Einführung in Python

Christian Wilms

Computer Vision Group
Universität Hamburg

Wintersemester 2023/24

auf Basis des offiziellen Python-Tutorials

01. November 2023

Gliederung

1. Einleitung
2. Basistypen
3. Datenstrukturen
4. Kontrollstrukturen
5. Dies und Das
6. Python programmieren
7. Zusammenfassung

Übersicht

1 Einleitung

2 Basistypen

3 Datenstrukturen

4 Kontrollstrukturen

5 Dies und Das

6 Python programmieren

7 Zusammenfassung

Motivation Python

Warum Python?

- sehr flexible
- leicht zu schreiben/lesen
- riesiger Support

Motivation Python

Warum Python?

- sehr flexible
- leicht zu schreiben/lesen
- riesiger Support

- interpretierte Sprache (langsamer)
- Skripte vs. Notebook
- dynamisches Typsystem
- automatisches Speichermanagement
- einfache Einbindung von (schnellem) C/C++-Code

Wie bekomme ich Python?

Anaconda

- Python Distribution mit vielen Features
- Paketverwaltung, IDE, Notebooks,... vorhanden

Wie bekomme ich Python?

Anaconda

- Python Distribution mit vielen Features
- Paketverwaltung, IDE, Notebooks,... vorhanden

Online Notebook Editoren

- Google Colab
- Datalore (Kollaboration in Echtzeit möglich)
- CoCalc
- ...

Wie bekomme ich Python?

Anaconda

- Python Distribution mit vielen Features
- Paketverwaltung, IDE, Notebooks,... vorhanden

Online Notebook Editoren

- Google Colab
- Datalore (Kollaboration in Echtzeit möglich)
- CoCalc
- ...

DIY

Ihr installiert einfach alles selbst.

Übersicht

1 Einleitung

2 Basistypen

3 Datenstrukturen

4 Kontrollstrukturen

5 Dies und Das

6 Python programmieren

7 Zusammenfassung

Wahrheitswerte

Literale für Falsch `False`, `0`

Literale für Wahr `True`, `1`

Operatoren für 'Booleans' `not`, `and`, `or`, mathematische Operatoren

Wahrheitswerte

Literale für Falsch `False`, `0`

Literale für Wahr `True`, `1`

Operatoren für 'Booleans' `not`, `and`, `or`, mathematische Operatoren

```
>>> True and False
False
>>> not True
False
>>> False == 0
True
```

Zahlen

Datentypen für Zahlen `int`, `float`, ...

Operatoren für Zahlen `+`, `-`, `*`, `/`, `==`, `<=`, `<`, ...

Das `math`-Modul bietet viele weitere Funktionen.

Zahlen

Datentypen für Zahlen `int`, `float`, ...

Operatoren für Zahlen `+`, `-`, `*`, `/`, `==`, `<=`, `<`, ...

Das `math`-Modul bietet viele weitere Funktionen.

```
>>> 2 + 2
4
>>> (50 - 5.0*6) / 4
5.0
>>> 5**2  # 5 squared
25
>>> float(5)
5.0
```

Strings

- Literale mit einfachen oder doppelten Anführungszeichen

Strings

- Literale mit einfachen oder doppelten Anführungszeichen

```
>>> 'hello world' == "hello world"
True
>>> 'hello ' + 'world'
'hello world'
```

Übersicht

1 Einleitung

2 Basistypen

3 Datenstrukturen

4 Kontrollstrukturen

5 Dies und Das

6 Python programmieren

7 Zusammenfassung

Listen - Erzeugung I

- einfacher Container, der sehr oft genutzt wird
- kein Elementtyp, es können Äpfel und Birnen in einer Liste sein
- Notation mit eckigen Klammern

Listen - Erzeugung I

- einfacher Container, der sehr oft genutzt wird
- kein Elementtyp, es können Äpfel und Birnen in einer Liste sein
- Notation mit eckigen Klammern

```
>>> []
[]
>>> [0,1,2]
[0, 1, 2]
>>> [0]*3 #Liste mit drei Nullen
[0, 0, 0]
```

Listen - Erzeugung II

- Funktion `range`: Listen von ganzen Zahlen erzeugen
- `list(range(stop))`
- `list(range(start,stop[,step]))`

Listen - Erzeugung II

- Funktion `range`: Listen von ganzen Zahlen erzeugen
- `list(range(stop))`
- `list(range(start,stop[,step]))`

```
>>> list(range(3)) #Liste mit drei Elementen
[0, 1, 2]
>>> list(range(3,7)) #Liste von 3 bis vor 7
[3, 4, 5, 6]
>>> list(range(3,7,2)) #jedes zweite Element von 3
    bis vor 7
[3, 5]
```

Listen - Zugriff

```
>>> l = [1,2,3,4]
>>> l[1] #Index startet bei 0
2
>>> l[-1] #mit einem Minus kann man die Liste von
    hinten aufrollen
4
>>> l[0:2] #von Index 0 bis vor Index 2
[1, 2]
>>> l[:2] #wie l[0:2]
[1, 2]
>>> l[::2] #jedes zweite Element
[1, 3]
>>> l[::-1] #Liste umdrehen
[4, 3, 2, 1]
```

Listen - Operationen I

```
>>> l
[1, 2, 3, 4]
>>> l[1] = 10 #veraendert einzelnes Element
>>> l
[1, 10, 3, 4]
>>> l.append(2) #fuegt ein Element hinzu
>>> l
[1, 10, 3, 4, 2]
>>> l.extend([5,4]) #fuegt eine Liste an
>>> l
[1, 10, 3, 4, 2, 5, 4]
```

Listen - Operationen II

```
>>> l
[1, 10, 3, 4, 2, 5, 4]
>>> l.index(10) #Index des Elements '10'
1
>>> l.pop(2) #entfernt Element an Pos. 2
10
>>> l.sort() #sortiert die Liste in-place
>>> len(l) #gibt die Laenge der Liste
5
```

Alles Weitere in der Dokumentation ↗ !

Dictionaries (Map)

- bildet eindeutige Schlüssel auf Elemente ab
- Notation mit geschweiften Klammern

Dictionaries (Map)

- bildet eindeutige Schlüssel auf Elemente ab
- Notation mit geschweiften Klammern

```
>>> d = {1:'a', 2:'b', 3:'c'}
>>> 1 in d
True
>>> d[1] #liest den Wert zum Schluessel 1 aus
'a'
>>> d[4]='d' #fuegt ein neuen Paar hinzu
>>> d
{1: 'a', 2: 'b', 3: 'c', 4: 'd'}
```

Alles Weitere in der Dokumentation ↗ !

Übersicht

1 Einleitung

2 Basistypen

3 Datenstrukturen

4 Kontrollstrukturen

5 Dies und Das

6 Python programmieren

7 Zusammenfassung

Funktionen

- statt geschweifter Klammern werden Doppelpunkt und Einrückung genutzt
- keine Typen an den formalen Parametern
- Vorfestlegung von Parametern auf Default-Werte möglich

Funktionen

- statt geschweifter Klammern werden Doppelpunkt und Einrückung genutzt
- keine Typen an den formalen Parametern
- Vorfestlegung von Parametern auf Default-Werte möglich

```
>>> def summe(a,b):
...     return a + b
...
>>> summe(5,9)
14
>>> summe('hello ','world')
'hello world'
```

Funktionen - Default-Parameter

- können beim Aufruf gesetzt werden
- ermöglichen einfache und komplexe Benutzungen einer Funktion

Funktionen - Default-Parameter

- können beim Aufruf gesetzt werden
- ermöglichen einfache und komplexe Benutzungen einer Funktion

```
>>> def summe3(a,b=42,c=24): #b und c sind D-P
...     return a + b + c
>>> summe3(5,9) #b=9, c=24
38
>>> summe3(5, c = 1) #b42, c=1
48
>>> summe3(5,9,1) #b=9, c=1
15
```

Bedingungen

- `if`
- `else`
- `elif` = `else if`

Bedingungen

- if
- else
- elif = else if

```
>>> x = 42
>>> if x < 0:
...     print('Negative')
... elif x == 0:
...     pass #do nothing, implement later
... else:
...     print('Positive')
...
'Positive'
```

Schleifen - `for` I

- `for` und `while`
- `for`-Schleifen meist als erweiterte `for`-Schleife (ohne Index)

Schleifen - for I

- for und while
- for-Schleifen meist als erweiterte for-Schleife (ohne Index)

```
>>> tiere = ['Hund', 'Vogel', 'Fisch']
>>> for t in tiere:
...     print(t, len(t))
...
Hund 4
Vogel 5
Fisch 5
```

Schleifen - for II

Durch die Funktion enumerate, lässt sich auch der Index explizit nutzen.

```
>>> tiere = ['Hund', 'Vogel', 'Fisch']
>>> for i, t in enumerate(tiere):
...     print(i, t)
...
0 Hund
1 Vogel
2 Fisch
```

Schleifen - `for` III

Durch die Funktion `range`, lässt sich auch nur der Index erzeugen.

```
>>> for i in range(3):
...     print(i)
...
0
1
2
```

Schleifen - while

Benutzung der while-Schleife ähnlich zu anderen Sprachen.

```
>>> ergebnis = 0
>>> while (ergebnis < 3):
...     ergebnis+=1
...     print(ergebnis)
...
1
2
3
```

Übersicht

1 Einleitung

2 Basistypen

3 Datenstrukturen

4 Kontrollstrukturen

5 Dies und Das

6 Python programmieren

7 Zusammenfassung

Dokumentation

- kommentieren von Funktionen mit `"""Inhalt"""`
- Zeilenkommentare mit `#` einleiten

Dokumentation

- kommentieren von Funktionen mit `"""Inhalt"""`
- Zeilenkommentare mit `#` einleiten

```
def helloWorld(a = 0):
    """Schreibt 'Hello World' und a auf die Konsole.

    Keyword arguments:
    a -- fancy parameter (default 0)
    """
    print('Hello World', a) #Implementationskommentar
```

Style Guides: PEP 8 und PEP 257

Module

- Module fassen Funktionen, Konstanten oder Klassen zusammen $\rightarrow$ Bibliotheken
- Import einzelner Elemente (oben) oder aller Elemente (unten)

Module

- Module fassen Funktionen, Konstanten oder Klassen zusammen → Bibliotheken
- Import einzelner Elemente (oben) oder aller Elemente (unten)

```
>>> from math import cos, pi
>>> cos(pi)
-1.0
```

oder

```
>>> import math
>>> math.cos(math.pi)
-1.0
```

Übersicht

1 Einleitung

2 Basistypen

3 Datenstrukturen

4 Kontrollstrukturen

5 Dies und Das

6 Python programmieren

7 Zusammenfassung

Möglichkeit 1: Klassische IDE und .py Skripte

- ein Skript kann viele Klassen, Funktionen oder auch direkte Aufrufe einhalten
- gut, wenn man größere Systeme baut
- Ausführung von Teilen des Skripts mitunter kompliziert

Möglichkeit 1: Klassische IDE und .py Skripte

- ein Skript kann viele Klassen, Funktionen oder auch direkte Aufrufe einhalten
- gut, wenn man größere Systeme baut
- Ausführung von Teilen des Skripts mitunter kompliziert

Gute IDEs für Python

- Spyder, gut für Einsteiger und in Anaconda schon vorhanden
- PyCharm, gut für Profis

Möglichkeit 2: Python Notebooks (.ipynb)

- hybride Form aus Direkteingabe und klassischer IDE
- Notebook besteht aus einer Sequenz von Code-Zellen
- jede Zelle kann einzeln ausgeführt werden
- Variablen werden zwischen den Zellen geteilt
- Notebooks werden im Browser bearbeitet und ausgeführt

Möglichkeit 2: Python Notebooks (.ipynb)

- hybride Form aus Direkteingabe und klassischer IDE
- Notebook besteht aus einer Sequenz von Code-Zellen
- jede Zelle kann einzeln ausgeführt werden
- Variablen werden zwischen den Zellen geteilt
- Notebooks werden im Browser bearbeitet und ausgeführt

Tools zum Nutzen: Jupyter Notebook (bspw. über Anaconda), Google Colab, Datalore, ...

Demo

Jupyter Notebooks

Schaut euch auch das Demo-Video und die Folien im Moodle an!

Übersicht

1 Einleitung

2 Basistypen

3 Datenstrukturen

4 Kontrollstrukturen

5 Dies und Das

6 Python programmieren

7 Zusammenfassung

Python Quick Facts

- dynamisches Typsystem

Python Quick Facts

- dynamisches Typsystem
- kein Semikolon

Python Quick Facts

- dynamisches Typsystem
- kein Semikolon
- Einrückung und Doppelpunkt statt Klammern

Python Quick Facts

- dynamisches Typsystem
- kein Semikolon
- Einrückung und Doppelpunkt statt Klammern
- Listen (`[]`) haben keine Elementtyp

Python Quick Facts

- dynamisches Typsystem
- kein Semikolon
- Einrückung und Doppelpunkt statt Klammern
- Listen (`[]`) haben keine Elementtyp
- meist erweiterte `for`-Schleifen über Listen

Python Quick Facts

- dynamisches Typsystem
- kein Semikolon
- Einrückung und Doppelpunkt statt Klammern
- Listen (`[]`) haben keine Elementtyp
- meist erweiterte `for`-Schleifen über Listen
- formale Parameter in Funktionen ohne Typ

Python Quick Facts

- dynamisches Typsystem
- kein Semikolon
- Einrückung und Doppelpunkt statt Klammern
- Listen (`[]`) haben keine Elementtyp
- meist erweiterte `for`-Schleifen über Listen
- formale Parameter in Funktionen ohne Typ
- Default-Parameter mit Vorfestlegung

Python Quick Facts

- dynamisches Typsystem
- kein Semikolon
- Einrückung und Doppelpunkt statt Klammern
- Listen (`[]`) haben keine Elementtyp
- meist erweiterte `for`-Schleifen über Listen
- formale Parameter in Funktionen ohne Typ
- Default-Parameter mit Vorfestlegung
- Programmierung in Notebooks

Und wenn ich nicht mehr weiter weiß?

- Python-Tutorial
- Python-Referenz

Nutzt Google!

Eure Probleme sind nicht exklusiv!

Aber kein blindes Kopieren!

Aufgabenblatt 1

- Rechner unter Ubuntu starten (Informatik-Login)
- Jupyter Notebook starten (s. Folien im Moodle)
- neues Notebook anlegen
- im MIN-Moodle einloggen (STiNE-Login)
- Aufgabenblatt 1 lösen

`https://lernen.min.uni-hamburg.de/course/view.php?id=2773`

- Name: Praktikum Computer Vision WiSe 23/24 (CVPrak23/24)
- Einschreibeschlüssel: CVPrak2324

Übersicht

1 Einleitung

2 Basistypen

3 Datenstrukturen

4 Kontrollstrukturen

5 Dies und Das

6 Python programmieren

7 Zusammenfassung

Einstieg in Python

- Offizielles Tutorial zu Python (Kapitel 1-6 und 10.2 sowie 10.6 relevant) The Python Tutorial
- A. Sweigart, Automate the Boring Stuff with Python (Kapitel 0,1,2,3,4,5,6 relevant) E-Book unter CC-Lizenz
- Kombiniertes Kurztutorial zu Python und NumPy (mit Extra-Hinweisen für Matlab-Umsteiger) Python Numpy Tutorial von Justin Johnson
- J. Bernard, Python Recipes Handbook, 1st ed., apress, 2016 (Kapitel 1,2,3,6,10,11 tlw. relevant) Bib-Katalog/E-Book