Werkzeuge für das wissenschaftliche Arbeiten

Python for Machine Learning and Data Science

Magnus Bender
bender@ifis.uni-luebeck.de
Wintersemester 2023/24

Inhaltsübersicht

1. Programmiersprache Python
   a) **Einführung, Erste Schritte**
   b) Grundlagen
   c) Fortgeschritten
2. Auszeichnungssprachen
   a) LaTeX, Markdown
3. Benutzeroberflächen und Entwicklungsumgebungen
   a) Jupyter Notebooks lokal und in der Cloud (Google Colab)
4. Versionsverwaltung
   a) Git, GitHub
5. Wissenschaftliches Rechnen
   a) NumPy, SciPy
6. Datenverarbeitung und -visualisierung
   a) Pandas, matplotlib, NLTK
7. Machine Learning (scikit-learn)
   a) Grundlegende Ansätze (Datensätze, Auswertung)
   b) Einfache Verfahren (Clustering, …)
8. DeepLearning
   a) TensorFlow, PyTorch, HuggingFace Transformers

Themen

I. Einführung
   - Organisatorisches
   - Projektaufgaben
   - Abschlussquiz

II. Erste Schritte
   - Installation
   - Das erste *Skript*
   - Kontrollstrukturen
   - Datentypen
   - Operatoren

Organisatorisches

- Leistungszertifikat Typ B mit 2 KP
  - Projektaufgaben & Abschlussquiz
  - Keine Klausur
- Teil des fächerübergreifend Wahlpflichtkataloges
  - Pflicht Informatik
  - Wahlpflicht u.a. Medizinische Informatik, Medieninformatik

- Dienstags 14 - 16 Uhr
- ***Vorlesungstermine*** im Seminarraum Informatik 5 (von Neumann)
- ***Übungstermine*** im PC Pool Geb. 64

Siehe Semesterübersicht

Voraussetzungen

- Keine!
- Aber, hilfreich sind
  - Logisches Denken
  - Mathematisches Verständnis
  - Programmiererfahrung

- Anwendung von Inhalten aus
  - Einführung in die Programmierung
  - Algorithmen und Datenstrukturen
  - Einführung in Web und Data Science
  - Einführung in die Logik
  - Lineare Algebra und Diskrete Strukturen

Projektaufgaben

- Fünf Projektaufgaben über das Semester verteilt
  - Fünfte Projektaufgaben ist Bonusaufgabe (Ausgleich einer Projektaufgabe)
- Projektaufgaben (und Modul) sind bestanden, wenn vier Projektaufgaben und Abschlussquiz erfolgreich bearbeitet

Aufgabe 1, 2, 4

- VPL im Moodle
- Alle Testfälle & volle Punktzahl → erfolgreich bearbeitet

Aufgabe 3

- Git und LaTeX
- Lösung in Git-Repository *hochladen*

Aufgabe 5 (Bonus)

- Machine (& Deep) Learning mit Python
- Abgabe im Moodle

Abschlussquiz

- Nach der letzten Vorlesung
  - Voraussichtlich 6.2. bis 9.2. je 18 Uhr
- E-Test im Moodle
- Zwei Versuche mit je 2 Stunden Bearbeitungszeit → Versuch mit den meisten Punkten *zählt*
- 50 % der Punkte müssen erreicht werden

- Thematisch über die Inhalte des Moduls, Fokus auf Themenbereiche
  - Machine Learning
  - Deep Learning
- Modul ist bestanden, wenn vier Projektaufgaben und Abschlussquiz erfolgreich bearbeitet

Semesterübersicht

KW	Dienstag	Donnerstag	Freitag
42	Vorlesungstermin Seminarraum 5	Veröffentlichung Projektaufgabe 1	
43	Vorlesungstermin Seminarraum 5		
44	**Feiertag**		
45	Vorlesungstermin Seminarraum 5	Veröffentlichung Projektaufgabe 2	
46	Übungstermin PC Pool		Abgabe Projektaufgabe 1
47	Vorlesungstermin Seminarraum 5		
48	Vorlesungstermin Seminarraum 5		
49	Vorlesungstermin Seminarraum 5	Veröffentlichung Projektaufgabe 3	

KW	Dienstag	Donnerstag	Freitag
50	Übungstermin PC Pool		Abgabe Projektaufgabe 2
51	Vorlesungstermin Seminarraum 5		
52	**Weihnachten & Neujahr**		
1			
2	Vorlesungstermin Seminarraum 5	Veröffentlichung Projektaufgabe 4	Abgabe Projektaufgabe 3
3	Vorlesungstermin Seminarraum 5		
4	Vorlesungstermin Seminarraum 5	Veröffentlichung Projaufg. 5 (Bonus)	
5	Vorlesungstermin Seminarraum 5		Abgabe Projektaufgabe 4
6	Vorlesungstermin Seminarraum 5	Abschlussquiz	Abschlussquiz
			Abgabe Projektaufgabe 5

II.
Erste Schritte mit Python

Aber zuerst:

Fragen zum Organisatorischen?

Installation

- Download unter https://www.python.org/downloads/
  - Vorinstalliert auf macOS und den meisten Linux Distributionen
    - Terminal `python3`

python gibt es nicht (mehr) oder es startet Python 2
Wir wollen Python 3!

- Weitere Installationsmöglichkeiten im späteren Verlauf des Moduls
  - Virtuelle Umgebungen, um verschiedene Abhängigkeiten und Pakete auf dem gleichen Computer zu installieren

Informationsquellen

- Offizielle Dokumentation
  - https://docs.python.org/3/
  - Z.B. Listen https://docs.python.org/3/tutorial/datastructures.html#more-on-lists
  - *Leider oft etwas schwierig das Gesuchte zu finden*
- Eine weitere Dokumentation
  - https://devdocs.io/python~3.11/
- Tutorials und Beispiele
  - Z.B. https://www.w3schools.com/python/

Python

Code wird nicht kompiliert, sondern vom (Python) *Interpreter* gelesen und direkt ausgeführt.

Einfach `python3` im Terminal eingeben und Python-Code kann eingegeben werden, `exit()` zum Verlassen.

- Interpretiert
- Interaktives Terminal
- Objektorientiert
- Multi-Paradigma
- Einrückung *zählt*
- 0-indiziert

- Einfach zu lernen
- Schneller Einstieg
- Viele Pakete für Data Science
- Open Source

Kästen dieser Art dienen als Hinweise, in den PDFs und beim Nacharbeiten der Unterlagen.

s erste Skript

Programmcode kann direkt in das Skript geschrieben werden

Laden einer (Standard-)Bibliothek

```
import time

timestamp = time.time()
print(timestamp, "Sekunden seit 1.1.1970")

timestamp = int(timestamp)
print(timestamp, "Sekunden seit 1.1.1970")

if timestamp % 60 == 0:
  print("Eine volle Minute")
else:
  print("Sekunde {sec}".format(
    sec=timestamp % 60
  ))
```

Typenumwandlung

Aufruf im Terminal

```
$> python3 name.py

1665149962.844817 Sekunden seit 1.1.1970
1665149962 Sekunden seit 1.1.1970
Sekunde 22

$> python3 name.py

1665150180.510071 Sekunden seit 1.1.1970
1665150180 Sekunden seit 1.1.1970
Eine volle Minute
```

If-Else Kontrollstruktur, auf : achten

Ausgabe auf Terminal, mit , trennen

If-Else – Details

`elif` statt `else if` für weitere Fälle

```
duration = 60

if duration < 0:
  print("Zeit kann nicht negativ sein!")
elif duration == 60:
  print("Genau eine Minute.")
elif duration > 60:
  print("Mehr als eine Minute.")
else:
  print("Es hat "+ str(duration) +"Sekunden gedauert.")

message = "Das war " + ( "schnell" if duration < 30 else "leider zu langsam" ) + "!"
print(message)
```

```
$> python3 name.py

Genau eine Minute.
Das war leider zu langsam!
```

Auch als Ausdruck möglich, *Inline-If*

„Rückgabe wahr `if` Bedingung `else` Rückgabe falsch“

Warum brauchen wir `str()` hier?

Warum brauchen wir die Klammern hier?

Schleifen und Funktionen

Funktionsdefinition

Typ einer Variable

For-Schleife

Funktionsaufruf

Standardeingabe

```
import sys

def process_line(s):
  print(type(s))
  print(s)

for line in sys.stdin:
  process_line(line)
```

Standardeingabe „befüllen“

```
$> echo "a\nb" | python3 name.py
<class 'str'>
a

<class 'str'>
b
```

Wo kommen die Leerzeilen her?

Schleifen – Details

Erstellen und ausgeben einer Liste

```
values = [1, 2, 3, 4]
print(values)

for v in values:
  print(v)

values2 = [ v*2 for v in values ]
print(values2)


while len(values2) > 0:
  print(values2.pop())

print(values2)
```

Inline-Schleife, gen. *List Comprehensions*

While-Schleife

Letztes Element aus Liste entfernen, analog zum Stack

```
$> python3 name.py

[1, 2, 3, 4]
1
2
3
4
[2, 4, 6, 8]
8
6
4
2
[]
```

Funktionen – Details

Definition: `Name(Parameterliste):`

```
# print(add_or_multiply(1,2))

def add_or_multiply(x, y, add=True):

  if add:
    return x + y
  else:
    return x * y

print(add_or_multiply(1, 2))
print(add_or_multiply(1, 2, False))

print(add_or_multiply(x=5, y=6, add=True))
print(add_or_multiply(x=5, add=False, y=6))

add_or_multiply = "Hallo"
add_or_multiply(1, 2)
```

Was passiert, wenn man hier die # entfernt?

Default-Parameter: `Name=Wert`

Rückgabe

```
$> python3 name.py
3
2
11
30

Traceback:
  File "name.py", line 19, in <module>
    add_or_multiply(1,2)
TypeError: 'str' object is not callable
```

Man kann übrigens Funktionen in Funktionen definieren, diese sind dann nur in der Elternfunktion verfügbar.

Aufruf mit Werte in Reihenfolge

Aufruf mit Namen und Werte, Reihenfolge dann egal

Funktionsnamen sind selbst Variablen

Datentypen

- Wahr, Falsch und Null `True, False, None`
- Zahlen (`int` und `float`) `12, 12.5, 12e3, -20`
- Zeichenketten `"Hallo Welt", 'Hallo Welt'`
- Tupel `(1, 2, 3, 4), ("A", 2, "C", None), tuple("ABCD")`
- Listen `[1, 2, 3, 4], ["A", 2, "C", None], list((1, 2, 3))`
- Mengen `{"a", "b"}, {"a", "a", "b"}, set(("a", "b", "b"))`
- Wörterbücher `{"a" : 1, "b" : 2}, dict((("a", 1 ), ("b", 2)))`
- Klassen (eigene Typen), weite Typen … → Später im Modul

GROSS geschrieben!

Drei Mal dieselbe Menge!

Zei

Zugriff auf Buchstaben und Teilzeichenketten, *Slicing*

```
s = "Hallo Welt "

print(s[0])
print(s[:-2])
print(s[1:3])
```

```
H
Hallo Wel
al
```

Zeichenkettenfunktionen, geben neue Zeichenk. zurück

```
print(s.strip() + "!")
print(s.lower())
print(s.replace("e", "or").replace("a", "e").replace("t", "d"))
```

```
Hallo Welt!
hallo welt
Hello World
```

Einfache und Doppelt Anführungszeichen machen keinen Unterschied

```
print(s == 'Hallo Welt ')
```

```
True
```

```
s += "!"
print(s * 2)
print("Welt" in s)
```

```
Hallo Welt !Hallo Welt !
True
```

```
print(s.split())
print('-'.join(["Hallo", "Welt!"]))
```

```
['Hallo', 'Welt', '!']
Hallo-Welt!
```

```
print("Hallo {du}, mein Name ist {ich}".format(du="A", ich="M"))
```

```
Hallo A, mein Name ist M
```

Tupel – Details

Runde Klammern kann man weglassen

```
tup1 = (1, 2, 3)
tup2 = 5, 6, 7

print(tup1[0])
print(tup2)

a, b = "A", "B"
print(a, b)

for (i, j) in ((1,"a"), (2, "b"), (3, "c")):
  print(i, j)
```

Tupel kann man auch bei Zuweisungen benutzen

```
$> python3 name.py

1
(5, 6, 7)

A B

1 a
2 b
```

Tupel können in Schleifen „entpackt“ werden.

Tupel sind nicht veränder- oder erweiterbar!

Diese Folien stellen *längst* nicht alle unterstützen Operationen und Funktionen dar!

Listen – Details

```
lis1 = list((1, 2, 3))
lis2 = [5, 6, 7]

print(lis1[:-1])
print(lis1 + lis2)

lis1.append(False)
lis1.extend(lis2)
print(lis1)

print(sorted(lis1),
  lis1.sort(), lis1)

for i, v in enumerate(lis2):
  print(i, v)

lis3 = [i for i in range(10)]
print(lis3)
```

Wo liegt der Unterschied?

Wo liegt der Unterschied?

```
$> python3 name.py

[1, 2]
[1, 2, 3, 5, 6, 7]

[1, 2, 3, False, 5, 6, 7]

[False, 1, 2, 3, 5, 6, 7]
   None [False, 1, 2, 3, 5, 6, 7]

0 5
1 6
2 7

[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Wörterbücher – Details

```python
dic = {"a" : 1, "b" : 2}

print(dic["a"])
dic["c"] = 3
print(dic)

del dic["b"]
print("b" in dic, "b" not in dic)

for k in dic: # dic.keys()
  print(k)

for v in dic.values():
  print(v)

for k, v in dic.items():
  print(k, v)
```

Wert unter Schlüssel ändern, hinzufügen, abfragen

Iteration und Prüfung ob enthalten auf Schlüssel

```
$> python3 name.py

1
{'a': 1, 'b': 2, 'c': 3}

False True

a
c

1
3

a 1
c 3
```

Operatoren

Vergleichend	==	Gleichheit
	> und <	Größer und kleiner
	>= und <=	Größer und kleiner gleich
Logisch	`not`	Negation
	`and`	Und
	`or`	Oder
Mathematisch	* und **	Multiplikation und Potenz
	/ und //	Division und Ganzzahldivision
	+ und -	Addition und Subtraktion
	%	Modulo/ Rest
	`in`	Prüfung ob enthalten

=	Zuweisung
+=	Addition und Zuweisung
-=	Subtraktion und Zuweisung
/=	Division und Zuweisung

- Dies ist nur eine kleine Auswahl
- Operatoren sind für den jeweiligen Datentypen definiert
- Klassen können die Operatoren spezifisch belegen

Klassenspezifische Operatoren – Beispiel

```python
s1 = {1, 2, 3}
s2 = {2, 3, 4}

print(
  s1 - s2,
  s1.difference(s2)
)
print(
  s1 & s2,
  s1.intersection(s2)
)
print(
  s1 | s2,
  s1.union(s2)
)

s1 |= {4}
print(s1)
```

```
$> python3 name.py

{1}
{1}

{2, 3}
{2, 3}

{1, 2, 3, 4}
{1, 2, 3, 4}

{1, 2, 3, 4}
```

Nützliche Funktionen

Zeichenketten	`s.strip()`	Entfernt Whitespace (Leerzeichen) am Anfang und Ende einer Zeichenkette.
	`s.lower()`	Übersetzt in einer Zeichenkette alle Zeichen in ihre kleingeschriebene Version.
	`s.replace(x, y)`	Ersetzt alle Vorkommen von x mit y in einer Zeichenkette.
	`s.split(x)`	Teilt eine Zeichenkette bei jedem Vorkommen von x auf und erstellt eine Liste.
	`s.join(x)`	Fügt die Element
Listen	`l.append(x)`	Fügt ein neues El
Wörterbücher	`d.items()`	Iteriert über alle E
	`d.values()`	Iteriert über alle V
Iteration	`enumerate(l)`	Enumeriert alle E
	`zip(l1, l2)`	Iteriert über zwei nmen aus.
	`range(x)`	Erlaubt die Iterati
Typen	`str(x)`	Wandelt x in eine
	`int(x)`	Wandelt x in ei anzzahl um (rundet dabei ab).
	`float(x)`	Wandelt x i eine Fließkommazahl um.
	`type(x)`	Besti mt den Typ von x aus.
Allgemein	`print(x)`	Gibt x aus.
	`eval(s)`	Führt den Inhalt einer Zeichenkette s als Python-Code aus.
	`open(f, r)`	Öffnet eine Datei f mit der Berechtigung r („r“ für lesen, „w“ für schreiben).
	`len(x)`	Bestimmt die Länge von x.

„Eval is Evil!“

- Die Eingabe s muss aus sicherer Quelle stammen!
- Es gibt Fälle, wo `eval()` sehr praktisch ist, um Ausdrücke direkt auszuwerten, z.B.

```
eval("True and (False or not False)")
```

Ein Beispiel

CSV Datei einlesen, zeilenweise alle Zahlen rausfiltern und nur die Zahlen wieder als CSV ausgeben.

```
$> python3 name.py

144,4182025
169,32364721
7921,1489496836
81,8008900
```

```python
def extract_numbers(l):
  l = l.strip()
  numbers = []
  for p in l.split(","):
    if p.strip().isnumeric():
      numbers.append(int(p))
  return numbers

def build_csv(nl):
  csv = ""
  for line in nl:
    csv += ','.join([
      str(n) for n in line
    ]) + "\n"
  return csv
```

```python
f = open("name.csv", "r")
lines = f.readlines()
f.close()

new_lines = []
for line in lines:
  numbers = extract_numbers(line)
  new_lines.append([n ** 2 for n in numbers])

print(build_csv(new_lines))
```

Zusammenfassung

- 4 (5 mit Bonus) Projektaufgaben & Abschlussquiz im Semester
- Installation & erste Skripte
- Schleifen & Funktionen
- Datentypen
- Operatoren
- Nützliche Funktionen

Die 1. Aufgabe erscheint am Donnerstag im Moodle!

Nächste Woche stelle ich die Aufgabe auch einmal vor und wir klären Fragen dazu.

Inhaltsübersicht

1. Programmiersprache Python
   a) *Einführung, Erste Schritte*
   b) Grundlagen
   c) Fortgeschritten
2. Auszeichnungssprachen
   a) LaTeX, Markdown
3. Benutzeroberflächen und Entwicklungsumgebungen
   a) Jupyter Notebooks lokal und in der Cloud (Google Colab)
4. Versionsverwaltung
   a) Git, GitHub
5. Wissenschaftliches Rechnen
   a) NumPy, SciPy
6. Datenverarbeitung und -visualisierung
   a) Pandas, matplotlib, NLTK
7. Machine Learning (scikit-learn)
   a) Grundlegende Ansätze (Datensätze, Auswertung)
   b) Einfache Verfahren (Clustering, …)
8. DeepLearning
   a) TensorFlow, PyTorch, HuggingFace Transformers