) [Community](#) ▾ [Contribute](#)

pandas

pandas is a fast, powerful, flexible and easy to use open source data analysis and manipulation tool, built on top of the Python programming language.

[Install pandas now!](#)

Latest version: 1.1.3

- [What's new in 1.1.3](#)
- [Release date:](#)
Oct 05, 2020
- [Documentation \(web\)](#)
- [Documentation \(pdf\)](#)
- [Download source code](#)

Follow us

Follow [@pandas_dev](#)

Get the book



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

Documentation

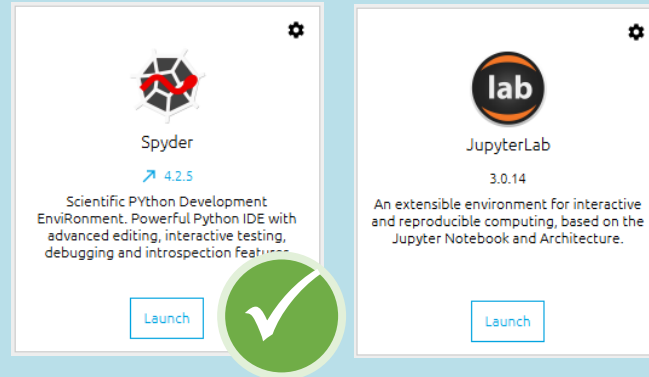
- [User guide](#)

Community

- [About pandas](#)

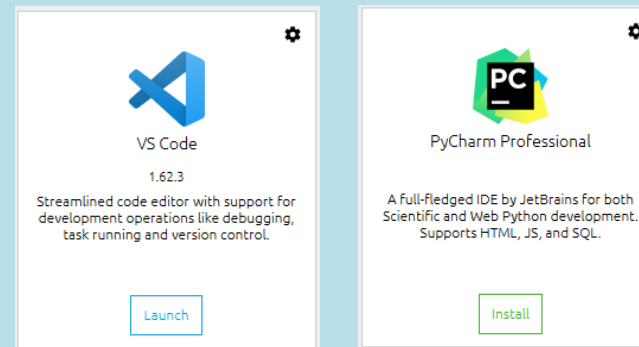
3.3 Popular Python IDEs Supported by Anaconda for AI Programming

Data Science



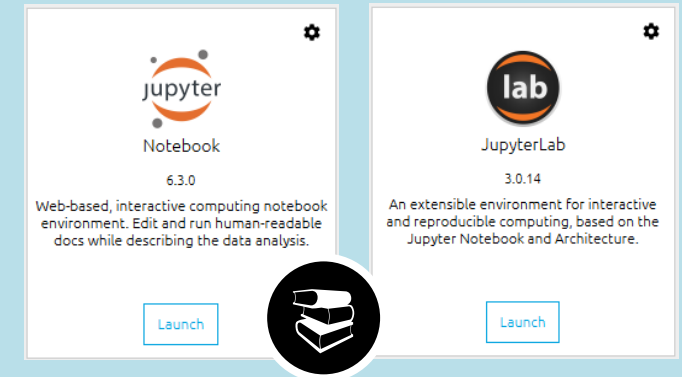
- Spyder is the most popular Python IDE for data analytics and diving deep in your data. It runs on your local machine
- Jupyter Lab is a kind of mix of Jupyter Notebook and Spyder that runs in your browser

AI Development



- VS Code is my favorite IDE for software development and Python coding. It is free to use and supports many other programming languages
- PyCharm is an IDE that is also very popular in software development

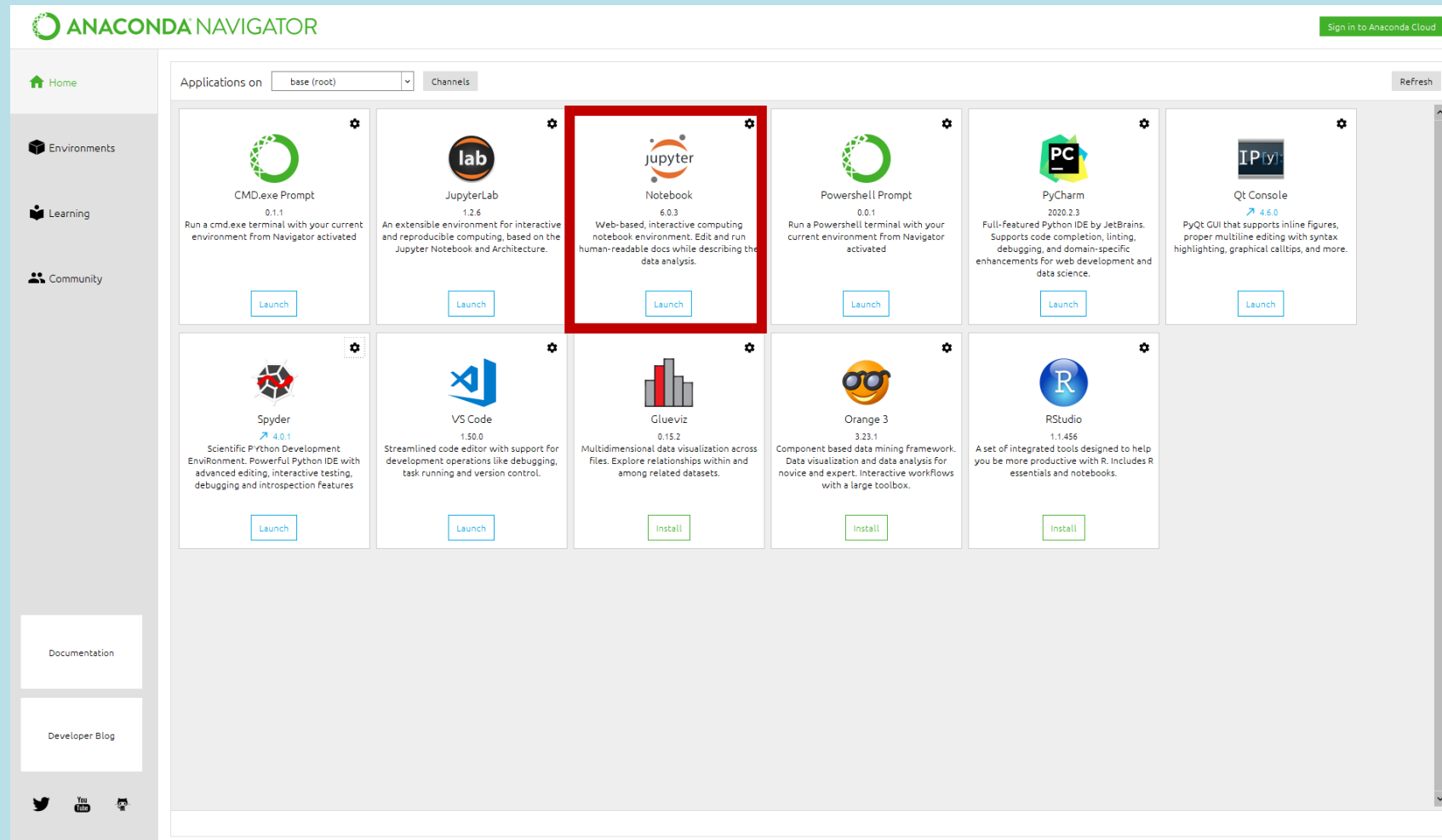
AI Development / Data Science



- Jupyter Notebook and Jupyter Lab are tools to develop Python in your browser or on a AI-server
- Due to their code buckets they are great to share and discuss code with non-AI specialists or other experts

Adapted from Russell, S., & Norvig, P. (2016)

3.3 Start Jupyter from Anaconda

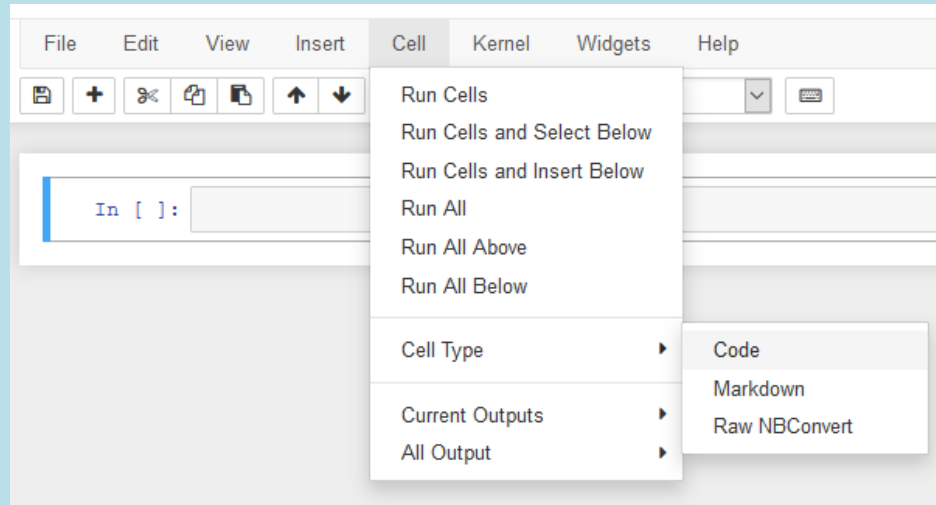


3.3 Notebooks

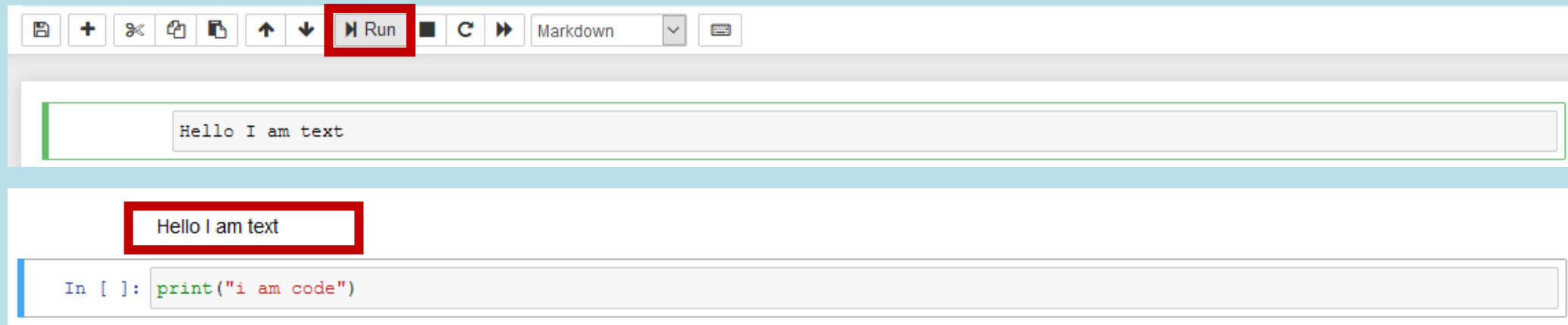


- Jupyter allows you to build simple code/text documents with your Python code

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 Other Hidden Champions: Colaboratory

The screenshot displays the Google Colaboratory (Colab) web interface. At the top, there's a header with the Colab logo, a welcome message 'Willkommen bei Colaboratory', and navigation links: 'Datei', 'Bearbeiten', 'Anzeige', 'Einfügen', 'Laufzeit', 'Tools', and 'Hilfe'. On the right, there are links for 'Teilen' (Share), a settings gear, and a blue 'Anmelden' (Sign in) button.

Below the header, a sidebar on the left titled 'Inhalt' (Content) shows a table of contents with sections like 'Erste Schritte' (First Steps), 'Data Science', 'Maschinelles Lernen' (Machine Learning), 'Weitere Ressourcen' (Further Resources), and 'Beispiele für maschinelles Lernen' (Examples for machine learning). The main content area is titled 'Was ist Colaboratory?' (What is Colaboratory?). It explains that Colab allows users to write and execute Python code in their browser. It lists three key features: 'Keine Konfiguration erforderlich' (No configuration required), 'Kostenlosen Zugriff auf GPUs' (Free access to GPUs), and 'Einfache Freigabe' (Simple release). It also mentions that both students and researchers can benefit from Colab, with a link to a video 'Einführung in Colab' (Introduction to Colab).

The 'Erste Schritte' (First Steps) section is expanded, showing that the document is an interactive 'Colab-Notebook'. It provides an example of a code cell with a Python script that calculates the number of seconds in a day (24 * 60 * 60) and stores the result in a variable named 'seconds_in_a_day'. The output of the code is displayed as '86400'. The text explains how to execute the code by clicking the cell or using the 'Befehlstaste/Strg + Enter' (Command/Control + Enter) keyboard shortcut. It also notes that variables defined in one cell can be used in subsequent cells.

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?

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



Artificial Intelligence

Algorithms and Applications with Python

Lectorial

Dr. Dominik Jung
dominik.jung42@gmail.com

python

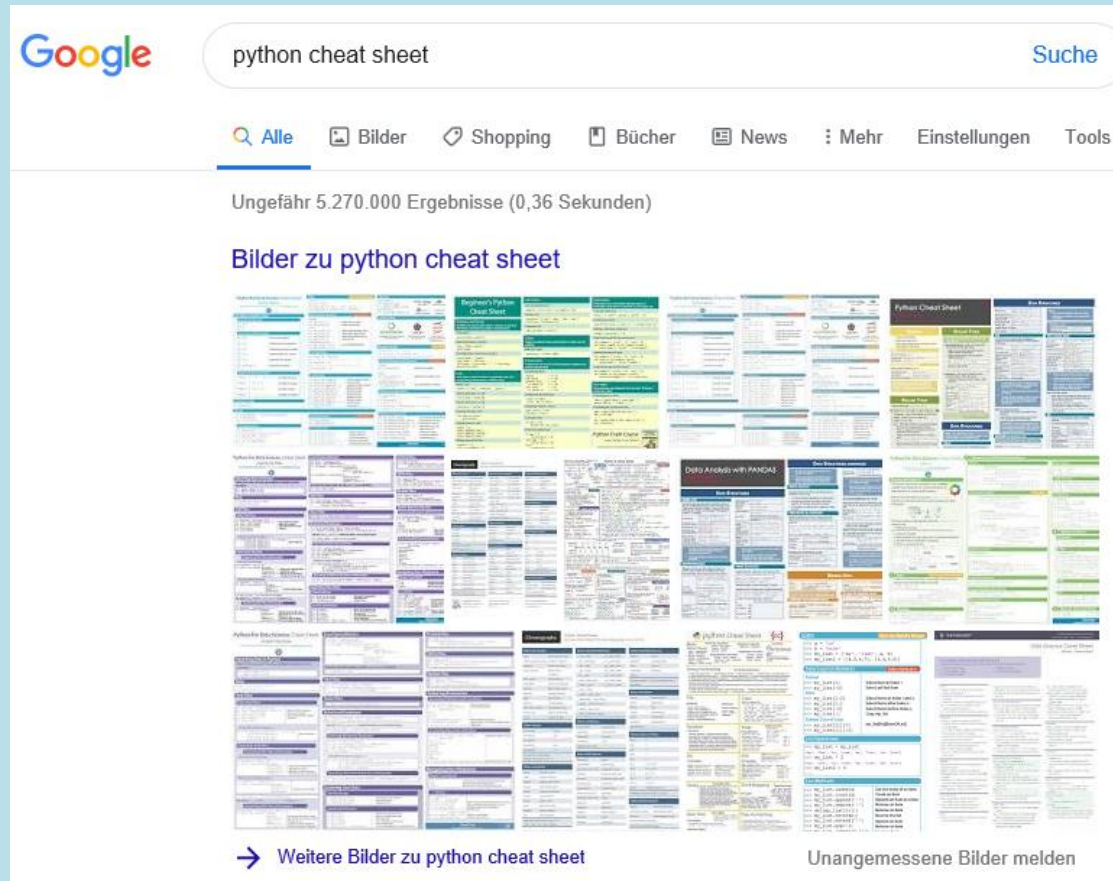
dominikjung42 updated syllabus for WT2020		c3d2300 5 days ago	🕒 60 commits
Capstone project	added report LATEX template	5 months ago	
Code	added sample data files for lecture 3	20 days ago	
Exams	added old exam for exercise	5 days ago	
Guest lectures	Updated lecture syllabus	8 months ago	
Lecture	updated syllabus for WT2020	5 days ago	



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



Python Cheatsheet



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

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 definitely help you to deepen your understanding of the presented concepts and algorithms.

Literature

1. Ernesti, J., & Kaiser, P. (2017). *Python 3: das umfassende Handbuch*. Rheinwerk Verlag.
2. Official 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/>

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 its design principles, which you can find inside the Python interpreter itself by typing `import this`. Check it out!

Data Frame	<i>A two-dimensional data structure, where data is aligned in a tabular fashion in rows and columns.</i>
IDE	<i>Integrated Development Environment. Applications that facilitate the development of software or code by giving the user an interface to work with.</i>
Libraries/Packages	<i>Collection of pre-written code (functions and methods) to perform certain task.</i>
Notebooks	<i>Virtual environment that enables literate programming.</i>
Programming language	<i>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).</i>
Software Tool	<i>Software that is designed to be used for a specific use case (e.g. Dashboarding, Data Visualization, Modelling).</i>

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!

