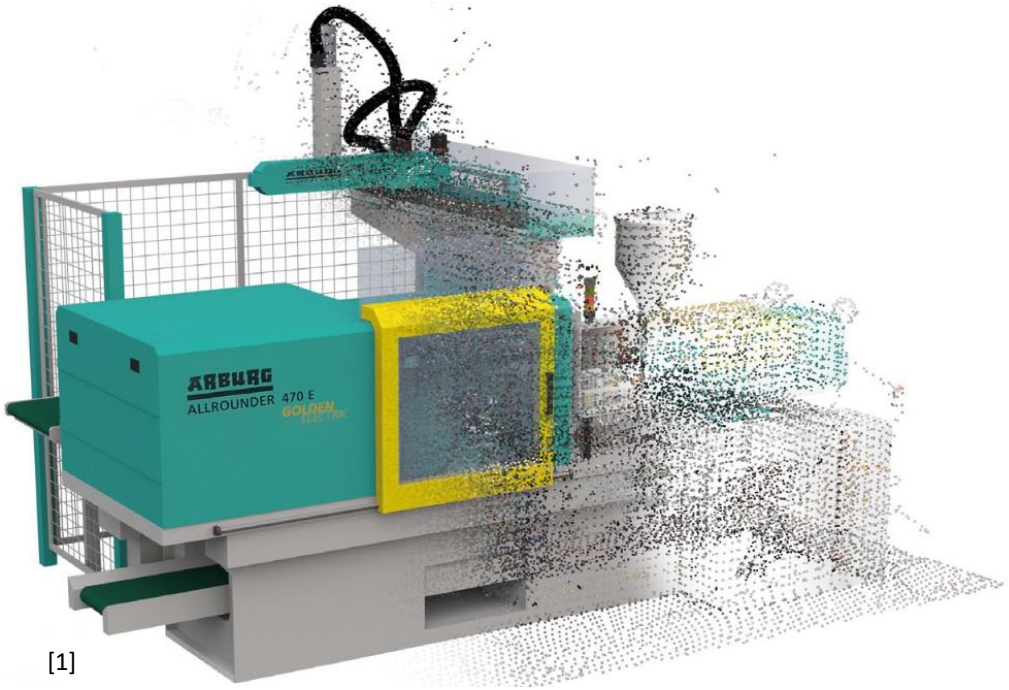
EUROPÄISCHE UNION:
Investition in Ihre Zukunft
Europäischer Fonds für regionale Entwicklung

Python Seminar & Workshop
Digital Twin of Injection Molding (DIM)

11.01.2021



[1]

Forschungskonsortium

- Institut für Werkstofftechnik / FG Kunststofftechnik, Prof. Dr.-Ing. H.-P. Heim
- FG Mess- und Regelungstechnik, Prof. Dr.-Ing. A. Kroll

Marco Klute

+

Alexander Rehmer

+

Stefan Rosenbach

+

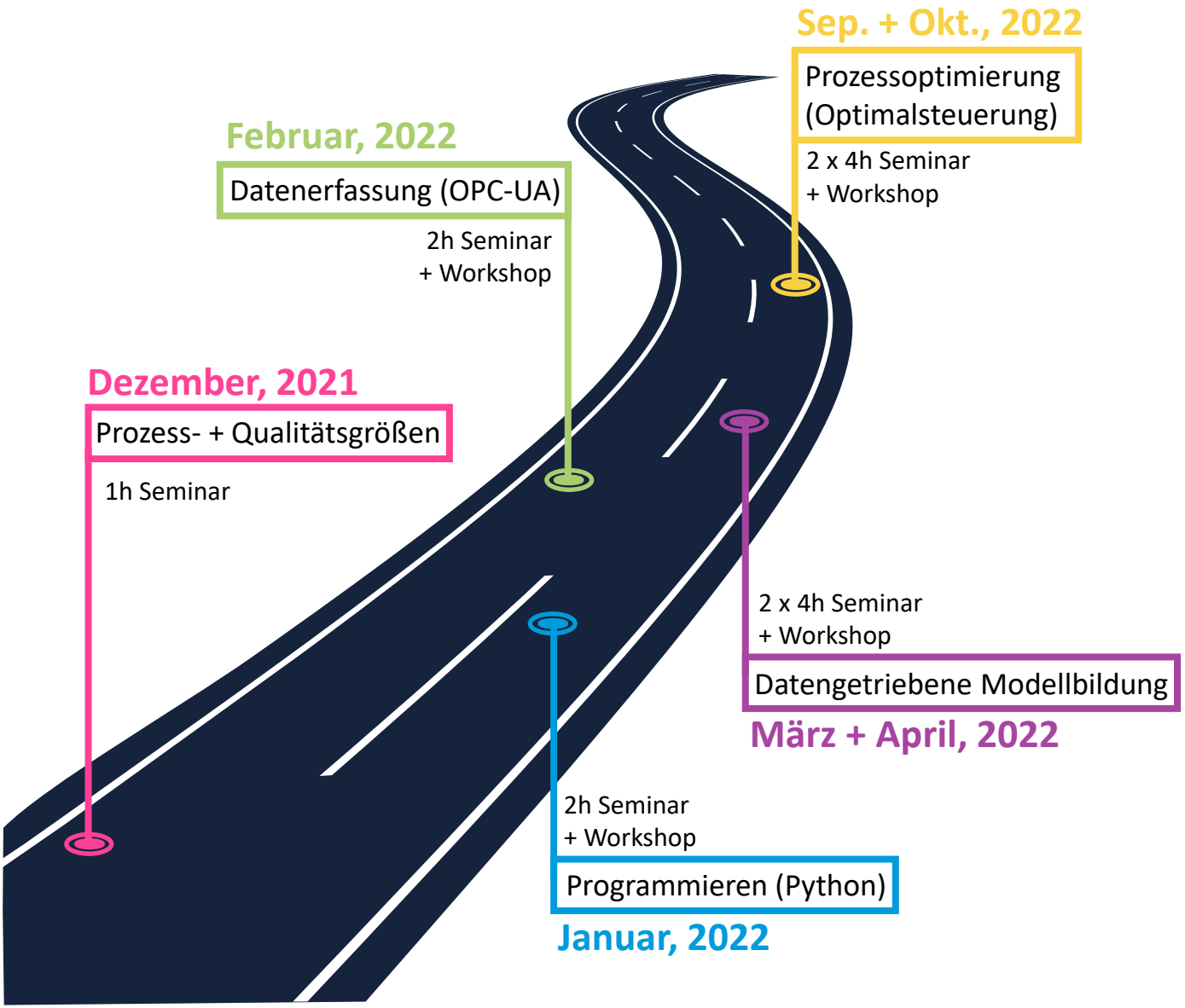
Studentische Hilfskräfte

Kontakt:

- dim@uni-kassel.de
- www.uni-kassel.de/go/DIM

Überblick über die Workshopreihe

EUROPÄISCHE UNION:
Investition in Ihre Zukunft
Europäischer Fonds für regionale Entwicklung



- Erfassung von Prozess- und Qualitätsgrößen
- **Programmieren mit Python**
  - Seminar: Vermittlung grundlegender und fortgeschrittener Aspekte der objektorientierten Programmierung mit Python
  - Workshop: Selbstständige Bearbeitung von Programmieraufgaben
  - Ziele:
    - Installieren und Ausführen von Python
    - Programmieren und Lesen von Python Code
- Datenerfassung mit OPC-UA
- Datengetriebene Modellbildung
- Prozessoptimierung mittels numerischer Optimalsteuerung

Agenda

EUROPÄISCHE UNION:
Investition in Ihre Zukunft
Europäischer Fonds für regionale Entwicklung

Seminar (1 h)

- **Warum Python?**
- **Python installieren und ausführen**
- **Programmieren mit Python**
  - Datentypen
  - Objekte
  - Funktionen
  - Klassen und Methoden
  - Flow Control
  - Nutzung von Bibliotheken

Rechnerübung (1 h)

Agenda

EUROPÄISCHE UNION:
Investition in Ihre Zukunft
Europäischer Fonds für regionale Entwicklung

Seminar (1 h)

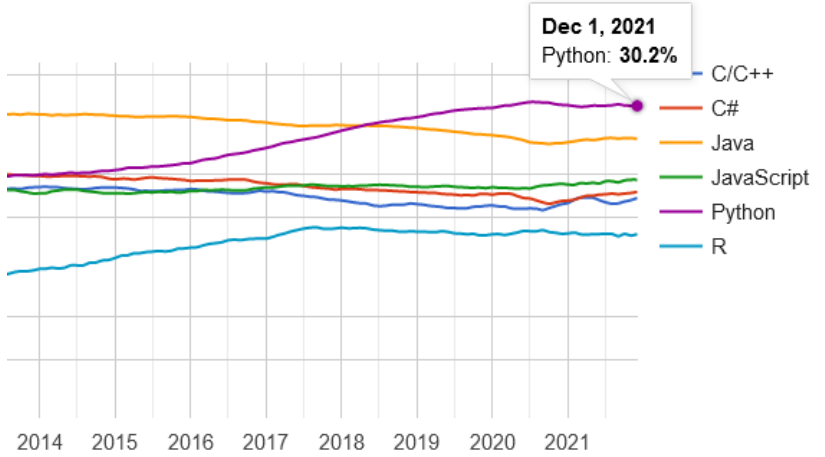
- **Warum Python?**
- **Python installieren und ausführen**
- **Programmieren mit Python**
  - Datentypen
  - Objekte
  - Funktionen
  - Klassen und Methoden
  - Flow Control
  - Nutzung von Bibliotheken

Rechnerübung (1 h)

Warum Python?

EUROPÄISCHE UNION:
Investition in Ihre Zukunft
Europäischer Fonds für regionale Entwicklung

Python rangiert seit Jahren unter den nachgefragtesten Programmier-sprachen und ist nach dem PYPL-Index die beliebteste Sprache der Welt



Gründe für die Beliebtheit von Python:

- Lernbarkeit: Durch eine einfache, der englischen Sprache beinahe ähnliche Syntax ist Python intuitiv lernbar und der Code besitzt eine hohe Lesbarkeit

```
print('Hello World!')
```

- Flexibilität: Als interpretierte Sprache kann derselbe Python-Code auf Linux, Windows oder Mac ausgeführt werden, Python kann einfach mit Java, .NET, oder C/C++ Bibliotheken verwendet werden
- Hunderte von Bibliotheken und Frameworks: Machine Learning, Data Science, Cloud Computing, Web Development, Robotik, ...
- Community Support: Zahlreiche Tutorials auf YouTube und im Web (www.w3schools.com, docs.python.org), hilfreiche Comunities (Stack Overflow, GitHub, etc.) mit erfahrenen Programmierern.
- Python ist kostenlos und open-source

- Aber: Python ist eine interpretierte Sprache und daher langsamer als bspw. C. Für Anwendungen mit harten Echtzeitanforderungen, in denen Verzögerungen von einer Sekunde bereits inakzeptabel sind, ist Python deshalb ungeeignet.
- Es existieren Compiler, welche ausführbaren C-Code aus Python Skripten erstellen, bspw. Cython.

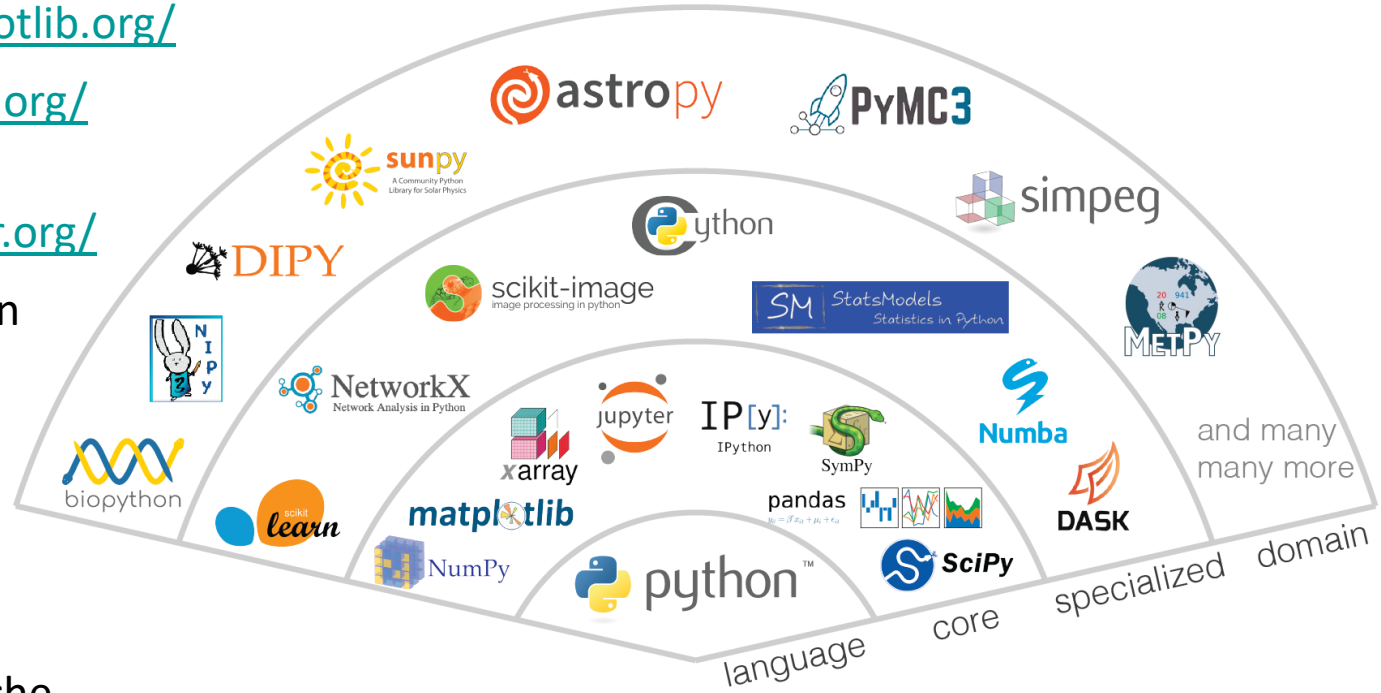
Python Ökosystem (+ CasADi)

EUROPÄISCHE UNION:
Investition in Ihre Zukunft
Europäischer Fonds für regionale Entwicklung

- **Numpy**: Methoden zum Rechnen mit numerischen Array-Objekten, http://www.numpy.org/
- **Scipy** : High-Level Methoden für Optimierung, Regression, Interpolation, Lineare Algebra, etc. http://www.scipy.org/
- **Matplotlib** : Visualisierung von Daten http://matplotlib.org/
- **IPython**: ausgereife Python console http://ipython.org/
- **Jupyter**: Erstellung von Notebooks (Python-Code angereichert mit Texten und Bildern) http://jupyter.org/
- **Pandas**: Speichern und Manipulieren von Zeitreihen https://pandas.pydata.org/

+

- **CasADi:** Nichtlineare Optimierung und algorithmische Differentiation https://web.casadi.org/



+

CasADi

Agenda

EUROPÄISCHE UNION:
Investition in Ihre Zukunft
Europäischer Fonds für regionale Entwicklung

Seminar (1 h)

- **Warum Python?**
- **Python installieren und ausführen**
- **Programmieren mit Python**
  - Datentypen
  - Objekte
  - Funktionen
  - Klassen und Methoden
  - Flow Control
  - Nutzung von Bibliotheken

Rechnerübung (1 h)

Python installieren

EUROPÄISCHE UNION:
Investition in Ihre Zukunft
Europäischer Fonds für regionale Entwicklung

PIP

- Empfohlenes Tool zur Installation und Verwaltung von Python Paketen aus dem Python Package Index
- Pip verwaltet ausschließlich Python-Pakete
- Um die Installation von systemseitiger Software, welche von den Python Paketen benötigt werden (Dependencies), aber keine Python Pakete sind, muss sich der Nutzer selbst kümmern
- Installation von Third-Party-Software über pip mit sogenannten wheel-Paketen möglich
- Virtual-Environments über zusätzliche Tools wie virtualenv oder venv realisierbar
- Empfehlenswert für Anwendungen, die wenige Abhängigkeiten von Third-Party-Software haben und für erfahrene Anwender, die alle Abhängigkeiten selbst verwalten können / möchten.

ANACONDA.

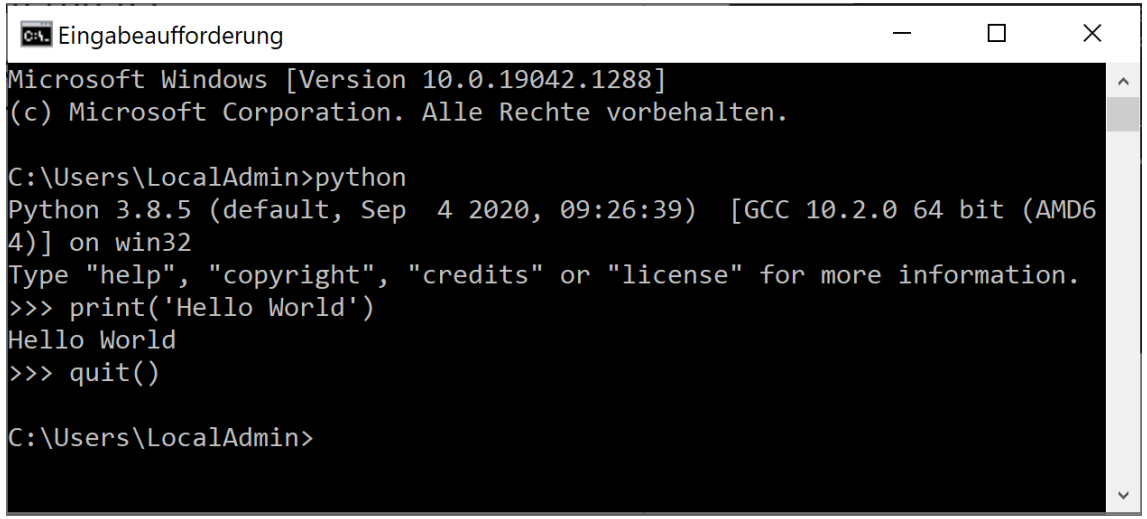
- Cross-Plattform Tool zur Installation und Verwaltung von Conda Paketen aus dem Anaconda Repository
- Fast alles kann ein Conda-Paket sein: Python-Pakete, C-Pakete, C-Compiler, Python Interpreter, ...
- Systemseitige Software sind somit nur weitere Conda-Pakete, die aus dem Anaconda Repository installiert werden kann
- Die Paketierung aller Dependencies ermöglicht
  - Portabilität: Dieselbe Conda-Umgebung kann auf allen Betriebssystemen verwendet werden
  - Reproduzierbarkeit: Verwendung der gleichen Conda-Pakete über alle Plattformen hinweg
  - Konsistenz: Python-Pakete und System-Pakete werden auf die gleiche Weise (als Conda-Paket) installiert
- Virtual-Environments (unterschiedliche Python-Installationen mit unterschiedlichen Paketen) sind built-in
- Insbesondere für wissenschaftliche Anwendungen wegen der hohen Abhängigkeit von Third-Party-Software empfehlenswert
- Nachteil: Installation von Paketen zum Teil langsam

Python ausführen

EUROPÄISCHE UNION:
Investition in Ihre Zukunft
Europäischer Fonds für regionale Entwicklung

Kommandozeile (interaktiv)

- Eingabe von `python` in der Kommandozeile startet den Python-Interpreter
- Nachfolgend kann Python Code zeilenweise eingegeben und ausgeführt warden
- Beenden des Interpreters mit `quit()`



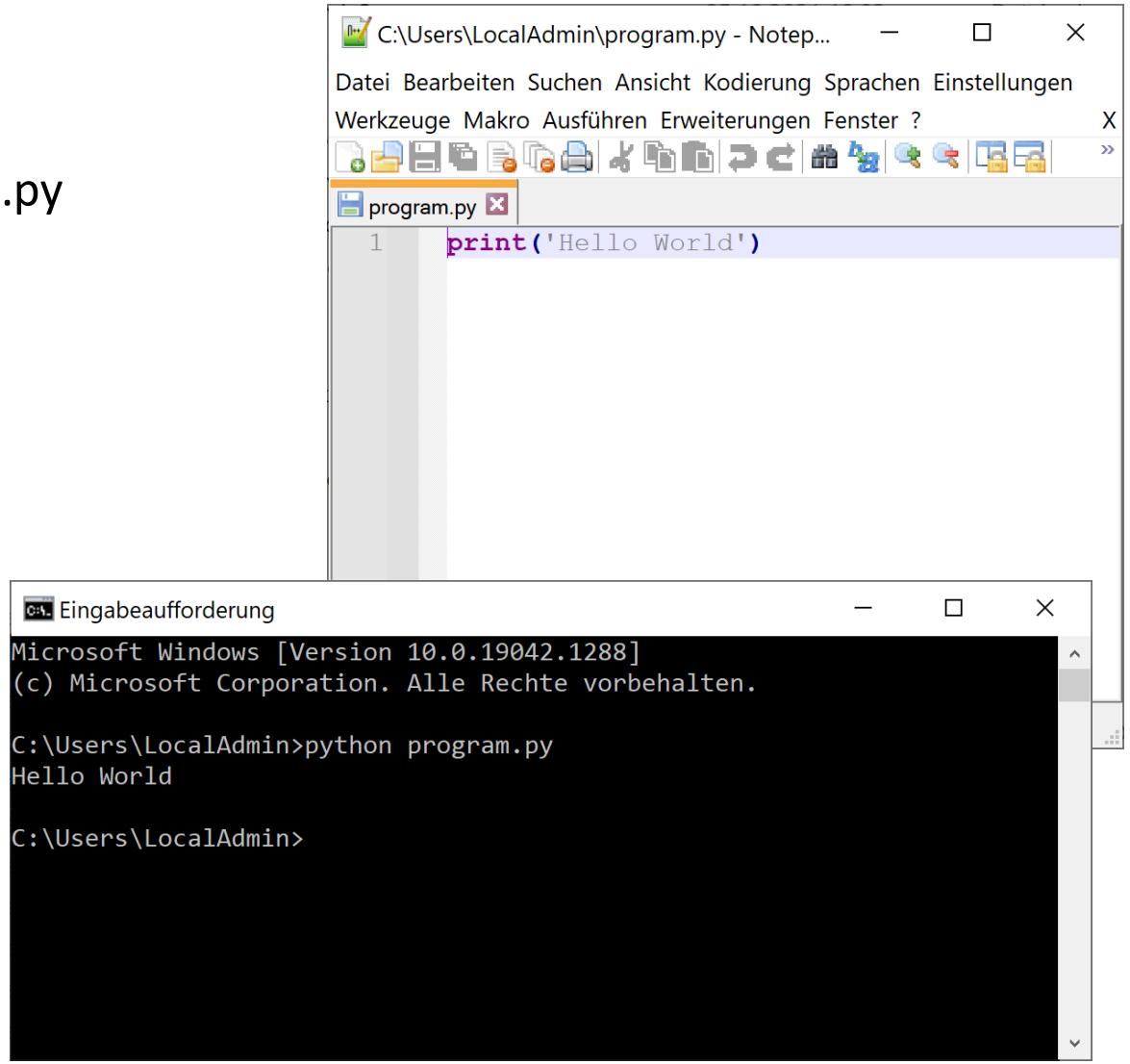
- Immer nur Eingabe eines einzigen Python Befehls und dessen sofortige Ausführung
- REPL: Read, Evaluate, Print, Loop
- Command history steht zur Verfügung
- Befehlszeilenergänzung (je nach Betriebssystem, falls installiert)

Python ausführen

EUROPÄISCHE UNION:
Investition in Ihre Zukunft
Europäischer Fonds für regionale Entwicklung

Editor und Kommandozeilenaufruf

- .py-Datei mit Programmcode in Editor erstellen
- Aufruf über die Kommandozeile mit python <filename>.py
- Verbreitete Editoren:
  - Emacs
  - Visual Studio Code
  - Atom
  - Vim



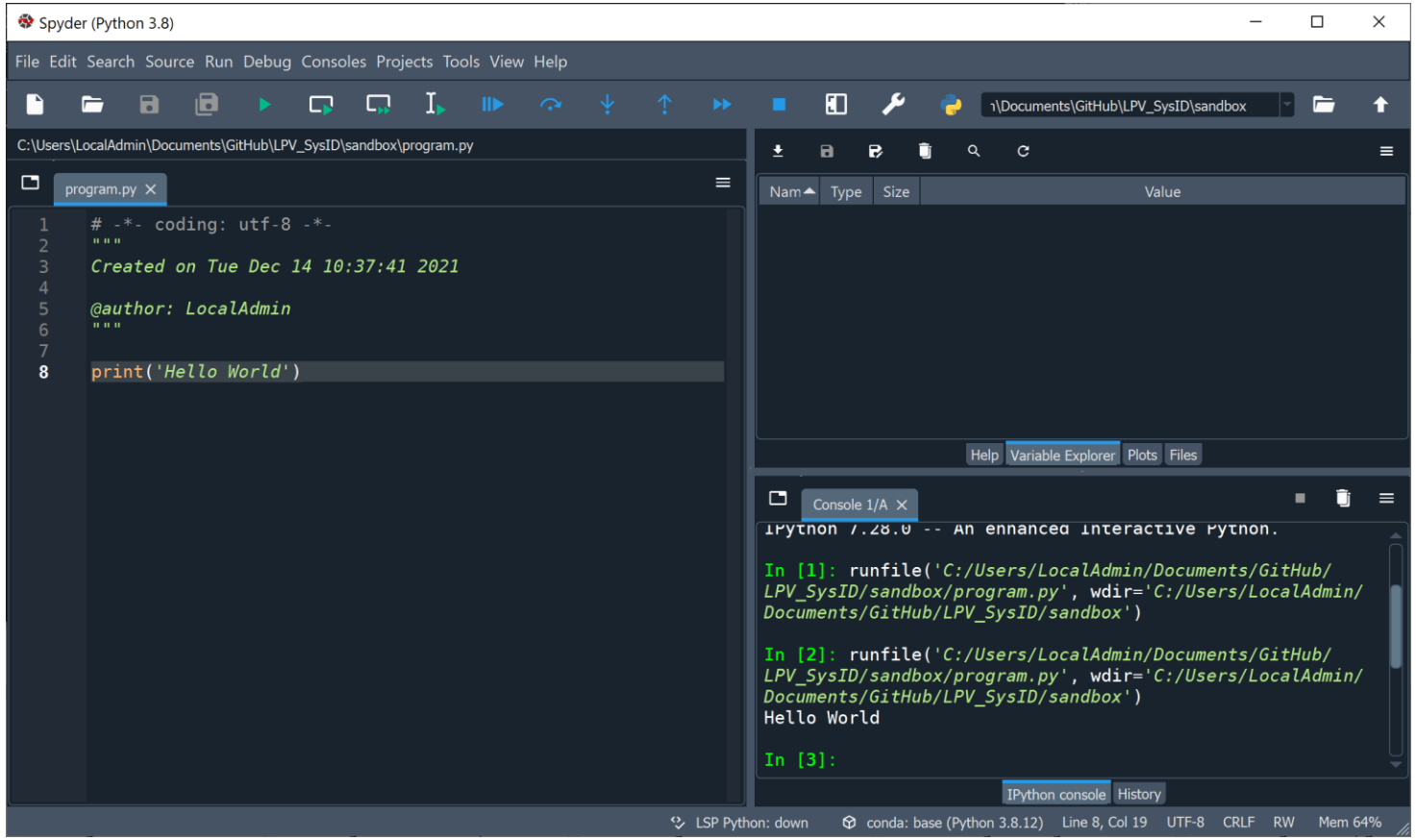
- Ausführung mehrerer Befehle auf einmal
- Syntax-Highlighting
- Befehlszeilenergänzung (abhängig von Editor)

Python ausführen

EUROPÄISCHE UNION:
Investition in Ihre Zukunft
Europäischer Fonds für regionale Entwicklung

Integrated Development Environment (IDE)

- Editor, Python-Shell, Variable Explorer, und File-Browser in einer Oberfläche
- Spyder und Eclipse sind die verbreitetsten IDEs



- Command History
- Befehlszeilenergänzung
- Syntax Highlighting
- Debugging Tools

Agenda

EUROPÄISCHE UNION:
Investition in Ihre Zukunft
Europäischer Fonds für regionale Entwicklung

Seminar (1 h)

- Warum Python?
- Python installieren und ausführen
- **Programmieren mit Python**
  - Datentypen
  - Objekte
  - Funktionen
  - Klassen und Methoden
  - Flow Control
  - Nutzung von Bibliotheken

Rechnerübung (1 h)

Datentypen

EUROPÄISCHE UNION:
Investition in Ihre Zukunft
Europäischer Fonds für regionale Entwicklung

Built-in Datentypen

In Python vorimplementierte Datentypen, die ohne das Einbinden von Modulen zur Verfügung stehen

Text:	`str`
Numerisch:	`int`, `float`, `complex`
Sequenzen:	`list`, `tuple`, `range`
Mapping:	`dict`
Set:	`set`, `frozenset`
Boolean:	`bool`
Binär:	`bytes`, `bytearray`, `memoryview`

Spezielle Datentypen

Spezielle Datentypen für die Ausführung bestimmter Aufgaben werden von Modulen als Objekte zur Verfügung gestellt, bspw.

- `ndarray` für algebraische Operationen von `numpy`
- `DataFrame` für die Speicherung von Zeitreihen in einer tabellenähnlichen Struktur von `pandas`
- `date` für die Repräsentation von Kalenderdaten von `datetime`
- …

Deklaration von Variablen

EUROPÄISCHE UNION:
Investition in Ihre Zukunft
Europäischer Fonds für regionale Entwicklung

- In Python wird der Datentyp beim Zuweisen eines Wertes zu einer Variablen automatisch gesetzt

Beispiel	Datentyp
`x = „Hello World“`	`str`
`x = 20`	`int`
`x = 20.0`	`float`

- Möchte man den Datentyp selbst spezifizieren, können dafür Konstruktorfunktionen genutzt werden

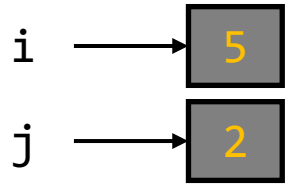
Beispiel	Datentyp
`x = str(5)`	`str`
`x = int(20.8)`	`int`
`x = float(20)`	`float`

Zeigerprinzip

EUROPÄISCHE UNION:
Investition in Ihre Zukunft
Europäischer Fonds für regionale Entwicklung

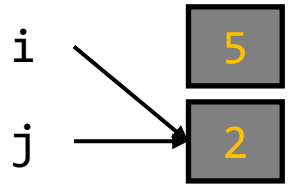
- Variablen in Python sind lediglich Zeiger oder Referenzen auf Objekte

```
i = 5        # Erzeuge ein Integer Objekt mit Wert 5, lasse i darauf zeigen
j = 2        # Erzeuge ein Integer Objekt mit Wert 2, lasse j darauf zeigen
```



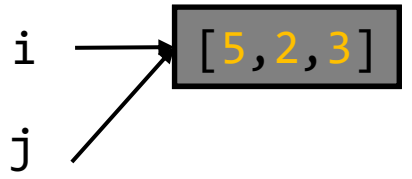
- Zwei Variablen gleichzusetzen, bedeutet, dass sie auf das gleiche Objekt zeigen. Es wird kein neues Objekt erzeugt

```
i = j        # Lasse i auf das gleiche Objekt zeigen wie j
```



- Inbesondere, wenn Objekte manipuliert werden, sollte dies beachtet werden:

```
i=[1,2,3]    # Erzeuge Instanz einer Liste und Lasse i darauf zeigen
j=i          # j zeigt auf dasselbe Objekt
i[0] = 5     # Ändere erstes Objekt der Liste
print(j)     # Ergebnis ist [5,2,3]
```



Integer & Float

EUROPÄISCHE UNION:
Investition in Ihre Zukunft
Europäischer Fonds für regionale Entwicklung

- Integers sind ganze Zahlen, positive oder negative (ohne Dezimalstellen), Floats sind reelle Zahlen mit Dezimalstellen

```
a = -1
b = 10
c = 3.5
d = 2.5e2
```

- Python stellt Funktionen für mathematische Operationen auf numerischen Datentypen bereit

Operation	Ergebnis
`x + y`	Summe von `x` und `y`
`x - y`	Differenz von `x` und `y`
`y * y`	Produkt von `x` und `y`
`x / y`	Quotient von `x` und `y`
`x**y`	Potenz von `x` mit Exponent `y`
`x // y`	Quotient von `x` und `y` ohne `Rest`

Integer & Float

EUROPÄISCHE UNION:
Investition in Ihre Zukunft
Europäischer Fonds für regionale Entwicklung

- Numerische Datentypen können mittels der Konstruktorfunktionen konvertiert werden

```
x = 1
y = 2.8

a = float(x)# konvertiert von int zu float 1.0
b = int(y)  # konvertiert von float zu int 2 !!!
```

- Numerische Datentypen sind nicht iterierbar und immutierbar

String

EUROPÄISCHE UNION:
Investition in Ihre Zukunft
Europäischer Fonds für regionale Entwicklung

- Strings werden in Python in einfache oder doppelte Anführungszeichen gesetzt

```
a = "Hello"
a = 'Hello'
```

- Sich über mehrere Zeilen erstreckende Strings werden in drei Anführungszeichen gesetzt

```
a = '''Lorem ipsum dolor sit amet,
    consectetur adipiscing elit ... '''
```

- Strings sind Arrays

```
a = 'Hello'
print(a[0])
```

- und können als solche per Schleife durchiteriert werden

```
for letter in 'Hello':
    print(letter)
```

- Strings sind immutierbar, d.h. nicht veränderbar. Nicht möglich ist also

```
a = 'Hello'
a[1] = 'a'
```

Listen & Tupel

EUROPÄISCHE UNION:
Investition in Ihre Zukunft
Europäischer Fonds für regionale Entwicklung

- Listen sind geordnete Sammlungen von Objekten beliebiger Datentypen.
- Sie werden generiert, indem die Listenelemente durch Kommata getrennt in [] gesetzt werden, oder indem eine Liste oder ein Tupel an die Konstruktorfunktion `list()` übergeben wird:

```
my_list = ['Hello', True, 50, ['a',1,False]]
my_list = list(('Hello', True, 50, list(['a',1,False])))
```

- Listenelemente sind indiziert

```
print(my_list[0])           # zeigt erstes Listenelement
print(my_list[-2])          # zeigt vorletztes Listenelement
print(my_list[1:4])         # zeigt zweites bis inkl. drittes Listenelement (slicing)
print(my_list[1:-1])        # zeigt zweites bis inkl. vorletztes Listenelement (slicing)
```

- Listen sind mutierbar

```
my_list[1] = 42                     # Ersetze zweites Listenelement
my_list.append('new_element')       # Hänge String an Listenende an
my_list.extend(['new_element'])     # Hänge mehrere Elemente auf einmal an Listenende an
my_list.insert(2,[1,'b',False])     # Füge eine Liste an dritter Position in die Liste ein
```

list & tuple

- Listen sind iterierbar

```
for x in my_list:
    print(x)
```

```
for i in range(0,len(my_list)):
    print(my_list[i])
```

- List Comprehension: Eine einfache Syntax zum Erzeugen von Listen.

```
my_list = [1,2,3,4,5,6,7,8,9,10]

new_list = []
for x in my_list:
   if x%2==0:
      new_list.append(x)
```

```
new_list = [x for x in my_list if x%2==0]
```

- Alle Methoden bspw. unter https://www.w3schools.com/python/python_lists_methods.asp

list & tuple

EUROPÄISCHE UNION:
Investition in Ihre Zukunft
Europäischer Fonds für regionale Entwicklung

- Tupel sind ebenfalls geordnete Sammlungen von Objekten beliebiger Datentypen.
- Sie werden generiert, indem die Tupelelemente durch Kommata getrennt in () gesetzt werden, oder indem eine Liste oder Tupel an die Konstruktorfunktion `tuple()` übergeben wird:

```
my_tuple = ('Hello', True, 50, ['a',1,False])
my_tuple = tuple(['Hello', True, 50, list('a',1,False)])
```

- Tupel sind wie Listen indiziert und iterierbar
- Tupel sind immutierbar, d.h. nicht möglich ist

```
my_tuple[0] = 1      # Ändern von Elementen in Tupeln ist nicht möglich
```

- Eine Liste in einem Tupel hingegen ist mutierbar

```
my_tuple[3][1] = 2   # ändere zweites Element in Liste an vierter Stelle des Tupels
```

- Die einzigen Methoden von Tupeln sind

```
tuple.count(50)      # Zählt wie oft der Integer 50 im Tupel vorhanden ist
my_tuple.index(True) # Gibt den ersten Index zurück, an dem das Element True auftaucht
```

Dictionaries

EUROPÄISCHE UNION:
Investition in Ihre Zukunft
Europäischer Fonds für regionale Entwicklung

- Dictionaries sind eine geordnete Sammlungen von Objekten beliebiger Datentypen. Die Elemente werden als key:value-Paare gespeichert. Ein key ist immer einzigartig und muss von einem immutierbaren Datentyp sein.
- Dictionaries werden generiert, indem die key:value-Paare durch Kommata getrennt in {} gesetzt werden

```
my_dict = {'Marke':'Ford','PS':130,'Gewicht':1236.5, 42:4242}
```

- Elemente in dictionaries werden über ihren key referenziert:

```
my_dict ['Marke']                    # referenziere erstes Element
```

- Dictionaries sind mutierbar

```
my_dict ['Marke'] = 'Opel'           # ändere erstes Element
```

- Durch Referenzieren eines nicht vorhandenen keys wird ein neues Element hinzugefügt

```
my_dict ['Baujahr'] = 2011           # Füge dem Dictionary Element hinzu
```

Dictionaries

EUROPÄISCHE UNION:
Investition in Ihre Zukunft
Europäischer Fonds für regionale Entwicklung

- Dict-comprehension: Eine einfache Syntax zum Erzeugen von Dictionaries

```
new_dict = {x:x**2 for x in (2,4,6)}  # new_dict = {2:4,4:16,6:36}
```

- Die Methode `keys` gibt eine iterierbare Sammlung aller keys eines Dictionaries zurück:

```
for key in my_dict.keys():
    print(key)                # zeige alle keys
    print(my_dict[key])       # zeige alle Elemente
```

- Alle Methoden bspw. unter: https://www.w3schools.com/python/python_dictionaries_methods.asp

Funktionen

EUROPÄISCHE UNION:
Investition in Ihre Zukunft
Europäischer Fonds für regionale Entwicklung

- Eine Funktion ist Code, der nur ausgeführt wird, wenn er aufgerufen wird. Es ist möglich Daten an eine Funktion zu übergeben (Argumente) und eine Funktion kann Daten zurückgeben.
- Funktionen werden in Python mittels des Schlüsselwortes def definiert

```
def norm(x1,x2):
    '''Calculates the euclidian norm of two vectors x1
    and x2 passed as lists'''
    r = [(i-j)**2 for i,j in zip(x1,x2)]      # print(list(zip(x1,x2))) um zu sehen was zip() macht
    norm = sum(r)**(1/2)
    return norm
```

- Funktionen werden aufgerufen, indem die korrekte Anzahl an Argumenten in Klammern übergeben wird

```
q = norm(x1=[1,2,2],x2=[2,3,3])          # x1,x2 sind Parameter, die Variablen die bei der
                                         # Definition der Funktion verwendet werden
                                         # [1,2,2],[2,3,3] sind Argumente, die konkreten Werte
                                         # die beim Aufruf der Funktion übergeben werden
```

Funktionen

EUROPÄISCHE UNION:
Investition in Ihre Zukunft
Europäischer Fonds für regionale Entwicklung

- Bei der Definition von Funktionen können Standardwerte für die Argumente festgelegt werden

```
def norm(x1=[],x2=[]):
    ...
q = norm()                                  # q=0.0 da sum([])=0.0
```

*- und **-Operator

EUROPÄISCHE UNION:
Investition in Ihre Zukunft
Europäischer Fonds für regionale Entwicklung

- Ist die Anzahl der Argumente einer Funktionen unklar oder soll flexibel gehalten werden, können die *args und **kwargs Parameter bei der Funktionsdefinition genutzt werden.
- Der *-Operator und **-Operator können nur innerhalb eines Funktionsaufrufs verwendet werden und dienen zum Entpacken eines Tupels bzw. Dictionaries in Positional Arguments bzw. Keyword Arguments (Unpacking):

```
function(1,2,3) # Funktionsaufruf mit 3
                  Positional Arguments

ist äquivalent zu

values = (1,2,3)  # Definiere Tupel
function(*values) # Entpacke Tupel in
                  Funktionsaufruf  in
                  Positional Arguments
```

```
function(x=1,y=2,z=3) # Funktionsaufruf mit 3
                        Keyword  Arguments

ist äquivalent zu

values = {'x':1,'y':2,'z':3} # Definiere Dictionary
function(**values)       # Entpacke Dictionary in
                         Funktionsaufruf in Keyword
                         Arguments
```

*args

EUROPÄISCHE UNION:
Investition in Ihre Zukunft
Europäischer Fonds für regionale Entwicklung

- Möchte man, dass eine Funktion eine unbekannte Anzahl an Positional Arguments übergeben bekommen kann, nutzt man den *args Parameter bei der Funktionsdefinition
- Dieser Parameter sorgt dafür, dass alle nicht anderweitig definierten Positional Arguments beim Aufruf der Funktion in den Tupel args gepackt werden (Packing)

```
def calcsum(*args):                          # Definition der Funktion mit Argument *args
    s=0
    for arg in args:
        s = s + arg
    return s

calcsum(12,20)                               # Ausgabe: 32

values = (12,30,5,1)                         # Definiere Tupel
calcsum(*values)                             # Ausgabe: 38
```

**kwargs

EUROPÄISCHE UNION:
Investition in Ihre Zukunft
Europäischer Fonds für regionale Entwicklung

- Möchte man, dass eine Funktion eine unbekannte Anzahl an Keyword Arguments übergeben bekommen kann, nutzt man den **kwargs Parameter bei der Funktionsdefinition
- Dieser Parameter sorgt dafür, dass alle nicht anderweitig definierten Keyword Arguments beim Aufruf der Funktion in das Dictionary args gepackt warden (Packing)

```
def print_key_value(**kwargs):                    # Definition der Funktion mit Argument **kwargs
    for key, value in kwargs.items():
        print(key + ': ' + value)
    return None
print_key_value(first='Python', second='ist', third='toll' )  # Ausgabe: first: Python
                                                                      second: ist
                                                                      third: toll


arguments = {'first': 'Python', 'second': 'ist', 'third': 'toll’}     # Definiere Dictionary
print_key_value(**arguments)                                          # Ausgabe: first: Python
                                                                                second: ist
                                                                                third: toll
```

Lambda Funktionen

EUROPÄISCHE UNION:
Investition in Ihre Zukunft
Europäischer Fonds für regionale Entwicklung

- Lambda Funktionen, auch annonyme Funktionen (Funktionen ohne Namen) genannt, können mithilfe des lambda keywords erstellt werden.
- Lambda Funktionen ermöglichen eine kompaktere Definition von Funktionen.
- Beispiel zur Verdeutlichung des Unterschiedes zwischen normaler Funktion und lambda function:

```
def square(x):                          # Funktion die Quadrat einer Zahl zurückgibt
    return x**2
square(8)                               # Ausgabe: 64


square = lambda x: x**2                 # lambda Funktion die Quadrat einer Zahl zurückgibt
square(8)                               # Ausgabe: 64
```

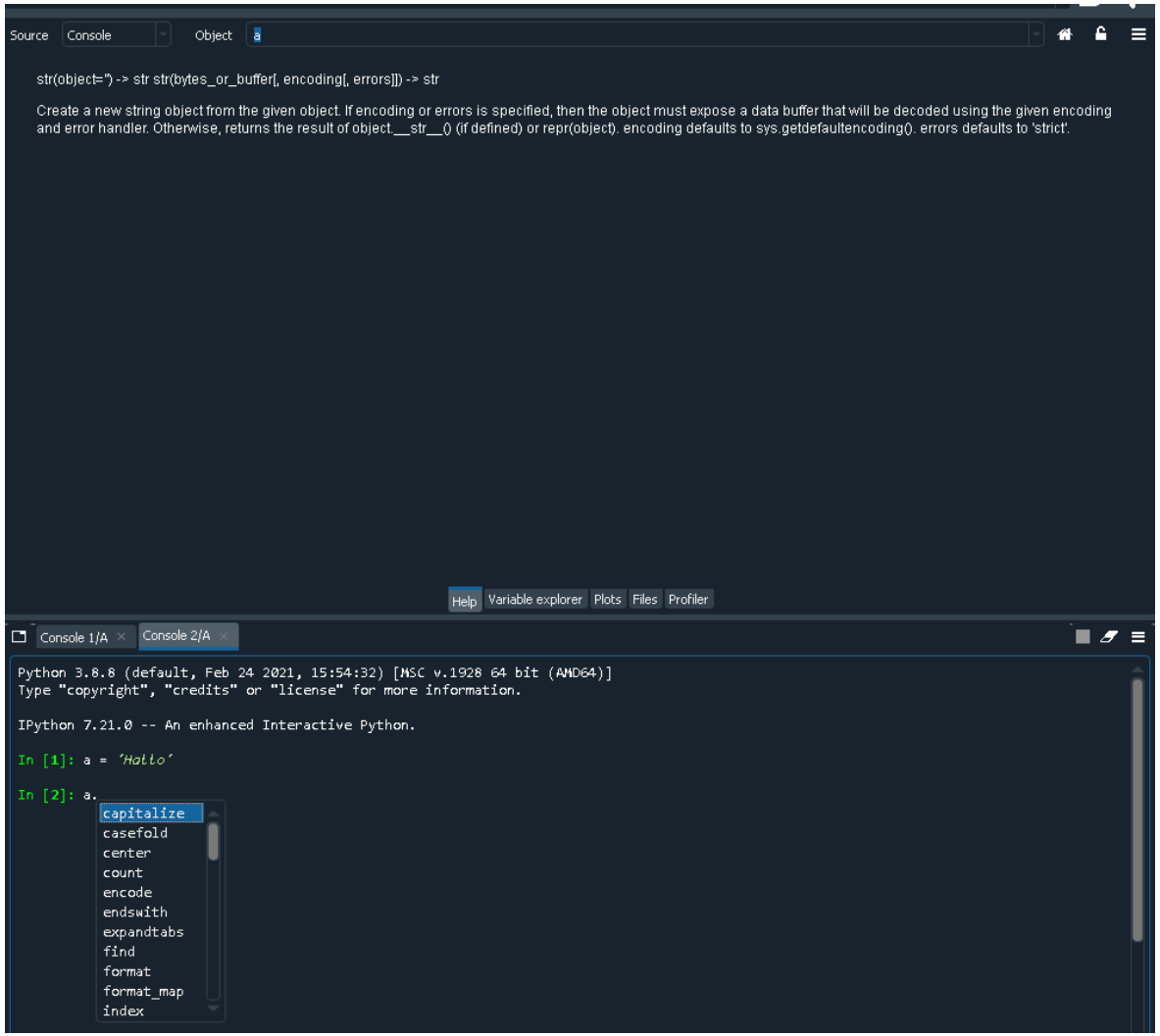
- Lambda Funktionen werden verwendet, falls eine Funktion nur einmal benötigt wird, oder um eine (Lambda) Funktion sofort einer anderen Funktion als Argument zu übergeben:

```
print('The square of 2 is ' + str((lambda x: x**2)(2))
```

Objekte

EUROPÄISCHE UNION:
Investition in Ihre Zukunft
Europäischer Fonds für regionale Entwicklung

- In Python ist alles ein Objekt:
  - Datentypen
  - Funktionen
  - Instanzen von Klassen
  - …
- Jedes Objekt hat
  - einen Typ
  - Methoden
  - Attribute
  - eine Dokumentation!



Oben: Dokumentation des Objektes a, Unten: Liste von Methoden des Objektes a

Klassen und Methoden

EUROPÄISCHE UNION:
Investition in Ihre Zukunft
Europäischer Fonds für regionale Entwicklung

- Klassen werden mittels des Schlüsselwortes `class` definiert. Klassennamen sollten mit einem Großbuchstaben beginnen.

```
class Tier:                                  # Erstellen der Klasse Tier, die als Vorlage für alle
     Beine = 4                               # Instanzen dient. Die Klasse enthält ein Attribut
                                             # (property) mit dem Namen Beine.


t1 = Tier()                                  # Erstellen einer Instanz t1 der Klasse Tier.

print(t1.Beine)                              # Gebe den Wert des Attributs Beine des Objekts t1
                                             # aus. Ausgabe: 4
```

- Methoden sind Funktionen, die innerhalb einer Klasse definiert werden und auf die über eine Referenz auf ein Objekt dieser Klasse zugegriffen werden kann
- Für den Zugriff auf ein Attribut oder eine Methode einer Klasse wird die Punkt-Notation verwendet.

```
class Tier:
     Beine = 4


     def printhello(self,name):              # Erstellen der Methode printhello, in der Klasse
              self.name = name               # Tier. Self referenziert die aktuelle Klasse

              print('My name is ' + self.name)

t1 = Tier()                                  # Aufruf der Methode printhello mit einem Argument
t1.printhello('Bello')                       # Ausgabe: My name is Bello.
```

Klassen und Methoden

EUROPÄISCHE UNION:
Investition in Ihre Zukunft
Europäischer Fonds für regionale Entwicklung

- Die __init__ Methode (Initialisierungsmethode) wird automatisch aufgerufen, sobald ein Objekt einer Klasse instanziiert wird.

```
class Tier():                                   # Die Init Methode akzeptiert name und age als
        legs = 4                                # Übergabeargumente und weist diese den Attributen zu,

        def __init__(self, name, age):          # die dann über das Objekt aufrufbar sind.
           self.name = name
           self.age  = age

        def geräusch(self):                     # Hinzufügen einer Methode, die ein zu erwartendes
           print(self.name +' bellt o. miaut')  # Geräusch ausgibt.


meinhund = Tier('Bello',8)                      # Erstellen einer Instanz meinhund der Klasse Tier
                                                # mit den Argumenten 'Bello' und 8.
meinhund.legs                                   # Zugriff auf Attribute und Methoden über
meinhund.age                                    # Punktnotation

meinhund.geräusch()                             # Ausgabe: Bello bellt o. miaut.

meinhund.legs = 3                               # Ändern von Attributen per Punktnotation
```

Klassen und Methoden - Vererbung

EUROPÄISCHE UNION:
Investition in Ihre Zukunft
Europäischer Fonds für regionale Entwicklung

- Vererbung erlaubt es, neue Klassen von bestehenden abzuleiten. Dies bietet sich an, falls unterschiedliche Klassen gleiche Attribute oder Methoden verwenden. So müssen diese nur einmalig in einer Oberklasse definiert werden und können dann an alle Unterklassen vererbt werden.

```
class Hund(Tier):                                   # Erstellen der Klasse Hund, welche die Eigenschaften
        def __init__(self, name, age):              # und Methoden der Oberklasse Tier erbt.
            super().__init__(name, age)
                                                    # Dabei ist es möglich, einzelne Attribute o. Methoden
        def geräusch(self):                         # zu überschreiben. Überschreiben der Methode geräusch
            print(self.name +' bellt')              # für die Klasse Hund.
```

- Methoden und Attribute der Oberklasse können überschrieben werden. Überschreiben von Methoden oder Attributen kann nützlich sein, wenn sich viele Unterklassen eine Eigenschaft teilen, es aber Einzelne gibt, die sich anders verhalten.

```
class Katze(Tier):
        def __init__(self, name, age):
        super().__init__(name, age)

        def geräusch(self):
            print(self.name + ' miaut')
```

```
meinhund = Hund('Bello',8)                          # Erstellung der Instanz mein Hund. Dabei wird sowohl
meinkatze = Katze('Tiger',8)                                # eine Instanz der Klasse Tier erstellt (über den
                                                    # Auruf von super__init__, als auch eine Instanz der
                                                    # Klasse Hund, welche dieser erbt.

meinhund.name                                               # Ausgabe: Bello
meinhund.geräusch()                                         # Ausgabe: Bello bellt
meinekatze.geräusch()                                       # Ausgabe: Tiger miaut
```

Flow Control - while

EUROPÄISCHE UNION:
Investition in Ihre Zukunft
Europäischer Fonds für regionale Entwicklung

- Eine while-Schleife erlaubt es eine Gruppe von Anweisungen (einen Block) solange auszuführen wie eine Bedingung wahr ist.
- Beispiel while-Schleife:

```
i = 1                          # Zuweisen eine Startwerts
while i < 3:                   # Durchlaufe die Schleife solange i<3 ist.
   print(i)                    # Ausgabe: 1 (erster Durchlauf), 2 (zweiter Durchlauf)
   i = i+1                     # Erhöht den Wert von i bei jedem Durchlauf um 1
```

- Der Indexvariable muss vor der Schleife ein Startwert zugewiesen werden.
- Vergißt man den innerhalb der Schleife den Wert der Variable zu verändern, erschafft man eine Endlosschleife.

Flow Control - for

EUROPÄISCHE UNION:
Investition in Ihre Zukunft
Europäischer Fonds für regionale Entwicklung

- Eine for-Schleife erlaubt es eine Gruppe von Anweisungen (einen Block) wiederholt auszuführen. Dafür wird über eine Sequenz/einen Bereich iteriert. Verwendet werden können dafür z.B. Listen, Tupel und Dictionaries.
- Beispiele for-Schleife:

```
my_list = [1,2,3,4]                     # Erstellen einer Liste mit 4 Elementen

for x in my_list:                       # Iterieren über alle Elemente in der Liste(my_list)
   print(x)                             # Ausgabe: 1 2 3 4 (jeweils in neuer Zeile)


for i in range(0,len(my_list)-2):       # Iterieren beginnt mit dem ersten Element und stoppt
   print(my_list[i])                    # vor dem drittem (len(my_list)=4 => 4-2 =2
```

Flow Control - if/else

EUROPÄISCHE UNION:
Investition in Ihre Zukunft
Europäischer Fonds für regionale Entwicklung

- Eine if-Anweisung erlaubt es eine Gruppe von Anweisungen nur unter bestimmten Bedingungen auszuführen.

```
x = 5                                   # x wird 5 zugewiesen

if x == 5:                              # Falls die Bedingung zutriff, wird der eingerückte
    print('x ist 5.')                   # Code ausgeführt


print('Ende')                           # Trifft die Bedingung nicht zu wird zur nächsten nicht
                                        # eingerückten Zeile gesprungen
```

- Mithilfe des keywords elif können Verkettungen von if Anweisungen effizienter geschrieben werden

```
x = 5

if x == 5:                              # Der Code nach der ersten if Anweisung wird nur
    print('x ist 5.')                   # ausgeführt, wenn die Bedingung nicht erfüllt ist.
elif x == 6:                            # Mit elif können beliebig viele Bedingungen abgefragt
    print('x ist 5.')                   # werden.
print('Ende')
```

Flow Control – try/except

EUROPÄISCHE UNION:
Investition in Ihre Zukunft
Europäischer Fonds für regionale Entwicklung

- Tritt bei der Ausführung des Codes ein Fehler (exception) auf, wird dessen Ausführung beendet und Python gibt eine Fehlermeldung aus
- Der Programmabbruch ist üblicherweise unerwünscht, stattdessen möchte der Nutzer Fehler systematisch abfangen und behandeln
- Dies wird durch `try`, `except` und `finally` ermöglicht

```
try:                                    # Versuche den folgenden (eingerückten) Code auszuführen
   print(x)                             # x ist nicht definiert, Ausführung generiert einen Fehler
except:                                 # Falls ein Fehler auftritt, führen den folgenden Code aus
   print('Ein Fehler ist aufgetreten')
finally:                                # Unabhängig von der Evaluation der try und except Blöcke, wird
   print('Ende')                        # der finally-Block immer ausgeführt
```

Flow Control – try/except

EUROPÄISCHE UNION:
Investition in Ihre Zukunft
Europäischer Fonds für regionale Entwicklung

- Falls die Ausführung des try-Blocks zu einem Fehler führt, wird eine Exception ausgegeben
- Eine Exception ist ein Python Objekt und verfügt somit über einen Typ. Für verschiedene Arten von Fehler existieren verschiedene Typen von Exceptions. Der Nutzer kann prüfen, um welche Art Exception es sich handelt, um verschiedene Exceptions unterschiedlich zu behandeln

```
try:                                    # Versuche den folgenden (eingerückten) Code auszuführen
   print(x)                             # x ist nicht definiert, Ausführung generiert einen Fehler

except NameError:                       # Führe den folgenden Code nur aus, falls Exception vom Typ
   print('NameError')                   # NameError ist

except:                                 # Führe den folgenden Code unabhängig vom Fehlertyp aus
   print('Different Error’)
```

- Eine Liste aller Built-in Exceptions gibt es hier: https://docs.python.org/2/library/exceptions.html

Nutzung von Bibliotheken

EUROPÄISCHE UNION:
Investition in Ihre Zukunft
Europäischer Fonds für regionale Entwicklung

- Eine Python-Biobliothek oder Python-Package sind mehrere Python-Dateien, welche Datentypen, Klassen, Methoden und Funktionen enthalten können
- Um diese nutzen zu können, muss das Package vom Nutzer importiert werden

```
import numpy                          # Importiere das numpy-Package


A = numpy.array((4,5))                # Nutze das Array-Objekt des numpy-Packages
                                      # um eine 4x5 Matrix zu erstellen
```

- Aus Platzgründen und zugunsten der Lesbarkeit können Packages beim Importieren umbenannt werden

```
import numpy  as np                   # Importiere das numpy-Package


A = np.array((4,5))
```

- Alle Methoden und Attribute eines Packages (trifft auf alle Objekte zu) können mittels dir() angezeigt werden

```
dir(np)              # Gibt alle Methoden, Objekte, Attribute und Funktionen des numpy-Packages aus

dir(A)               # Gibt alle Methoden und Attribute des Array-Objektes A aus
```