Geburtstage

Dickbauer Y., Moser P., Perner M.

PS Computergestützte Modellierung, WS 2016/17

November 28, 2016

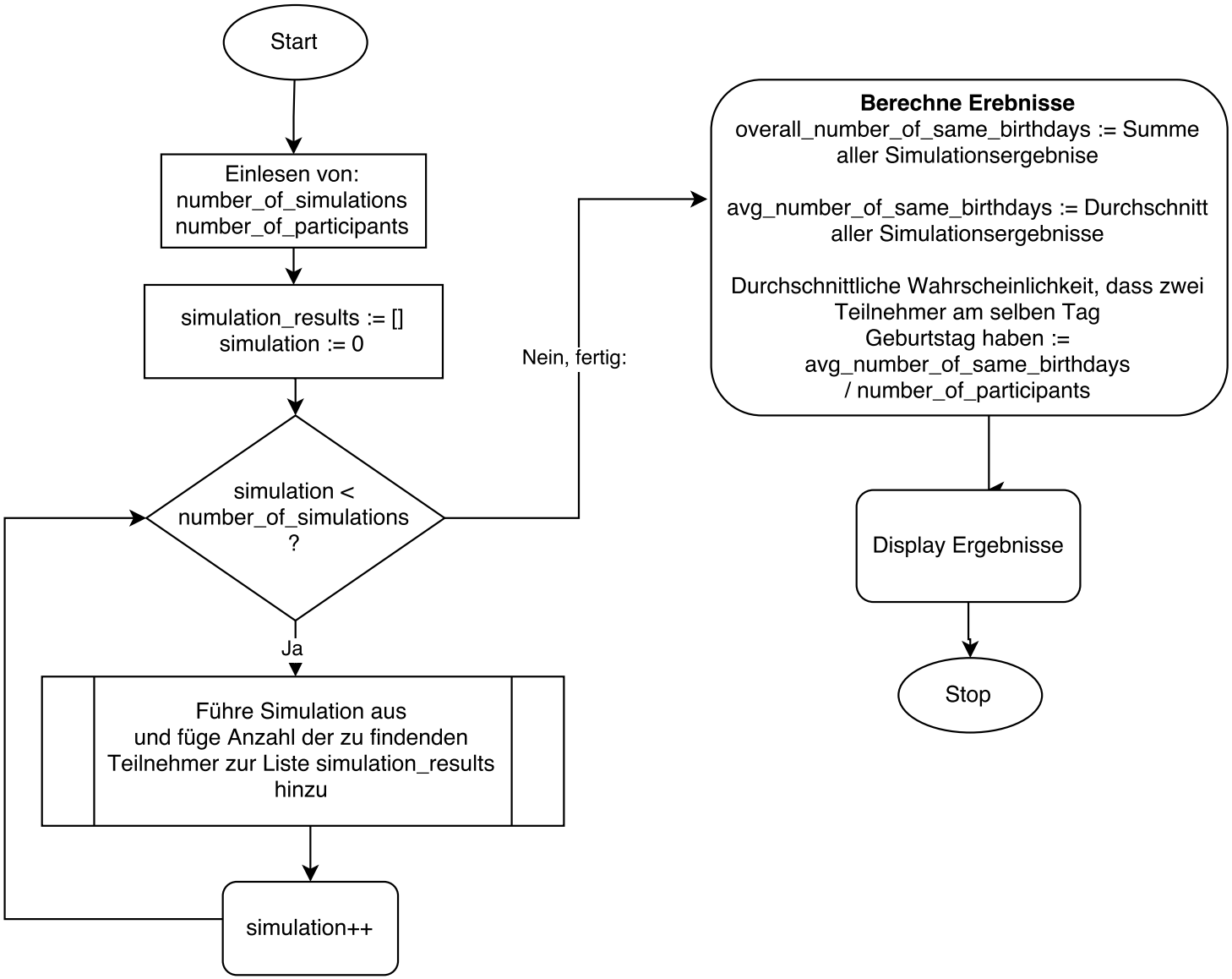
Outline

Aufgabenstellung

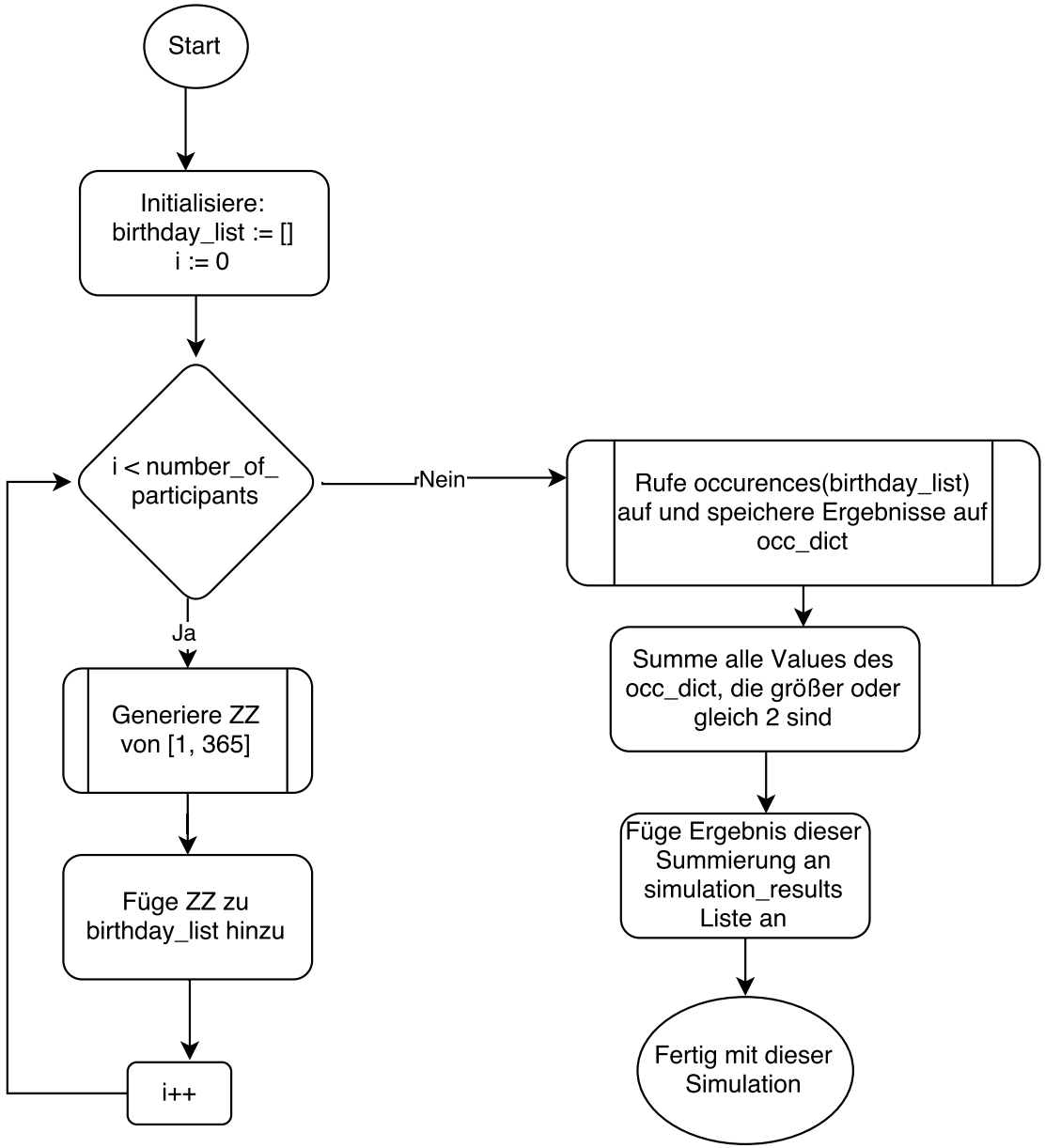
In einem Seminar sitzen x Teilnehmer. Ermitteln Sie n¨aherungsweise mittels Simulation die Wahrscheinlichkeit, dass zwei Teilnehmer am selben Tag Geburtstag haben.

- Eingabe: Anzahl an Teilnehmer, Anzahl an Simulationsdurchläufen
- Durchschnittliche Wahrscheinlichkeit, dass zwei Teilnehmer am selben Tag Geburtstag haben.
- Output optional: Ausgabe der Geburtstage und Markierung der Tage, wo mehr als eine Person Geburtstag hat.

Flow Chart


Start
Einlesen von:
number_of_simulations
number_of_participants
simulation_results := []
simulation := 0
simulation < number_of_simulations ?
Nein, fertig:
Ja
Führe Simulation aus
und füge Anzahl der zu findenden Teilnehmer zur Liste simulation_results hinzu
simulation++
Berechne Erebnisse
overall_number_of_same_birthdays := Summe aller Simulationsergebnisse
avg_number_of_same_birthdays := Durchschnitt aller Simulationsergebnisse
Durchschnittliche Wahrscheinlichkeit, dass zwei Teilnehmer am selben Tag Geburtstag haben := avg_number_of_same_birthdays / number_of_participants
Display Ergebnisse
Stop

Flow Chart - Ein Simulationsdurchgang



Main Funktion - Programmeinstieg

```
1   def main():
2       #user input:
3       number_of_simulations, number_of_participants = user_input((
4           ('Number␣of␣simulations', int, 100),
5           ('Number␣of␣participants', int, 40)), DEBUG)
6
7       simulation_results = []
8       for simulation in range(number_of_simulations):
9           # generate a list of birthdays - with length of participants
10          birthday_list = []
11          for i in range(number_of_participants):
12              birthday_list.append( int(random_number_from_interval(0, DAYS_IN_A_YE
13          # count the same candidates:
14          occ_dict = occurrences(birthday_list)
15          assert sum(occ_dict.values()) == number_of_participants
16          number_of_same_values = sum([value if value >= 2 else 0 for key, value ir
17          simulation_results.append(number_of_same_values)
18
19          # output only if option enabled:
20          if OPTION:
21                  # print those where there are more than two... see source
```

Main Funktion - Programmeinstieg

```
1       # result of all simulations
2       overall_number_of_same_birthdays = sum(simulation_results)
3       avg_number_of_same_birthdays = overall_number_of_same_birthdays /
4                                                       number_of_simulations
5       simulation_result_p = avg_number_of_same_birthdays /
6                                           number_of_participants
7
8       print('Ergebnisse aller {0}/{0} Simulationen:'.format(number_of_simulations))
9       print('Durchschnittlich haben {} von {} Personen am gleichen Tag Geburtstag.
10          avg_number_of_same_birthdays, number_of_participants))
11      print('WSKL ueber alle Simulationen: p={}%'.format(simulation_result_p * 100)
```

Funktion random_number_from_interval(..)

- Diese Funktion verlangt zwei Eingabeparameter *lower* und *upper*
- Gibt eine (pseudo)Zufallszahl (*float*) im Intervall [*lower*, *upper*) zurück
- *random.random()* ist eine Funktion der Python Standardbibliothek, welche ein Zufallszahl (*float*) im Intervall [*lower*, *upper*) zurück gibt
- Mersenne Twister Methode wird als Generator der ZZ verwendet[1] [2]

```
1  def random_number_from_interval(lower, upper):
2      val = random.random()
3      return lower + (upper -lower) * val
```

[1] https://docs.python.org/3.5/library/random.html
[2] https://en.wikipedia.org/wiki/Mersenne_Twister

Funktion user_input(input_vars, [use_defaults])

- Diese Funktion verlang vom User die geforderten Eingabeparameter und gibt diese als von der Programmiererin gewünschten Datentyp wieder zurück
- Funktion verlangt als ersten Eingabeparameter die Liste *input_vars*
- Falls *use_defaults* == *True* wird der User nicht nach Eingabe gefragt (Dient zum Testen)
- Diese Liste besteht wiederrum aus Listen mit je Länge = 3:
  - 0: Text, welcher dem User ausgegeben wird
  - 1: Datentyp (int/float/str)
  - 2: Default value: Dieser Wert wird zurueckgegeben, falls *use_defaults* == *True*

```
1   x, y = user_input((
2       ('Geben Sie einen X Wert ein', int, 10),
3       ('Geben Sie einen Y Wert ein', int, 5), False):
```

Funktion occurences(input_list)

- Diese Funktion verlangt eine Liste voller Zahlen als Eingabeparameter *input_list*
- Diese Liste wird durchsucht auf alle vorkommenden Zahlen und zählt mit, wie oft welche Zahl in der Liste enthalten ist
- Zurückgegeben wird ein *dictionary*, welches jeweils die Zahl als *key* und die Anzahl dieser keys in der Eingabeliste als *value*
- Eingabeliste [1,2,3] gibt zurück: {1: 1, 2: 1, 3: 1}
- Eingabeliste [1,1,2] gibt zurück: {1: 2, 2: 1}

Code:

```
1  def occurrences(input_list):
2      occ_dict = {}
3      for elem in input_list:
4          if elem in occ_dict:
5              occ_dict[elem] += 1
6          else:
7              occ_dict[elem] = 1
8      return occ_dict
```

Beispiel anhand fixer Zufallszahlen

- number_of_simulations := 2
- number_of_participants := 10

Simulationsdurchgang 0:

- Geburtstage := {1, 10, 20, 50, 10, 20, 40, 12, 40, 365}
- Ergebnis: 6 (2x10, 2x20, 2x40)

Simulationsdurchgang 1:

- Geburtstage := {2, 10, 77, 40, 15, 20, 77, 12, 40, 365}
- Ergebnis: 4 (2x40, 2x77)

Im Durchschnitt haben 5 Personen am gleichen Tag Geburtstag $\frac{6+4}{2} = 5$.
Das sind 50% der Teilnehmer $\frac{6+4}{2*10}$ oder $\frac{5}{10}$

Anhang: Modifikation des Source Codes um Demo Beispiel zu erhalten

```
1     # Aendere random_number_from_interval() in lib.py wie folgt:
2   ZZ = [1, 10, 20, 50, 10, 20, 40, 12, 40, 365,
3         2, 10, 77, 40, 15, 20, 77, 12, 40, 365]
4   i = -1
5   def random_number_from_interval(lower, upper):
6       global i
7       i += 1
8       return ZZ[i]-1
```