E.Tutorial
www.et.ethz.ch

Programmieren mit Python Modul 3

Sequenzielle Datentypen, Such- und Sortieralgorithmen, Sequenzanalyse

Selbstständiger Teil

Inhaltsverzeichnis

1 Überblick **4**

2 Teil A: To-Do-Liste **4**
2.1 Einführung . . . 4
2.2 Programmanforderungen . . . 4
2.3 Zwischenschritte . . . 6

3 Teil B: Such- und Sortieralgorithmen **6**
3.1 Vorbereitendes . . . 6
3.1.1 Pollendaten . . . 6
3.2 Suchalgorithmen . . . 7
3.2.1 Aufgaben . . . 7
3.2.2 Erweiterungen . . . 7
3.3 Sortieralgorithmen . . . 7
3.3.1 Aufgaben . . . 7
3.3.2 Erweiterungen . . . 7

4 Teil C: DNA-Sequenzanalyse **8**
4.1 Einführung und Übersicht . . . 8
4.2 Teilaufgaben . . . 8
4.2.1 Auswerten von Einzelbasen . . . 8
4.2.2 Auswerten von Basen-Sequenzen . . . 8
4.2.3 Auswerten einer Liste von Basen-Sequenzen . . . 9

4.3 Testen einer Hypothese mittels DNA-Sequenzanalyse 10
4.3.1 Einführung und Aufgabenstellung 10
4.3.2 Zwischenschritte 12
4.3.3 Erweiterungen 13

5 Bedingungen für die Präsentation **13**

Begriffe

Liste	Listen-Bereich	Suchalgorithmus
Tupel	geschachtelte Liste	
Index	Listen-Abstraktion	Sortieralgorithmus
Dimension	Dictionary	
Listen-Durchlauf	Schlüssel-Wert Paar	Sequenzanalyse

Autoren:

Lukas Fässler

E-Mail:

et@ethz.ch

Datum:

18 October 2024

Version: 1.1

Hash: c539f64

Trotz sorgfältiger Arbeit schleichen sich manchmal Fehler ein. Die Autoren sind Ihnen für Anregungen und Hinweise dankbar!

1 Überblick

Der selbstständige Teil dieses Moduls besteht aus folgenden Teilen:

- Teil A: To-Do-Liste
- Teil B: Such- und Sortieralgorithmen
- Teil C: DNA-Sequenzanalyse

2 Teil A: To-Do-Liste

2.1 Einführung

Bei dieser Aufgabe soll eine To-Do-Liste angelegt und verwaltet werden können. Dabei kommen einfache Listen-Operationen zum Einsatz und es werden Schleifen und Bedingungsprüfungen repetiert.

2.2 Programmanforderungen

Bei diesem Programm soll eine To-Do-Liste erstellt werden. Mit einer Benutzereingabe soll entschieden werden können, ob ein neuer To-Do-Eintrag hinzugefügt, entfernt oder ausgegeben werden soll.

So könnte Ihre Programmausgabe aussehen:

```
Was willst du?
N=Neues ToDo | L=ToDo Löschen | P=ToDoListe |
Beliebige andere Taste=Beenden
N
Neues ToDo: Chemie repetieren
Anzahl ToDos:  1

Was willst du?
N=Neues ToDo | L=ToDo Löschen | P=ToDoListe |
Beliebige andere Taste=Beenden
N
Neues ToDo: Mathe Aufgabe 4
Anzahl ToDos:  2

Was willst du?
N=Neues ToDo | L=ToDo Löschen | P=ToDoListe |
Beliebige andere Taste=Beenden
N
Neues ToDo: Informatik E.Tutorial 3
Anzahl ToDos:  3

Was willst du?
N=Neues ToDo | L=ToDo Löschen | P=ToDoListe |
Beliebige andere Taste=Beenden
L
['Chemie repetieren', 'Mathe Aufgabe 4',
'Informatik E.Tutorial 3']
Welches löschen? 1
Anzahl ToDos:  2

Was willst du?
N=Neues ToDo | L=ToDo Löschen | P=ToDoListe |
Beliebige andere Taste=Beenden
P
Meine ToDos: ['Chemie repetieren',
'Informatik E.Tutorial 3']
Anzahl ToDos:  2

Was willst du?
N=Neues ToDo | L=ToDo Löschen | P=ToDoListe |
Beliebige andere Taste=Beenden
Z
Meine ToDos: ['Chemie repetieren',
'Informatik E.Tutorial 3']
Anzahl ToDos:  2
Viel Erfolg bei der Arbeit!
```

2.3 Zwischenschritte

- Erstellen Sie eine leere Liste.
- Erstellen Sie eine wiederholte User-Eingabe mit einer Menü-Auswahl für verschiedene Aktionen. Das Programm fragt jeweils nach der nächsten Aktion.

> **Tipp:** Verwenden Sie eine **while True-Schleife** mit `break` zum Abbruch der Schleife.

- Prüfen Sie die User-Eingabe und schreiben Sie die Funktionalitäten (Hinzufügen und Entfernen eines Listenelements).
- Geben Sie bei jeder Aktion die Anzahl der Listenelemente aus.

3 Teil B: Such- und Sortieralgorithmen

In diesem zweiten selbstständigen Teil werden Sie eine kleine Auswahl an Such- und Sortieralgorithmen implementieren. Eine kurze Beschreibung finden Sie im Theorieteil.

> **Hinweis:** Bei dieser Aufgabe geht es um das Verständnis der Such- und Sortieralgorithmen. Auch wenn es bei Python Funktionen (z.B. `sort`) gibt, welche die Ergebnisse direkt liefern, wollen wir bei dieser Aufgabe die ganzen Algorithmen ausprogrammieren, um sie anschliessend auch optimieren zu können.

3.1 Vorbereitendes

Laden Sie das Ausgangsprogramm **Pollen.py** auf Ihren Rechner (Code Expert: Öffnen Sie das Projekt `03_Teil_B: Such- und Sortieralgorithmen: Pollen`). Studieren Sie das Programm.

3.1.1 Pollendaten

Die Datei **Pollen.py** (Code Expert: **main.py**) enthält 122 Messwerte für Gräserpollen (Anzahl Pollen pro Kubikmeter Luft). Hierbei handelt es sich Tagesmittelwerte vom 01.03. bis 30.06.2015 in Zürich.

3.2 Suchalgorithmen

3.2.1 Aufgaben

Durchsuchen Sie die Liste mit *linearer Suche* nach dem **höchsten Wert** und geben Sie den **Wert** und die **Position** der Daten in der Konsole aus.

So könnte Ihre Ausgabe aussehen:

```
Laenge:  122
Maximum: 320
Datum: 21.05.2015
```

3.2.2 Erweiterungen

- Suchen Sie den Pollenwert, der am 07.06.2015 gemessen wurde.
- Überlegen Sie sich, was passiert, wenn der gesuchte Wert mehr als einmal vorkommt. Wie müsste ihr Programm darauf reagieren?
- Wie beurteilen Sie den Suchaufwand? Haben Sie Ideen für eine Optimierung?
- Versuchen Sie (nach der Umsetzung der Sortierung) die Liste mittels Binärer Suche zu durchsuchen.

3.3 Sortieralgorithmen

3.3.1 Aufgaben

Implementieren Sie den *Bubble-Sort-Algorithmus* für die Liste **pollen** (Gräserpollendaten), so dass die Werte auf- oder absteigend sortiert dargestellt werden (Beschreibung Bubble-Sort siehe Theorie-Teil).

3.3.2 Erweiterungen

- Geben Sie zu jedem Eintrag der sortierten Pollendaten-Liste das zugehörige Datum aus.
- Wie könnte die Effizienz von *Bubble-Sort* erhöht werden?
- Quantifizieren Sie die Effizienz Ihrer Such- und Sortieralgorithmen (z.B. mit einer Zählung der Rechenschritte oder einer Zeitmessung).
- Speichern Sie die Pollen-Liste in einem separaten File und importieren Sie die Daten (für Details siehe Theorieteil im nächsten Modul).

4 Teil C: DNA-Sequenzanalyse

4.1 Einführung und Übersicht

Die Nukleotiden der **Desoxyribonukleinsäure** (engl. *DNA*) unterscheiden die **4 Basen** **A**denin, **T**hymin, **G**uanin und **C**ytosin. Bei dieser Aufgabe werden Sie verschiedene **DNA-Sequenzen** auswerten, die aus einer **Abfolge der Basen A, T, G, C** bestehen (z.B. `ATCCTG`).

Diese Aufgabe besteht aus folgenden **Teilaufgaben**:

- Auswerten von Einzelbasen
- Auswerten von Basen-Sequenzen
- Auswerten einer Liste von Basen-Sequenzen
- Testen einer Hypothese mittels DNA-Sequenzanalyse

4.2 Teilaufgaben

4.2.1 Auswerten von Einzelbasen

Ein erstes Programm soll die **Häufigkeit und die Stelle einer gegebenen Einzelbase** in einer **DNA-Sequenz beliebiger Länge** auswerten.

Die Eingabe einer gegebenen DNA-Sequenz und einer Base (z.B. `"A"`) kann wie folgt aussehen:

```
dna = "ATCCTATCGAT"
seq = "A"
```

Mögliche Ausgabe:

```
A: 3
pos: [0, 5, 9]
```

> **Tipp:** Speichern Sie die gefundenen DNA-Stellen in einer weiteren Liste.

4.2.2 Auswerten von Basen-Sequenzen

Ein zweites Programm soll auch **längere Basen-Sequenzen** in einer DNA-Sequenz zählen können und die Stellen in der Sequenz angeben.

Die Eingabe einer gegebenen DNA-Sequenz und einer Basen-Sequenz kann wie folgt aussehen:

```
dna = "ATCCTATCGAT"
seq = "AT"
```

Mögliche Ausgabe:

```
AT: 3
pos: [0, 5, 9]
```

> **Tipp:** Verwenden Sie für die Prüfung der Basen-Sequenz eine Teilliste (*Slicing*), die der Länge der gesuchten Sequenz entspricht.

4.2.3 Auswerten einer Liste von Basen-Sequenzen

Ein nächstes Programm soll die **Anzahl einer Liste von Teilsequenzen** in einer gegebenen DNA-Sequenz auswerten und jeweils die Stellen angeben.

Die Eingabe einer gegebenen DNA-Sequenz und einer Liste von Basen-Sequenzen beliebiger Länge kann wie folgt aussehen:

```
dna = "ATCATGAGGGCCTATCTA"
gesucht = ["A","T","G","AT","ATG"]
```

Mögliche Ausgabe:

```
A: 5
pos: [0, 3, 6, 13, 17]
T: 5
pos: [1, 4, 12, 14, 16]
G: 4
pos: [5, 7, 8, 9]
AT: 3
pos: [0, 3, 13]
ATG: 1
pos: [3]
```

> **Tipp:** Verwenden Sie für den Durchlauf der DNA-Sequenz aller Teilsequenzen eine geschachtelte Schleife (*nested loop*). Die innere Schleife durchläuft dabei die gesuchte Teilsequenz, die äussere die einzelnen Teilsequenzen.

4.3 Testen einer Hypothese mittels DNA-Sequenzanalyse

4.3.1 Einführung und Aufgabenstellung

Mabuya ist eine Reptiliengattung aus der Familie der *Skinks* (Glattechsen, Abbildung 1).

Abbildung 1: Mabuya mabouya (Abbildung Mark Stevens from Warrington, UK, CC BY 2.0, https://creativecommons.org/licenses/by/2.0).

Von zahlreichen *Mabuya*-Arten weltweit ist mitochondriale DNA sequenziert und in Bio-Datenbanken abgelegt worden[1]. Bei dieser Aufgabe werden Sie verschiedene DNA-Sequenzen analysieren, um die Verwandschaft und Herkunft dieser Arten zu untersuchen.

Das Archipel *Fernando de Noronha* befindet sich etwa 350 km vor der Küste Basiliens (Abbildung 2). *Mabuya atlantica* ist auf der Insel endemisch. Da das Archipel vulkanisch entstanden ist, stellt sich die Frage, wie und woher die Art auf die Insel gekommen ist. Eine Hypothese besagt, dass die Art mit der Meeresströmung aus Afrika angeschwemmt wurde und nicht vom viel näheren südamerikanischen Festland stammt[2].

Um diese Hypothese zu prüfen, wollen wir DNA-Sequenzen von vier verschiedenen *Mabuya*-Arten bezüglich ihrer prozentualen Übereinstimmung miteinander vergleichen (Abbildung 2):

- *M. agilis* (Südamerikanisches Festland, Brasilien)
- *M. atlantica* (Fernando de Noronha, Brasilien)
- *M. capensis* (Afrikanisches Festland, Südafrika)

[1] z.B. GenBank (www.ncbi.nlm.nih.gov/genbank) betrieben vom National Center for Biotechnology Information(NCBI).

[2] S. Carranza, E.N. Arnold (2003). Investigating the origin of transoceanic distributions: Mtdna shows Mabuya lizards crossed the Atlantic twice, Systematics and Biodiversity, 1:2, 275-282.

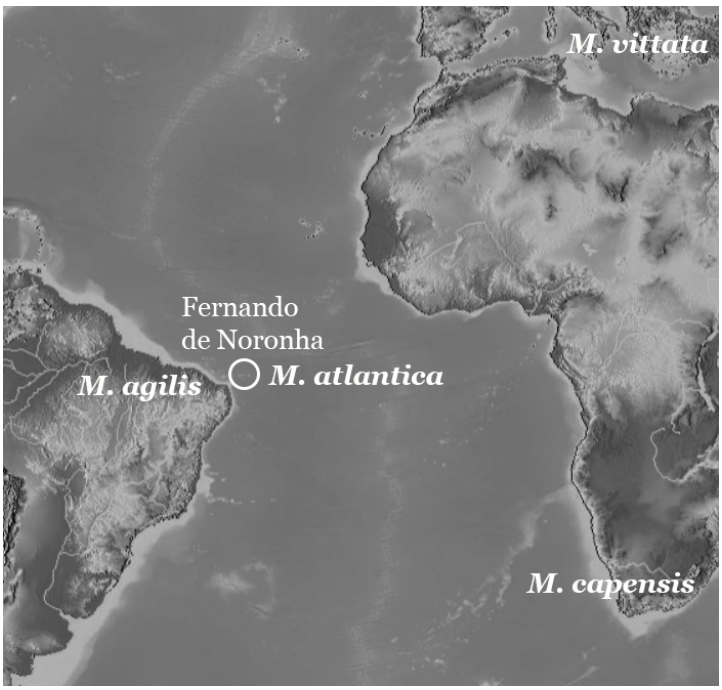

Abbildung 2: Verbreitung der vier Mabuya-Arten. Details siehe Text.

- *M. vittata* (Türkei)

Die gegebenen DNA-Sequenzen bestehen aus den Basen `"A"`, `"T"`, `"G"`, `"C"` sowie Bindestrichen `"-"`. Diese stehen für eine neu eingefügte oder entfernte Base.

	M. atlantica	*M. agilis*	*M. vittata*	*M. capensis*
M. atlantica	1.00			
M. agilis		1.00		
M. vittata			1.00	
M. capensis				1.00

Abbildung 3: Resultate Ihrer DNA-Sequenzanalyse. Prozentuale Übereinstimmung der vier Mabuya-Arten.

4.3.2 Zwischenschritte

- Öffnen Sie die Python-Datei mit den vier DNA-Sequenzen.
- Werten Sie die Einzelbasen der vier *Mabuya*-Arten aus.

Mögliche Ausgabe:

```
Mabuya vittata
A: 142
T: 98
G: 81
C: 106
-: 12
Anzahl Elemente: 439
```

- Vergleichen Sie die DNA-Sequenzen der vier *Mabuya*-Arten bezüglich ihrer **Länge** und ihrer **prozentualen Übereinstimmung** und ergänzen Sie die Vergleichstabelle mit den gefundenen Werten (Abbildung 3).

Mögliche Ausgabe:

```
Mabuya capensis vs. Mabuya vittata
Die beiden Sequenzen haben dieselbe Länge
Anzahl Vergleiche: 415
Anzahl Treffer: 364
Übereinstimmung: 0.8771084337349397
```

Beachten Sie folgende Punkte:

- Der DNA-Sequenzvergleich wird nur ausgeführt, wenn die beiden Sequenzen **dieselbe Länge** haben. Ansonsten wird eine Meldung angezeigt (z.B. `Die Sequenzen müssen dieselbe Länge haben.`) und das Programm wird beendet.
- Haben wir in einer der beiden Sequenzen eine Lücke (Zeichen `"-"`), gehen wir direkt zur nächsten Position ohne den Vergleich durchzuführen.
- Wird ein Vergleich durchgeführt (d.h. weder Sequenz 1 noch Sequenz 2 haben ein `"-"`), wird ein erster Zähler um eins erhöht.
- Enthält Sequenz 1 und 2 dieselbe Base, wird ein zweiter Zähler um eins erhöht.
- Die Prozentuale Übereinstimmung ergibt sich aus der Division von Zähler 2 und Zähler 1.

- Diskutieren Sie auf der Basis Ihrer Resultate die Verwandtschaftsbeziehungen zwischen den *Mabuya*-Arten. Haben Sie eine Erklärung, wie *M. atlantica* auf das Archipel *Fernando de Noronha* gekommen sein könnte?

4.3.3 Erweiterungen

- Schreiben Sie den Code für den Sequenzvergleich einmal als Funktion. Rufen Sie die Funktion auf und übergeben Sie die beiden zu vergleichenden Sequenzen an die Funktion. Die Theorie dazu finden Sie im nächsten Modul.

5 Bedingungen für die Präsentation

Führen Sie einer Assistenzperson die erstellten Programme (To-Do-Liste, Pollen, DNA-Sequenzanalyse) am Bildschirm vor und diskutieren Sie die Resultate.

Überlegen Sie sich, wie Sie einem Laien folgende Fragen erklären würden:

- Was ist der Unterschied zwischen Listen, Tupel und Dictionaries?
- Wie werden Elemente von Listen adressiert?
- Wie funktioniert ein Listen-Durchlauf bei einfachen und geschachtelten Listen?
- Wie funktionieren Listen-Abstraktionen?
- Wie funktioniert die *lineare Suche* und *Bubble-Sort*?
- Wo liegt punkto Effizienz die Schwäche der *linearen Suche* und von *Bubble-Sort* im Vergleich zu anderen Sortier-/Suchalgorithmen?
- Wie funktioniert ein Sequenzvergleich zweier Listen?

Die Begriffe dieses Kursmoduls sollten Sie mit einfachen Worten erklären können.