

Programmieren für Naturwissenschaften

PD Dr. Kaspar Riesen

Zusammenfassung & Musterlösung Wiederholungsserie

FS 2023

Lukas Batschelet

 Sämtliches Material ist auch auf GitHub abgelegt: https://github.com/lbatschelet/23FS_PN

Dieses Werk ist lizenziert unter einer [Creative Commons](#) “Namensnennung – Nicht-kommerziell – Weitergabe unter gleichen Bedingungen 4.0 International” Lizenz.



Inhaltsverzeichnis

1	Excel	2
1.1	Wichtige Excel-Formeln	2
1.2	Übliche Diagrammarten und ihre Anwendungen	4
2	Python Datentypen, Funktionen und Methoden	5
2.1	Datentypen in Python	5
2.2	Grundlegende Funktionen	5
2.3	Methoden für Datentyp <code>str</code>	6
2.4	Methoden für Datentyp <code>list</code>	7
2.5	Funktionen im Modul <code>random</code>	7
2.6	Funktionen im Modul <code>math</code>	7
2.6.1	Konstanten im Modul <code>math</code>	8
2.7	Funktionen im Modul <code>statistics</code>	8
3	Wiederholungsserie	9
3.1	Range und for Schleifen	9
3.2	Quadratfunktion	10
3.3	Listenoperationen und dictionary	10
3.4	Boolesche Operatoren	11
3.5	Lokale Maxima	11
3.6	Summe einer Liste < Threshold	12
3.7	FizzBuzz	12
3.8	Zufallszahlen	12
3.9	Geldspielautomat	13
3.10	Würfelsimulation	13
3.11	Quadratische funktion mit zwei Lösungen	14
3.12	Dateiimport und Keywordsuche	14
3.13	scipy und Kurvenplot	15
3.14	Verständnisfragen	16
3.15	Gemischt	17
3.15.1	Counter und bool	17
3.15.2	while zu for Schleife	17
3.15.3	Zufällige Liste erstellen und auftrennen	17
3.15.4	Aufsummieren einer Liste mit threshold	18
3.16	Funktionen mit Userinput	18
3.17	string-Manipulationen und Matrix	19
3.18	Gemischte Fragen, Grundlegendes	20
3.19	Listenmanipultaionen	21
3.20	Mathematische Funktionen und statistik über Datei	21
3.21	list_splitting und multiply into table	23

1 Excel

1.1 Wichtige Excel-Formeln

Excel bietet eine Vielzahl von Formeln und Funktionen zur Datenanalyse und -manipulation. Hier sind einige der wichtigsten:

- **ZÄHLEN(Bereich)** - Gibt die Anzahl der Zellen mit Zahlen im angegebenen **Bereich** zurück.
- **SUMME(Bereich)** - Addiert alle Zahlen in einem **Bereich** von Zellen.
- **WENN(Bedingung, Dann_Wert, Sonst_Wert)** - Überprüft eine **Bedingung** und gibt einen Wert zurück, wenn die Bedingung wahr ist (**Dann_Wert**), und einen anderen Wert, wenn sie falsch ist (**Sonst_Wert**).
- **WENNS(Bedingung1, Wert1, Bedingung2, Wert2, ...)** - Ähnlich wie **WENN**, aber ermöglicht mehrere Bedingungen und Rückgabewerte.
- **ZÄHLENWENN(Bereich, Kriterium)** - Zählt, wie oft ein bestimmtes **Kriterium** im angegebenen **Bereich** erfüllt wird.
- **ZÄHLENWENNS(Bereich1, Kriterium1, Bereich2, Kriterium2, ...)** - Ähnlich wie **ZÄHLENWENN**, aber ermöglicht mehrere Bedingungen.
- **SUMMEWENN(Bereich, Kriterium, Summe_Bereich)** - Summiert die Werte in einem **Summe_Bereich**, die ein bestimmtes **Kriterium** erfüllen.
- **SUMMEWENNS(Summe_Bereich, Bereich1, Kriterium1, Bereich2, Kriterium2, ...)** - Ähnlich wie **SUMMEWENN**, aber ermöglicht mehrere Bedingungen.
- **ANZAHL(Bereich)** - Gibt die Anzahl der Einträge in einem **Bereich** zurück.
- **GANZZAHL(Zahl)** - Rundet eine **Zahl** ab auf die nächste ganze Zahl.
- **NICHT(Logischer_Wert)** - Kehrt einen logischen Wert um; z.B. wird **WAHR** zu **FALSCH**.
- **UND(Bedingung1, Bedingung2, ...)** - Gibt **WAHR** zurück, wenn alle Bedingungen erfüllt sind.
- **ODER(Bedingung1, Bedingung2, ...)** - Gibt **WAHR** zurück, wenn mindestens eine der Bedingungen erfüllt ist.
- **WAHR()** - Gibt den logischen Wert **WAHR** zurück.
- **FALSCH()** - Gibt den logischen Wert **FALSCH** zurück.
- **WOCHENTAG(Datum, Typ)** - Gibt den Wochentag eines bestimmten **Datums** als Zahl zurück (1 für Sonntag bis 7 für Samstag).
- **RUNDEN(Zahl, Stellen)** - Rundet eine **Zahl** auf eine bestimmte Anzahl von Dezimalstellen.

- **AUFRUNDEN(Zahl, Stellen)** - Rundet eine **Zahl** immer auf, unabhängig von den Dezimalstellen.
- **MIN(Bereich)** - Gibt den kleinsten Wert in einem **Bereich** zurück.
- **MAX(Bereich)** - Gibt den größten Wert in einem **Bereich** zurück.
- **MITTELWERT(Bereich)** - Gibt den Durchschnittswert in einem **Bereich** zurück.
- **MEDIAN(Bereich)** - Gibt den mittleren Wert (Median) eines **Bereichs** zurück.
- **MODALWERT(Bereich)** - Gibt den am häufigsten vorkommenden Wert in einem **Bereich** zurück.
- **QUANTIL(Bereich, Quantil)** - Gibt den Wert an der angegebenen Quantilposition in einem **Bereich** zurück.
- **STABW.S(Bereich)** - Gibt die Stichprobenstandardabweichung der Werte in einem **Bereich** zurück.
- **HÄUFIGKEIT(Daten_Bereich, Klassen_Bereich)** - Gibt eine Häufigkeitsverteilung der Daten in einem **Bereich** zurück.
- **GROSS(Text)** - Wandelt einen **Text** in Großbuchstaben um.
- **KLEIN(Text)** - Wandelt einen **Text** in Kleinbuchstaben um.
- **LÄNGE(Text)** - Gibt die Anzahl der Zeichen in einem **Text** zurück.
- **FINDEN(Suchtext, In_Text, Start)** - Findet eine Zeichenkette innerhalb einer anderen und gibt die Position des ersten Zeichens zurück.
- **SVERWEIS(Suchkriterium, Tabelle, Spaltenindex, Bereich_Verweis)** - Sucht in der ersten Spalte einer Tabelle nach einem Wert und gibt einen Wert in derselben Zeile aus einer angegebenen Spalte zurück.
- **WVERWEIS(Suchkriterium, Tabelle, Zeilenindex, Bereich_Verweis)** - Ähnlich wie **SVERWEIS**, sucht aber in der ersten Zeile einer Tabelle.
- **INDEX(Bereich, Zeile, Spalte)** - Gibt den Wert einer Zelle in einem **Bereich** basierend auf der angegebenen **Zeile** und **Spalte** zurück.
- **VERGLEICH(Suchwert, Suchbereich, Typ)** - Gibt die relative Position eines angegebenen Elements in einem **Suchbereich** zurück.

1.2 Übliche Diagrammarten und ihre Anwendungen

- **Balkendiagramm (Säulendiagramm)**

- *Eignung*: Geeignet zur Darstellung von diskreten Datenkategorien und deren Mengen oder Werten. Nützlich für den Vergleich von Datenmengen zwischen verschiedenen Kategorien.
- *Ungeeignet für*: Kontinuierliche Daten und Darstellung von Trends über die Zeit.

- **Liniendiagramm**

- *Eignung*: Ideal zur Darstellung von Trends und Veränderungen über einen Zeitraum hinweg. Geeignet für kontinuierliche Daten.
- *Ungeeignet für*: Vergleich von diskreten Datenkategorien.

- **Kreisdiagramm (Tortendiagramm)**

- *Eignung*: Geeignet zur Darstellung von Proportionen und prozentualen Anteilen innerhalb eines Ganzen.
- *Ungeeignet für*: Darstellung von Trends über die Zeit und wenn es viele Kategorien gibt, da dies die Lesbarkeit beeinträchtigt.

- **Streudiagramm (Punktdiagramm)**

- *Eignung*: Ideal zur Darstellung von Beziehungen zwischen zwei kontinuierlichen Variablen.
- *Ungeeignet für*: Darstellung von kategorischen Daten oder Zeitreihen.

- **Histogramm**

- *Eignung*: Gut geeignet für die Darstellung von Verteilungen und Häufigkeiten innerhalb von kontinuierlichen Daten.
- *Ungeeignet für*: Kategorische Daten und Darstellung von Beziehungen zwischen zwei Variablen.

- **Boxplot (Box-Whisker-Diagramm)**

- *Eignung*: Gut geeignet zur Darstellung der Verteilung von Daten und zur Identifizierung von Median, Quartilen und Ausreißern.
- *Ungeeignet für*: Detaillierte Darstellung von einzelnen Datenwerten und Trends über die Zeit.

- **Heatmap**

- *Eignung*: Geeignet zur Darstellung von Dichteverteilungen oder zur Visualisierung von Daten in einer Matrixform, bei der Farben die Werte repräsentieren.
- *Ungeeignet für*: Darstellung von Trends über die Zeit oder Darstellung von Beziehungen zwischen zwei kontinuierlichen Variablen.

- **Blasendiagramm**

- *Eignung*: Ähnlich wie ein Streudiagramm, aber mit einer zusätzlichen Dimension, die durch die Größe der Blasen dargestellt wird. Geeignet zur Darstellung von Beziehungen zwischen drei Variablen.
- *Ungeeignet für*: Kategorische Daten und einfache Zeitreihen.

2 Python Datentypen, Funktionen und Methoden

2.1 Datentypen in Python

Python unterstützt eine Vielzahl von Datentypen, die in veränderbare (mutable) und unveränderbare (immutable) Typen eingeteilt werden können.

- `int` (Ganzzahl) - z.B. 1, 42, -10. (Immutable)
- `float` (Gleitkommazahl) - z.B. 3.14, -0.8, 2.0. (Immutable)
- `complex` (Komplexe Zahl) - z.B. 1 + 2j, -3.2 + 4.1j. (Immutable)
- `str` (Zeichenkette) - z.B. "Hello", 'Python'. (Immutable)
- `list` (Liste) - eine geordnete Sammlung von Elementen, z.B. [1, 2, 3]. (Mutable)
- `tuple` (Tupel) - ähnlich einer Liste, aber unveränderbar, z.B. (1, 2, 3). (Immutable)
- `set` (Menge) - eine ungeordnete Sammlung von einzigartigen Elementen, z.B. {1, 2, 3}. (Mutable)
- `frozenset` - ähnlich wie ein Set, aber unveränderbar, z.B. `frozenset([1, 2, 3])`. (Immutable)
- `dict` (Wörterbuch) - eine Sammlung von Schlüssel-Wert-Paaren, z.B. {'a': 1, 'b': 2}. (Mutable)
- `bool` (Boolescher Wert) - entweder `True` oder `False`. (Immutable)
- `None` - ein spezieller Typ, der das Fehlen eines Wertes repräsentiert. (Immutable)
- `bytes` - eine Sequenz von Bytes (Werten zwischen 0 und 255), z.B. `b'hello'`. (Immutable)
- `bytearray` - ähnlich wie Bytes, aber veränderbar. (Mutable)
- `memoryview` - ein Objekt, das den Zugriff auf interne Daten eines anderen Objekts ermöglicht, ohne dass eine Kopie gemacht wird. (Mutable)

2.2 Grundlegende Funktionen

- `print(*objects, sep=' ', end='\n', file=sys.stdout, flush=False)`: Gibt die Objekte aus, die übergeben werden. Trennt diese standardmäßig mit einem Leerzeichen und fügt am Ende einen Zeilenumbruch hinzu. Die Parameter sind anpassbar.
- `input([prompt])`: Zeigt optional eine Eingabeaufforderung an und gibt die vom Benutzer eingegebene Zeile als Zeichenkette zurück.
- `int(x[, base])`: Konvertiert eine Zahl oder einen String x in eine Ganzzahl. Optional kann eine Basis angegeben werden.

- `float(x)`: Konvertiert eine Zahl oder einen String `x` in eine Gleitkommazahl.
- `range([start,] stop[, step])`: Erstellt ein Objekt vom Typ `range`, das eine Sequenz von Zahlen repräsentiert. Wird oft in Schleifen verwendet.
- `len(s)`: Gibt die Länge (Anzahl der Elemente) eines Objekts (einer Liste, eines Strings, etc.) zurück.
- `type(object)`: Gibt den Typ eines Objekts zurück.
- `str(object)`: Konvertiert ein Objekt in einen String.
- `list(iterable)`: Konvertiert ein iterierbares Objekt (z.B. einen String oder ein Tupel) in eine Liste.
- `tuple(iterable)`: Konvertiert ein iterierbares Objekt in ein Tupel.
- `set(iterable)`: Erzeugt ein Set aus einem iterierbaren Objekt. Ein Set ist eine ungeordnete Sammlung einzigartiger Elemente.
- `dict(**kwarg)`, `dict(mapping, **kwarg)`, `dict(iterable, **kwarg)`: Erzeugt ein neues Wörterbuch.
- `sorted(iterable, *, key=None, reverse=False)`: Gibt eine neue sortierte Liste der Elemente in einem iterierbaren Objekt zurück.
- `sum(iterable, start=0)`: Addiert die Elemente eines iterierbaren Objekts, beginnend mit dem Wert `start`.
- `max(iterable, *[, key, default])` oder `max(arg1, arg2, *args[, key])`: Gibt das größte Element zurück.
- `min(iterable, *[, key, default])` oder `min(arg1, arg2, *args[, key])`: Gibt das kleinste Element zurück.
- `abs(x)`: Gibt den Absolutwert einer Zahl zurück.

2.3 Methoden für Datentyp `str`

- `st.find(s)`: Gibt die Position der gesuchten Zeichenkette `s` in `st` aus (-1, falls sich `s` nicht in der Zeichenkette `st` befinden sollte).
- `st.endswith(suffix)` und `st.startswith(prefix)`: Gibt den Wahrheitswert `True` zurück, wenn die Zeichenkette mit dem angegebenen Suffix endet bzw. startet, ansonsten `False`.
- `st.count(sub)`: Liefert die Anzahl der nicht überlappenden Vorkommen des Teilstrings `sub` in der Zeichenkette `st`.
- `st.lower()` und `st.upper()`: Liefert eine Kopie der Zeichenkette `st` zurück, in der alle Großbuchstaben in Kleinbuchstaben bzw. alle Kleinbuchstaben in Großbuchstaben umgewandelt sind.
- `st.replace(old, new)`: Gibt eine Kopie der Zeichenkette `st` zurück, bei der alle Vorkommen von `old` durch `new` ersetzt werden.
- `st.strip([chars])`: Gibt eine Kopie der Zeichenkette `st` mit entfernten Zeichen `chars` zurück. Wird der Parameter weggelassen, werden führende und abschließende Leerzeichen entfernt.

- `st.split(sep)`: Gibt eine Liste der Wörter aus der Zeichenkette `st` zurück, wobei `sep` als Begrenzungszeichen verwendet wird.

2.4 Methoden für Datentyp `list`

- `lst.append(x)`: Fügt das Element `x` am Ende der Liste hinzu.
- `lst.insert(i, x)`: Fügt ein Element an einer bestimmten Position ein. Das erste Argument `i` ist der Index des Elements, vor dem eingefügt werden soll. Beispiel: `lst.insert(0, x)` fügt das Element `x` am Anfang der Liste ein.
- `lst.remove(x)`: Entfernt das erste Element aus der Liste, dessen Wert gleich `x` ist (löst einen Fehler aus, wenn es kein solches Element gibt).
- `lst.pop([i])`: Entfernt das Element an der angegebenen Position in der Liste und gibt es zurück. Wenn kein Index `i` angegeben wird, entfernt `lst.pop()` den letzten Eintrag in der Liste und gibt ihn zurück.
- `lst.clear()`: Entfernt alle Elemente aus der Liste.
- `lst.index(x)`: Liefert den Index des ersten Elements, dessen Wert gleich `x` ist (löst einen Fehler aus, wenn es kein solches Element gibt).
- `lst.count(x)`: Liefert die Häufigkeit von `x` in der Liste.
- `lst.sort()`: Sortiert die Liste.

2.5 Funktionen im Modul `random`

- `random.randint(a, b)`: Liefert eine zufällige ganze Zahl `x`, so dass $a \leq x \leq b$.
- `random.randrange(a, b[, step])`: Liefert eine zufällige ganze Zahl `x`, so dass $a \leq x < b$ und $(x+a) \% \text{step} = 0$. Der Standardwert für `step` ist 1.
- `random.random()`: Liefert eine zufällige Gleitkommazahl aus dem Intervall $[0.0, 1.0[$.
- `random.uniform(a, b)`: Liefert eine zufällige Gleitkommazahl aus dem Intervall $[a, b[$.
- `random.gauss(mu, sigma)`: Liefert eine normalverteilte Zufallszahl mit Mittelwert `mu` und Standardabweichung `sigma`.
- `random.shuffle(x)`: Mischt die Elemente einer Liste `x`.
- `random.choice(x)`: Liefert ein zufälliges Element aus der Liste `x`.
- `random.sample(x, k)`: Liefert eine Liste mit `k` Elementen, die aus der Liste `x` ausgewählt wurden (ohne Zurücklegen).

2.6 Funktionen im Modul `math`

- `math.ceil(x)`: Liefert die kleinste ganze Zahl größer oder gleich `x`.
- `math.floor(x)`: Liefert die größte ganze Zahl kleiner oder gleich `x`.
- `math.comb(n, k)`: Liefert die Anzahl der Möglichkeiten, `k` Elemente aus `n` Elementen auszuwählen (ohne Zurücklegen, ohne Beachtung der Reihenfolge).

- `math.perm(n, k)`: Liefert die Anzahl der Möglichkeiten, k Elemente aus n Elementen auszuwählen (ohne Zurücklegen, mit Beachtung der Reihenfolge).
- `math.fabs(x)`: Liefert den Absolutwert von x .
- `math.exp(x)`: Gibt e^x zurück, wobei e der Euler'schen Zahl entspricht.
- `math.log(x[, base])`: Mit einem Argument wird der natürliche Logarithmus von x (zur Basis e) zurückgegeben. Mit zwei Argumenten wird der Logarithmus von x zur angegebenen Basis $base$ zurückgegeben.
- `math.pow(x, y)`: Gibt x^y zurück (im Gegensatz zum eingebauten `**`-Operator konvertiert `math.pow()` beide Argumente x und y in Gleitkommazahlen).
- `math.sqrt(x)`: Gibt die Quadratwurzel von x zurück.
- `math.cos(x)`: Liefert den Kosinus eines Winkels x . Analog dazu gibt es `math.sin(x)`, `math.tan(x)`.
- `math.acos(x)`: Liefert den Winkel zurück, dessen Cosinus x ist. Analog dazu gibt es `math.asin(x)`, `math.atan(x)`.

2.6.1 Konstanten im Modul `math`

- `math.pi`: Die mathematische Konstante $\pi \approx 3.141592$.
- `math.e`: Die mathematische Konstante $e \approx 2.718281$.
- `math.inf`: Eine Gleitkomma-Unendlichkeit.

2.7 Funktionen im Modul `statistics`

- `statistics.mean(data)` und `statistics.geometric_mean(data)`: Ermittelt den arithmetischen Mittelwert bzw. das geometrische Mittel der Daten.
- `statistics.median(data)`: Ermittelt den Median der Daten. Bei ungerader Anzahl Datenpunkte: mittlerer Wert; bei gerader Anzahl Datenpunkte: Mittelwert der beiden mittleren Werte.
- `statistics.mode(data)`: Ermittelt den häufigsten Datenpunkt aus diskreten oder nominalen Daten. Bei mehreren Modi wird der erste zurückgegeben.
- `statistics.pvariance(data)` und `statistics.pstdev(data)`: Ermittelt die Populationsvarianz bzw. die Populationsstandardabweichung der Daten.
- `statistics.quantiles(data, n=x)`: Teilt die Daten in x Intervalle mit gleicher Wahrscheinlichkeit und liefert eine Liste von $x - 1$ Schnittpunkten, die die Intervalle trennen.

3 Wiederholungsserie

3.1 Range und for Schleifen

Welche Ausgaben produzieren die folgenden Code-Fragmente?

```
1 # (a)
2 for i in range(3, 0, -1):
3     print(i * " * ")
4
5 ***
6 **
7 *
8
9 # (b)
10 lst = [1, 2, 3, 4, 5]
11 for i in range(1, len(lst)):
12     lst[i] += lst[i - 1]
13 print(lst)
14
15 [1, 3, 6, 10, 15]
16
17 # (c)
18 def list_mutation(lst):
19     for i in range(1, len(lst)):
20         lst[i] = lst[i - 1] + lst[i]
21
22 lst = [1, 2, 3, 4]
23 list_mutation(lst)
24 print(lst)
25
26 [1, 3, 6, 10]
27
28 # (d)
29 for i in range(1, 6):
30     print(i, i ** (i % 3 + 1), sep=" -> ")
31
32 1 -> 1
33 2 -> 8
34 3 -> 3
35 4 -> 16
36 5 -> 125
```

3.2 Quadratfunktion

Schreiben Sie eine Funktion, die als Parameter eine ganze Zahl x erwartet und danach eine Liste der Grösse x mit den Werten $[12, 22, \dots, x2]$ generiert und sie zurückgibt.

```
1 def quadratliste(x):
2     lst = []
3     for i in range(1, x):
4         lst.append(i ** 2)
5     return lst
```

3.3 Listenoperationen und dictionary

Schreiben Sie je eine Funktion, welche eine Liste values (beliebiger Länge) als Parameter erwartet und dann Folgendes erledigt: (a) Weisen Sie jedem Element in values den entsprechenden Index zu (das erste Element referenziert also 0, das zweite Element 1, usw.). (b) Geben Sie alle Werte von values einzeln aus. (c) Erhöhen Sie jedes Element in values um 1. (d) Berechnen Sie die Summe aller Werte aus values und geben Sie das Resultat aus.

```
1 values = [6, 24, 3, 55, 3, 0]
2
3 #Aufgabe a
4 def indexierung(values):
5     indizes = {}
6     for i in values:
7         indizes.update({i : values.index(i)})
8     return indizes
9
10 print("Aufgabe a:", indexierung(values))
11
12 #Aufgabe b
13 def printer(values):
14     for i in values:
15         print(i)
16
17 print("Aufgabe b:")
18 printer(values)
19
20 #Aufgabe c
21 def erhoeher(values):
22     for i in range(len(values)):
23         values[i] += 1
24     return values
25
26 print("Aufgabe c: ", erhoeher(values))
27
28 #Aufgabe d
29 def summarize(values):
30     sum = 0
31     for i in range(len(values)):
32         sum += values[i]
33     return sum
34 print(values)
35 print("Aufgabe d: ", summarize(values))
```

3.4 Boolesche Operatoren

Sind folgenden Booleschen Aussagen "True" oder "False"?

```
1 "Hei" < "Hoi" -> True
2 "gruss" < "Ahoi" -> False
3 "aaa" < "aa" -> False
4 "abrakadabra" < "abrakadabro" -> True
```

3.5 Lokale Maxima

Für diese Aufgabe soll zunächst eine neue Liste erstellt werden, in welcher die einzelnen Elemente mittels Eingabe des Benutzers bestimmt werden. Sie können ähnlich wie in Aufgabe 5 in Serie 10 eine Abbruchbedingung für die Eingabe festlegen. Dann schreiben Sie eine Funktion, welche die vom Benutzer definierte Liste als Parameter entgegennimmt, um dann eine neue Liste mit den *lokalen Maxima* zu definieren und auszugeben. In dieser Übung ist eine Zahl genau dann ein lokales Maximum, wenn ihre beiden Nachbarn strikt kleiner sind. Die erste und letzte Zahl in der Liste haben jeweils nur einen Nachbarn und sind somit nach unserer Definition keine lokalen Maxima. Sie können Ihren Code beispielsweise mit der Liste [2,5,4,3,3,7,6,1] testen. Die Ausgabe sollte [5,7] lauten.

```
1 another = "y"
2 lst = []
3
4 while another == "y":
5     lst.append(input("Geben Sie eine Zahl der Liste hinzu: "))
6     another = input("Wollen Sie eine weitere Zahl hinzufügen? (y/n)")
7
8 def maxima(values):
9     maximas = []
10
11     for i in range(1, len(values) - 1):
12         if values[i] > values[i - 1] and values[i] > values[i + 1]:
13             maximas.append(values[i])
14     return maximas
15
16 print(maxima(lst))
```

3.6 Summe einer Liste < Threshold

Schreiben Sie eine Funktion, welche eine Liste `lst` und einen Schwellwert `threshold` entgegennimmt. Falls die Summe der Einträge in `lst` kleiner ist als `threshold`, gibt Ihre Funktion `True` und sonst `False` zurück. Testen Sie Ihre Funktion mit zwei Eingaben.

```
1 def list_threshold(lst, threshold):
2     sum = 0
3     for i in range(len(lst)):
4         sum += lst[i]
5     check = sum < threshold
6     return check
7
8 my_list = range(0, 11)
9 print(list_threshold(my_list, 3))
10
11 False
```

3.7 FizzBuzz

Schreiben Sie eine Funktion, die für eine ganze Zahl `num` (die als Parameter an die Funktion übergeben wird) überprüft, ob diese durch 3 und/oder durch 5 teilbar ist. Falls die Zahl weder durch 3 noch durch 5 teilbar ist, soll die Zahl ausgegeben werden. Ist die Zahl durch 3 teilbar, soll statt der Zahl „Fizz“ ausgegeben werden. Ist die aktuelle Zahl durch 5 teilbar, soll statt der Zahl „Buzz“ ausgegeben werden. Ist die Zahl sowohl durch 3 als auch durch 5 teilbar, soll „FizzBuzz“ ausgegeben werden. Rufen Sie die Funktion für alle ganzen Zahlen von 1 bis 100 auf.

```
1 def fizzbuzz(num):
2     if num % 3 == 0 and num % 5 != 0:
3         return "Fizz"
4     elif num % 5 == 0 and num % 3 != 0:
5         return "Buzz"
6     elif num % 3 == 0 and num % 5 == 0:
7         return "FizzBuzz"
8     else:
9         return num
10
11 for i in range(1, 101):
12     print(i, fizzbuzz(i), sep=" -> ")
```

3.8 Zufallszahlen

Schreiben Sie je eine Programmieranweisung, die Ihnen eine ganzzahlige Zufallszahl aus folgenden Intervallen erzeugt:

```
1 import random
2
3 print(random.randint(0, 100))
4 print(random.randint(1, 3))
5 print(random.randint(5, 10))
6 print(random.randint(-10, 0))
```

3.9 Geldspielautomat

Schreiben Sie eine Funktion, welche einen Geldspielautomaten simuliert. Hierzu sollen drei zufällige Zahlen zwischen 0 und 9 generiert werden und nebeneinander ausgegeben werden. Wenn zwei bzw. drei Ziffern gleich sind, gewinnt der Benutzer den kleinen bzw. grossen Preis (erzeugen Sie entsprechende Ausgaben). Das Spiel soll so lange fortgeführt werden, bis sich der Benutzer entscheidet, das Spiel zu beenden.

```
1 import random
2
3 another = "y"
4
5 while another == "y":
6     zahl_1 = random.randint(0, 2)
7     zahl_2 = random.randint(0, 2)
8     zahl_3 = random.randint(0, 2)
9     print(zahl_1, zahl_2, zahl_3)
10    if zahl_1 == zahl_2 and zahl_1 == zahl_3:
11        print("Gratulation! Sie haben den grossen Preis gewonnen! Gehen Sie Lotto spielen!")
12    elif zahl_1 == zahl_2 or zahl_1 == zahl_3 or zahl_2 == zahl_3:
13        print("Gratulation! Sie haben den kleinen Preis gewonnen!")
14    else:
15        print("Scchade! Viel Erfolg beim nächsten Versuch!")
16    another = input("Wollen Sie weiterspielen? (y/n)")
```

3.10 Würfelsimulation

Schreiben Sie eine Funktion, welche 100Würfelwürfe simuliert und dabei jeweils zählt, wie oft jede Augenzahl auftaucht. Benutzen Sie dafür eine Liste counter mit 6 Einträgen, welche der Würfelzahl entsprechen. Benutzen Sie dann die Funktion print, um die Liste auszugeben.

```
1 import random
2
3 counter = [0, 0, 0, 0, 0, 0]
4
5 for i in range(1, 101):
6     wurf = random.randint(1, 6)
7     counter[wurf - 1] += 1
8
9 print(counter)
10
11 [18, 11, 21, 14, 18, 18]
```

3.11 Quadratische Funktion mit zwei Lösungen

Schreiben Sie eine Funktion, welche p und q als Parameter erwartet. Die Funktion soll für die Berechnung von x_1 und x_2 die folgende Formel verwenden und diese dann zurückgeben:

$$x_{1,2} = -\frac{p}{2} \pm \sqrt{\left(\frac{p}{2}\right)^2 - q}$$

```
1 import math
2
3 def calculate_x(p, q):
4     # Berechnung innerhalb der Wurzel
5     inner_sqrt = (p / 2) ** 2 - q
6
7     # Überprüfen, ob die Wurzel einen realen Wert hat
8     if inner_sqrt < 0:
9         return "Keine Lösung in den reellen Zahlen"
10
11     sqrt_val = math.sqrt(inner_sqrt)
12
13     # Berechnung von x_1 und x_2
14     x1 = -p / 2 + sqrt_val
15     x2 = -p / 2 - sqrt_val
16
17     return x1, x2
```

3.12 Dateiimport und Keywordsuche

Auf Ilias finden Sie die Datei data.txt. Schreiben Sie eine Funktion, die einen Parameter search_string entgegennimmt. Innerhalb der Funktion wird die Datei data.txt geöffnet und danach wird überprüft, ob einer der Einträge in data.txt dem Parameter search_string entspricht. Ist dies der Fall, gibt die Funktion True und sonst False zurück. Testen Sie Ihre Funktion mit zwei verschiedenen Parametern.

```
1 def search_in_file(search_string):
2     with open("data.txt", "r") as file:
3         for line in file:
4             if search_string in line:
5                 return True
6     return False
7
8 print(search_in_file("Haus"))
```

3.13 scipy und Kurvenplot

In dieser Aufgabe beschäftigen wir uns mit der scipy-Funktion `curve_fit`. Diese Funktion nimmt als Parameter eine Funktion, und zwei Listen `xdata` und `ydata` und gibt ein 2-Tupel zurück. Gehen Sie Schritt für Schritt durch die Aufgabe:

1. Importieren Sie Bibliotheken `numpy`, `matplotlib.pyplot`, `scipy.optimize` und `random`
2. Definieren Sie die folgende Funktion: $f(x, a, b, c) = a \cdot e^{-bx} + c$
3. Kreieren Sie eine Variable `xdata` und weisen Sie ihr eine Liste mit Werten zwischen 0 und 4 in 0.1-Schritten zu. **Tipp:** Die Funktion `numpy.arange(a,b,c)` gibt eine Liste zwischen `a` und `b` mit Einträgen in Schritt `c` zurück.
4. Kreieren Sie eine Variable `ydata` und weisen Sie ihr eine Liste zu. Das Element an Stelle `i` in dieser Liste soll folgenden Wert haben: $f(x_i, 2.5, 1.3, 0.5) + \text{random.uniform}(-0.1, 0.1)$, wobei x_i das `i`-te Element in der Liste `xdata` ist.
5. Rufen Sie die Funktion `curve_fit` auf mit den passenden Parametern und speichern Sie das Resultat in Variablen `popt` und `pcov` ab.
6. Rufen Sie die Funktion `plot` mit den Parametern `xdata` und `ydata` auf.
7. Rufen Sie die Funktion `plot` mit den Parametern `xdata` und $f(xdata, \text{popt}[0], \text{popt}[1], \text{popt}[2])$ auf.

```
1 import numpy
2 import matplotlib.pyplot
3 from scipy.optimize import curve_fit
4 import random
5
6 # Definition der Funktion
7 def f(x, a, b, c):
8     return a * numpy.exp(-b * x) + c
9
10 # Erzeugen der x-Daten
11 xdata = numpy.arange(0, 4, 0.1)
12
13 # Erzeugen der y-Daten
14 ydata = f(xdata, 2.5, 1.3, 0.5) + numpy.random.uniform(-0.1, 0.1, len(
15     xdata))
16
17 # Durchführen der Kurvenanpassung
18 popt, pcov = curve_fit(f, xdata, ydata)
19
20 # Plotten der Daten
21 matplotlib.pyplot.plot(xdata, ydata, 'o', label='data')
22
23 # Plotten der fit-Funktion
24 matplotlib.pyplot.plot(xdata, f(xdata, *popt), label='fit')
25
26 # Hinzufügen einer Legende
27 matplotlib.pyplot.legend()
28
29 # Anzeigen des Plots
30 matplotlib.pyplot.show()
```


3.14 Verständnisfragen

1. Python verfügt über eine Reihe von sogenannten eingebauten Funktionen. Nennen Sie zwei Beispiele solcher Funktionen.
2. Wie rufen Sie in Python eine Funktion bar (ohne Parameter) aus dem importierten Modul foo auf?
3. Python Quellcode muss vor der Ausführung nicht kompiliert werden. Weshalb nicht?
4. Wie können Sie Python Quellcode kommentieren. Geben Sie zwei Möglichkeiten mit je einem Beispiel an.
5. Die folgende Funktion printmax soll den grösseren der beiden Parameter val1 oder val2 ausgeben. Im Quellcode befinden sich ein semantischer und ein syntaktischer Fehler. Markieren Sie beide Fehler (mit sem bzw. syn) und korrigieren Sie diese.

```
1 # a bspw. print() oder round()
2
3 #b
4 import foo
5 foo.bar()
6
7 #c
8 # Python ist eine interpretierte Sprache und wird zur Laufzeit
   interpretiert.
9
10 #d
11 # Beispiel Inline
12
13 """
14 Beispiel grössere Blöcke, für Bspw. Codeinformationen
15 """
16 #e
17 def printmax(val1, val2)
18     if vall < val2:
19         print(val1)
20     else:
21         print(val2)
22
23 def printmax(val1, val2): #Doppelpunt hinzugefügt
24     if val1 > val2: #< zu > geändert und vall auf val1 korrigiert
25         print(val1)
26     else:
27         print(val2)
```

3.15 Gemischt

3.15.1 Counter und bool

Gegeben seien die Variable `full` vom Typ `bool`, die Variable `counter` und eine Konstante `MAX` (beide vom Typ `int`). Schreiben Sie eine Python Anweisung, die zu `counter` den Wert 3 addiert, falls `full` falsch und `counter` kleiner als `MAX` ist. Andernfalls wird `counter` auf 0 gesetzt.

```
1 full = False
2 counter = 0
3 MAX = 10
4
5 while True:
6     if not full and counter < MAX:
7         counter += 3
8     else:
9         counter = 0
10        break
```

3.15.2 while zu for Schleife

Schreiben Sie folgende while-Schleife in eine äquivalente for-Schleife um.

```
1 i = 10
2 while i >= 0:
3     print(i)
4     i -= 2
5
6 for i in range(10, -1, -2):
7     print(i)
```

3.15.3 Zufällige Liste erstellen und auftrennen

Gegeben seien eine Liste `values`, welche positive und negative Zahlen beinhaltet, sowie zwei leere Listen `pos = []` und `neg = []`. Schreiben Sie mithilfe einer for-Schleife ein Code Fragment, so dass sich am Schluss in `pos` alle Werte aus `values` grösser-gleich 0 und in `neg` alle Werte kleiner als 0 befinden.

```
1 import random
2
3 values = random.sample(range(-100, 100), 20) # erstellen einer Liste mit
        20 zufälligen Werten
4 pos = []
5 neg = []
6
7 for i in range(len(values)):
8     if values[i] >= 0:
9         pos.append(values[i])
10    else:
11        neg.append(values[i])
```

3.15.4 Aufsummieren einer Liste mit threshold

Gegeben seien eine Liste `values`, welche Zahlen speichert, eine Variable `total = 0` und eine Variable `threshold`. Schreiben Sie ein Code Fragment, das – solange die Summe `total` kleiner ist als `threshold` und es noch ungelesene Werte in `values` hat – die Werte aus `values` von vorne beginnend zu `total` addiert.

```
1 import random
2
3 values = random.sample(range(-100, 100), 20) # Erstellen einer Liste mit
        20 zufälligen Werten
4 total = 0
5 threshold = 100
6 index = 0
7
8 while total < threshold and index < len(values):
9     total += values[index]
10    index += 1
11 print(total)
```

3.16 Funktionen mit Userinput

- (a) Schreiben Sie eine Funktion ohne Parameter, welche vom Benutzer zunächst eine Zahl über die Tastatur einliest und danach alle geraden, ganzen Zahlen von 0 bis zu dieser Zahl (inklusive) ausgibt.
- (b) Schreiben Sie eine Funktion, die als Parameter eine ganze Zahl `x` erwartet und danach eine Liste der Größe `x` mit den Werten `[1, 2, ..., x]` generiert und zurückgibt.
- (c) Schreiben Sie eine Funktion, welche eine Liste `lst` und einen Wert `value` entgegennimmt. Falls der Wert `value` noch nicht in `lst` vorhanden ist, wird dieser zu `lst` hinzugefügt, andernfalls wird die Liste nicht verändert.

```
1 # (a)
2 num = int(input("Geben sie eine Zahle ein: "))
3 for i in range(0, num + 1, 2):
4     print(i)
5
6 # (b)
7 num = int(input("Geben Sie eine Zahl ein: "))
8 lst = list(range(1, num + 1))
9 print(lst)
10
11 # (c)
12 def is_in_list(lst, value):
13     if value not in lst:
14         lst.append(value)
```

3.17 string-Manipulationen und Matrix

- (a) Schreiben Sie eine Funktion, welche eine Zeichenkette string entgegennimmt, in string alle Grossbuchstaben mit Kleinbuchstaben und alle Zeichen 'c' mit 'k' ersetzt. Die resultierende Zeichenkette soll zurückgegeben werden.
- (b) Gegeben sei eine Datei data.txt, in der auf jeder Zeile ein Eintrag steht. Schreiben Sie eine Funktion namens dataContains(searchString), welche data.txt öffnet und danach überprüft, ob einer der Einträge in data.txt dem Parameter search-string entspricht. Ist dies der Fall, gibt die Funktion True und sonst False zurück.
- (c) Schreiben Sie eine Funktion, welche eine zweidimensionale Liste matrix mit Zahlen entgegennimmt. Ihre Funktion gibt den maximalen Wert zurück, der sich in matrix befindet.

```
1 # (a)
2 def lowercase_and_c_to_k(input_string):
3     mod_string = input_string.replace("c", "k")
4     mod_string = mod_string.lower()
5     return mod_string
6
7 # (b)
8 def dataContains(search_string):
9     with open("data.txt", "r") as file:
10         for line in file:
11             if search_string in line:
12                 return True
13     return False
14
15 print(dataContains("Frog"))
16
17 # (c)
18 def max_matrix(matrix):
19     max = matrix[0][0] #festsetzten des ersten Wertes
20     for i in range(len(matrix)):
21         for j in range(len(matrix[i])):
22             if matrix[i][j] > max:
23                 max = matrix[i][j]
24     return max
```

3.18 Gemischte Fragen, Grundlegendes

- (a) Beurteilen Sie, ob folgende Aussagen jeweils wahr oder falsch sind:

In Python geschriebene Programme werden vor der Ausführung komplett kompiliert.	falsch
Objekte vom eingebauten Datentypen int oder str sind unveränderlich.	wahr
Die Ausgabe von: <pre>title = "Python" print(title[-3:])</pre> lautet: hon	wahr
Die Ausgabe von: <pre>lst1 = [1] lst2 = lst1 lst2 = [2] print(lst1)</pre> lautet: [1]	wahr

- (b) Schreiben Sie den Kopf einer Funktion `summe`, welche einen obligatorischen und zwei optionale formale Parameter definiert (den beiden optionalen Parametern weisen Sie als Standardwert je den Wert 0 zu).
- (c) Schreiben Sie ein Python Code-Fragment, das den Benutzer nach einer Gleitkommazahl fragt, die Eingabe einliest und diese einer Variablen vom Typ `float` zuweist. Mögliche Ein- und Ausgabe:
 Wert eingeben: 7.1
- (d) Gegeben sind zwei Variablen `num1 = 17.8` und `num2 = 99.7`. Schreiben Sie eine `print` Anweisung, welche die aktuellen Werte von `num1` und `num2` mithilfe von Ersatzfeldern ausgibt. Die Ausgabe soll so aussehen:
 Messung 1 ist 17.8 und Messung 2 ist 99.7.
- (e) Gegeben sei ein Wörterbuch `dct`. Schreiben Sie ein Code-Fragment, das überprüft, ob der Schlüssel 17 im Wörterbuch vorhanden ist. Falls ja, soll Schlüssel 17 vorhanden! ausgegeben werden.

```

1 # (b)
2 def summe(c, a=0, b=0, ):
3
4 # (c)
5 num = float(input("Wert eingeben: "))
6
7 # (d)
8 num1, num2 = 17.8, 99.7
9 print(f"Messung 1 ist {num1} und Messung 2 ist {num2}.")
10
11 # (e)
12 dct = {12: "abc", 17: "def", 29: "ghi"}
13
14 if 17 in dct:
15     print("Schlüssel 17 vorhanden!")

```

3.19 Listenmanipulationen

- (a) Gegeben sei eine Funktion `do_something`, welche eine ganze Zahl als formalen Parameter definiert (diese Funktion müssen Sie nicht definieren). Rufen Sie die Funktion `do_something` mit den tatsächlichen Parametern 1, 2, 3, ... , 1000 auf.
- (b) Gegeben sei eine Liste `values`, welche ganze Zahlen speichert (die Liste müssen Sie nicht definieren). Solange die Liste `values` nicht leer ist, entfernen Sie ein zufälliges Element aus der Liste und geben dieses aus.
- (c) Gegeben sei eine Zeichenkette `name` (z.B. `name = "Ürs"`). Schreiben Sie ein Code-Fragment, das jeden einzelnen Buchstaben und den zugehörigen Index auf je einer Zeile ausgibt. Also beispielsweise:
- ```
0 : U
1 : r
2 : s
```
- (d) Gegeben sei eine Liste `values = [1, 1]`. Schreiben Sie ein Code-Fragment, das die Liste mit zehn weiteren Einträgen erweitert, so dass das Element an Position  $i \geq 2$  die Summe der Elemente an Position  $i-1$  und  $i-2$  ist. Die Liste sollte also so aussehen: `[1, 1, 2, 3, 5, 8, ...]`

```
1 # (a)
2 for i in list(range(1, 1000 + 1)):
3 do_something(i)
4
5 # (b)
6 import random
7 while len(values) > 0:
8 values.pop(random.randint(0, len(values) - 1)) # -1 damit Grenze nicht
9 unterschritten wird
10
11 # (c)
12 wort = "Ürs"
13 for i in range(len(wort)):
14 print(f"{i} : {wort[i]}")
15
16 # (d)
17 values = [1, 1]
18 while len(values) < 100: #Bedingung damit keine Endlosschleife entsteht
19 values.append(values[-1] + values[-2])
```

### 3.20 Mathematische Funktionen und statistik über Datei

- (a) Schreiben Sie eine Funktion `func`, welche zwei Gleitkommazahlen `a` und `b` als Parameter erwartet, den Term  $\frac{a^2+b}{b+1}$  berechnet und diesen zurückgibt.
- (b) Schreiben Sie eine Funktion `compare_to`, welche zwei Gleitkommazahlen `f1` und `f2` als Parameter erwartet: Falls die beiden Werte identisch sind, gibt Ihre Funktion 0 zurück, ist der erste Parameter `f1` grösser als `f2`, gibt Ihre Funktion 1 und sonst -1 zurück.
- (c) Schreiben Sie eine Funktion `reset_list`, die eine Liste `lst` als Parameter erwartet. Ihre Funktion ersetzt jeden vorhandenen Wert in der Liste mit dem Wert 0.
- (d) Ihre Aufgabe ist es:

- (a) Numerische Daten aus der Datei data.txt einzulesen
  - (b) Auf den eingelesenen Daten den Mittelwert  $m$  und die Standardabweichung  $s$  zu berechnen
  - (c) Jedes Element  $x$  aus den Daten mit der Formel  $x - ms$  zu normieren.
  - (d) Die Liste mit den normierten Daten auszugeben.
- Hierzu stehen Ihnen die Funktionen `mean_std_dev`, `normalize` und `fill_list` zur Verfügung – nutzen Sie diese Funktionen für Ihr Python Programm.

```
1 # (a)
2 def func(a, b):
3 value = (a ** 2 + b) / (b + 1)
4 return value
5
6 # (b)
7 def compare_to(f1, f2):
8 if f1 == f2:
9 return 0
10 elif f1 > f2:
11 return 1
12 else:
13 return -1
14
15 # (c)
16 def reset_list(lst):
17 for i in range(len(lst)):
18 lst[i] = 0
19
20 # (d)
21 import statistics
22
23 def mean_std_dev(data):
24 return statistics.mean(data), statistics.stdev(data)
25
26 def normalize(data, m, s):
27 normalized_data = []
28 for val in data:
29 normalized_data.append((val - m) / s)
30 return normalized_data
31
32 def fill_list():
33 lst = []
34 with open("data.txt", "r") as file:
35 for line in file:
36 lst.append(float(line.strip("\n")))
37 return lst
38
39 values = fill_list()
40 mean, stdev = mean_std_dev(values)
41
42 normalized_data = normalize(values, mean, stdev)
43
44 print(normalized_data)
```

### 3.21 list\_splitting und multiply into table

- (a) Schreiben Sie eine Funktion `list_splitting`, die eine Liste `lst` und eine ganze Zahl `split_index` als Parameter erwartet. Die Funktion teilt Ihre Liste in zwei Teillisten `lst1` und `lst2` auf, so dass in `lst1` die ersten `split_index` Elemente und in `lst2` die restlichen Elemente gespeichert sind. Also beispielsweise:

`values = [2, 4, 5, 1, 9]` und `split_index = 3`

-> `lst1 = [2, 4, 5]`, `lst2 = [1, 9]`

Ihre Funktion gibt beide Teillisten zurück. Behandeln Sie den folgenden Fall: Falls der Parameter `split_index` grösser ist als die Länge der Liste `lst`, entspricht `lst1` der ursprünglichen Liste `lst` und `lst2` ist eine leere Liste.

- (b) Schreiben Sie eine Funktion `multiply`, die zwei Listen `vals1` und `vals2` als Parameter erwartet. Die Funktion erstellt eine zweidimensionale Liste `table`, so dass in der *i*-ten Zeile und *j*-ten Spalte das Resultat von `vals1[i] * vals2[j]` gespeichert wird. Also beispielsweise:

`vals1 = [2, 5]` und `vals2 = [5, 6, 1]`

-> `table = [[10, 12, 2], [25, 30, 5]]`

Ihre Funktion gibt die zweidimensionale Liste `table` zurück.

```
1 # (a)
2 def list_splitting(lst, split_index):
3 lst1 = []
4 lst2 = []
5 for i in range(len(lst)):
6 if i < split_index:
7 lst1.append(lst[i])
8 else:
9 lst2.append(lst[i])
10 return lst1, lst2
11
12 values = [2, 4, 5, 1, 9]
13 split_index = 3
14
15 lst1, lst2 = list_splitting(values, split_index)
16
17 print(values, lst1, lst2, sep="\n")
18 # Output:
19 # [2, 4, 5, 1, 9]
20 # [2, 4, 5]
21 # [1, 9]
22
23 # (b)
24 def multiply(vals1, vals2):
25 table = []
26 for i in range(len(vals1)):
27 row = []
28 for j in range(len(vals2)):
29 row.append(vals1[i] * vals2[j])
30 table.append(row)
31 return table
32
33 vals1 = [2, 5]
34 vals2 = [5, 6, 1]
35 print(multiply(vals1, vals2))
36 # Output:
37 # [[10, 12, 2], [25, 30, 5]]
```