# Inhaltsverzeichnis

# Grundlagen

- Print gibt Text auf Bildschirm aus
- # gibt Kommentar und wird nicht ausgeführt
- Korrekte Syntax ist wichtig
- Korrekte Einrückung (4 Leerzeichen) ist wichtig

## Variablen

- starten mit Buchstaben (Kleinbuchstaben) oder Unterstrich, nicht mit Zahl
- darf nur Buchstaben Zahlen und Unterstriche enthalten
- Aussagekräftig (Alter, Gewicht etc.)

## Datentypen

- Type () um Datentyp festzustellen
- Indizierung:
    - Fängt mit Index 0 an
    - [1:8:2] -> von Index 1 bis 8 in 2er Schritten
    - [ -1] -> letzte Stelle

| **Name** | **Typ** |
|---|---|
| Numerisch/numeric | int, float, complex |
| Wahrheitswerte/boolean | bool |
| Text/string | str |
| Folge/sequence | list, tuple, range |
| Abbildung/mapping | Dict |
| Menge/set | set, frozenset |
| Leere Variable | none |
| Callable | Funktionen, Methoden |
| Modules | Python-Module |

### *Numerisch*

- **Int**
    - Ganze Zahl
    - In Zeichenkette umwandeln: str(int)
    - Länge bestimmen mit len()
    - Mit input() kann Zeichenkette eingegeben werden und mit int() konvertiert werden
- **Float**
    - Reelle Zahl (markiert durch punkt am Ende)
    - Round(float, n): auf n-Nachkommastellen gerundet, sonst auf nächste ganze Zahl
    - Int(float) -> Nachkommastelle wird abgeschnitten (=immer abrunden)
    - Float(1) -> 1. -> Type: float
- **Complex**
    - z = 1.4 + 1.2j -> z = complex (1.4,1.2)

### *Boolean*

- TRUE oder FALSE
- 1 (bzw.>0) = true, 0 = false

### *String*

- Mit ‘’ oder “” gekennzeichnet
- Zeichenkette = unveränderbar
- Mit + miteinander kombiniert
- Mit * beliebig wiederholt
- Sonderzeichen: \n: Zeilenumbruch, t: Tabulator, \\: Backlash

- **Funktionen:**
  - Suchen nach Teilstrings:
    - .count(''): ob und wie oft 'hallo'.count('l') -> 2
      - Start/Stopp-Positionen festlegen 'hallo'.count('l', start, stopp)
    - .find(''): Indexposition des ersten Vorkommens
      - Suche von Rechts: rfind()
      - Suche nach Endung: endswith() -> True/False
      - Suche nach Start: startswith() -> True/False
      - komplexe Suchmuster: RegularExpressions
  - Ersetzen von Teilstrings:
    - .replace() #siehe unten
    - komplexe Ersetzungen: RegularExpressions
  - Groß/Kleinschreibung:
    - .lower(): alle in KB
    - .upper(): alle in GB
    - .capitalize(): nur der 1. Buchstabe groß
    - .title(): alle 1. Buchstaben groß
    - .swapcase(): KB -> GB; GB -> KB
  - Leerzeichen entfernen:
    - .lstrip(): links
    - .rstrip(): rechts
    - .strip(): LZ Anfang und Ende entfernen
  - Aufteilen & Zusammenfügen:
    - .split(): Argument = Zeichen/-folge gemäß welcher Aufteilung erfolgen soll #siehe unten
    - .join(): Liste von Zeichenketten zu einer zusammensetzen

### *Listen[]*

- Sammlung beliebiger Python-Objekte
- kann mehrere identische Objekte enthalten
- geordnet: mit Indizes adressierbar
- veränderbar: random[1] = 1.4
- slicen [x:y]: über Indizes
- Konvention Elemente in einer Liste sind oft homogen und werden in Iterationen verwendet
- **Funktionen:**
  - Listen verknüpfen:
    - .extend(): Add the elements of a list (or any iterable), to the end of the current list
    - .append(): Adds an element at the end of the list
    - +: Zusammenfügen der Listen
    - *: Wiederholung der gesamten Liste
  - Liste sortieren:
    - .reverse():Absteigende RF
    - .sort(): Aufsteigende RF
  - Elemente indizieren, einfügen, entfernen:
    - .copy(): Returns a copy of the list
    - .count(): Returns the number of elements with the specified value
    - .index(): Returns the index of the first element with the specified value
    - .insert(indexnummer, element): Adds an element at the specified position
    - .remove(): Removes the item with the specified value
    - clear(): Removes all the elements from the list
    - .pop(indexnummer): Removes the element at the specified position

### *Tuple ()*

- Sammlung beliebiger Python-Objekte
- kann mehrere identische Objekte enthalten
- geordnet: mit Indizes adressierbar
- unveränderbar: random[1] = 1.4 -> FEHLER
- **Funktionen:**
    - .count(): Returns the number of elements with the specified value
    - .index(): Returns the index of the first element with the specified value

### *Dictionary {}*

- "key-value" Paare
- Jeder Key darf nur einmal enthalten sein, Values können doppelt drankommen
- Key und value getrennt mit ":" -> 'apple' : 3.99
- Key-value-Paare getrennt mit ","
- Key abrufen -> value wird ausgegeben

### *Sets {}*

- Sammlung beliebiger Python-Objekte
- kann jedes Objekt nur einmal enthalten
- ungeordnet
- unveränderbar

### *Umwandlungen von Datentypen*

- Int <-> float
- Int -> str
- List <-> Tuple
- Str -> tuple
- Str -> List

## Operatoren

- Sind Symbole, die Python veranlassen spezifische Aktionen auszuführen
- **Zuweisungsoperatoren/Assignmen: =, +=, -=**
    - ‚=': weist Operanden auf linker Seite Objekt von rechter Seite zu
    - ‚+='/'-=': addiert/subtrahiert Operanden auf linker Seite zum Operanden auf rechter Seite und weist Ergebnis dem linken Operanden zu
- **Mathematische Operatoren/Arithmetic: +, -, *, /, %, ****
    - Wie in der Mathematik, aber nicht nur auf Zahlen, sondern auch anderen Datentypen anwendbar
    - % = Modulo -> Rest berechnen; ** = Potenz
- **Vergleichsoperatoren/Relational: ==, !=, >, <, >=, <=**
    - Vergleicht Objekte
    - Liefert immer true oder false zurück
- **Logische Operatoren/Logical: and, or, not, in**
    - AND: True, wenn beide Operanden True, sonst false
    - OR: True, wenn mindestens einer von beiden true, sonst false
    - NOT: Kehrt Wahrheitswert um
- **Präzedenz:**
    - * vor +
    - AND vor OR
    - NOT vor AND
    - * vor NOT

    - Vergleichsoperatoren vor AND
- **Verrechnungen: MÖGLICH**
    - int + float -> float
    - float * complex -> complex
    - int * list -> list
    - list + list -> list
    - tuple * int -> tuple
    - tuple + tuple -> tuple
    - str * int -> str
    - str + str -> str
- **Verrechnungen: NICHT möglich**
    - int > complex
    - dict + dict
    - set + set

## Konditionen

- Programmsteuerung, das erlaubt abhängig von Variablen verschiedene Befehle auszuführen
- **If-Statement: if Bedingung: Anweisung**
    - Keyword: if
    - Es folgt Wahrheitswert
    - Code wird ausgeführt, wenn Wahrheitswert = true
    - Verschachtelt: mehrere Bedingungen gleichzeitig auswerten
- **If-else-Statement:**
    - Wenn Ausdruck nach if False liefert, wird else ausgeführt
- **If-elif-ladder:**
    - Wenn mehr als 2 Fälle getestet werden, um abhängig vom Ergebnis zwischen mehreren Alternativen wählen zu können

## Funktionen

- **Aufgaben:**
    - Übernimmt bestimmte und klar definierte Aufgabe
    - Zerlegt Programme in kleinere, übersichtlichere, wiederverwertbare Bestandteile
    - Wird erst ausgeführt, wenn aufgerufen
- **Syntax:**
    1. Keyword: def
    2. Name der Funktion (Regeln wie Variablen)
    3. Parameter welche der Funktion übergeben werden in () -Klammern (Parameter optional, Klammern nicht) -> Parameter nur innerhalb Funktion verfügbar
    4. : markiert Ende des Headers
        - Bsp.: def funktion_name (x, y,…):
    5. Optional: Docstring um Funktion zu beschreiben (“““ markieren Anfang und Ende)
    6. Befehle, eingerückt angeben, die Bei Funktionsaufruf ausgeführt werden
        - Bsp.: function_code
            result = x + y + …
    7. Optional: return-Statement, um Objekt zurückzugeben
        - Bsp.: return result

## Iteration/Wiederholung

- Bei Iterationen geht man durch jedes einzelne Element der Gruppe

- Z.B. Listen, strings

### *While-loop*

- z.B. while a<1000:
- Solange die Kondition true ist, wird durchgeloopt
- Einrückung!
- End = ‘ ,’ trennt Ergebnisse
- Vorsicht! Wenn Kondition nie falsch ist, wird es für immer laufen

### *For-loop*

- z.B. For m in animals:
- Kann über jede Sequenz (list, str) in der angegebenen Reinfolge iterieren
- len () kann länge der variable aufrufen

### *range()-Funktion*

- z.B. for i in range(5): print (i) -> 0 1 2 3 4
- ist ein Generator (=jeder Wert ist berechnet und nur zur gegebenen Zeit da)
- in listen speichern: list(range(2,8,2) -> von 2 bis 8 in 2er Schritten
- range() und len() kombinieren: gibt Indices zusätzlich an:
    - for i in range(len(list)): print (i, sw[i])

### *enumerate()*

- Nummeriert Einträge einer Liste, Tuple, String (mit Indices)
- .format(x,y): x,y werden in Platzhalter {} eingesetzt
    - z.B. print (‘Der {}.Eintrag in Liste is{}‘.format(index, zahl))

### *zip()*

- Verbindet Elemente aus versch. Iterierbaren Objekten zum neuen Iterator-Objekt (=Zusammensetzung der ursprünglichen Elemente)
- Enumerate() und zip() kombiniert: bei mehreren Listen mit Indices:
    - for idx (subj, cond) in enumerate (zip(subj, cond)): print

### *durch dict {} iterieren*

- jedes Element des dict{} gibt seinen key wieder
    - for condition in subj_score: print(condition, subj_score[condition])
- nur VALUE: dict.value()
- Key-Value-Paare: dict.items()

## Modules

- Eine Datei welche Python Definitionen und Anweisungen enthält
- Import math (m)/numpy (np)/matplotlib.pyplot (plt)

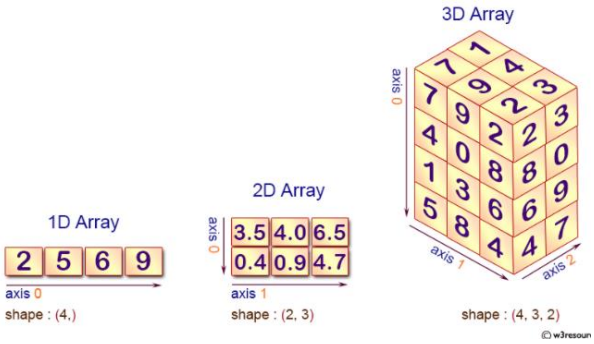
### *Math-Module*

- m.pow(Zahl, n): n-te Potenz von Zahl; Wurzel bei -> 0<n<1
- m.sqrt(Zahl): Wurzel
- m.exp(x): e^x
- m.log(a,x): log von x zur Basis a
- m.sin(x), m.cos(x), m.tan(x): sin, cos, tan
- m.arcsin(x), m.arccos(x), m.arctan(x): arcsin, arccos, arctan
- m.floor(): abrunden auf int
- m.ceil(): aufrunden auf int

- m.pi: pi
- m.e: e

## *Numpy*

- numerische Berechnungen
- arrays, Matrizen erstellen, rechnen, transformieren
- from numpy import * (alle Rechenoperatoren: Achtung! Kann andere funktionen überschreiben)
- Hilfe: z.b. ?rdm.random
- **Arrays: np.array ([x, y, z], dtype =float/int)**
    - Fixe Länge
    - Nur ein Datentyp (muss nicht angegeben werden -> dtype
    - Mehr dimensional
    - Slicen und indexing




- **Befehle:**
    - .shape: gibt (rows, columns) aus
    - np.where(): Indizes entlanf eines arrays finden für die eine Bedingung WAHR ist (z.B. a>5 -> gibt NUR noch die die wahr sind wieder)
    - np.arange(start, stop, step): Zahlenfolge
    - np.linspace(start, stop, number of samples, endpoint = True/False): True: nimmt Stop-Wert mit; False: endet vor Stop
        - Unterschied zwischen arange und linspace: entweder ist letzte Zahl in Klammer die Schrittgröße oder wie viele Werte ausgegeben werden sollen (z.B. 12 -> gibt 12 Werte wieder)
    - np.logspace(): Zahlen auf log-Skala
    - np.random.random(x): Zufallszahl zwischen 0,1
    - np.random.randint(start, stop, number): Zufallszahlen von start bis stop, number mal
    - np.random.randint(0,10,(2,10)): 2 Zeilen, 10 Spalten von Zufallszahlen zwischen 0 und 10
    - np.random.normal(mean, stdeviation, (x,y)): normalverteilte Zufallszahlen mit einem Mittelwert von 0, einer Standartabweichung von 2; ein Array mit x-Zeilen und y-Elemente in Zeilen
    - np.zeros(): array: 0
    - np.ones(): array: 1
    - np.eye(x): Einheitsmatrix
    - np.concatenate(): verbindet zwei Arrys ((a1,a2), axis = 0,1) -> 0: untereinander, 1: nebeneinander
    - np.tile(x, Zahl): wiederholt array Zahl mal
    - np.repeat(x,Zahl): wiederholt einzelne Elemente im Array Zahl mal
    - np.intersect1d(a,b): findet gemeinsame Elemente in a und b
    - np.setdiff1d(a,b) = np.in1d(a,b): findet Elemente, die inn a aber Nicht in b
    - np.arange(x).reshape(x,y): bringt Zahlenfolge in Matrizenform: x-Zeilen, y-Spalten
    - np.mean(): Mittelwert
    - np.max(): Maximalwert
    - np.min(): Minimalwert
    - np.sum(): Summe

## *Matplotlib*

- figures = Fenster in dem Grafik dargestellt wird
- axes = titles, lables, lines markers
- **Befehle:**
    - plt.axis([xmin, xmax, ymin, ymax]): Wertebereich Achsen formatiere`
    - plt.grid(True/False): Gitter passend zur Achsenskalierung
    - plt.xscale('log'): x-Achse logarithisch

    - plt.yscale('log'): y-Achse logarithmisch
    - plt.xlabel('text')
    - plt.ylabel('text')
    - plt.title('text')
    - plt.plot():
        - nur eine Zahlenliste: automatisch durchnummeriert, Start = 0; x-Achse= Nummerierung =Wertebereich
        - mehr als 1 Zahlenlisten: 1. Liste = x-Achse; 2. Liste = y-Achse
        - letztes Argument: Farbe/Form Diagramm
    - plt.legend(): Legende einfügen mit Namen nach label =
- **Farbe/Form:**
    - red, green, yellow, blue, orange, purple,...
    - darkmagenta, fuchsia, magenta, orchid, mediumvioletred
    - -: Linie
    - --: gestrichelte Linie
    - -.: Strichpunkt
    - :: gepunktet
    - o: farbige Kreise
    - s: farbige squares
    - D: Diamantform
    - ^: Dreiecke
    - x: x-Zeichen
    - *: *Zeichen
    - +: +Zeichen
- **Subplots = mehrere Plots in einem figure**
    - figures = fig
    - axes = ax
    - plt.subplot(zeile, spalte, #welches)
        - durch hinzufügen von sharey = True wird y-Achse geteilt
        - durch hinzufügen tight_layout = True schöneres Zusammenfügen
        - fig, ax = plt.subplots(#rows,#columns)
        - ax[row,column].plot(x,y)
        - ax[row,column].set_title('Text')
    - ax.plot(x-Achse, y-Achse, 'Form Datenpunkt', 'Farbe Datenpunkt', label ='text')
    - ax.set_title('text'): Teildiagramm Titel geben
    - ax.set_xlabel('text')
    - ax.set_ylabel('text')
    - ax.legend()
- **Histogramm**
    - plt.hist(daten, bins=FeinheitDesDiagramms)
    - bei größeren Datenmengen: plt.hist(Daten, bins=Zahl,range =(x, bisy), 'Farbe des Histogramms', label ='txt', umrandungsfarbe = 'black', Transparenz= 0.5)

# Komplexität

## Datei importieren

1. myfile = open('NameDatei.txt','r')
    - Bearbeitungsmodi: 'r' read, 'w' write, 'rw' read and write
    - Einlesen von Dateien:
        - .read(): im Ganzen als einzigen string einlesen

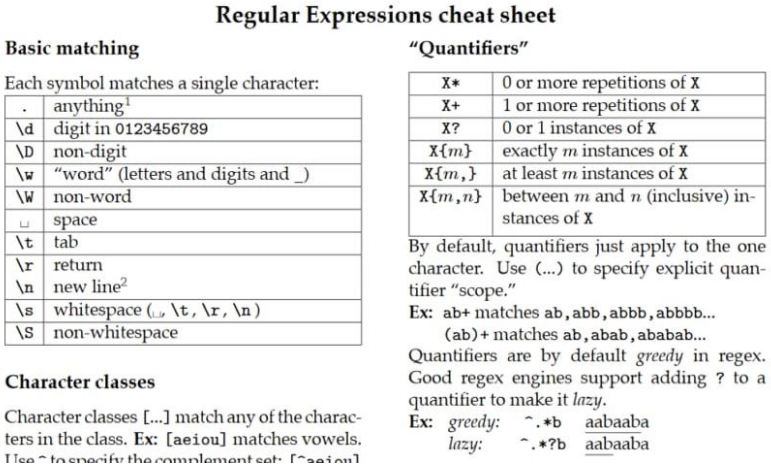
        - .readline(): Zeile f Zeile einlesen
            - .readlines(): zeilenweise in Liste von Zeichenketten einlesen
        - for-Schleifen: zeilenweise einlesen for line in myfile: print(line)
    - Schreiben in Dateien:
        - myfile = open('NameDatei.txt','r')
        - .write(): myfile.write('HALLO')
    - !! Schließen mit: close(myfile) oder .close

## Big O's

- welcher Term dominiert wird betrachtet
- Term mit konstanten (z.B. range) -> O(1)
- **for i in x -> O(N)**
    - wenn verschachtelt: i in x und j in y -> O(N^2) usw.
- if -> O(1)
- elif -> O(logn)
- else -> O(N^2)
- Reinfolge: O(1)<O(logN)< O(N)<O(NlogN)<O(N^2)<O(2^N)<O(N!)

## Regular Expression

- import re
- **Anchors**
    - ^: Start of string, or start of line in multi-line pattern
    - \A: Start of string
    - $: End of string, or end of line in multi-line pattern
    - \Z: End of string
    - \b: Word boundary
    - \B: Not word boundary
    - <: Start of word
    - >: End of word
- **Quantifier**
    - *: 0 or more
    - {3}: Exactly 3
    - +: 1 or more
    - {3,}: 3 or more
    - ?: 0 or 1
    - {3,5}: 3, 4 or 5
    - Add a ? to a quantifier to make it ungreedy
    - *; +; -; ? : Suche nach +,*,-,?
- **Character Classes**
    - \c: Control character
    - \s: White space
    - \S: Not white space
    - \d: Digit 0-9
    - \D: Not digit
    - \w: Word, A-Za-z0-9
    - \W: Not word
    - \w?: match 0 or 1 alphanumeric character
    - \w*: match any number of alphanumeric character 0-inf
    - \w+: match at least one alphanumeric character but as many as possible 1-inf
    - \w{3}: match exactly 3 alphanumeric characters



**Regular Expressions cheat sheet**

**Basic matching**

Each symbol matches a single character:

| | |
|---|---|
| `.` | anything[1] |
| `\d` | digit in `0123456789` |
| `\D` | non-digit |
| `\w` | "word" (letters and digits and _) |
| `\W` | non-word |
| `␣` | space |
| `\t` | tab |
| `\r` `\n` | return new line[2] |
| `\s` | whitespace (`␣`, `\t`, `\r`, `\n`) |
| `\S` | non-whitespace |

**Character classes**

Character classes [...] match any of the characters in the class. **Ex:** `[aeiou]` matches vowels. Use `^` to specify the complement set: `[^aeiou]`

**"Quantifiers"**

| | |
|---|---|
| `X*` | 0 or more repetitions of `X` |
| `X+` | 1 or more repetitions of `X` |
| `X?` | 0 or 1 instances of `X` |
| `X{m}` | exactly $m$ instances of `X` |
| `X{m,}` | at least $m$ instances of `X` |
| `X{m,n}` | between $m$ and $n$ (inclusive) instances of `X` |

By default, quantifiers just apply to the one character. Use (...) to specify explicit quantifier "scope."

**Ex:** `ab+` matches `ab,abb,abbb,abbbb...`
`(ab)+` matches `ab,abab,ababab...`

Quantifiers are by default *greedy* in regex. Good regex engines support adding `?` to a quantifier to make it *lazy*.

**Ex:** *greedy:* `^.*b` aabaaba
*lazy:* `^.*?b` aabaaba

    - \w{3,4}:match eather 3 or 4 alphanumeric characters
    - \x: Hexadecimal digit
    - \O: Octal digit
    - a?: a kein/einmal
    - a+: a mind. einmal
    - a*: a kein oder mehrmals
    - a{3}: a genau 3 mal
    - a{3,}: a mind 3 mal
    - a{,3}: a max 3 mal
    - a{3,4}: a mind 3, max 4 mal
    - ausdruck$: Zeichenkette endet auf ausdruck
- **Groups/Ranges**
    - .: Any character except new line (\n)
    - (a|b): a or b
    - (...): Group
    - (?:...): Passive (non-capturing) group
    - [abc]: Range (a or b or c)
    - [^abc]: Not (a or b or c)
    - [a-q]: Lower case letter from a to q
    - [A-Q]: Upper case letter from A to Q
    - [0-7]: Digit from 0 to 7
    - \x: Group/subpattern number "x"Ranges are inclusive

# Anwendung in der Biologie

## Lotka-Volterra-Modell

- **Nur eine Population:**
    - Neue Generation = Alter Generation + (Geburten – Sterberate)
- **2 Populationen ohne Konkurrenz:**
    - $B'(t) = B(t) \cdot (g - d \cdot R(t))$
    - $R'(t) = R(t) \cdot (b \cdot B(t) - s)$
- 2 Populationen mit Konkurrenz:
    - $B'(t) = B(t) \cdot (g - d \cdot R(t) - k \cdot B(t))$
    - $R'(t) = R(t) \cdot (b \cdot B(t) - s - c \cdot R(t))$

### *Refactoring*

- Bevor Funktionalität eines Programms ändern, Struktur des Programms ändern, damit Erweiterungen möglich sind
- Dazwischen testen, ob Programm noch dieselben Ergebnisse liefern wie vor dem Verändern

### *Ansätze des Programmentwurfs*

- **Top-Down:**
    - Zerlegen eines Problems in einfachere Teilprobleme
    - Beginn mit Funktion und Verwendung von Platzhalter für noch zu schreibende Funktion
- **Bottom-Up:**
    - Beginn mit Einzelfunktionen und Zusammensetzung am Ende zur großen Funktion

### Räuber-Beute-Simulation mit Zufall

- Annahme: Binomialverteilung der Wahrscheinlichkeit der Beute/Räuber-Individuen sich zu vermehren/sterben

#### *Pseudozufälle*

- Pseudo-Zufallsgenerator: **rng** (numpy)
- Zur Erzeugung von pseudo-binomialverteilte Werte
    - z.B. mit n = 20 und p = 0.3: Wahrscheinlichkeitsverteilung der Erfolge bei 20 Versuchen, die alle unabhängig voneinander eine Erfolgswahrscheinlichkeit von 0.3 haben
- **Funktionen:**
    - rng = np.random.default_rng(seed)
    - rng.uniform(): gibt einen Zufallswert zwischen 0,1 aus
    - rng.binomial(n, p, #Würfe): Binomialverteilung
    - x = rng.integers(n,m,#Würfe): integer-Zufallszahlen zwischen n und m #Würfe-mal
    - np.unique(x, return_counts=True): Welche Werte in x vorkommen/einzigartig sind
    - np.count_nonzero(x == Zahl): Wieviele Werte haben den Wert Zahl

# Machine Learning

### Rosenblatt Perceptron

- einfach:
    - 2 Eingabewerte und 1 Ausgabeneuron
    - Darstellung der logischen Operatoren: AND, OR und NOT
- Mehrlagig:
    - Zwischen Ausgabeschicht noch mindestens eine weitere Schicht verdeckter Neuronen
    - Darstellung von XOR

### Definitionen

- **Klassifikation**
    - Vorhersage von Gruppenzugehörigkeit (Overfit, Underfit, Good fit)
- **Regression**
    - Vorhersage von stetigen Werten (Overfit, Underfit, Good fit)
    - Linear Regression:
        - y = Steigung * x + (y-Achsenabschnitt)
    - Ridge Regression:
        - minimum sum of squared residuals R^2 + lamda * Steigung^2
            - lamda = 0: lineare Regression
            - Landa >>: Asymptomatische Annäherung an 0, Steigung/Y-Achsenabschnitt wählen, dass R^2 zw. 0 und 1
        - Inkludiert alle Daten
    - Lasso Regression:
        - The minimum sum of squared residuals R^2 + lamda * |Steigung|
            - Steigung kann 0 werden
        - Löscht unwichtige, redundante Daten

### Kreuzvalidierung/Cross validation

- Auf mehr Teilmengen anwendbar
- In k Teilmengen aufteilen und eine dieser Teilmenge auf k-1 trainieren (letzte Teilmenge für Test)

### *Daten k Teilmengen*

- K-1 Trainingsatz: Modell an Daten anpassen
- Enthält bekannte Ausgabe
- Modell lernt anhand der Daten
- Lernt andere Daten zu verallgemeinern
- Testdatensatz: Vorhersage, ob unser Modell für diese Teilmenge zu testen ist

### *K-Folds Cross Validation*

- Daten in k verschiedene Teilmengen/Falten aufteilen
- (k-1)-Teilmengen, um Daten zu trainieren
- Teilmenge (oder die letzte Falte) als Testdaten zu belassen
- das Modell gegen jede der Falten Mitteln und finalisieren des Modells
- danach gegen Testset testen

### *Leave-one-out-CV*

- Anzahl der Teilmengen = Anzahl der Beobachtungen, die im Datensatz sind
- mitteln ALLER Teilmengen und Bau des Modells mit dem Durchschnitt
- testen des Modells gegen die letzte Teilmenge

## K-Means Algorithmus

- **Ziel: Datensatz so in k Partitionen zu teilen, dass Summe der quadrierten Abweichung von Cluster-Schwerpunkten minimal ist**
- effiziente Methode um Clustrt in Datenmengen zu finden
- Cluster von nicht-annotierten Daten finden
- Anzahl der Cluster, muss im Voraus bekannt sein
- Cluster im Datensatz müssen etwa gleich groß sein
- Datensatz darf nicht viel Rauschen bzw. nicht viele Ausreißer enthalten
- Cluster überlappen nicht
- Non determinitic:
    - Random initialisation: Zudallsinitialisierung
    - Run multiple times

## Cluster

- **Miteinander vergleichen: Silhouettenkoeffizient**
    - Von Cluster unabhängige Bewertung der Gruppierungen
    - 1>S>0.75: gute Strukturierung, robust
    - 0.75>S>0.5: mittel
    - 0.5>S>0.25: schwach
    - 0.25>S>0: keine Struktur It is possible to data driven identify the most stable cluster, however, best results will be obtained with prior knowledge
- **Hierarchische Clusteranalyse: Dendrogramm**
    - distanzbasierten Verfahren zur Strukturentdeckung in Datenbeständen. Cluster bestehen hierbei aus Objekten, die zueinander eine geringere Distanz (oder umgekehrt: höhere Ähnlichkeit) aufweisen als zu den Objekten anderer Cluster.

# Data Visualisation

## Pandas

- Import pandas as pd
- Iris dataset is gespeichert als „iris_csv.csv“ und kann mit pandas library geladen werden

- sep= um bestimmtes Symbol zum trennen der Spalten (z.B. Komma, tab etc.)
- .head() gibt die ersten 5 Reihen wieder
- .info() gibt mehr Informationen wieder
- .unique() gibt alle einzigartigen Werte in einer Reihe oder Spalte
    - Um zu überprüfen wie viele verschiedenen Spezies es in dem dataset gibt
- .loc() entzieht die Reihen, die einer bestimmten logical expression folgen, in quadratischen Blöcken
    - Z.B. iris_df.loc[iris_df["class"] == "Iris-setosa"] entziehen alle Reihen in denen class Spalte iris-setosa entspricht
- .values() gibt die Pandas Objekte in numpy arrays wieder
- .iloc() entziehen Reihen und Spalten anhand deren Indices
    - iris_df.iloc[:10]: alle ersten 10
    - iris_df.iloc[1,3]: index 1 = Reihe 2; index 3 = Spalte 4
    - iris_df.iloc[2]: Jene Zeile mit Index 2

## Seaborn

- import seaborn as sns
- sns.relplot() = relational plot: plot figures that show relationships between different variables
- hue: sets the column that will be represented as different colors
- sns.set_style(“ which style”): plotting styles
- **Plotting distributions**
    - displot: visualization of distributions
    - sns.displot(data=iris_df, x="sepallength", hue="class")
    - kde = kernel density estimation: estimates underlying probability distribution => smooth function (smoothing parameter)
    - sns.displot(data=iris_df, x="sepallength", hue="class", kde=True)
    - kind = 'kde': only plot densities, removing histograms
    - sns.displot(data=iris_df, x="sepallength", hue="class", kind="kde")
    - high bandwidths => smooth curves, but not so meaningful
    - low bandwidths => noisier curves; may over-emphasize random noise

### *Catplot*

- kins = Darstellungstyp
- BOX Plot: sns.catplot (data=iris_df, x="class", y="sepallength", kind="box")
- VIOLIN Plot: sns.catplot(data=iris_df, x="class", y="sepallength", kind="violin")
- BAR Plot: sns.catplot(data=iris_df, x="class", y="sepallength", kind="bar", ci="sd")
    - mean values by default, ci="sd" used to make error bars represent standard deviation
- **Figure-level Funktionen:**
    - produce different subplots
    - sns.displot(data=iris_df, x="sepallength", hue="class", kind="kde")
- **Figure-level Subplots: col**
    - Every class has an own plot
    - sns.displot(data=iris_df, x="sepallength", hue="class", col="class", kind="kde")
- **Axes-level:**
    - produce self-contained plots that can be dropped into matplotlib figures using ax
    - sns.kdeplot(data=iris_df, x="sepallength", hue="class")
- **Axesl-level-Subplots:**
    - fig, axes = plt.subplots(1,2)
    - sns.kdeplot(data=iris_df, x="sepallength", hue="class", bw_adjust=.25, ax=axes[0])
    - sns.kdeplot(data=iris_df, x="sepallength", hue="class", bw_adjust=1, ax=axes[1])