
# Introduction to Python

*Sources: https://www.python.org/*

Prof. Dr. Reinhard Madlener

FCN | Future Energy Consumer Needs and Behavior

E.ON Energy Research Center

RWTH AACHEN UNIVERSITY

# Agenda

- Optimization software: Fields of application

- Spyder
  - Interface overview
  - Editor

- Python
  - Variables
  - Some data types
  - Control flow tools

# Agenda

- Optimization software: Fields of application

- Spyder
  - Interface overview
  - Editor

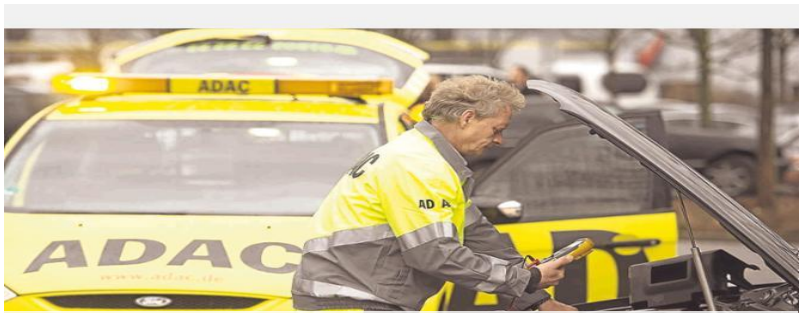
- Python
  - Variables
  - Some data types
  - Control flow tools

## Optimization questions from everyday life


Robust, "delay-resistant" timetable at Deutsche Bahn

Determination of prices for seats in the aircraft


Construction of a new Zalando site




Deployment planning of ADAC roadside assistance

**Can you think of any other examples?**

# Fields of application

Optimization question

External influencing variables


Problems in practice:

- Large data volumes
- Complex problem contexts
- Non-intended effects

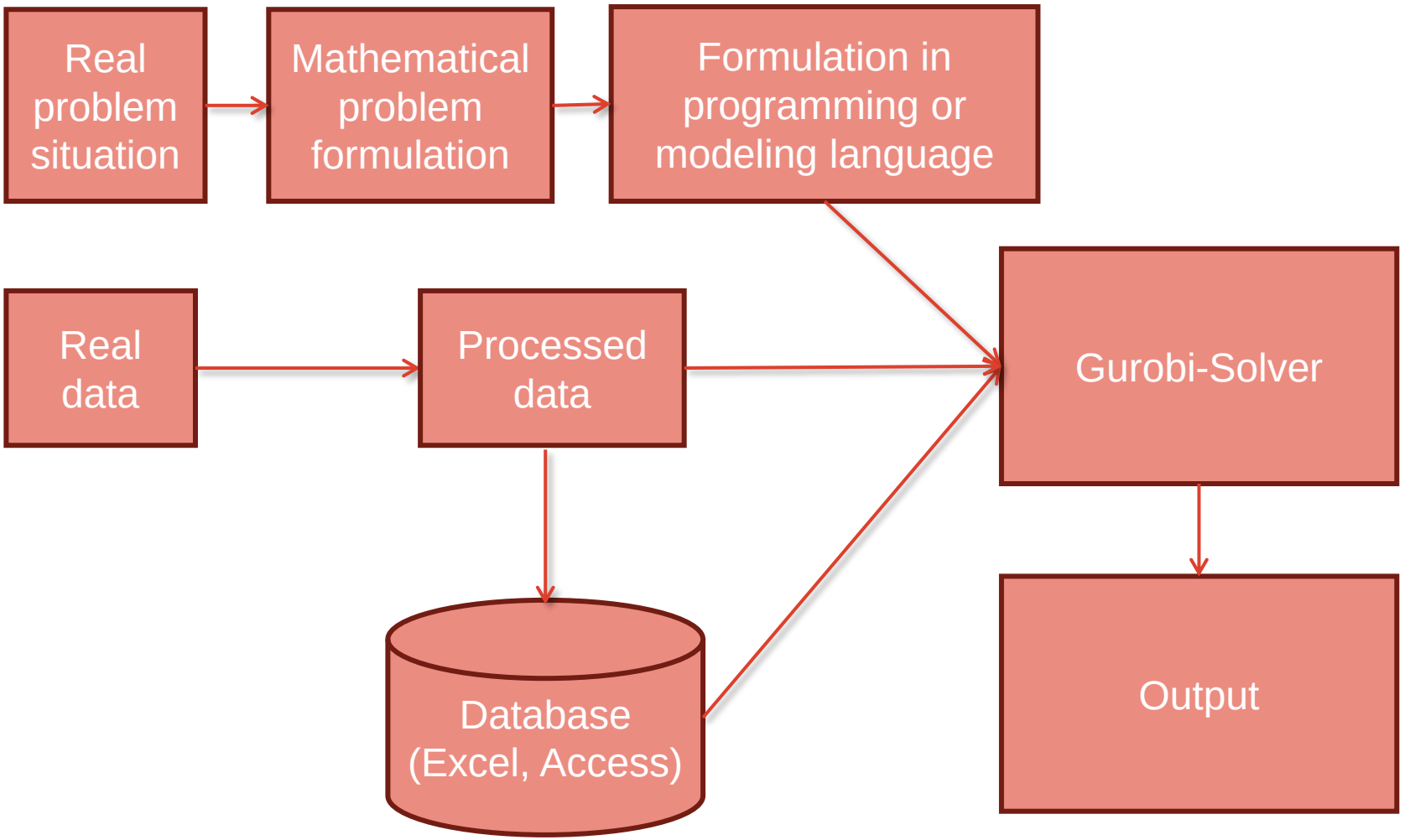
Deployment of optimization software

Software support

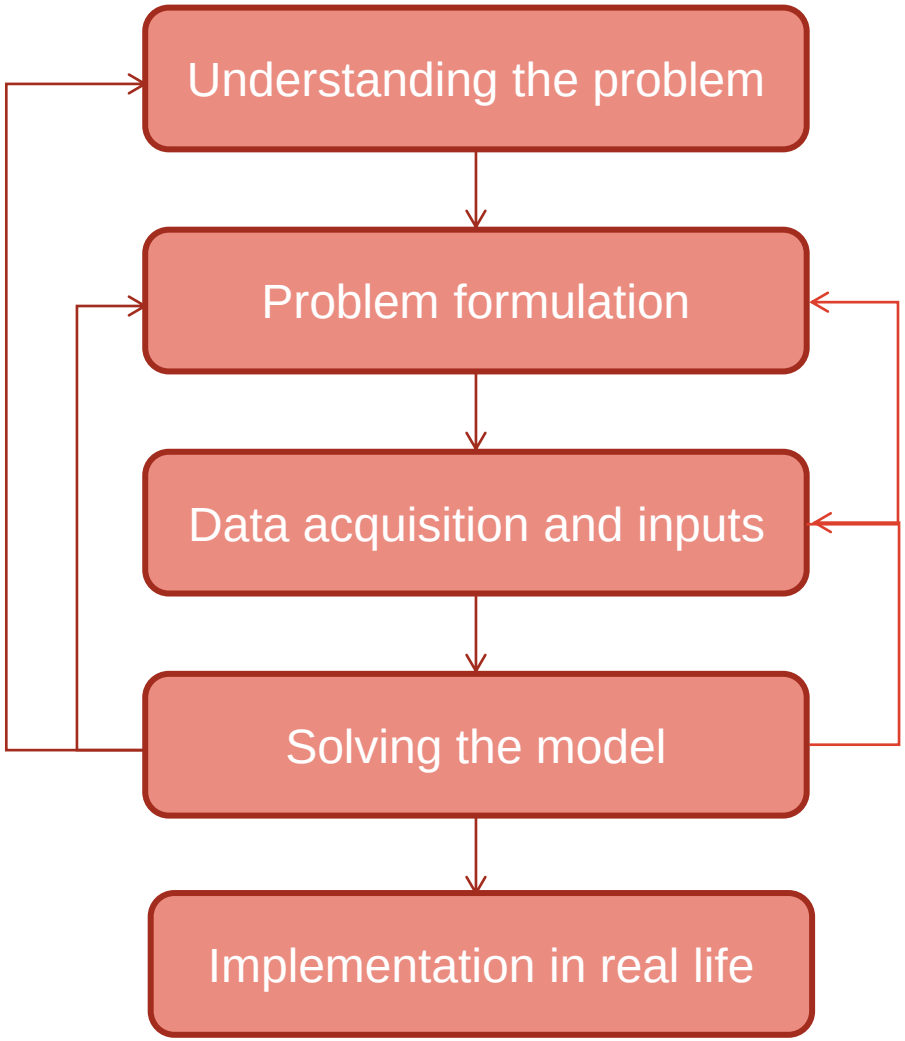
- Different solution algorithms and heuristics implemented
- Possibilities to analyze the solution
- Better understanding of the problem context

- Problems cannot be solved "by hand"
- Inefficient sequential solution

# Fields of application



# Fields of application



# Agenda

- Optimization software: Fields of application

- Spyder
  - Interface overview
  - Editor

- Python
  - Variables
  - Some data types
  - Control flow tools

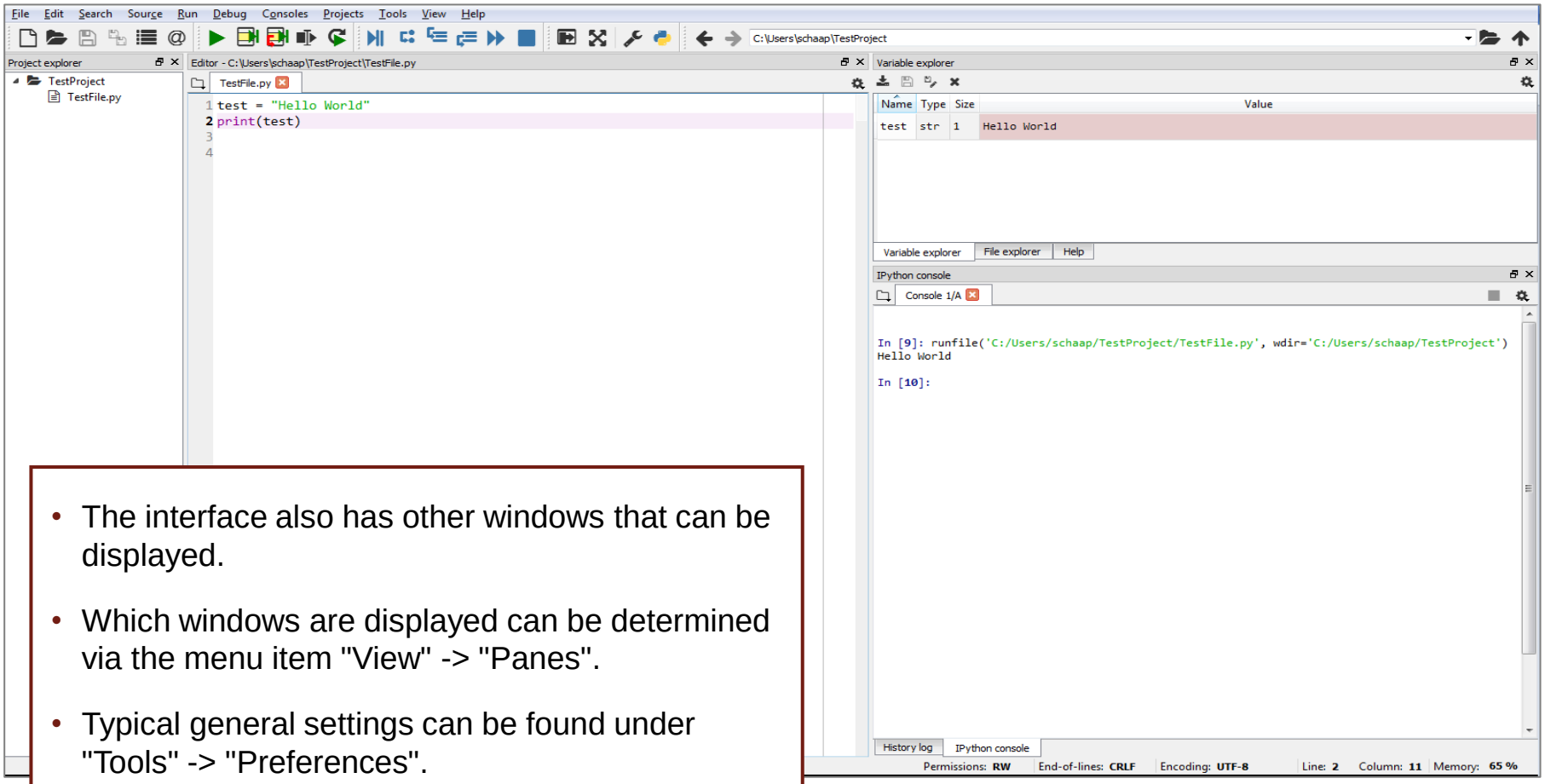
# Agenda

- Optimization software: Fields of application

- Spyder
  - Interface overview
  - Editor

- Python
  - Variables
  - Some data types
  - Control flow tools

# Interface overview




- The interface also has other windows that can be displayed.
- Which windows are displayed can be determined via the menu item "View" -> "Panes".
- Typical general settings can be found under "Tools" -> "Preferences".

# Agenda

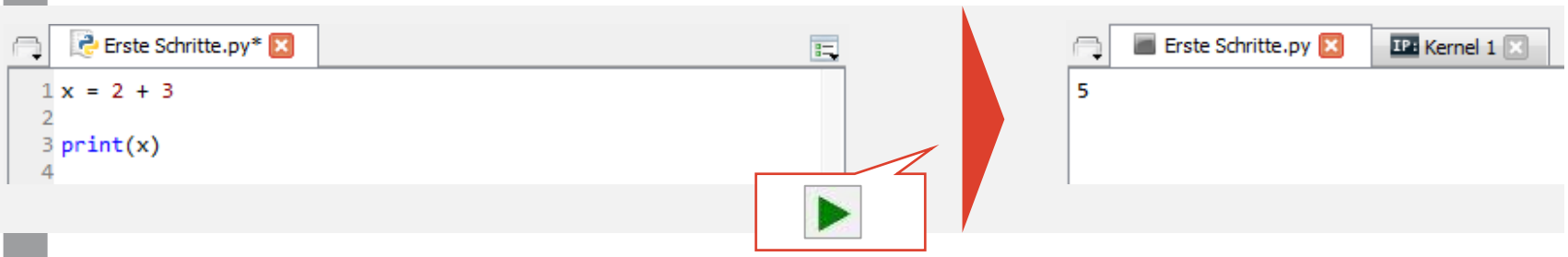
- Optimization software: Fields of application

- Spyder
  - Interface overview
  - Editor

- Python
  - Variables
  - Some data types
  - Control flow tools

# Editor


- In the editor window you can write Python code which is saved as a file.
- The code is executed only by the "Run file" (F5).
- Line numbers, syntax errors and "warnings" are displayed in the margin.




Step 1: Write program in the editor

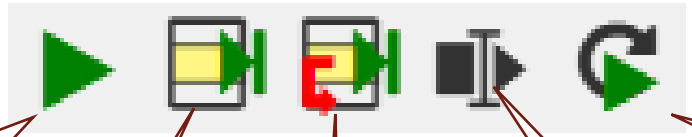
Step 2: Run via "Run file"

Step3 : The print() produce the output



E.ON Energy Research Center | RWTH AACHEN UNIVERSITY

# Editor – Run file



**Run file**
Run file open in the editor. How exactly to run → "Run Settings".

**Run again last file**
Runs the last executed program again, regardless of current selection.

**Run current cell**
If the code is divided into cells, these can be executed individually with this.

**Run current cell and go to the next one**
Enables manual-sequential execution of the cells.

**Run selection or current line**
Executes the code step by step and displays the intermediate results.

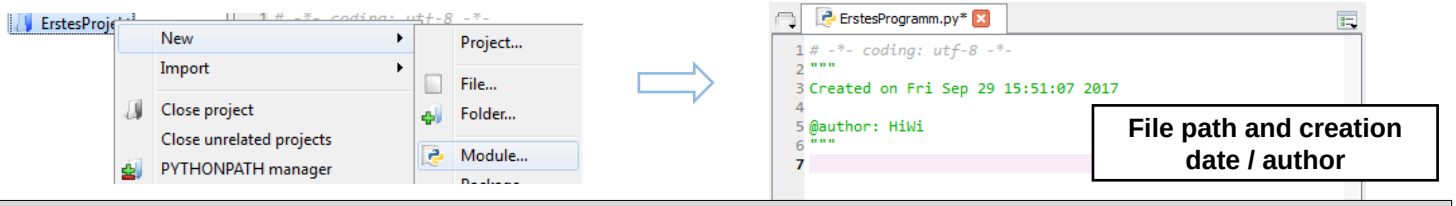
# Editor – First program

## Step 1: Open/run Spyder

## Step 2: Create a new project

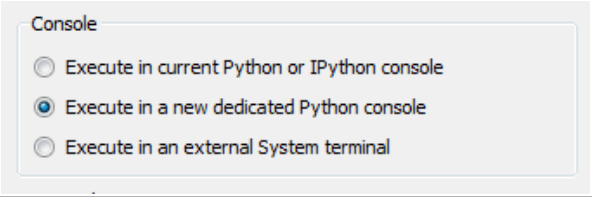

## Step 3: Create a new module

The new project is now displayed in the Project explorer. In this project we want to create a new module, the first program. We name the module "FirstProgram" and save it.




## Step 4: Set run settings

Select "Run Settings" (Under "Run" "Configuration per file"). Only one option should be changed, all other settings can remain unchanged.

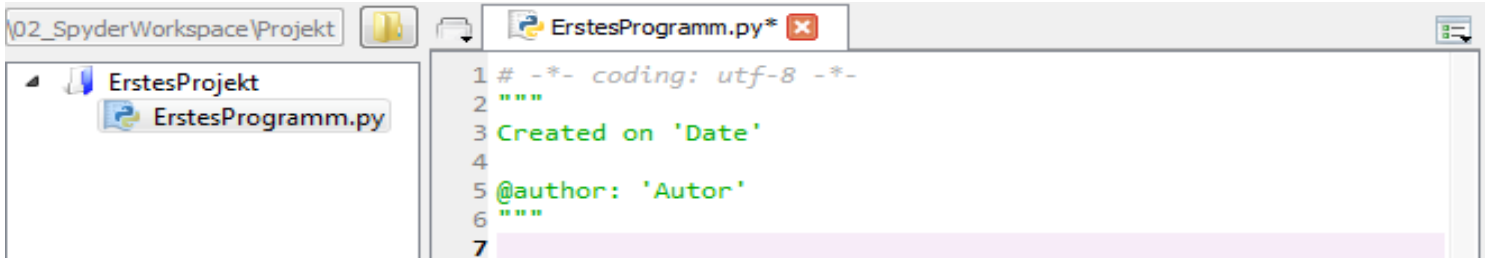




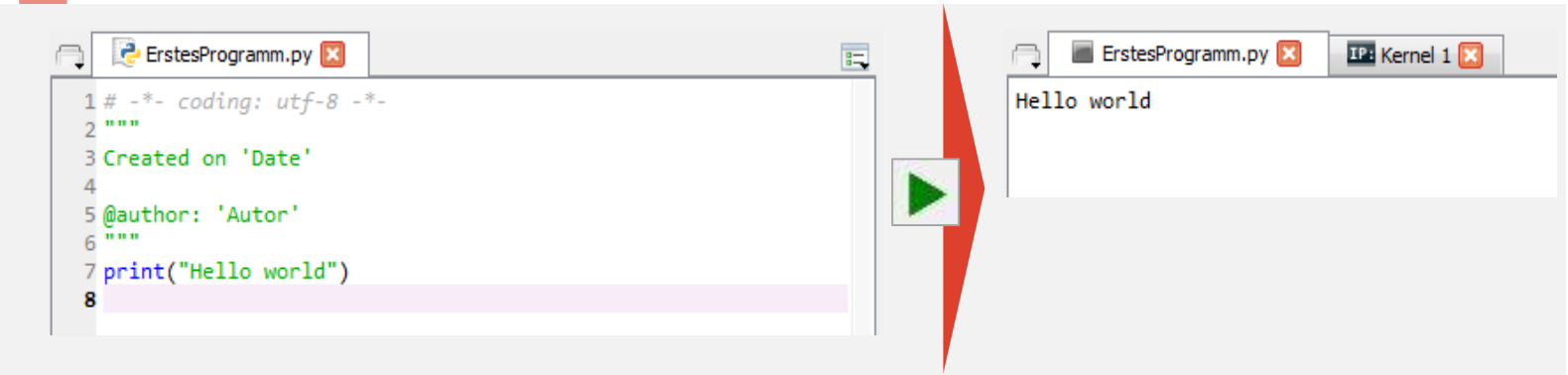

E.ON Energy Research Center | RWTH AACHEN UNIVERSITY

# Editor – First Program

## Step 5: Overview of the result




## Step 6: Write first program



# Agenda

- Optimization software: Fields of application

- Spyder
  - Interface overview
  - Editor

- Python
  - Variables
  - Some data types
  - Control flow tools

# Agenda

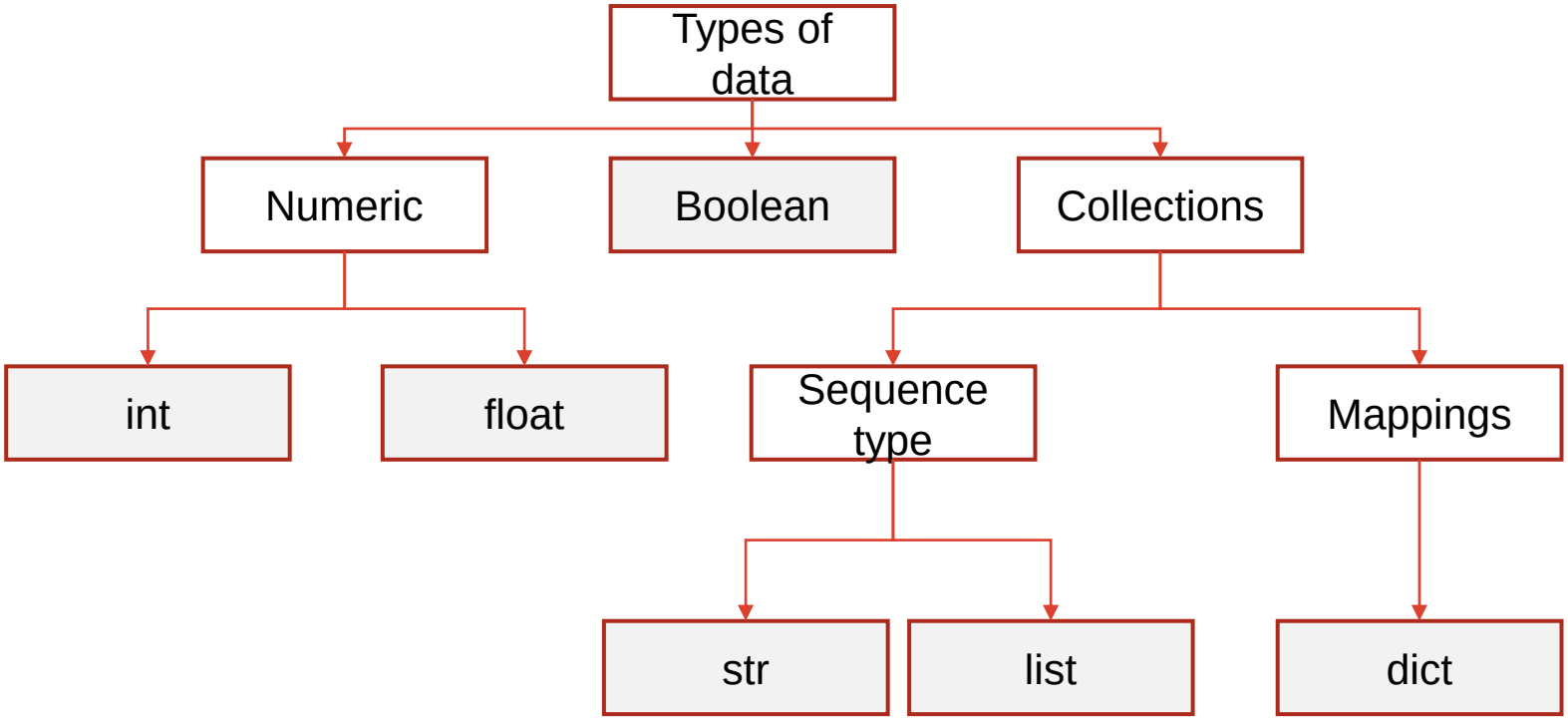
- Optimization software: Fields of application

- Spyder
  - Interface overview
  - Editor

- Python
  - Variables
  - Some data types
  - Control flow tools

# Variables

Containers/variables can be assigned values of different data types, or can themselves contain further containers. Some of them are discussed here:



# Variables

- Variables represent containers → Values can be " stored"

```
1 x = "Haus"
2
3 print(x)
4
```

The print function: prints the argument to the console.

```
Haus
```

- Attention: The "=" does not correspond to the mathematical equal sign, but to an **assignment**

```
1 Hoehe = 20
2 Breite = 30
3 Laenge = 25
4 Volumen = Hoehe * Breite * Laenge
5
6 print(Volumen)
7
```

```
15000
```

- Variables are **changeable** during the run

```
1 x = 1
2 print(x)
3 x = x + 1
4 print(x)
5
```

```
1
2
```

# Agenda

- Optimization software: Fields of application

- Spyder
  - Interface overview
  - Editor

- Python
  - Variables
  - Some data types
  - Control flow tools

# Data types – Numeric

- Variables can be assigned numerical values of different data types

```
1 y = 4
2 print(type(y))
3
```

```
<type 'int'>
```

- *int* stands for integer whole numbers. Common arithmetic operations can be applied to numbers

```
1 x = 14/6
2 print(x)
3
```

```
2.3333333333333335
```

- Mostly, however, numbers of data type $float$ are used → floating point numbers

```
1 print(type(6.0))
2
3 print(14/6.0)
4
```

```
<type 'float'>
2.33333333333
```

```
1 x = 17.0/5
2 y = 2**3
3 z = x + y
4
5 print(y)
6 print(z)
7
```

```
8
11.4
```

# Data types – Sequences

- Also words and general strings can be used and assigned

```
1 MeinName = "Max Mustermann der Zweite !*"
2
3 print("Mein Name ist " + MeinName)
4
```

```
Mein Name ist Max Mustermann der Zweite !*
```

You can also print combined arguments of the same data type with +

Output with *print*

- Variables that are not strings themselves can also be combined with a string

```
1 x = 1
2 y = 2
3 z = 3
4
5 print("%s plus %s gibt %s " % (x, y, z))
6
```

```
1 plus 2 gibt 3
```

- You can use \n (for new line) to include line breaks in the output

```
1 x = 1
2 y = 2
3 z = 3
4
5 print("%s plus %s gibt \n%s " % (x, y, z))
6
```

```
1 plus 2 gibt
3
```

# Data types – Sequences

- Multiple values (of different data types) can also be assigned: Data type *list*

```
L1 = [1, "f", 3.14, "Lager"]

print(L1)

```

```
[1, 'f', 3.14, 'Lager']
```

- The values are stored in an **orderly manner** and can be retrieved and changed

```
L1 = [1, "f", 3.14, "Lager"]

print(L1[0])

L1[3] = "KeinLagerMehr"

print(L1)

```

Python counts starting from 0, not from 1

```
1
[1, 'f', 3.14, 'KeinLagerMehr']
```

- Via the *len* – function the number of elements in a list can be obtained

```
L1 = [41, "Datum", 1.032, 12]

print(len(L1))

```

```
4
```

# Data types – boolean

- The data type **boolean**: *bool* can only correspond to two different values: *True* or *False*

```
1 x = True
2 y = False
3
4 print(type(y))
5 print(x)
6
```

```
<type 'bool'>
True
```

- This corresponds as far as possible to the mathematical understanding of a **true** or **false** statement. Here in the example below are the available relation operators.

```
1 print(10 == 10)
2
3 print(9 <= 9)
4
5 print(9 < 8)
6
7 print(7 != 4)
8
```

```
True
True
False
True
```

- Mathematical logic operators are also applicable

```
1 print(10 < 10 and 5 == 5)
2
3 x = 2
4
5 print(1 < x < 3 or 4 < 3)
6
```

"or" corresponds to the mathematical operator ∨, i.e. either one statement or both statements

```
False
True
```

# Data types – Mappings

- It is also possible to store data as **key-value pairs**
- A variable to which key-value pairs are assigned is called a **dictionary** (*dict*)

```
1 Transportkosten = {"Lager1": 147, "Lager2": 256}
2
3 print(type(Transportkosten))
4 print(Transportkosten)
5
```

```
<type 'dict'>
{'Lager1': 147, 'Lager2': 256}
```

- The values can be retrieved via the keys

```
1 TK = {"Lager1": 147, "Lager2": 256}
2
3 print(TK["Lager1"])
4
5 print(TK["Lager1"]+TK["Lager2"])
6
```

```
147
403
```

- The values are variable during the run, also can be supplemented by pairs

```
1 TK = {"Lager1": 147, "Lager2": 256}
2
3 TK["Lager3"] = 320
4 TK["Lager1"] = 100
5
6 print(TK)
7
```

The key-value pair “Lager3” ↔ 320 is added, 147 at “Lager1” is replaced

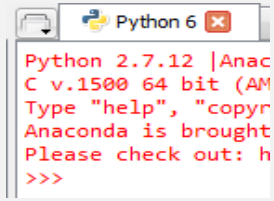
```
{'Lager1': 100, 'Lager3': 320, 'Lager2': 256}
```

Unlike lists, dictionaries are **not ordered**

# Data types – Mappings

- All declared variables can also be viewed in the variable explorer after execution

```
1 x = 1.0
2
3 y = 1
4
5 Die_Erde_ist_flach = True
6
7 L1 = {"a": 5, "b": 13}
8
```




Variable explorer

| Name | Type | Size | Value |
|---|---|---|---|
| Die_Erde_ist_flach | bool | 1 | True |
| L1 | dict | 2 | {'a': 5, 'b': 13} |
| x | float | 1 | 1.0 |
| y | int | 1 | 1 |

# Agenda

- Optimization software: Fields of application

- Spyder
  - Interface overview
  - Editor

- Python
  - Variables
  - Some data types
  - Control flow tools

# Control flow tools

- In Python, lines of code are executed in the order they appear in the code ("from top to bottom")
- Instructions can be used to skip parts of code or execute them repeatedly

| *if*- Conditon | *for*- Loop |
| --- | --- |
| Execute the code only if a condition is met | Execute the code for all elements of a list |

- Whether a line of code belongs to a statement is indicated in Python by the indentation

```
1 if 2 < 1:
2     print("Diese Zeile wird nur ausgeführt, wenn die Bedingung True ist")
3 print("Diese Zeile wird immer ausgeführt")
4
```

```
Diese Zeile wird immer ausgeführt
```

# Control flow tools

- ***if* – condition**: Couples instructions to be executed to conditions

```
x = 3
y = 2
z = 1

if x < y:
    print("x ist kleiner als y")
elif x < z:
    print("x ist kleiner als z")
else:
    print("x ist groesser als y und z")

```

Expression of type *boolean*. So it is queried whether the statement is *True* or *False*

```
x ist groesser als y und z
```

- Operators can also be used with control statements

```
Bedingung1 = 2 < 1
Bedingung2 = 2 < 3

if Bedingung1 and Bedingung2:
    print("Beide Bedingungen sind erfuellt")
else:
    print("Es sind nicht beide Bedingungen erfuellt")

```

```
Es sind nicht beide Bedingungen erfuellt
```

```
if 2 < 1 or 4 < 5:
    print("Eine oder beide Bedingung/en ist/sind erfuellt")
else:
    print("Keine der Bedingungen ist erfuellt")

```

```
Eine oder beide Bedingung/en ist/sind erfuellt
```

# Control flow tools

- ***for* – loop**: iterates over a set of objects, statements contained within are repeated for all objects

```
1 VL = ["PuL", "EBWL", "Mathe"]
2 for x in VL:
3     print (x)
4
```

```
PuL
EBWL
Mathe
```

- For example, the sum can be formed over a set of numbers

```
1 I = [43, 14, 87, 23, 9]
2 s = 0
3 for i in I:
4     s = s + i
5
6 print(s)
7
```

Can also be written in abbreviated form: `s += i`

```
176
```

- Or the sum over all values in a dictionary

```
1 L1 = {"K1": 200, "K2": 150, "K3": 225}
2 s = 0
3 for i in L1:
4     s = s + L1[i]
5
6 print(s)
7
```

Here we iterate over K1, K2 and K3 (in no particular order)

```
575
```

# Control flow tools

- Nested iterations are also possible

```
Lager = ["Aachen", "Koeln", "Bonn"]
Fabriken = ["Muenchen", "Stuttgart"]

for i in Fabriken:
    for j in Lager:
        print("Von " + i + " nach " + j)

```

```
Von Muenchen nach Aachen
Von Muenchen nach Koeln
Von Muenchen nach Bonn
Von Stuttgart nach Aachen
Von Stuttgart nach Koeln
Von Stuttgart nach Bonn
```

- Additionally, there is the possibility to iterate over indices. For this we need 2 functions:

```
I = [13, 54, "Haus", 9.0]

print(len(I))
print(range(4))

print(range(len(I)))

```

***len*** generates the number x of list elements as a number. ***range*** generates from it a list that counts up in integers from 0 to x-1.

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

- Instead of iterating over each element, the range list is iterated over

```
I = ["Aachen", "Koeln", "Bonn"]

for i in range(len(I)):
    print(i)

```

```
0
1
2
```


## Contact

Prof. Dr. Reinhard Madlener
T +49 241 80 49820
RMadlener@eonerc.rwth-aachen.de
http://www.eonerc.rwth-aachen.de/fcn

E.ON Energy Research Center,
Institute for Future Energy Consumer Needs and Behavior (FCN)
Mathieustraße 10, 52074 Aachen, Germany



## Acknowledgement

Many thanks to Prof. Dr. Grit Walther, Chair of Operations Management for providing the German version of these slides


FCN | Future Energy Consumer Needs and Behavior

E.ON Energy Research Center

RWTH AACHEN UNIVERSITY