

Einführung in die Statistik mit Python

RS-eco

Biodiversity & Global Change Lab
Technische Universität München

rs-eco@posteo.de

25. Oktober 2019



Einleitung

kurzes Kennenlernen

Über mich:

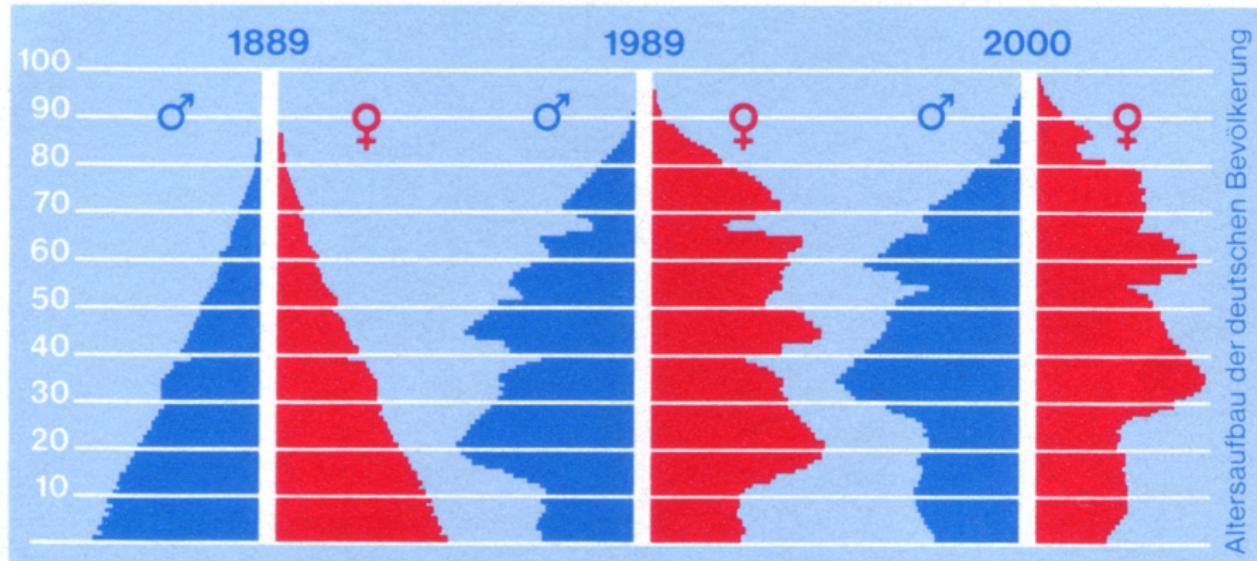
- BSc Applied Marine Biology, Bangor University, UK
- MSc Global Change Ecology, Universität Bayreuth
- Seit 05/2017 Daten- und IT-Manager (seit 11/2018 an der TU München)

- Geschlecht: Männlich
- Größe: 189 cm
- Schuhgröße: 43

Und nun zu euch: Name, Disziplin, Größe, Schuhgröße, Erwartungen?

Was ist Statistik?

- ursprünglich: "Die Lehre von den Daten über den Staat"



Was ist Statistik?

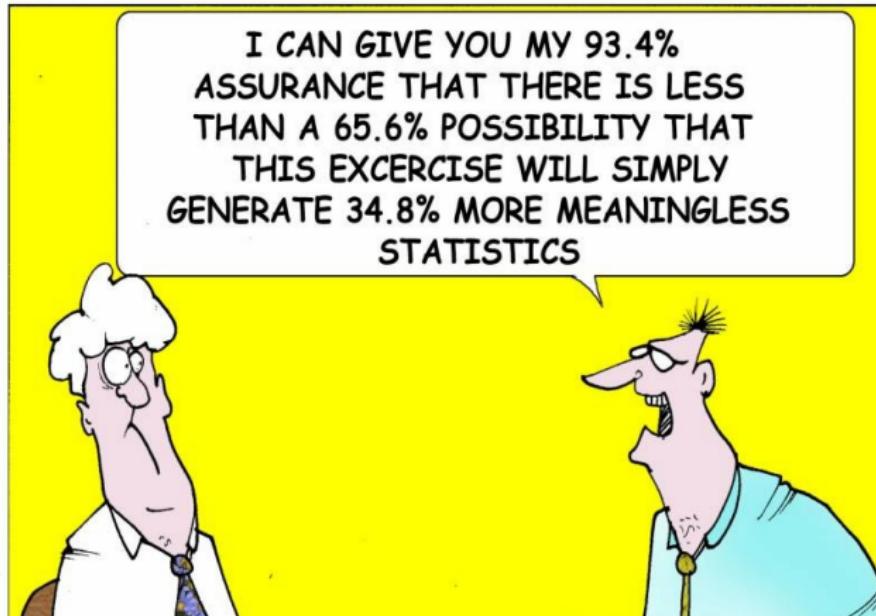
Entwicklung und Anwendung formaler Methoden zur **Gewinnung, Beschreibung und Analyse** sowie zur **Beurteilung quantitativer Beobachtungen (Daten)** - F. Vogel



“Data don't make any sense,
we will have to resort to statistics.”

Was ist Statistik?

Zusammenfassung von Methoden, die uns erlauben, **vernünftige optimale Entscheidungen im Falle von Ungewissheit** zu treffen. - A. Wald



Warum Statistik?

- kompakte Darstellung gewonnener Daten
- Bestätigung oder Widerlegung einer Hypothese
- vernünftige Vorhersagen zukünftiger Ereignisse auf Basis aktueller Gegebenheiten

- bildet die theoretische Grundlage aller empirischer Forschung
- vielfältige Anwendungsbereiche
- verzeichnet derzeit großes Wachstum (Big Data)

- hilft Statistik in anderen wissenschaftlichen Arbeiten zu verstehen
- hilft Statistiken selbstständig zu beurteilen und Missbräuche und Fehler leichter zu durchschauen

Begrifflichkeiten I

■ **deskriptive Statistik**

Daten werden in geeigneter Weise beschrieben, aufbereitet und zusammengefasst, meist in Form von Tabellen, graphischen Darstellungen und Kennzahlen, z.B. Volkszählung.

■ **induktive Statistik**

Herleitung von den Eigenschaften einer Grundgesamtheit, aus den Daten einer Stichprobe anhand von Schätz- und Testverfahren, z.B. Wahlprognosen.

■ **explorative Statistik**

Systematische Suche von möglichen Zusammenhängen (oder Unterschieden) zwischen Daten in vorhandenen Datenbeständen

Begrifflichkeiten II

■ univariate Statistik

Isolierte Betrachtung einzelner Variablen

■ bivariate Statistik

Gleichzeitige Betrachtung von zwei Variablen. Ziel ist es, etwas über die Beziehung der beiden Variablen zu erfahren, d.h. ihren Zusammenhang.

■ multivariate Statistik

Mehrere unabhängige oder abhängige Variablen werden gleichzeitig untersucht

Allgemeine Vorgehensweise

1. Datenerhebung

Sammeln und Auswerten von Daten. Bei einer Erhebung müssen die Daten nicht erst erzeugt werden, wie bei einem Experiment. Man unterscheidet zwischen Vollerhebung und Teilerhebung.

Vollerhebung = Untersuchung einer Grundgesamtheit, die die ganze zu befragende Masse darstellt

Teilerhebung = Entnahme einer Stichprobe der Grundgesamtheit (Nur ein Teil der ganzen Gruppe wird befragt)

Allgemeine Vorgehensweise

1. Datenerhebung

Sammeln und Auswerten von Daten. Bei einer Erhebung müssen die Daten nicht erst erzeugt werden, wie bei einem Experiment. Man unterscheidet zwischen Vollerhebung und Teilerhebung.

2. Datenaufbereitung

- Datenkodierung
- Datenbereinigung (Plausibilitätsprüfung und Korrektur, Ausreißer, fehlende Werte)
- Transformation der erhobenen Variablen
- Imputation von fehlenden Werte

Konventionen und Zeichen präzisieren die Ergebnisse einer sorgfältigen Aufbereitung!

Allgemeine Vorgehensweise

1. Datenerhebung

2. Datenaufbereitung

3. Datenanalyse

Anwendung von Methoden der explorativen, deskriptiven und induktiven Statistik. Eine Analyse ohne eine geeignete Statistik-Software (z.B. Python) ist heutzutage kaum möglich.

4. Interpretation

- unter Berücksichtigung des jeweiligen Fachgebietes
- treffsichere sprachliche Umsetzung der gewonnenen Ergebnisse
- Rückbezug auf aufgestellte Hypothesen und Fragestellungen
- Verweis und Querbezug auf andere wissenschaftlich gewonnene und valide Studienergebnisse

Achtung vor Fehlinterpretation

WELTSENSATION AUS DEUTSCHLAND

Erster Blut-Test erkennt zuverlässig Brustkrebs

Mediziner der Uniklinik Heidelberg erklären exklusiv alles zum neuen Test



Achtung vor Fehlinterpretation

Neuer Bluttest für Brustkrebs hat eine Trefferrate von 75 Prozent.

Die Trefferrate allein sagt aber nichts über die Zuverlässigkeit eines Tests aus!

=> Man muss immer zugleich die Falsch-Alarm-Rate kennen (z.B. wie häufig der Test bei gesunden Frauen fälschlicherweise einen Verdacht auf Krebs feststellt)

Ein Beispiel: Wenn man bei jeder Frau einen Tumor diagnostiziert, dann wird zwar jeder Tumor gefunden (Trefferrate = 100 %), aber auch jede gesunde Frau wird falsch diagnostiziert (Falsch-Alarm-Rate = 100 %).

Die Falsch-Alarm-Rate des neuen Bluttests ist 46 % = knapp die Hälfte aller gesunden Frauen würden einen verdächtigen Befund erhalten!

=> Ist das wirklich eine Weltsensation?

Datenerhebung

Variablen

- Variablen sind Dinge, die wir messen, kontrollieren oder manipulieren. Sie unterscheiden sich in vielerlei Hinsicht, vor allem in der Rolle, die ihnen in unserer Forschung zukommt, und in der Art der Maßnahmen, die auf sie angewendet werden können.
- Zur Vereinfachung der Nachvollziehbarkeit der Daten empfiehlt es sich, möglichst **selbsterklärende Variablennamen** zu verwenden, selbst wenn dies zu recht langen Bezeichnungen führt.
- Maßeinheiten der gemessenen Variablen sollten immer im Datensatz enthalten sein. Hier empfiehlt sich die Maßeinheit an das Ende des Variablenamens zu stellen, z.B. **bodylength_mm**.
- **fehlende Werte** im Datensatz sollten konkret spezifiziert sein (NA). Es macht einen großen Unterschied, ob Daten nicht gemessen wurden, ob sie den Wert 0 haben, oder ob es einen anderen Grund gibt, warum kein Eintrag erfolgt ist (z.B. selber Wert wie zuvor).

Einflussgröße & Zielgröße

Einflussgröße = beeinflusste/manipulierte Größe/Variable

Zielgröße = Merkmal, dass beinflusst wird, und damit
Untersuchungsgegenstand ist.

x	y	ε
Einflussgröße unabhängige Variable	Zielgröße abhängige Variable	Störgröße variabler Rest
Ursache	Wirkung	Nebenwirkung
Stellgröße	Messgröße	Fehler
Stimulus	Response	Nebenreaktion
erklärende Variable	erklärte Variable	unerklärter Anteil
generierte Variable	Interessierende Variable	ignorierte Variable
Treatment	Outcome	Nebeneffekt

Messskalen

- Variablen unterscheiden sich darin, wie gut sie gemessen werden können, d.h. wie viele messbare Informationen ihre Messskala liefern kann.
- Es ist offensichtlich, dass bei jeder Messung ein gewisser Messfehler vorliegt, der die Menge der Informationen bestimmt, die wir erhalten können.
- Ein weiterer Faktor, der die Menge an Informationen bestimmt, ist die Art der Messskala/des Skalenniveaus
- Prinzipiell kann zwischen **quantitativen Merkmalen**, die auf einer metrischen Skala messbar sind (wie **Körpergewicht** oder **Einkommen**), und **qualitativen Merkmalen**(wie **Geschlecht** oder **Farbe**) unterschieden werden.
- Im zweiten Fall spricht man auch von einem **kategorialen Merkmal**, da Ausprägungen in Form einer Kategorie angegeben werden.

Kategoriale Daten

Kategoriale Variablen umfassen eine endliche Anzahl von Kategorien oder eindeutigen Gruppen

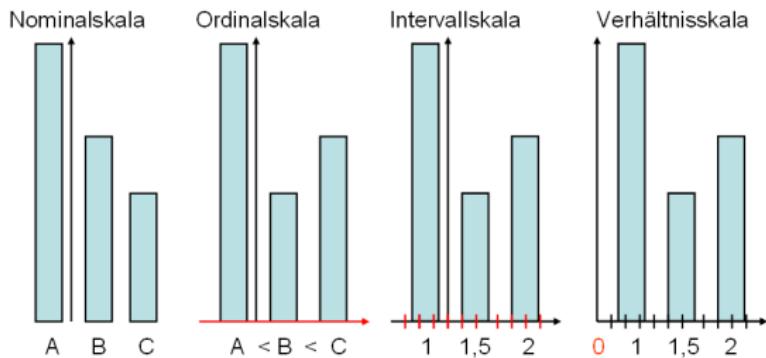
- **Boolesche Variable** = eine Variable, die nur zwei Zustände annehmen kann. Diese Zustände werden True und False (engl. für wahr und falsch) genannt und werden auch als Wahrheitswerte bezeichnet.
- **Nominale Variable** = eine Variable, die mehrere Zustände annehmen kann, aber keine Rangordnung hat, z.B. Geschlecht oder Blutgruppe.
- **Ordinal Variable** = Neben ihren Namen haben die Kategorien auch eine (gewisse) Rangordnung, z.B. Stadium einer Krankheit: Arterielle Verschluss krankheit (AVK) Stadium I, II, III, IV. Dies können auch metrische Variablen sein, die kategorisiert wurden (Beispiel: Variable „Einkommen“ mit den Kategorien „500–999 €“, „1000–1499 €“ usw.)

Numerische Daten

- **Intervaldaten** = kontinuierliche Daten, welche durch Messungen gesammelt werden, z.B. Körpergröße, Gewicht, Blutdruck, Cholesterin. Diese Daten können alle möglichen Werte annehmen und sind nur durch die Genauigkeit der Messung eingeschränkt.
- **Verhältnisvariablen** sind ebenfalls metrische Daten, im Unterschied zur Intervallskala existiert jedoch ein absoluter Nullpunkt (z. B. Blutdruck, absolute Temperatur, Lebensalter, Längenmaße). Bei diesem Skalenniveau sind Multiplikation und Division sinnvoll und erlaubt und Verhältnisse von Merkmalswerten dürfen gebildet werden.
- **Absolute Variablen** sind numerische Daten mit einem natürlichen Nullpunkt und einer natürlich gegebenen Maßeinheit (d.h. im weitesten Sinne „Stück“), z.B. Bevölkerungsgröße eines Landes.
- Bei Absolutskalen kann die Skaleneinheit nicht frei gewählt werden. Absolutskalen sind eindeutig festgelegt und Skalentransformationen nicht erlaubt.

Skalenniveaus

Skalenniveau	mathematische Operationen	Messbare Eigenschaften	Beispiel
Nominalskala	$= / \neq$	Häufigkeit	Postleitzahlen
Ordinalskala	$= / \neq; < / >$	Häufigkeit, Reihenfolge	Schulnoten
Intervallskala	$= / \neq; < / >; -/+$	Häufigkeit, Reihenfolge, Abstand	Zeitskala (Datum)
Verhältnisskala	$= / \neq; < / >; -/+; \% ; x$	Häufigkeit, Reihenfolge, Abstand, natürlicher Nullpunkt	Alter



Nominal: nur Häufigkeiten, ordinal: Reihenfolge, intervall: Abstände, verhältnisskaliert: Nullpunkt

Stetige und diskrete Daten

■ Stetige Daten

- ▶ numerische Variablen, die zwischen zwei beliebigen Werten eine unendliche Anzahl von Werten aufweisen
- ▶ Stetige Variablen können aus numerischen oder Datums-/Uhrzeitwerten bestehen
- ▶ Beispiel: die Länge eines Teils oder Datum und Uhrzeit eines Zahlungseingangs.

■ Diskrete Daten

- ▶ numerische Variablen, die zwischen zwei beliebigen Werten eine bestimmtezählbare Anzahl von Werten aufweisen
- ▶ Diskrete Daten sind immer numerisch und entstehen nahezu ausnahmslos durch Zählungen.
- ▶ Beispiel: Anzahl Kinder, Anzahl der Besuche beim Hausarzt im Jahr usw.

Häufbare und nicht-häufbare Merkmale

- Ein Merkmal gilt dann als häufbar, wenn es hinsichtlich der gleichen statistischen Einheit mehrere Ausprägungen annehmen kann, es also eine oder mehrere Antworten geben kann
- z.B. Merkmale wie Hobby oder Berufsausbildung, da eine Person ganz verschiedene Hobbies ausüben oder auch mehrere Berufsausbildungen durchlaufen haben könnte.
- Andere Merkmale wie beispielsweise Geburtsjahr müssen dagegen als nicht-häufbar betrachtet werden, da es hier sinnigerweise nur eine korrekte Angabe geben kann.
- Die Frage der Häufigkeit ist dann von großer Relevanz, wenn erhobene Daten in eine tabellarische Struktur überführt werden sollen

Es lohnt sich vor der Erfassung erhobener Daten am Rechner darüber nachzudenken, welche Tabellenstruktur sich für die vorliegenden Daten eignet.

Skalenniveaus



Übung - Skalenniveaus und Variablenarten

Welches Skalenniveau (metrisch, nominal, ordinal) haben die folgenden Merkmale – und sind sie stetig oder diskret?

- Wassertiefe eines Schwimmbeckens - metrisch, stetig
- Telefonnummern von Versandkunden - nominal, diskret
- Sorten von Speiseeis - nominal, diskret
- Schulnoten auf einer Skala von 1 bis 6 - ordinal, diskret
- Abstand zwischen zwei Gebäuden in cm - metrisch, stetig
- Preis eines Neuwagens in Euro und Cent - metrisch, diskret
- Haarfarbe von Kundinnen im Friseursalon - nominal, diskret
- Temperatur eines glimmenden Holzscheits - metrisch, stetig
- Produktwertung auf einer Skala von 1 bis 5 - ordinal, diskret
- Studiumsnoten auf einer Skala von 1,0 bis 5,0 - ordinal, diskret

Datenstruktur

- Idealerweise werden Daten als .csv-Datei gespeichert, diese können mit zahlreichen Programmen (z.B. Excel) geöffnet werden. Von einer Speicherung als .xls(x)-Datei ist abzusehen, da hier Änderungen der Einträge durch Formatierungs-Änderungen entstehen können.

Datenstruktur

- Idealerweise werden Daten als .csv-Datei gespeichert, diese können mit zahlreichen Programmen (z.B. Excel) geöffnet werden.
- Streng genommen handelt es sich bei diesem Format um reine Textdateien, diese weisen jedoch eine bestimmte Struktur auf und sind sehr häufig innerhalb der Forschung bzw. allgemein im Datenmanagement anzutreffen.
- CSV-Dateien können auch durch andere Zeichen getrennt sein, häufig ein Tab-Zeichen. Dann spricht man analog auch von „tab-separated values“ und Dateien im tsv-Format.
- Häufig kommt es zudem vor, dass Dateien zwar unter *.csv gespeichert werden, jedoch kein Komma als Trennzeichen genutzt wird.
- Ist nicht bekannt was für ein Trennzeichen verwendet wurde, muss die Datei mit einem Text-Editor geöffnet werden, um manuell das Trennzeichen zu bestimmen.

Datenstruktur

- Daten sollten immer in einem von zwei Formaten strukturiert sein:

Einer **breiten Tabelle**

oder

einer **langen Tabelle**

country	1999	2000
A	0.7K	2K
B	37K	80K
C	212K	213K



country	year	cases
A	1999	0.7K
B	1999	37K
C	1999	212K
A	2000	2K
B	2000	80K
C	2000	213K

Erste Schritte mit Python

Was ist Python?

Eine Schlangenfamilie



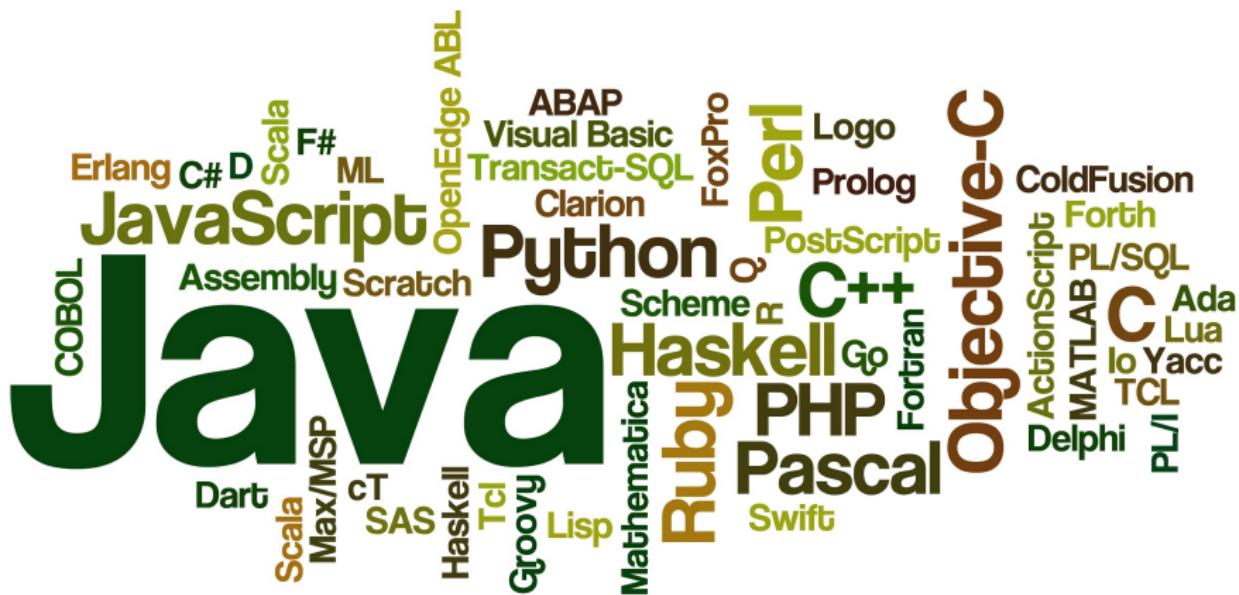
Was ist Python?

Eine Schlangenfamilie & eine Programmiersprache



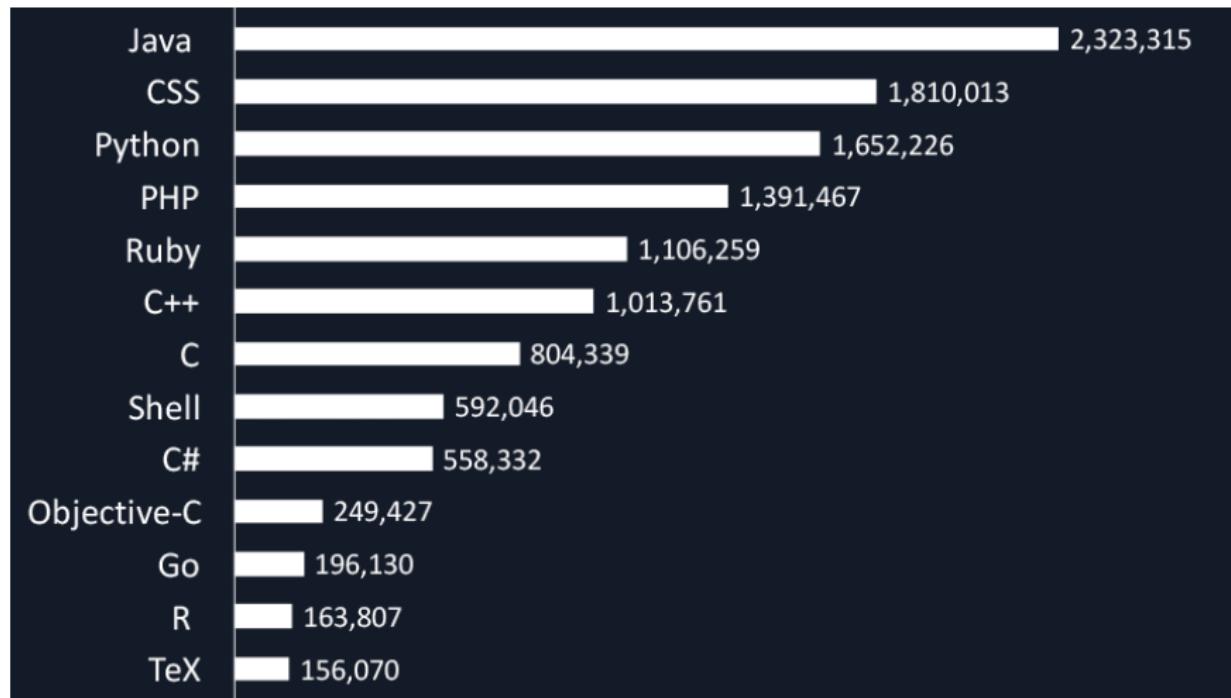
Was ist Python?

Aber auch eine Programmiersprache unter vielen:



Warum Python?

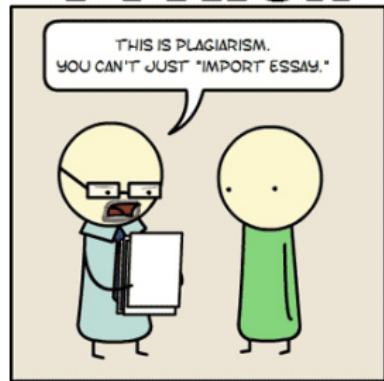
Eine der am häufigsten verwendeten Programmiersprachen



Warum Python?

- sehr vielseitig und flexibel
- leicht erlernbar
- platformunabhängig
- Open-Source
- Hochsprache statt Maschinensprache
- interpretierte Sprache
- Objekt-orientiert
- erweiterbar
- einbindbar in andere Sprachen und Programme
- verfügt über umfangreiche Bibliotheken

PYTHON



Sprachen lernt man, indem man sie spricht. Dies gilt für Python, wie für alle anderen Programmiersprachen.

Python installieren

Python kann unter "Downloads" von der offiziellen Website www.python.org heruntergeladen werden.

Es wird zwischen Python2.7 und Python3.7 unterschieden. Bitte stellt sicher, dass ihr Python3.7 installiert habt, da die Schreibweise zwischen den beiden Versionen zum Teil leicht variiert.

Zur Statistik mit Python benötigen wir zusätzliche Pakete (NumPy, Pandas, matplotlib, seaborn, Scipy, statsmodel). Mit der **Anaconda Distribution** werden diese zusätzlichen Pakete automatisch installiert, aber auch nützliche Werkzeuge wie das Jupyter Notebook.

Die Anaconda Distribution kann von www.anaconda.com/distribution heruntergeladen werden.

Nach der Installation können Python-Befehle direkt in der Konsole des Betriebssystems geschrieben werden, dazu öffnet man zunächst die Konsole und gebt dort wiederum **python** ein (Kleinschreibung beachten!).

IPython - Interaktive Python Umgebung

Damit das Schreiben von Python-Code noch besser gelingt, gibt es ein paar Werkzeuge, oder Schreibumgebungen, die uns bei der Programmierung unterstützen.

IPython = Open Source Software, die das interaktive Programmieren in Python vereinfacht (Vervollständigung des Codes, Syntaxhervorhebung)

Um die interaktive Umgebung zu nutzen, gibt man einfach **ipython** in die Konsole ein. Alternativ kann IPython auch über die grafische Benutzeroberfläche, dem Anaconda Navigator, geladen werden.

Der weitere Vorteil der IPython Umgebung ist, dass man eine Code-Historie erhältet, d. h. man kann mithilfe der Pfeiltasten durch den Code navigieren.

Mithilfe der Tabulator-Taste kann man außerdem alle verfügbaren Methoden eines Objektes anzeigen lassen.

Programmieren in Jupyter Notebooks

Jupyter Notebooks stehen auch mit der Anaconda Distribution bereit und können über den Anaconda Navigator geöffnet werden, oder indem man den Befehl **jupyter notebook** in die Konsole eingibt.

Bei einem Jupyter Notebook handelt es sich letztlich um nichts anderes als eine IPython-Konsole, die jedoch im Web-Browser läuft.

Ein neues Notebook kann man erstellen, indem man rechts auf „New“ klickt und dann „Python 3“ auswählt.

Der Vorteil ist, dass man den Code nicht nur speichern, sondern auch anderen Personen zur Verfügung stellen kann. Wenn man zum Beispiel eine Berechnung durchführt, dann bleibt das Ergebnis erhalten und man kann das Notebook anschließend exportieren und mit anderen teilen.

Mit Binder könnt ihr Jupyter Notebooks auch online verwenden (siehe <https://mybinder.org/>), die Kursunterlagen könnt ihr hier öffnen:
<https://hub.gke.mybinder.org/user/rs-eco-pystats-8b6ze63v/tree>.

Programmieren in Entwicklungsumgebungen (Spyder)

Zusätzlich steht zur Programmierung auch die Entwicklungsumgebung „Spyder“ zur Verfügung, die ebenfalls in der Anaconda Distribution integriert wurde. Auch Spyder kann mithilfe des Anaconda Navigators oder der Konsole geöffnet werden. Der Konsolen-Befehl lautet einfach **spyder**.

Der Name Spyder steht für „Scientific Python Development Environment“ und ist besonders für Forschungsarbeiten interessant, weil Sie mit dem „Variable Explorer“ ein Programm-Feature bereitstellt, das anderen Entwicklungsumgebungen fehlt.

Immer dann also, wenn man mit Tabellen arbeitet (bzw. Datensätzen), ist dieser Explorer sehr hilfreich, um einen besseren Einblick in die Daten zu erhalten.

Neben dieser Komponente bietet Spyder noch einen Editor zum Schreiben von Programmcode sowie eine IPython-Konsole an, aber auch einen Debugger, mit dem sich Fehler im Code leichter finden lassen.

Python Konventionen

- Ein Name beginnt mit einem Großbuchstaben (A-Z), Kleinbuchstaben (a-z) oder einem Unterstrich (_)
- **Namen enthalten niemals Leerzeichen!**
- Python akzeptiert die Zeichen @, \$ und % in den Namen nicht.
- Ein Python Befehl wird normalerweise auf eine Zeile geschrieben und das Zeilenende bedeutet Ende des Befehls
- Für Befehle über mehrere Zeilen, benutzt man das Zeichen \ um Python zu informieren, dass der Befehl auch die nächste Zeile betrifft.

```
value = 1 + \
2 + \
3
```

Python Konventionen

- Ein Name beginnt mit einem Großbuchstaben (A-Z), Kleinbuchstaben (a-z) oder einem Unterstrich (_)
- **Namen enthalten niemals Leerzeichen!**
- Python akzeptiert die Zeichen @, \$ und % in den Namen nicht.
- Ein Python Befehl wird normalerweise auf eine Zeile geschrieben und das Zeilenende bedeutet Ende des Befehls
- Für Befehle über mehrere Zeilen, benutzt man das Zeichen \um Python zu informieren, dass der Befehl auch die nächste Zeile betrifft.
- Man kann mehrere Befehle auf eine Zeile schreiben, diese werden dann mit einem ; voneinander getrennt

```
a = "One"; b = "Two"; c= "Three"
```

- Um eine Zeichenkette (String) zu schreiben, verwendet man Anführungszeichen (') oder ("")

```
str1 = 'Hello every body'
```

Python Konventionen

- Für einen Zeichenkette über mehrere Zeilen, verwendet man 3 Anführungszeichen ("") ohne das Zeichen \ zu benutzen

```
multiLineStr = """This is a paragraph. It is  
made up of multiple lines and sentences."""
```

- Kommentare beginnen immer mit einem #

```
# First comment  
print("Hello, Python!") # second comment  
  
# This is a comment.  
# This is a comment, too.  
# This is a comment, too.  
print("Finish")
```

PEP8 - der Stil-Guide für Python Code

- zur Einrückung 4 Leerzeichen verwenden
- Zeilenlänge auf 79 Zeichen begrenzen
- Module in folgender Reihenfolge importieren:
 - ▶ Standard Bibliothek
 - ▶ Module von Drittanbietern
 - ▶ lokale Module
- ein Import pro Zeile
- sogenannte „wildcard imports“ vermeiden (`from module import *`)

Auf der Seite www.pep8.org werden noch weitere Aspekte benannt und exemplarisch aufgezeigt.

Do it yourself - Python als Taschenrechner

```
# Addition  
2 + 3  
# Subtraktion  
3.3 - 2  
# Multiplikation  
2 * (3.141 - 2)  
# Division  
10 / 4  
# Division ohne Rest  
10 // 4  
# Modulo bzw. Rest  
10 % 3  
# Potenz  
5 ** 5
```

Do it yourself - Hello World

```
# Hello World  
print("Hello World.")
```

```
# Simple output  
print("Hello, I'm Python!")
```

```
# Input, assignment  
name = input('What is your name?\n')  
print('Hi, %s.' % name)
```

Do it yourself - Namensräume in Python

```
# Import pi from Math package
from math import pi

# Print pi
print(pi) # 3.14159

# Define a variable, also called pi
pi = 2

# Print pi
print(pi)
```

Python verstehen und anwenden

Datentypen: Zahlen und Zeichenketten

- 3 Zahlenformate:
 - ▶ Ganzzahlen (int)
 - ▶ Dezimal- bzw. Fließkommazahlen (float)
 - ▶ (komplexe Zahlen (complex))
- Dezimalzahlen werden nicht mit einem Komma sondern einem Punkt markiert!
- Zeichenketten (str) sind eine Aneinanderreihung von Zeichen, Leerzeichen oder weiteren Symbolen (Kann man als Text verstehen).
- **Achtung:** Nicht nur Buchstaben sondern auch Zahlen können als Buchstaben interpretiert werden.

Zeichenketten festlegen

```
# einfache Anführungszeichen
x = 'Hallo Welt.'

# doppelte Anführungszeichen
y = "Hallo Welt."
print(x ==y)

quote1 = "Sie sagte 'Hallo' zu mir."
quote2 = 'Sie sagte "Hallo" zu mir.'
print(quote1 == quote2)

# Eine mehrzeilige Zeichenkette
z = """Diese Zeichenkette besteht aus mehreren Zeilen.
Das hier ist die zweite Zeile,
das ist die dritte."""
```

Methoden von Zeichenketten

```
s = "Hallo Welt"
```

```
s.lower()
```

```
## 'hallo welt'
```

```
s.upper()
```

```
## 'HALLO WELT'
```

```
s.startswith("H")
```

```
## True
```

```
s.endswith("x")
```

```
## False
```

```
s = "Hallo Welt"
```

```
s.isdigit()
```

```
## False
```

```
str.isdigit("6")
```

```
## True
```

```
s.split(" ")
```

```
## ['Hallo', 'Welt']
```

```
"-".join(s.split(" "))
```

```
## 'Hallo-Welt'
```

Indexierung von Zeichenketten

```
n = "0123456789"
```

```
n[0]
```

```
## '0'
```

```
n[0:3] == n[:3]
```

```
## True
```

```
n[0:9]
```

```
## '012345678'
```

```
n = "0123456789"
```

```
n[:]
```

```
## '0123456789'
```

```
n[::-2]
```

```
## '02468'
```

```
n[1::2]
```

```
## '13579'
```

```
n[::-1]
```

```
## '9876543210'
```

Listen (list)

- Listen sind Datencontainer, die leer sein bzw. beliebig viele sowie auch unterschiedliche Datentypen enthalten können
- Oftmals wird eine Liste auch als Array bezeichnet, diese beinhalten jedoch (oftmals) nur einen Datentyp.
- Eine (leere) Liste wird durch eckige Klammern erstellt.
- Mit der Funktion `list()` können Objekte in eine Liste konvertiert werden.
- Listen haben den Vorteil, Elemente mehrfach enthalten zu können.

Listen erstellen

```
# leere Listen erstellen
liste_leer1 = []
liste_leer2 = list()
liste_leer1 == liste_leer2

# Listen erstellen mit unterschiedlichen Datentypen
liste_inhalt = [1, 2.0, "drei", ["Hallo", "Welt"],
                 {"key": "value"}]
liste_inhalt

# list()-Funktion auf Objekte anwenden
list("Hello")
```

Methoden von Listen

```
liste1 = []
liste2 = [4, 5]
liste3 = [4, 3, 2, 5, 1]

liste1.append(1) # Element anfügen
liste1.extend(liste2) # Liste erweitern
liste1.pop() # letztes Element herauslösen
liste3.sort() # Liste sortieren
liste3.remove(3) # spezifisches Element entfernen
```

Indexierung von Listen

Auf die Inhalte einer Liste können wir ebenfalls durch Indexierung zugreifen, wobei ein Index auch hier auf die Position eines Elementes verweist.

Wir erinnern uns, da in Python die Zählung bei „0“ beginnt, besitzt das erste Element einer Liste folglich als entsprechenden Index eben diesen Wert.

```
# Zuerst erstellen wir eine Liste  
liste = [9, 8, 7, 6, 5, 4]
```

```
# Nun indexieren wir die Liste  
liste[1:3]  
liste[:2]  
liste[3:]  
liste[1::2]  
liste[-1::-1]
```

Listen anwenden

```
# List comprehensions
fruits = ['Banana', 'Apple', 'Lime']
loud_fruits = [fruit.upper() for fruit in fruits]
print(loud_fruits)

## ['BANANA', 'APPLE', 'LIME']
```

```
# List and the enumerate function
list(enumerate(fruits))

## [(0, 'Banana'), (1, 'Apple'), (2, 'Lime')]
```

Listen entschachteln

Listen können verschachtelt sein, mit der Bibliothek *itertools* lassen sich verschachtelte Listen in Einzellisten konvertieren.

```
import itertools

liste = [[1,2,3], [4, 5, 6]]
chain1 = list(itertools.chain(*liste))
chain2 = list(itertools.chain.from_iterable(liste))

print(chain1 == chain2)

## True

print(chain1)

## [1, 2, 3, 4, 5, 6]
```

Dictionaries (dict)

- Ein Dictionary wird durch geschweifte Klammern instanziert und erhält zudem ein Schlüssel-Wert-Paar (Key-Value-Pair), wobei jeder Schlüssel (jeder Key) nur einmal vorhanden ist.
- Auch ein Dictionary kann bei der Instanziierung leer bleiben.
- Sollen jedoch Werte übergeben werden, wird zunächst der Schlüssel geschrieben, danach folgt – getrennt durch einen Doppelpunkt – der dazugehörige Wert

Ein Dictionary-Objekt erstellen

```
# Dictionary mit geschweiften Klammern instanzieren
d = {"key1": "val1", "key2": "val2"}
```



```
# dict indexing
print(d["key2"])
```



```
# Werte ändern
d["key2"] = "val2_neu"
print(d["key2"])
```



```
# Dictionary mit dict-Funktion
dict(hallo="welt")
```



```
dict(1="eins")
```

Ein Dictionary sortieren

```
d = {3: "drei", 1: "eins", 2: "zwei"}  
  
d_sorted = {k:v for (k,v) in  
            sorted(d.items(), key=lambda x: x[0])}  
  
print(d_sorted)  
  
## {1: 'eins', 2: 'zwei', 3: 'drei'}
```

Ein Dictionary sortieren

```
# Einfacher Datenabgleich mithilfe von Key-Value-Paaren
de_en = {"Hallo": "hello", "Welt": "world",
          "du": "you", "bist": "are", "schön": "beautiful",}

trans = (de_en.get("Hallo"), de_en.get("Welt"),
          de_en.get("du"), de_en.get("bist"),
          de_en.get("schön"),)

print("{} {}{}, {} {}{}!".format(*trans)) # tuple unpacking
## hello world, you are beautiful!
```

Einem Dictionary neue Werte hinzufügen

```
# default Parameter ändern  
de_en.get("großartig", "Wort nicht vorhanden")  
  
## 'Wort nicht vorhanden'
```

```
# Key-Value-Paar hinzufügen per Zuweisung  
de_en["großartig"] = "awesome"
```

```
# Key-Value-Paar hinzufügen per update()-Methode  
de_en.update({"großartig": "awesome"})
```

```
de_en.get("großartig", "Wort nicht vorhanden")  
  
## 'awesome'
```

Wert-Mapping

■ Text in Zahlen umwandeln

```
antworten = ["gut", "ok", "schlecht", "sehr gut",
             "sehr schlecht", "weiß nicht", "so lala",]

mapping = {"sehr schlecht": -2, "schlecht": -1, "ok": 0,
           "gut": 1, "sehr gut": 2, "weiß nicht": 8,}

numerisch = [mapping.get(antwort, 99)
             for antwort in antworten]
```

■ Mapping erstellen aus zwei Listen

```
labels = ["negativ", "neutral", "positiv"]
number = [-1, 0, 1]
mapping = {n: l for (n, l) in zip(number, labels)}
```

Tuple (tuple)

- Tuple sind weitere Datencontainer, jedoch mit der Besonderheit, dass sie unveränderbar sind („immutable“), man Werte also nicht überschreiben kann.
- Ein Tuple ist also ein Datentyp mit festen Werten und wird mit Rundklammern instanziert.

```
t = ("Hallo", "du", "schöne", "Welt")
```

Tuple (tuple)

- Tuple sind weitere Datencontainer, jedoch mit der Besonderheit, dass sie unveränderbar sind („immutable“), man Werte also nicht überschreiben kann.
- Ein Tuple ist also ein Datentyp mit festen Werten und wird mit Rundklammern instanziert.
- Ferner verfügt es lediglich über die Methoden `count()` und `index()`.
- Weil ihre Daten nicht verändert werden können, werden Tuple vor allem dann eingesetzt, wenn die Datenintegrität eine große Rolle spielt.
- Zudem können sie in einfacher Weise „entpackt“ werden, das sogenannte „Unpacking“.

```
t = ("Hallo", "du", "schöne", "Welt")
```

```
# tuple unpacking
x, *y, z = t
```

Tuple (tuple)

- Tuple sind weitere Datencontainer, jedoch mit der Besonderheit, dass sie unveränderbar sind („immutable“), man Werte also nicht überschreiben kann.
- Ein Tuple ist also ein Datentyp mit festen Werten und wird mit Rundklammern instanziert.
- Ferner verfügt es lediglich über die Methoden `count()` und `index()`.
- Weil ihre Daten nicht verändert werden können, werden Tuple vor allem dann eingesetzt, wenn die Datenintegrität eine große Rolle spielt.
- Zudem können sie in einfacher Weise „entpackt“ werden, das sogenannte „Unpacking“.
- Hilfreich ist diese Art der Zuweisung besonders bei der schnellen Vergabe von Namen für Variablen bzw. Objekten.
- Werden in einem Tuple drei Werte gespeichert, müssen wir das Tuple nicht drei Mal aufrufen, sondern lediglich einmal.

Unpacking

- Unpacking funktioniert auch mit Datencontainer: Listen, Dictionaries, Tuple und Sets Listen.

```
# list unpacking
liste = ["Hallo", "du", "schöne", "Welt"]
a, *b, c = liste

print(b)
```

- Stehen dabei weniger Variablen zur Verfügung, als Werte vorhanden sind, kann mithilfe des Asterisk-Zeichens (*) auch signalisiert werden, dass eine Variable mehrere Werte beinhalten soll – diese werden dann in einer Liste gespeichert.

Set / Frozenset (set, frozenset)

- Bei einem Set handelt es sich um eine Menge, die ihre Elemente genau einmal beinhaltet.
- Ein Set ist prinzipiell ungeordnet und kann verändert werden. Mit anderen Worten, Sets sind „mutable“, d. h. man kann ihnen Werte hinzufügen oder auch entfernen.
- Ein Set wird auch, ähnlich zu einem Dictionary, mit geschweiften Klammern instanziert – jedoch ohne die die Schlüssel-Wert-Beziehung, sondern lediglich als Aufzählung.
- Ein Sonderfall stellt das Frozenset dar. Dabei handelt es sich um ein Set, das jedoch unveränderlich ist. Folglich können hier keine Werte entfernt bzw. hinzugefügt werden.

Sets und ausgewählte Methoden

```
s1 = {1, 2, 3, 4}  
liste = [1, 3, 5, 7, 7, 3, 1]  
s2 = set(liste)
```

```
s1.union(s2) ## set([1, 2, 3, 4, 5, 7])  
s2.union(s1) ## set([1, 2, 3, 4, 5, 7])
```

```
s1.intersection(s2) ## set([1, 3])  
s2.intersection(s1) ## set([1, 3])
```

```
s1.difference(s2) ## set([2, 4])  
s2.difference(s1) ## set([5, 7])
```

```
s1.add(8)  
fset = frozenset(liste)  
fset.add(8)
```

Kontrollfluss: Programme mit Bedingungen steuern

- Die meisten Programme bzw. Aufgaben sind weitaus komplexer, als nur das Speichern von Einzeldaten
- Oft ist der Ablauf eines Programms an bestimmte Bedingungen geknüpft und einige Aufgaben sollen nur dann erledigt werden, wenn eine Bedingung erfüllt ist, sonst soll etwas anderes mit den Daten geschehen.
- Dieser Kontrollfluss lässt sich mit Wahrheitswerten steuern, die wiederum mithilfe von Vergleichsoperatoren kombiniert werden können.

Auch hier gilt: Die Programmanweisungen muss konkret formuliert sein, denn der Computer versteht nur das, was man ihm direkt mitteilt.

None, True, False

- Die Steuerung wird im Wesentlichen mithilfe der Werte **None**, **True** und **False** vorgenommen:

```
False == 0  
## True  
  
True == 1  
## True  
  
None == False  
## False  
  
not None == True  
## True
```

- Der Wert **True** wird zurückgegeben, wenn Variablen nicht den Wert **None** oder **False** erhalten, ein beliebiger Zahlentyp den Wert 1 hat, oder ein Datencontainer bzw. eine Daten-Sequenz nicht leer ist.
- Das bedeutet im Umkehrschluss, dass immer dann **False** ausgegeben

None, True, False

- Dies können wir uns zunutze machen, um Abfragen an Bedingungen zu knüpfen.

```
if True:  
    do_this(stuff)  
else:  
    do_that(stuff)
```

- Ausführung nur dann, wenn die Bedingung den Wert **True** annimmt.
- Dies lässt sich direkt angeben („if True“), oder indirekt anhand der Evaluation von Werten („if x“)

```
x = []  
if x:  
    print("Die Liste ist gefüllt.")  
else:  
    print("Die Liste ist leer.")
```

Vergleiche vornehmen

- Bedingungen lassen sich auch kombinieren, hierzu stehen zum Beispiel die Schlüsselwörter **and** und **or** bereit, die jeweils auch mit dem Wort **not** negiert werden können.
- Zudem kann das Schlüsselwort **in** genutzt werden, um die Existenz eines Wertes in einem Objekt zu überprüfen.
- Stellen wir uns folgendes Szenario vor:
 - ▶ Wir haben eine Umfrage durchgeführt und wollten nun prüfen, ob alle Angaben vollständig sind.
 - ▶ Oftmals wird die Angabe „weiß nicht“ mit dem Wert 8 codiert.
 - ▶ Haben wir beispielsweise fünf Fragen gestellt, die jeweils mit Werten von 1 bis 5 beantwortet werden können (und 8 bei „weiß nicht“), dann können wir mit jenen Schlüsselwörtern testen, ob alles angeben wurde.

Vergleiche vornehmen

- Bedingungen lassen sich auch kombinieren, hierzu stehen zum Beispiel die Schlüsselwörter **and** und **or** bereit, die jeweils auch mit dem Wort **not** negiert werden können.
- Zudem kann das Schlüsselwort **in** genutzt werden, um die Existenz eines Wertes in einem Objekt zu überprüfen.

```
antwort = [1, 2, 3, 2, 4, 1, 8]
```

```
if antwort and not 8 in antwort:  
    print("Die Antworten sind vollständig")  
else:  
    print("Antworten unvollständig.")  
## Antworten unvollständig.
```

Prüfung auf Gleichheit und Identität von Werten

- Neben diesen Schlüsselwörtern existieren noch weitere Operatoren, die genutzt werden können, um bestimmte Bedingungen zu überprüfen.

`x == y # x gleich y`

`x != y # x ungleich y`

`x > y # x größer y`

`x < y # x kleiner y`

`x < y < z # x kleiner y, y kleiner z`

`x <= y # x kleiner oder gleich y`

`x >= y # x größer oder gleich y`

`x & y, x and y # x und y -> bitwise, logical`

`x | y, x or y # x oder y -> bitwise, logical`

Prüfung auf Gleichheit und Identität von Werten

- Die Abfrage mit doppelten Gleichheitszeichen prüft, ob der eine Wert dem anderen Wert entspricht.

```
x = 4
```

```
y = 4.0
```

```
# Prüfung auf Gleichheit / Gleichwertigkeit
x == y # True
```

- Mit **is** wird überprüft, ob beide Werte identisch sind (und damit letztlich auf denselben Speicherplatz verweisen).

```
# Prüfung auf Selbigkeit / Identität
x is y # False
```

Unterschied zwischen Referenz und Kopie

```
# Referenz -> dieselbe ID
x = [1, 2, 3]
y = x
y.pop()

print(x)
```

```
# Kopie -> unterschiedliche ID
x = [1, 2, 3]
y = x.copy()
y.pop()

print(x, y)
```

Daten abfragen mit while- und for-Schleifen

- Daten selbst können auf unterschiedliche Art und Weise abgefragt werden.
- Grundsätzlich stehen hierbei zwei Optionen zur Verfügung, um mit den Einzelwerten in Datencontainern zu arbeiten.
- Während die sogenannten while-Schleifen ausgeführt werden, solange eine bestimmte Bedingung erfüllt ist (bzw. wahr bleibt), wird jedes Element in einem Datencontainer mit einer for-Schleife nacheinander abgearbeitet.
- Der Einsatz von while-Schleifen ist also dann sinnvoll, wenn eine Abfrage solange ausgeführt werden soll, bis eine Bedingung den Wahrheitswert False annimmt.

Abfragen mit while-Schleifen

```
i = 2
j = -1
while i > j:
    i = i**42
    print(i)
```

- Die Bedingung wird nie falsch und der Ablauf daher **nie** abgebrochen!
- Anders ist es, wenn die Bedingung irgendwann nicht mehr zutrifft.

```
i = 1
while i < 4:
    print(i, end=" ")
    i += 1 # kurz für: i = i + 1
```

- Hier ist die Bedingung ($i < 4$) nach einigen Durchläufen nicht mehr True, da wir den i-Wert nach jedem Durchlauf erhöhen.

Abfragen mit while-Schleifen

■ allgemeines Funktionsprinzip einer while-Schleife

```
process = True
while process:
    # Solange die Bedingung wahr ist,
    if some_condition:
        # ... rufe die Funktion auf.
        do_something_with(stuff)
    else:
        # Sonst: Schleife stoppen.
    process = False
```

for-Schleifen schreiben

- for-Schleifen werden genutzt, wenn wir auf jedes einzelne Element in einem Datencontainer zugreifen bzw. damit operieren wollen.
- Grundsätzlich beinhalten Datencontainer(z.B. Listen oder Dictionaries) 0 bis n-Elemente.
- Jedes Element in der Liste hat einen spezifischen Index-Wert, der angegeben werden kann, um das Objekt anzusprechen.
- Üblicherweise wird also festgestellt, wie viele Elemente eine Liste enthält und solange hochgezählt, bis der Wert überschritten wird.

```
names = ["Markus", "Julia", "Klaus"]
n = len(names) # n = 3

# als Schleife mit Index-Wert
for i in range(n):
    name = names[i] # names[0] == "Markus"
    print(name)

## Moritz
```

For-Schleife durch eine Liste

```
numbers = [2, 4, 6, 8]
product = 1
for number in numbers:
    product = product * number
print('The product is:', product)

## ('The product is:', 384)
```

List- und Dictionary-Comprehension

- Die Schreibweisen der sogenannten „List Comprehension“ und „Dict Comprehension“ bieten in Python die Möglichkeit, mit weniger Programmcode dieselbe Wirkung zu erzielen.
- Wenn wir eine Liste mit Elementen erstellen wollen, können wir dies mit einerseits mit einer for-Schleife und der append() Methode umsetzen, andererseits mit einer Comprehension.
- Der Vorteil einer Comprehension ist zwar, dass der Code schneller ausgeführt wird, andererseits kann eine solche Abkürzung auch von Nachteil sein, da der Code manchmal weniger gut lesbar ist und es etwa nicht auf die Geschwindigkeit, jedoch auf die Verständlichkeit ankommt.
- Weniger gut lesbar sind diese Abkürzungen etwa dann, wenn man mit verschachtelten Listen arbeitet

List Comprehensions I

```
# Liste mit for-loop erstellen
numbers_for = []
for i in range(10):
    numbers_for.append(i)

# Beispiel einer einfachen List-Comprehension
numbers_comp = [i for i in range(10)]
numbers_for == numbers_comp
```

List Comprehensions mit if-Bedingung

- Bei den Comprehensions muss man im Prinzip schon zu Beginn wissen, was die letzte Instanz ist.
- Mit anderen Worten, wir müssen wissen, was wir am Ende der Liste hinzufügen wollen (im Beispiel: die einzelne Nummer).
- List Comprehensions lassen sich im Grunde ganz einfach erstellen, indem man zunächst eine for-Schleife schreibt und diese dann einfach „in Form bringt“.
 - ▶ Man löscht also zunächst die Absätze, Einrückungen und Doppelpunkte, fügt die eckigen Klammern hinzu und schreibt das, was am Ende der Liste angehangen werden soll, einfach an den Anfang.
- List Comprehensions sind auch deshalb hilfreich, weil man Bedingungen an die Wertübergabe knüpfen kann.

List Comprehensions mit if-Bedingung

```
# als for-loop
even = []
for i in range(10):
    if i % 2 == 0:
        even.append(i)

# als list comp
even = [i for i in range(10) if i % 2 == 0]
print(even)
```

List Comprehensions II

```
# Eine Liste mit Listen entschachteln
list_of_lists = [[0, 1, 2], [3, 4, 5], [6, 7, 8]]  
  
# als for-loop
single_numbers = []
for num_list in list_of_lists:
    for num in num_list:
        single_numbers.append(num)  
  
# als list comp
single_numbers = [num for num_list in
                  list_of_lists for num in num_list]
```

Dictionary Comprehensions

- Auch ein Dictionary lässt sich anhand einer Comprehension erstellen, man spricht dann von einer „Dict-Comprehension“.
- Die Schreibweise ist dabei ähnlich, nur das eben geschweifte Klammern und Key-Value-Paare vergeben werden.
- Auch wenn die Comprehensions anfangs etwas komplizierter scheinen, so lohnt es sich, ihr Konzept zu verinnerlichen.
- Wenn man einmal das Prinzip verstanden hat, dann weiß man die Comprehensions schnell zu schätzen.
- Auch hier gilt, je öfter man es übt, desto leichter fällt es einem beim nächsten Mal.

```
numbers = [1, 2, 3]
labels = ["eins", "zwei", "drei"]
d = {k:v for (k, v) in zip(numbers, labels)}
```

Fehlerbehebung: Try und Except

```
# Code ohne try und except Schlagwörtern
nummern = [1, 2, "3.14", "Hallo. Wie geht's?", 4.0]
for num in nummern:
    if type(num) == str:
        if "." in num:
            num = float(num)
    else:
        num = int(num)
    print(num * 3)
```

Fehlerbehebung: Try und Except

```
nummern = [1, 2, "3.14", "Hallo. Wie geht's?", 4.0]

# Code mit try und except Schlagwörtern
for num in nummern:
    try:
        if type(num) == str:
            if "." in num:
                num = float(num)
        else:
            num = int(num)
        print(num * 3)
    except ValueError as e:
        print(e)
```

Funktionen aufrufen

- Bisher haben wir schon einige Funktionen von Python kennengelernt, zum Beispiel die Funktionen `print()`, `len()` oder `range()`.
- Funktionen zeichnen sich dadurch aus, dass man sie einmal formuliert bzw. definiert und sie später aufrufen kann – ohne den gesamten Programmcode erneut schreiben zu müssen.
- Python bietet zahlreiche, bereits eingebaute Funktionen an, die wir nutzen können, ohne den Code dafür schreiben zu müssen
- Die `range()`-Funktion stellt uns zum Beispiel in einfacher Weise eine Zahlenreihe bereit
- mit `list()` können wir Objekte in eine Liste umwandeln usw.

Wichtig ist, dass man die Namen der Funktionen nicht für eigene Funktionen oder Objekte nutzen sollte, da auch diese Objekte „überschrieben“ werden können und somit auf ein anderes Objekt verweisen.

Eingebaute Funktionen in Python

Built-In Functions				
abs()	delattr()	hash()	memoryview()	set()
all	dict()	help()	min()	setattr()
any()	dir()	hex()	next()	slice()
ascii()	divmod()	id()	object()	sorted()
bin	enumerate()	input()	oct()	staticmethod()
bool()	eval()	int()	open()	str()
breakpoint()	exec()	isinstance()	ord()	sum()
bytearray()	filter()	issubclass()	pow()	super()
bytes()	float()	iter()	print()	tuple()
callable()	format()	len()	property()	type()
chr()	frozenset()	list()	range()	vars()
classmethod()	getattr()	locals()	repr()	zip()
compile()	globals()	map()	reversed()	__import__()
complex()	hasattr()	max()	round()	

Eigene Funktionen definieren

- Es ist denkbar einfach, in Python eigene Funktionen zu schreiben.
- Sie werden allgemein mit dem Schlüsselwort def eingeleitet.
- Danach folgt der Name der Funktion, Rundklammern mit den Parametern sowie die Funktionsanweisung, die nach einem Doppelpunkt und eingerückt in einer neuen Zeile folgt.
- Ein konkretes Beispiel könnte eine simple Funktion sein, die eine Grußformel mit einem Namen ausgibt:

```
# Funktion definieren
def hello(name):
    return (f"Hello, {name}! Wie geht es dir?")

# Funktion aufrufen
hello("John")
```

Eine Funktion zur Berechnung des Quadrats

```
def squared(x):  
    return x**2  
  
for ii in range(6):  
    print(ii, squared(ii))  
  
## (0, 0)  
## (1, 1)  
## (2, 4)  
## (3, 9)  
## (4, 16)  
## (5, 25)
```

Eine Funktion mit Keyword-Parameter festlegen

```
def hello(name, anrede="dir"):  
    return f"Hallo, {name}! Wie geht es {anrede}?"  
  
print(hello("James", "ihnen"))  
  
print(hello("ihnen", "James"))  
  
print(hello(anrede="ihnen", name="James"))
```

Daten einlesen und exportieren

- Wollen wir unsere Forschungsdaten in Python nutzen, müssen wir diese einlesen. Zudem ist es wichtig, zu wissen, wie wir unsere Ergebnisse als Datei speichern können, um sie etwa mit anderen auszutauschen.
- Das Einlesen von verhandenen Daten geschieht mit Funktionen, die das Pandas-Paket bereitstellt
- Der Export findet hingegen mit Methoden des DataFrame-Objekts statt
- Man muss beim Aufruf mindestens den Dateipfad angeben
- Zudem können zahlreiche weitere Parameter übergeben werden, die je nach Dateiformat den Einlese- sowie Schreibprozess spezifizieren
- csv-formatierte Dateien stellen sich oftmals als durch Tabulator getrennte Dateien heraus, dies könnten wir wiederum dem Parameter sep mitteilen

Daten einlesen und exportieren

```
# Einlesen allgemein  
pd.read_<format>(fpath, **params)  
  
# Einlesen einer Datei mit Tabulator getrennten Werten  
data = pd.read_csv("my_tsv_data.csv", sep="\t")
```

```
# Export allgemein  
df.to_<format>(fpath, **params)  
  
# Export einer Datei mit Pipe getrennten Werten  
df.to_csv(fpath, sep="|")
```

Einen Beispiel-Datensatz generieren

```
# Libraries importieren
import numpy as np
import pandas as pd

# Seed festlegen, zur Reproduktion des Codes
np.random.seed(42)

# Daten und Index generieren
data = np.random.random((10, 4))
names = ["f1", "f2", "f3", "f4"]
idx = range(4, 14)

# DataFrame erstellen
df = pd.DataFrame(data, columns=names, index=idx)
```

Indexing: Attribute abfragen

```
# Attribute abfragen (Zeilen, Spalten und Werte)
df.index
df.columns
df.values

# Indexierung einer Spalte
df["f1"] == df.f1
```

Einem DataFrame neue Spalten hinzufügen

```
# neue Werte hinzufügen
np.random.seed(42)
cluster_values = np.random.randint(1, 4, 10)
df["cluster"] = cluster_values

mapping = {
    1: "Sport",
    2: "Wirtschaft",
    3: "Kultur",
}
df["label"] = df.cluster.map(mapping)
```

Werte durch Berechnungen hinzufügen und löschen

```
# neue Werte durch Berechnungen hinzufügen
df["sum_f1_f2"] = df.f1 + df.f2
df["mean_f1_f2"] = (df.f1 + df.f2) / 2

# neue Werte durch for-Loop hinzufügen
for column in df.columns[0:4]:
    name = f"{column}_mn_diff"
    df[name] = df[column] - df[column].mean()

# Spalten löschen
df.drop("sum_f1_f2", axis=1, inplace=True)
```

Indexing: Gezielt auf Daten zugreifen

- Prinzipiell können wir auf Einzelwerte zugreifen, auf Zeilen und Spalten sowie auf Ausschnitte („Slices“).
- Wollen wir uns exakt einen Wert ausgeben lassen, gibt es dafür zwei Varianten: `df.at[row, col]` und `df.iat[row, col]`
- In beiden wird zunächst der Zeilenname bzw. Zeilenindex und dann der Spaltenname bzw. Spaltenindex übergeben.
- Wesentlich ist hier die Verwendung der eckigen Klammern, da es sich dabei nicht um eine Funktion handelt.
- iat als Spaltenangabe nicht den Namen, sondern die Index-Position benötigt – daher auch das „i“, für „Index“.
- Anders als dem iat-Befehl, können wir dem at-Befehl auch Zeichenketten übergeben, etwa wenn der Zeilen-Index nicht aus Zahlen, sondern aus Buchstaben besteht.
- Man kann selbstredend nicht nur auf Werte zugreifen, sondern auch Werte verändern bzw. ersetzen.

Auf Werte eines DataFrame zugreifen

```
# Auf Einzelwerte zugreifen: [Zeile, Spalte]
df.at[4, "f1"]
df.iat[4, 0]
df.at[4, "f1"] == df.iat[0, 0]

# Einzelwerte festlegen
df.at[4, "f1"] = 1
```

Auf Werte eines DataFrame zugreifen

- Abfrage von kompletten Zeilen funktioniert mit: **loc** und **iloc**
- Ein DataFrame-Objekt erlaubt zudem das sogenannte „Slicing“, die stückweise Auswahl von Werten. Hierdurch können dann Zeilen- und Spaltenwerte gezielt angesprochen werden, etwa solche, die „mitten“ in der Datentabelle liegen.
- Auch der loc-Befehl benötigt die Namen der Objekte im jeweiligen Index, der iloc-Befehl wiederum deren genaue Position.
- Die Indexierung erfolgt, wie bei Listen auch, durch die Verwendung eines Doppelpunkts – und die Angabe von Start-, End- sowie Abstandswerten.

```
# Slicing mit Namen  
df.loc[5:9, ["f1", "f3"]]
```

```
# Indexierung mit Position  
df.iloc[1:6, [0, 2]]
```

Auf Werte eines DataFrame zugreifen

- Um Werte auszugeben, die eine bestimmte Bedingung erfüllen, wird das sogenannte „Boolean Indexing“ genutzt.
- Die Werte lassen sich also anhand einer Wahrheitsprüfung ausgeben

```
# einfaches boolean Indexing  
df[df > 0.4]  
df[df.label == "Kultur"]
```

Auf Werte eines DataFrame zugreifen

- Um Werte auszugeben, die eine bestimmte Bedingung erfüllen, wird das sogenannte „Boolean Indexing“ genutzt.
- Die Werte lassen sich also anhand einer Wahrheitsprüfung ausgeben
- Auch der Abgleich mehrerer Bedingungen ist möglich.
- **Die Bedingungen werden in dem Fall nicht mit and und or bestimmt, sondern mit den Zeichen & und |.**
- Jede Bedingung muss mithilfe von Rundklammern gruppiert werden.
- Die Bedingungen werden wie beim Indexing innerhalb der eckigen Klammern platziert.

```
# komplexes boolean Indexing
df[(df.label == "Kultur") | (df.cluster == 2)]
df[(df.label == "Kultur") & (df.f3 >= 0.3)]
```

Werte eines DataFrame sortieren

- Bei allen Varianten, die die Index-Position als Eingaben benötigen (iloc, iat), ist die Auswahl der Werte von der Sortierung der Datentabelle abhängt
- Dies ist bei der direkten Ansprache durch die Namen, nicht der Fall
- Die Werte einer Datentabelle können mit den Funktionen `sort_index()` sowie `sort_values()` sortiert werden.
- Die wertbasierte Sortierung erfolgt dabei durch Eingabe des Spaltennamens, wobei auch mehrere Spalten (als Liste oder Tuple) angegeben werden können
- Außerdem lässt sich die Richtung der Sortierung mithilfe des Parameters `ascending` bestimmen.

```
df.sort_index(inplace=True)  
df.sort_values("f1", ascending=False, inplace=True)  
df.sort_values(["f2", "f1"], ascending=False, inplace=True)
```

Überblick über die Daten beschaffen

- Um sich mit Pandas einen Überblick über die vorhandenen Daten zu verschaffen, stehen abermals unterschiedliche Methoden und Attribute bereit.
- Unter anderem lässt sich klären, welche Datentypen in der Datentabelle vorhanden sind, etwa wenn es sich um Daten aus einer fremden Quelle handelt.
- Mit der `head()`-Methode lässt sich der Anfang der Datentabelle ausgeben, standardmäßig werden die ersten fünf Zeilen angezeigt – auch dies lässt sich durch Eingabe eines Zahlenwertes ändern.
- Die Methode `tail()` stellt das Gegenstück dar und gibt ebenfalls die n-letzten Zeilen aus (auch hier ist der Standard-Wert „5“).

```
df.dtypes
```

```
df.head(2)
```

```
df.tail(7)
```

Daten mit eigenen Funktionen bearbeiten

Bisher haben wir Methoden kennengelernt, mit denen wir uns Ergebnisse generieren bzw. uns Statistiken ausgeben lassen können. Daten lassen sich mit deren Hilfe jedoch auch bearbeiten und transformieren, etwa mit der Methode `apply()`.

```
def multi_x(x, *args):
    for arg in args:
        x *= arg
    return int(x)

# xi mit den args-Werten multiplizieren [i: 0 ... n]
df.f1.apply(multi_x, args=(3, 2, 4))

# 0, wenn xi kleiner oder gleich 0.5, sonst 1 [i: 0 ... n]
df.f2.apply(lambda x: 0 if x <= 0.5 else 1)
```

Do it yourself

- Liest den Datensatz **size.csv** in Python ein
- Verschafft euch einen Überblick über die Daten (dtypes, head, tail, shape)
- Sortiert die Daten nach Größe
- Extrahiert alle Einträge mit einer Größe über 180 cm
- Nun extrahiert alle Einträge für weibliche Personen mit einer Größe von 180 cm und mehr
- und für alle männlichen Personen mit einer Größe unter 190 cm und einer Schuhgröße von 43
- Extrahiert alle Einträge für weibliche Personen mit einer Größe von 190 cm oder einer Schuhgröße von 40

Vielen Dank für eure Aufmerksamkeit!