E.Tutorial
www.et.ethz.ch

Programmieren mit Python Modul 4

Funktionen, Module und Simulationen

Selbstständiger Teil

Inhaltsverzeichnis

Begriffe

Modularität	Rückgabewert	Gültigkeitsbereich von Variablen
Subroutine	Rekursion	
Prozedur	Modul	Modell
Funktion	Bibliotheken	Simulation
Parameter	Zufallszahlen	Zelluläre Automaten

Autoren:

Lukas Fässler, Markus Dahinden

E-Mail:

et@ethz.ch

Datum:

30 October 2024

Version: 1.2

Hash: f4c2bd9

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: Bowling
- Teil B: Schere-Stein-Papier-Spiel
- Teil C: Pandemie-Simulation

2 Teil A: Bowling

2.1 Einführung

Bei dieser ersten Aufgabe soll das Konzept der verschachtelten Listen geübt werden, um 2-dimensionale Strukturen zu verarbeiten.

Beim Bowling werden die Resultate typischerweise in einer Tabelle aufgeschrieben und ausgewertet. Aufgeschrieben wird die Anzahl umgeworfener Pins jeder Runde. Es sind somit Zahlen zwischen 0 (keiner getroffen) und 10 (alle getroffen, ein sogenannter *Strike*) möglich.

	Spieler		
	1	2	3
1. Runde	4	6	2
2. Runde	2	8	0
3. Runde	10	2	5
4. Runde	3	4	5
5. Runde	6	8	10
Summe	25	28	22

Tabelle 1: Resultate eines Bowlingspiels.

2.2 Programmanforderungen

Ihr Programm soll die Resultate von 3 Spielenden über 5 Runden hinweg aufnehmen und auswerten (siehe Tabelle 1). Zum Speichern der Resultate wird eine **verschachtelte Liste** und zur Berechnung der Summen eine **einfache Liste** benötigt.

2.3 Zwischenschritte

Gehen Sie wie folgt vor:

- **Listen bereitstellen:** Stellen Sie mittels Listen-Abstraktion eine verschachtelte Liste `resultate` mit $m \times n$ Elementen (m= Anzahl Spielende und n= Anzahl Runden) und dem Einheitswert 0 sowie eine Liste `summen` mit n Elementen und Einheitswert 0 bereit.

Mögliche Ausgabe:

```
Punkte der 3 Spielenden über 5 Runden:
[[0,0,0,0,0],[0,0,0,0,0],[0,0,0,0,0]]

Totalpunkte:
[0,0,0]
```

- **Einlesen der Resultate:** Lesen Sie die Resultate in die Liste `resultate` ein. Es sollen in jeder Runde die Punkte für jeden Spielenden eingegeben werden können.

> **Tipp:** Setzen Sie hierfür die Funktion `input` zusammen mit der Speicherung der Werte in der geschachtelten Liste in eine geschachtelte Schleife.

> **Alternative:** Setzen Sie randomisierte Werte ein (siehe Theorie-Teil des nächsten Moduls).

Mögliche Ausgabe:

```
Runde 1
Spieler 1
Wert: 4
Spieler 2
Wert: 6
Spieler 3
Wert: 2
Runde 2
...
```

- **Berechnen der Resultate:** Hier soll in der Liste `summen` die Summe der Punkte jeder einzelnen spielenden Person gespeichert werden.
- **Ausgeben der Resultate:** Geben Sie die Punktetabelle (`resultate`) und die Summen (`summen`) auf dem Bildschirm aus.

Mögliche Ausgabe:

```
Punkte der 3 Spielenden über 5 Runden:
[[4, 4, 5, 3, 5], [6, 7, 9, 5, 7], [7, 3, 7, 7, 4]]

Totalpunkte:
[21, 34, 28]
```

Tipp: Um bei `print()` mehrere Elemente in einer Zeile auszugeben, kann mit `end=" "` der Zeilenumbruch unterdrückt werden (siehe Theorie-Teil).

2.4 Erweiterungen

- Erweitern Sie Ihr Programm für beliebig viele Spielende und beliebig viele Runden.
- Ersetzen Sie den Input durch randomisierte Werte.
- Geben Sie am Ende aus, wer wie viele Strikes geschafft hat, und wie oft jede Person keinen Pin getroffen hat.
- Geben Sie aus, wer die meisten Punkte hat.
- Berechnen Sie, in welcher Runde die Spielenden ihren ersten Strike geschafft haben, und geben Sie das Resultat am Bildschirm aus.

3 Teil B: Schere-Stein-Papier-Spiel

3.1 Einführung

Beim Spiel *Schere-Stein-Papier* (oder auch *Schnick-Schnack-Schnuck*) stehen sich zwei Spielende gegenüber und wählen gleichzeitig je eines von drei möglichen Figuren *Schere*, *Stein* oder *Papier* in Form eines Handzeichens (Abbildung 1). Dabei gibt es die folgenden Gewinner:

- *Stein* schlägt *Schere*,
- *Schere* schlägt *Papier*,
- *Papier* schlägt *Stein*.

Zeigen beide Spielenden dieselbe Figur, endet der Spielzug unentschieden. Das Spiel wird über mehrere Runden gespielt. Wer die meisten Spielzüge gewinnt, gewinnt das Spiel.

3.2 Aufgabenstellung und Programmanforderungen

Sie haben die Aufgabe, ein *Schere-Stein-Papier*-Spiel zu implementieren, welches folgende Anforderungen erfüllt:

- Es kann bestimmt werden, wie viele Runden gespielt werden.

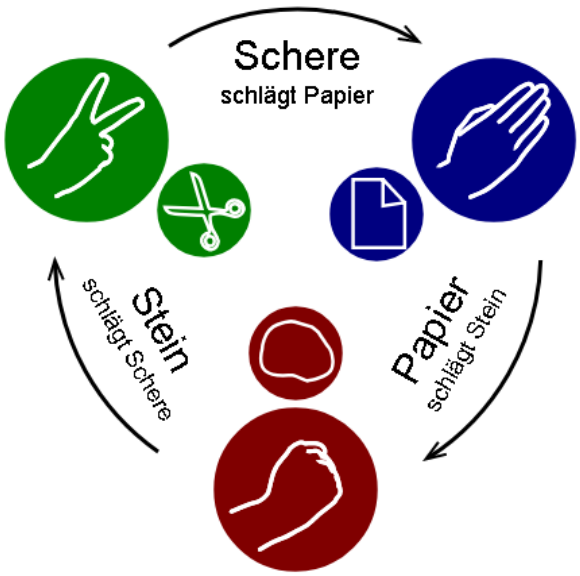

Abbildung 1: Figuren und Handzeichen des Spiels Schere-Stein-Papier (Abbildung VonEnzoklop, CCBY-SA3.0, https://commons.wikimedia.org/w/index.php?curid=27958795).

- Die gespielten Figuren werden zufällig bestimmt.
- Es wird ermittelt, wer das Spiel gewonnen hat.
- Sämtliche Funktionalitäten sind als Funktionen geschrieben.
- Innerhalb von Funktionen werden lokale Variablen verwendet.
- Nutzen Sie keine globalen Variablen.

Mögliche Ausgabe:

```
Wie viele Runden? 5
Spieler 1: Stein, Stein, Schere, Papier, Schere
Spieler 2: Papier, Stein, Papier, Papier, Stein
Spieler 1: 0, 0, 1, 0, 0
Spieler 2: 1, 0, 0, 0, 1
Spieler 2 hat gewonnen
1 : 2
```

Achten Sie bei dieser Aufgabe auf folgende Punkte:

- **Programmbereiche:** Beachten Sie die drei Programmbereiche *Importteil*, *Funktionsteil* und *Hauptprogramm*.
- **Leere Funktionen:** In den noch leeren Funktionen steht der Platzhalter `pass`. Ersetzen Sie diesen Befehl durch Ihren Programmcode.
- **Globale oder lokale Variablen:** Nutzen Sie innerhalb von Funktionen nur lokale Variablen. Benötigt eine Funktion Werte einer globalen Variable, versuchen Sie diese in der Funktion aus den vorliegenden Informationen abzuleiten (z.b. mit `len()`). Falls das nicht geht, fügen Sie einen Funktionsparameter ein.
- **Testfälle nutzen (nur Code Expert-User):** Achten Sie bei dieser Aufgabe besonders auf die Testfälle, die bei jeder Programmausführung automatisch ausgeführt werden. Schauen Sie sich dafür das Resultat im Tab "Test Results" am unteren Bildschirmrand an. Im Folgenden wird jeweils angegeben, welcher Testfall ein bestimmtes Kriterium prüft.

3.3 Ausgangssituation

Bei dieser Aufgabe erhalten Sie einen Ausgangscode, bestehend aus folgendem Programmgerüst:

Importteil:

Enthält die Import-Anweisungen. Ist zu Beginn noch leer.

Funktionsteil:

Enthält als Grundgerüst folgende **5 Funktionen**:

- `spielzug()`: noch leer, keine Eingangsparameter, siehe Schritt 2.
- `ausgabe()`: noch leer, 2 Eingangsparameter, siehe Schritt 3.
- `ermittle_punkte()`: noch leer, 4 Eingangsparameter, siehe Schritt 4.
- `spiel()`: Anzahl Runden als Eingangsparameter, enthält folgende Anweisungen:

  - Stellt für jede:n Spieler:in je eine Liste mit den gespielten Figuren und den erreichten Punkten bereit.
  - Ruft für jede Spieler:in so oft `spielzug()` auf, wie Runden gespielt werden und speichert die gespielten Figuren in die Liste.
  - Ruft für jede Spieler:in `ausgabe()` auf und übergibt die gespielten Figuren.
- `ermittle_gewinner()`: noch leer, 2 Eingangsparameter, siehe Schritte 5 bis 7.

Hauptprogramm:

- Variable `runden` mit 0 initialisiert.
- Funktionsaufruf `spiel()`, wobei `runden` als Funktionsparameter übergeben wird.

3.4 Mögliche Zwischenschritte

Die Zwischenschritte der Aufgabe im Überblick:

- Schritt 1: Benutzereingabe für Anzahl Runden
- Schritt 2: Funktion für zufälligen Spielzug
- Schritt 3: Funktion zur Ausgabe der Spielzüge
- Schritt 4: Funktion zur Bewertung einzelner Spielzüge
- Schritt 5: Funktion zur Ermittlung des Gewinners

3.4.1 Schritt 1: Benutzereingabe für Anzahl Runden (Code Expert Testfall 1)

Schreiben Sie im Hauptprogramm eine **Benutzereingabe** für die Anzahl zu spielenden Runden. Dies wird vom Testfall 1 geprüft.

Mögliche Ausgabe:

```
Wie viele Runden? 5
```

3.4.2 Schritt 2: Funktion für zufälligen Spielzug (Testfall 2)

Schreiben Sie die Funktion `spielzug()`, die mit derselben Wahrscheinlichkeit `"Schere"`, `"Stein"` oder `"Papier"` zurückgibt. An der Ausgabe des Programms ändert sich noch nichts.

3.4.3 Schritt 3: Funktion zur Ausgabe der Spielzüge (Testfall 3)

Schreiben Sie die Funktion `ausgabe()`, die eine Liste mit den gespielten Figuren für jede:n Spieler:in ausgibt. Eingangsparameter sind `name` und `liste`.

Mögliche Ausgabe:

```
Wie viele Runden? 5
Spieler 1: Schere, Papier, Schere, Stein, Schere
Spieler 2: Papier, Papier, Stein, Papier, Papier
```

3.4.4 Schritt 4: Funktion zur Bewertung einzelner Spielzüge (Testfall 4)

Schreiben Sie die Funktion `ermittle_punkte()`, welche die einzelnen Spielzüge bewertet und die erreichten Punkte für jede der Spielenden in je einer Liste speichert. Eingangsparameter sind die Listen `spieler_1`, `punkte_spieler_1`, `spieler_2`, `punkte_spieler_2`.

Rufen Sie die neue Funktion in der Funktion `spiel()` auf, damit die Punkte für jede Runde berechnet werden. Rufen Sie anschliessend die Funktion `ausgabe()` mit den berechneten Punkten für die beiden Spielenden auf, um die Resultate in der Konsole anzuzeigen.

Mögliche Ausgabe:

```
Wie viele Runden? 5
Spieler 1: Schere, Papier, Schere, Stein, Schere
Spieler 2: Papier, Papier, Stein, Papier, Papier
Spieler 1: 1, 0, 0, 0, 1
Spieler 2: 0, 0, 1, 1, 0
```

3.4.5 Schritt 5: Funktion zur Ermittlung des Gewinners (Testfälle 5 bis 7)

Schreiben Sie die Funktion `ermittle_gewinner()`, die bestimmt, wer das Spiel gewonnen hat. Eingangsparameter sind die beiden Listen `punkte_spieler_1` und `punkte_spieler_2`.

Berechnen Sie für jede der Spielenden das Total der erreichten Punkte und geben Sie aus, wer gewonnen hat. Rufen Sie die neue Funktion in der Funktion `spiel()` auf.

Mögliche Ausgabe:

```
Wie viele Runden? 5
Spieler 1: Schere, Stein, Stein, Schere, Stein
Spieler 2: Schere, Schere, Stein, Papier, Schere
Spieler 1: 0, 1, 0, 1, 1
Spieler 2: 0, 0, 0, 0, 0
Spieler 1 hat gewonnen
3 : 0
```

3.5 Erweiterungen

Bei einer **Monte-Carlo-Simulation** werden eine grosse Anzahl gleichartiger Zufallsexperimente durchgeführt und statistisch ausgewertet (siehe Theorie-Teil in einem späteren Modul).

- Erweitern Sie Ihr Schere-Stein-Papier-Spiel aus dem letzten Modul zu einer Monte-Carlo-Simulation, in dem Sie mindestens eine spielende Person durch den Computer spielen lassen, viele Experimente durchführen und auswerten.
- Berechnen Sie ein paar statistische Grössen, um die Resultate vergleichen zu können.
- Testen Sie verschiedene Strategien, wie z.B.:
  - Spieler 2 wählt immer diejenige Figur, die Spieler 1 ein Spielzug vorher gespielt hat.
  - Spieler 2 wechselt die Figur, sobald er ein Spielzug verloren hat.
  - etc.

4 Teil C: Pandemie-Simulation

4.1 Einführung und Übersicht

Bei dieser dritten Modul-Aufgabe werden Sie die **Ausbreitung einer ansteckenden Krankheit** (z.B. Grippevirus) in einer Population mittels **zellulärer Automaten** simulieren. Dabei werden Sie Funktionen schreiben, Python Standardmodule einsetzen und Zeitintervalle mittels 2-dimensionale Listen abbilden. Einige Elemente können aus dem *Game of Life* im E.Tutorial® übernommen werden.

4.2 Aufgabenstellung

Die Pandemie-Simulation soll folgende Anforderungen erfüllen (Abbildung 2):

- Die Population besteht aus einem 2-dimensionalen Spielfeld (auch *Gitter* oder *Grid*) der **Grösse 50x50 plus Rand**.
- Ein einzelnes Feld stellt ein Individuum dar.
- Ein Individuum kann folgende **Zustände** haben:
  - **Wert 0**: Das Individuum ist *gesund* und *ansteckbar*.
  - **Werte 1 bis 7**: Das Individuum ist *erkrankt* und *ansteckend*. Die Zahl zeigt an, wie lange ein Individuum bereits krank ist.
  - **Wert 8**: Das Individuum ist *genesen* und *nicht mehr ansteckbar* (immun).
- Für jede Zeiteinheit (z.B. Tag) wird neu berechnet, ob ein Individuum angesteckt wird oder nicht.
- Die Ansteckung geschieht mit einer definierten Wahrscheinlichkeit von einem infizierten Individuum auf ihre direkten vier Nachbarzellen (siehe Abbildung 2).

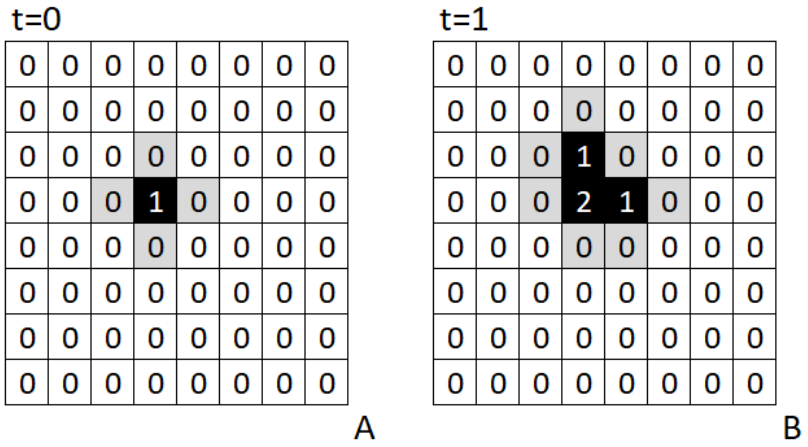

Abbildung 2: Pandemie-Simulation als zellulärer Automat (verkleinerte Darstellung). In Gitter A (Ausgangssituation t=0) ist ein Individuum erkrankt (Wert=1). Vier direkte Nachbarzellen (grau eingefärbt) können mit einer Wahrscheinlichkeit p angesteckt werden. In der nächsten Zeiteinheit (t=1) sind zwei weitere Individuen infiziert worden. Der Zustand der in Gitter A erkrankten Zelle hat auf den Wert 2 gewechselt. In Gitter B sind nun potentiell 7 Individuen ansteckbar.

4.3 Ausgangssituation

Bei dieser Aufgabe erhalten Sie als Grundgerüst ein Code-Skelett aus den **4 Funktionen** `setup()`, `output()`, `count()` und `update()` sowie eine Variable `grid` im Hauptprogramm.

4.4 Zwischenschritte

Eine Grundversion der Pandemie-Simulation kann in folgenden Zwischenschritten erstellt werden:

- Schritt 1: Spielfeld mit Ausgangssituation erstellen
- Schritt 2: Alle Individuen werden krank
- Schritt 3: Simulation läuft eine bestimmte Zeit
- Schritt 4: Zufall einbauen
- Schritt 5: Zustände werden für nächste Zeiteinheit übernommen
- Schritt 6: Ansteckung einbauen
- Schritt 7: Genesung und Immunität einbauen
- Schritt 8: Zählung durchführen und grafisch darstellen

4.4.1 Schritt 1: Spielfeld mit Ausgangssituation erstellen (Testfälle 2 bis 4)

Als Erstes erstellen wir ein Spielfeld in Form eines Gitters, welches in der Simulation unsere Population repräsentieren soll. Der Grundzustand eines Individuums ist gesund und ansteckbar (Zustand=0).

- Erstellen Sie in der Funktion `setup()` eine geschachtelte Liste `A` der Grösse 52x52 (50x50 plus Rand von der Breite einer Zelle).
- Setzen Sie alle Listen-Elemente auf den Wert 0 (=*gesund* und *ansteckbar*).
- Geben Sie die Liste `A` mit `return` zurück.
- Rufen Sie die Funktion `setup()` im Hauptprogramm auf und speichern Sie die zurückgegebene Liste unter dem Namen `grid`.

Programmieren Sie die vorgegebene Funktion `output()` so aus, dass das Spielfeld *ohne* Randzellen auf der Konsole geprintet wird:

- Rufen Sie die Funktion im Hauptprogramm auf und übergeben Sie die Liste `grid`.
- Nutzen Sie zwei verschachtelte Schleifen um die Liste zu printen.
- Geben Sie die Zeitinheit `Tag 0` auf der Konsole aus.

Mögliche Ausgabe: Ausgabe des Gitters `grid` in der Konsole nach dem Funktionsaufruf von `output()` (hier verkleinert auf die Grösse 8x8):

```
Tag 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
```

4.4.2 Schritt 2: Alle Individuen werden krank (Testfälle 5 bis 7)

Nach einer ersten Zeiteinheit sollen alle Individuen auf den Wert 1 (=*infiziert*) gesetzt werden. Dies erledigen wir in der Funktion `update()`. Die Werte zu Beginn dieser Zeiteinheit übergeben wir an Gitter `A`. Die Werte *nach* dieser Zeiteinheit speichern wir in einem neuen Gitter `B`. Dieses Gitter dient uns zum Speichern der Resultate eines Zeitschrittes, bevor wir die Werte als Ausgangspunkt für den nächsten Zeitschritt mit `return` ins Hauptprogramm zurückgeben und dort ins Gitter `grid` zurückspeichern (siehe Abbildung 2).

- Erstellen Sie eine neue Funktion `update()` mit einem neuen lokalen Resultate-Gitter `B` der gleichen Grösse wie `A`.

- Setzen Sie alle Elemente von `B` *ohne* Randzellen auf den Wert 1 (der Rand soll 0 bleiben).
- Rufen Sie die Funktion `update()` im Hauptprogramm auf und übergeben Sie das `grid` als Parameter. Geben Sie die Werte von `B` zurück ins Hauptprogramm und aktualisieren Sie damit die Werte von `grid`.
- Geben Sie im Hauptprogramm die geänderten Werte von `grid` erneut mit `output()` in der Konsole aus und ergänzen Sie die entsprechende Zeiteinheit.

Mögliche Ausgabe: Zweimalige Ausgabe des Gitters `grid` mit der Funktion `output()` vor und nach dem Funktionsaufruf von `update()` (verkleinerte Darstellung von `grid`):

```
Tag 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0

Tag 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
```

Eine erste Veränderung ist bereits zu sehen. Alle Individuen werden auf einen Schlag krank (Zustandswert=1).

4.4.3 Schritt 3: Simulation läuft eine bestimmte Zeit (Testfall 8 und 9)

Nun kommt die Zeitreihe ins Spiel. Die Funktionsaufrufe von `update()` und `output()` sollen eine bestimmte Anzahl Mal wiederholt werden.

- Definieren Sie eine neue Variable (z.B. `tEnd`) und setzen Sie sie auf einen Wert (z.B. 10).
- Wiederholen Sie den Funktionsaufruf `update()` und der zweite Aufruf von `output()` mit einer Schleife so oft, wie in der Variable `tEnd` definiert.
- Passen Sie die Zeitangaben entsprechend an, indem Sie die Laufvariable nutzen.

Konsoleninhalt nach jedem Zeitschritt mit Zeitverzögerung löschen:

Um die Konsole nach jedem Zeitschritt zu löschen und die berechneten Werte mit einer Zeitverzögerung anzuzeigen, können sie bei jedem Schritt `sleep` des Moduls `time` mit der Variable `delay = 1` einsetzen und die Konsolenausgabe mit `clear` des Moduls `os` löschen. Damit Sie berechnete Werte überprüfen können, kann es aber auch hilfreich sein, den `clear` Befehl zwischenzeitlich auszukommentieren oder `delay` auf 0 zu setzen. Auch zum Laufenlassen der Tests sollten Sie den Delay auf 0 setzen (sonst laufen die Tests möglicherweise sehr lange).

Mit folgendem Code wird der Text `Hallo Nr.` gefolgt von der Nummer zehn Mal ausgegeben und die Konsolenausgabe mit einer zeitlichen Verzögerung gelöscht:

```
import time
import os

delay = 1
for i in range(0,10):
  print("Hallo Nr.", i+1)
  time.sleep(delay)
  os.system('clear')
```

Mögliche Ausgabe: Ausgabe des Gitters `grid` über die Zeit. Noch erkranken alle Individuen nach dem ersten Zeitschritt, die Dauer kann über eine Variable eingestellt werden (die Zeitschritte 2 bis 9 sind nicht dargestellt):

```
Tag 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0

Tag 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
...

Tag 10
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
```

4.4.4 Schritt 4: Zufall einbauen (Testfall 10 und 11)

In einer nächsten Version sollen nur noch gewisse Individuen mit einer **bestimmten Wahrscheinlichkeit** erkranken.

- Führen Sie für die Ansteckungswahrscheinlichkeit eine neue (globale) Variable (z.B. `p`) ein und setzen Sie sie auf einen Wert (z.B. 0.25).
- Setzen Sie in der Funktion `update()` den Wert 1 nur noch mit einer Wahrscheinlichkeit von `p`. Übergeben Sie hierzu `p` an `update()`.

Tipp: So führen Sie eine Anweisung mit einer bestimmten Wahrscheinlichkeit p aus:

```
import random

p = 0.5
if random.random() < p:
    # Anweisung wird mit der
    # Wahrscheinlichkeit p ausgeführt.
```

Mögliche Ausgabe: Bei jedem Durchgang wird für alle Individuen neu bestimmt, ob Sie angesteckt werden ($p = 0.25$, die Resultate einiger Zeitschritte sind nicht dargestellt):

```
Tag 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0

Tag 1
1 0 0 0 0 0 1 0
0 0 1 0 0 1 0 0
0 1 0 1 1 0 0 0
0 0 0 0 0 1 0 1
0 0 0 0 0 0 0 0
0 1 1 0 1 0 0 0
1 0 0 0 0 0 1 1
0 0 1 1 0 0 0 0
...

Tag 10
0 1 0 0 0 0 0 1
0 0 0 0 1 0 0 0
0 0 1 1 0 0 0 1
0 0 0 0 0 1 0 1
0 0 0 1 0 0 0 0
1 0 1 0 1 0 1 0
0 1 0 0 0 0 0 0
1 0 1 1 0 0 0 1
```

4.4.5 Schritt 5: Zustände werden für nächste Zeiteinheit übernommen

Nun sollen nur gesunde und ansteckbare Individuen (Zustand=0) erkranken können und einmal infizierte Individuen über die Zeit krank bleiben.

- Ergänzen Sie in der Funktion `update()` die Bedingungsprüfung wie folgt:

```
FALLS Zelle in Gitter A den Wert 0 hat:
    wird der Wert in B mit der
    Wahrscheinlichkeit p auf 1 gesetzt.
SONST:
    wird Zelle in Gitter B der Wert derselben
    Zelle von A übernommen.
```

Mögliche Ausgabe: Nur gesunde und ansteckbare Individuen können krank werden. Die Anzahl der kranken Individuen nehmen über die Zeit zu ($p = 0.5$, die Resultate einiger Zeitschritte sind nicht dargestellt):

```
Tag 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0

Tag 1
0 1 1 0 1 0 0 1
0 1 0 1 1 1 1 0
1 0 0 0 0 0 1 1
1 1 0 1 1 0 1 1
0 0 0 1 0 1 1 1
0 1 0 1 0 1 0 1
1 0 0 1 0 1 1 0
0 0 1 1 1 0 1 0

Tag 2
1 1 1 0 1 1 0 1
0 1 0 1 1 1 1 0
1 1 0 1 0 1 1 1
1 1 1 1 1 0 1 1
0 1 0 1 0 1 1 1
1 1 0 1 1 1 1 1
1 0 0 1 1 1 1 0
0 1 1 1 1 0 1 1
...

Tag 10
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
```

4.4.6 Schritt 6: Ansteckung einbauen (Testfall 12)

Bis hierher können unsere Individuen zwar erkranken, es findet aber noch keine Ansteckung statt. In der Realität findet eine **Ansteckung** dann statt, wenn sich ein Individuum in der Nähe eines bereits infizierten Individuums aufhält. Die Ansteckung soll von nun an nur noch möglich sein, wenn **einer der vier direkten Nachbarn** krank ist (siehe Abbildung 2). Da wir auf diese Weise in unserer Simulation zu Beginn keine kranken Individuen haben, müssen wir beim Programmstart ein erstes Individuum auf *krank* setzen. Um die Ausbreitung der Krankheit besser verfolgen zu können, wollen wir jedoch zuerst in der Konsole nur noch die kranken Individuen anzeigen.

- Passen Sie die Funktion `output()` so an, dass nur Werte die grösser als 0 sind angezeigt werden.
- Setzen Sie bei Programmstart ein Individuum (z.B. A[10][15] oder ein zufällig bestimmtes) auf den Zustand *krank*.
- Ergänzen Sie in der Funktion `update()` die Verzweigung wie folgt:

```
FALLS Zelle in Gitter A Wert 0 hat:
    FALLS ein Nachbar von A ein Wert grösser als 0 hat:
        wird der Wert in B mit der
        Wahrscheinlichkeit p auf 1 gesetzt.
SONST:
    wird Zelle in Gitter B der Wert derselben
    Zelle von A übernommen.
```

Tipp: Um ein Individuum aus dem Gitter zufällig zu bestimmten, können für die x- und y-Koordinate je eine Zufallszahl bestimmen.

Mögliche Ausgabe: Die Anzahl der erkrankten Individuen sollten durch die Ansteckung über die Zeit ausgehend vom Erstinfizierten zunehmen (verkleinerte Darstellung, die Resultate einiger Zeitschritte sind nicht dargestellt):

```
Tag 1


    1



Tag 2

    1
    1 1



Tag 3

    1 1
  1 1 1
    1
...
```

4.4.7 Schritt 7: Genesung und Immunität einbauen (Testfall 13)

Angenommen der Krankheitserreger verursacht eine Krankheit, die eine bestimmte Zeit (z.B. 8 Tage) dauert. Danach sind die Individuen **wieder gesund**. Bei vielen Krankheitserregern führt eine einmalige Infektion nach der Genesung zur **Immunität**, d.h. sie können nicht mehr erneut angesteckt werden.

Bei der nächsten Version unserer Simulation möchten wir den **Status** der Individuen mit den **Werten 2 bis 8 erweitern**. Der Wert gibt an, wie lange das Individuum schon krank ist. Zeigt der Status den Wert 1 bis 7, kann das Individuum selber nicht mehr angesteckt werden, kann aber während dieser Zeit andere Nachbar-Individuen anstecken. Wird der Wert 8 erreicht, ist das Individuum *genesen* und *immun*. Das heisst, es kann nicht mehr angesteckt werden und steckt auch keine anderen Individuen mehr an.

- Erweitern Sie die Verzweigung in der Funktion `update()` wie folgt:

```
FALLS Zelle in Gitter A den Wert 0 hat:
    FALLS ein Nachbar von A den Wert grösser als 0 und
    kleiner als 8 hat:
        wird der Wert in B mit der
        Wahrscheinlichkeit p auf 1 gesetzt.
ANDERNFALLS Zelle von Gitter A einen Wert kleiner als 8 hat:
    wird der Wert von Zelle A um 1 erhöht
    und in B gespeichert.
SONST:
    wird Zelle in Gitter B der Wert
    derselben Zelle von A übernommen.
```

Mögliche Ausgabe: Bei den infizierten Individuen wird der Krankheitsstatus angezeigt und nach jeder Zeiteinheit erhöht. Bei gesunden (Wert=0) und genesenen (Wert=8) Individuen wird der Status nicht angezeigt (verkleinerte Darstellung, die Resultate einiger Zeitschritte sind nicht dargestellt):

```
Tag 1


    2
    1

Tag 2


    3
    2

Tag 3

    1
    4
  1 3
...

Tag 6

  2
  3 4
1 4 7 3
1 2 6 1

Tag 7

  3
  4 5 1
2 5   4
2 3 7 2
    1

Tag 8

  4 1 1
  5 6 2
3 6   5
3 4   3
1   2
...
```

4.4.8 Schritt 8: Zählung durchführen und grafisch darstellen (Testfall 14 bis 16)

Um die Entwicklung über die Zeit zu veranschaulichen, möchten wir zum Abschluss ein paar System-Werte berechnen und eine Auswertung mit den Gesundheitsstatus der Individuen erstellen. Anschliessend sollen diese Werte noch mit der Bibliothek `matplotlib` visualisiert werden.

- Erstellen Sie im Hauptprogramm drei neue Listen (z.B. `n_gesund`, `n_infiziert`, `n_genesen`) der Länge `tEnd`, um die Anzahl infizierter Individuen pro Zeiteinheit abzuspeichern.
- Rufen Sie für jedes Zeitintvervall die Funktion `count()` im Hauptprogramm auf und übergeben Sie das `grid` als Parameter.
- Passen Sie die gegebene Funktion `count()` so an, dass diese die Anzahl *gesunder* (Status = 0), *infizierter* (Status grösser als 0 und kleiner als 8) und *genesener* Individuen (Status = 8) zählt. Speichern Sie diese drei Werte in die vorgegebene, lokale Liste und geben Sie diese mit `return` ins Hauptprogramm zurück.
- Speichern Sie die einzelnen Werte der zurückgegebenen Liste in die jeweilige Liste im Hauptprogramm.
- Geben Sie bei jedem Durchgang die aktuellen Werte in der Konsole aus, wie z.B.:

```
Tag 20
Gesund:     1976    (79.04%)
Infiziert:  283     (11.32%)
Genesen:    241     (9.64%)
```

> **Tipp:** Mit dem Steuerzeichen `\t` können Sie innerhalb eines Strings einen Tabulator setzen.

- Visualisieren Sie die Resultate in einem matplotlib-Diagramm.

Bei einer hohen Ansteckungsrate und genügend Laufzeit erwarten wir gemäss Modell folgende zeitliche Verteilung (Abbildung 3):

- Schauen Sie die Verteilung an. Inwiefern unterscheidet sich Ihre Verteilung mit der Verteilung des Modells?

4.5 Erweiterungen

In dieser Projektaufgabe haben Sie eine Grundversion der Pandemie-Simulation erstellt. Diese können wir verwenden, um einige **Untersuchungen am Modell** vorzunehmen. Wie Sie sicher bemerkt haben, entsprechen einige Elemente der Simulation noch nicht ganz der Realität. Hier finden Sie einige Ideen, wie Sie Ihre Simulation erweitern und mit dem Verlauf der Grundversion vergleichen können;:

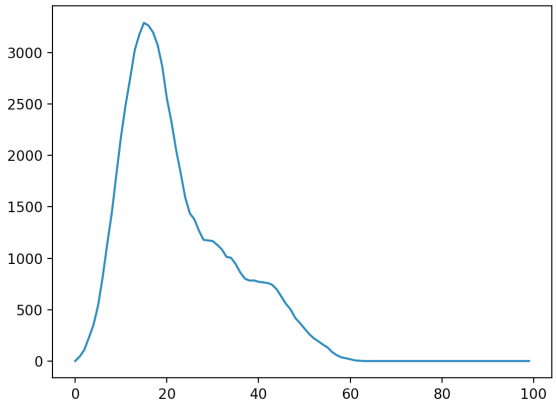

Abbildung 3: Grafische Darstellung des zeitlichen Verlaufs der Anzahl angesteckter Individuen während der Pandemie.

- Effekt von Kontakteinschränkungen
- Impfungen einbauen
- Mortalität einbauen
- Abnahme der Ansteckungswahrscheinlichkeit und der Immunität
- Reisen einbauen

4.5.1 Effekt von Kontakteinschränkungen (leicht)

Überschreitet die Zunahme der Ansteckungen einen bestimmten Wert, sollen behördliche Massnahmen (z.B. *Social Distancing*, *Lockdown*) ergriffen werden.

- Reduzieren Sie im Fall einer starken Zunahme von Neuansteckungen die Anzahl der Kontakte.
- Untersuchen Sie den Einfluss der die Reduktion der Anzahl der Kontakte hat. Wie verändert sich die Anteckungskurve?

4.5.2 Impfungen einbauen (mittel)

Durch die Verabreichung eines Impfstoffes sollen Individuen vor einer Ansteckung geschützt werden.

- Simulieren Sie den Effekt einer Impfung. Bei jedem Durchgang werden zufällig eine bestimmte Anzahl an Individuen direkt auf den Gesundheitsstatus *genesen, nicht ansteckbar* gesetzt.
- Wie verändert sich die Kurve, wenn Sie die Anzahl Impfungen erhöhen?

4.5.3 Mortalität einbauen (mittel)

Viele Infektionskrankheiten können Individuen nicht nur krank machen, sondern auch töten. Die Wahrscheinlichkeit, mit der infizierte Individuen sterben, wird Tödlichkeit oder *Letalität* genannt. Infektionskrankheiten haben häufig eine charakteristische Todesrate (z.B. 1%). Diese soll in unser Modell einfliessen und beim Programmstart als Simulationsparameter eingebaut werden.

- Passen Sie Ihre Simulation so an, dass bei jedem Zeitschritt infizierte Individuen mit einer bestimmten Wahrscheinlichkeit dem Krankheitserreger erliegen. Erweitern Sie hierzu auch ihre Statistik, z.B.:

```
Tag 20
Gesund:     1976    (79.04%)
Infiziert:  269     (10.76%)
Genesen:    212     (8.48%)
Gestorben:  16      (0.76%)
```

4.5.4 Abnahme der Ansteckungswahrscheinlichkeit und der Immunität (mittel)

Viele Krankheiten sind zu Beginn am ansteckendsten. Machen Sie die Ansteckungswahrscheinlichkeit abhängig von der Dauer der Krankheit der Überträger.

- Passen Sie Ihre Simulation so an, dass geheilte Personen nach einer gewissen Zeit (z.B. vier Tagen) wieder angesteckt werden können.
- Wie verhält sich die Kurve, wenn die Ansteckungswahrscheinlichkeit und die Immunität mit der Zeit abnimmt?

4.5.5 Reisen einbauen (anspruchsvoll)

Bei unserem bisherigen Modell bleiben alle Individuen an ihrem Platz und haben nur Kontakt zu ihren direkten 4 Nachbaren. Dies soll sich nun ändern. Die Inidividuen sollen innerhalb eines gegebenen Radius "reisen" können. Hierfür wird ein neuer Simulationsparameter eingeführt: der *Reise-Radius*. Innerhalb diser Entfernung sollen sich Individuen verschieben und dort Individuen anstecken können. Für unser Modell bedeutet das, dass ein Individuum bei jedem Zeitschritt die Möglichkeit bekommt, innerhalb des Reiseradius zufällig andere Individuen anzustecken. Bisher hatten wir einen Reiseradius von 1, d.h. nur direkte Nachbarn können angesteckt werden. Nun kann sich dieser Radius erhöhen, z.B. auf 2 oder 3.

- Passen Sie Ihre Simulation so an, dass der Reiseradius eingestellt werden kann und in diesem Bereich vier Individuen zufällig angesteckt werden können.
- Wie verhält sich die Kurve mit und ohne Reisebeschränkung?

5 Bedingungen für die Präsentation

Führen Sie einer Assistenzperson die erstellten Programme am Bildschirm vor und diskutieren Sie die durch die Simulation erzeugten Resultate. Überlegen Sie sich, wie Sie einem Laien folgende Fragen erklären würden:

- Wozu setzt man beim Programmieren Funktionen ein?
- Welche Arten von Funktionen gibt es und wie kann man Sie unterscheiden?
- Wie funktioniert bei Funktionen die Wertübergabe?
- Wo sind lokale und globale Variablen gültig?
- Was ist die Aufgabe von `return`?
- Was ist ein rekursiver Aufruf?
- Was ist der Sinn von Modulen? Wie können Module in Python-Programmen verwendet werden?
- Wie werden auf Elemente von verschachtelten Listen zugegriffen?
- Welche Elemente gehören zu einer Simulation?
- Wofür braucht man Testfälle?

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