[**LABOR 1** 2](#_Toc184653335)

[**COMBINARI + VARIATIONEN** 2](#_Toc184653336)

[**MODALITATI DE A DISTRIBUI CATEVA BILE IN MAI PUTINE COMPARTIMENTE** 3](#_Toc184653337)

[**LABOR 2** 4](#_Toc184653338)

[**WKT. AUS X PERSONEN MINDESTENS Y DEN GL. GEBURTSTAG** 4](#_Toc184653339)

[**GEOMETRISCHE WKT.** 4](#_Toc184653340)

[**ZEICHNUNG- ZUFALLIGER PUNKT- KREIS-QUADRAT** 5](#_Toc184653341)

[**THEORETISCHE WKT- DER PUNKT IN KREISINNEN** 5](#_Toc184653342)

[**LABOR 3** 10](#_Toc184653343)

[**URNE-SIMULATIONEN UND TH. WKT.** 10](#_Toc184653344)

[**HISTOGRAM ZEICHNEN -RELATIVE HAUFIGKEIT** 11](#_Toc184653345)

[**ESTIMARI DACA SE ARUNCA UN ZAR/ ZILE DE NASTERE** 14](#_Toc184653346)

[**LABOR 4** 16](#_Toc184653347)

[**Zufallige Werte der ZG** 16](#_Toc184653348)

[**ZG = Wert, generiere Werte + Schatzungen** 16](#_Toc184653349)

[**ZG mit binomialer Verteilung** 17](#_Toc184653350)

[**Computer + Rechner mit Virus (avem Wkt. dat deja)** 18](#_Toc184653351)

[**Zufallsgenerator fur Verteilung** 19](#_Toc184653352)

[**Erwartungswert + Summe Kugeln + rel. Hauf.** 20](#_Toc184653353)

[**LABOR 5** 22](#_Toc184653354)

[**Standabweichung + Erwrtungswert + Dichtefunktion** 22](#_Toc184653355)

[**Exponentialverteilung** 23](#_Toc184653356)

[**Anhand Simulationen schatzen** 24](#_Toc184653357)

[**Dichtefunktion + Verteilungsfunktion** 25](#_Toc184653358)

**LABOR 1**

**Variationen, Kombinationen, Permutationen**

|  |
| --- |
| **import** random **from** random **import** sample **import** math **from** math **import** factorial, perm |
| **import** itertools **from** itertools **import** permutations, combinations  **help**("random.sample") **help**("math.factorial") |
| *#help("itertools.permutations")*  **print**(perm(4,4)) **print**(perm(4,2)) |

!!!LA VARIATION CONTEAZA ORDINEA

**COMBINARI + VARIATIONEN**

**ALLE** PERMITATIONEN VON SICHER + COUNT:

print("Alle Permutationen von 'sicher' sind: ", list(permutations("sicher")))  
anzahl = list(permutations("sicher"))  
print(len(anzahl))

**ZWEI** ZUFALLIGE PERMUTATIONEN VON SICHER:

print("2 zufallige Permutationen von 'sicher' sind: ", (random.sample("sicher", len("sicher"))), "und", random.sample("sicher", len("sicher")))

ALLE VARIATIONEN JE **ZWEI** BUCHSTABEN AUS SICHER + COUNT:

print("Alle Variationen von 'sicher' sind: ", list(("sicher")), 2)  
anzahl\_v = list((permutations("sicher"),2))  
print(len(anzahl\_v))

GESAMTE ANZAHL DER VARIATIONEN JE **DREI BUCHSTABEN** VON MATHE:

word= "MATHE"  
var = list(permutations(word, 3))  
count = len(var)  
print("Die Anzahl der Variationen von 3 Buchstaben: ", count

ALLE KOMBINATIONEN (**NU** CONT. ORDINEA) JE **DREI** BUCHST. VON MATHE:

comb = list(combinations(word, 3))  
print("Alle Kombinationen von 3 Buchstaben von MATHE sind: ",comb)

GESAMTE ANZAHL DER KOMBINATIONEN JE **DREI** BUCHST. AUS MATHE:

anz\_comb = list(combinations(word, 3))  
count = len(anz\_comb)  
print("Gesamte Anzahl der Kombinationen ist: ",count)

ALLE PERMUTATIONEN VON AABB:

import more\_itertools  
from more\_itertools import distinct\_permutations  
M=list(distinct\_permutations("AABB"))  
print(M)  
m = len(M)  
print("Anzahl Permutationen von AABB mit Wiederholung:",m)  
for p in distinct\_permutations("1112"): print("".join(p))  
n = len(list(distinct\_permutations("1112")))  
print("Anzahl Permutationen von 1112 mit Wiederholung:",n)

KOMBINATION **MIT WIEDERHOLUNG** VON ABC JE **ZWEI**:

import itertools  
from itertools import combinations\_with\_replacement  
print("Alle Kombinationen von ABC je 2, mit Wiederholung")  
combin = list(combinations\_with\_replacement("ABC", 2))  
print(combin)  
k = len(combin)  
print("Anzahl Kombinationen von ABC je 2 mit Wiederholung:",k)

**MODALITATI DE A DISTRIBUI CATEVA BILE IN MAI PUTINE COMPARTIMENTE**

**MOGLICHKEITEN** 6 ROTE KUGELN IN 4 KARTONS AUFZUTEILEN?

import math  
from itertools import combinations\_with\_replacement  
  
kugel = 6  
karton = 4  
  
comb= list(combinations\_with\_replacement(range(1, karton + 1), kugel))  
count=(len(comb))  
print("Anzahl: ",count)

**LABOR 2**

**Probabilitati**

**WKT. AUS X PERSONEN MINDESTENS Y DEN GL. GEBURTSTAG**

**WKT.** DURCH WIEDERHOLTE SIMULATIONEN DASS AUS **23** PERSONEN **MINDESTENS** **ZWEI** DEN GLEICHEN GEBURTSTAG HABEN + **THEORETISCHE** **WKT.**

23 Personen- mindestens 2 denselben Geburtstag- 365 Tage (scrie 366 fiind pana la n-1)

import numpy  
  
n = 1000  
c = 0  
for i in range(n):  
 s = numpy.random.randint(low = 1, high = 366, size = 23)  
 s = set(s)  
 if len(s) < 23:  
 c = c + 1  
  
print("Wk. Simulationen:" , c/n)  
  
#andere Variante  
p = 1  
for i in range(23):  
 p = p \* ((365-i)/365)  
  
p = 1 - p  
print("Wk. theoretische:" , p)

**GEOMETRISCHE WKT.**

**GEOMETRISCHE WKT.** (ZUFALLIGE PUNKTE ZEICHNEN)

|  |
| --- |
| **import** numpy **from** matplotlib.pyplot **import** axis,plot,figure,show,legend fig = figure() axis("square") |
| axis((0, 1, 0, 1))  X=numpy.random.random(25)  Y=numpy.random.random(25)  plot(X,Y,"bo")  fig.suptitle("Beispiel 1 ",fontweight ="bold") |
| show() fig = figure() axis("square") axis((0, 1, 0, 1)) plot(X,numpy.square(X),"g\*") *# zufallige Punkte auf dem Bild der Funktion f(x)=xˆ2* plot(X[-1],numpy.square(X[-1]),"g\*",label="$f(x)=xˆ2$") *#Beispiel fur label* |
| legend(loc="upper left")  fig.suptitle("Beispiel 2 ",fontweight ="bold")  show() |

**ZEICHNUNG- ZUFALLIGER PUNKT- KREIS-QUADRAT**

EIN ZUFALLIG GEWAHLTER **PUNKT IM QUADRAT [0,1] × [0,1]** SICH AUCH IN DEM **EINGESCHRIEBENEN KREIS** BEFINDET:

N ZUFALLIGE PUNKTE + COUNT W.V. KREISINNEREN (K) + ZEICHNE DIE PUNKTE (VERSCH FABRE IN + AUS)

* EUKL. DISTANZ ZW. 2 PUNKTEN

import numpy  
from matplotlib.pyplot import axis,plot,figure,show,legend  
fig = figure()  
axis("square")  
axis((0, 1, 0, 1))  
X=numpy.random.random(25)  
Y=numpy.random.random(25)  
plot(X,Y,"bo")  
fig.suptitle("Beispiel 1 ",fontweight ="bold")  
show()  
fig = figure()  
axis("square")  
axis((0, 1, 0, 1))  
plot(X,numpy.square(X),"g\*") # zufallige Punkte auf dem Bild der Funktion f(x)=xˆ2  
plot(X[-1],numpy.square(X[-1]),"g\*",label="$f(x)=xˆ2$") #Beispiel fur label  
legend(loc="upper left")  
fig.suptitle("Beispiel 2 ",fontweight ="bold")  
show()

**THEORETISCHE WKT- DER PUNKT IN KREISINNEN**

**THEORETISCHE WKT.** DASS DER PUNKT IM **KREISINNEN** IST:

import numpy as np  
from matplotlib.pyplot import axis, plot, figure, show, legend  
  
N = 1000  
X = np.random.random(N)  
Y = np.random.random(N)  
  
r = 1/2  
colors = []  
l =1  
aria\_quadrat = l\*\*2  
aria\_kreis= 3.14 \* r\*\*2  
  
  
count = 0  
for i in range(N):  
 d = (X[i] - 1/2)\*\*2 + (Y[i] - 1/2)\*\*2  
 if d <= r\*\*2:  
 colors.append('g')  
 count += 1  
 else:  
 colors.append('r')  
  
  
fig = figure()  
axis("square")  
axis((0, 1, 0, 1))  
plot(X, Y, "bo")  
  
for i in range(N):  
 plot(X[i], Y[i], marker='o' ,color=colors[i])  
  
p1 = count/N  
p2 = aria\_kreis/ aria\_quadrat  
print(f"Punkte im Kreis: ", p1)  
print(f"Wk. theoretisch: ", p2)  
  
fig.suptitle("Beispiel 1: Punkte im Quadrat und im Kreis", fontweight="bold")  
  
show()

QUADRAT MIT SEITENLANGE 1- ZUFALLIG. PUNKT A

MAN VERBINDET A MIT DEN SPITZEN DES QUADRATES => 4 DREIECKE MIT GEMEINSAMER SPITZE IN A

ANHAND SIMULATIONEN

WKT. GENAU 1 WINKEL IN A = STUMPT

WKT. GENAU 2 WINKEL IN A = STUMPF

+ ZEICHNUNG

import numpy as np  
import matplotlib.pyplot as plt  
import math  
  
  
def angle\_between(v1, v2):  
 *"""Calculate the angle in radians between vectors 'v1' and 'v2'."""* v1\_u = v1 / np.linalg.norm(v1)  
 v2\_u = v2 / np.linalg.norm(v2)  
 return np.arccos(np.clip(np.dot(v1\_u, v2\_u), -1.0, 1.0))  
  
  
def count\_obtuse\_angles(point, square\_corners):  
 *"""Count the number of obtuse angles formed with point A and square corners."""* obtuse\_angles = 0  
 for i in range(len(square\_corners)):  
 v1 = square\_corners[i] - point  
 v2 = square\_corners[(i + 1) % len(square\_corners)] - point  
 angle = angle\_between(v1, v2)  
 if angle > np.pi / 2:  
 obtuse\_angles += 1  
 return obtuse\_angles  
  
  
def simulate\_and\_plot(N, square\_corners):  
 one\_obtuse = []  
 two\_obtuse = []  
  
 for \_ in range(N):  
 point = np.random.rand(2)  
 obtuse\_count = count\_obtuse\_angles(point, square\_corners)  
  
 if obtuse\_count == 1:  
 one\_obtuse.append(point)  
 elif obtuse\_count == 2:  
 two\_obtuse.append(point)  
  
 one\_obtuse = np.array(one\_obtuse)  
 two\_obtuse = np.array(two\_obtuse)  
  
 plt.figure(figsize=(6, 6))  
 if len(one\_obtuse) > 0:  
 plt.scatter(one\_obtuse[:, 0], one\_obtuse[:, 1], color='blue', label='Exactly 1 Obtuse Angle')  
 if len(two\_obtuse) > 0:  
 plt.scatter(two\_obtuse[:, 0], two\_obtuse[:, 1], color='red', label='Exactly 2 Obtuse Angles')  
 plt.legend()  
 plt.title('Random Points with 1 or 2 Obtuse Angles in Square')  
 plt.xlabel('x')  
 plt.ylabel('y')  
 plt.axis('equal')  
 plt.show()  
  
 return len(one\_obtuse) / N, len(two\_obtuse) / N  
  
  
# Square corners  
square\_corners = np.array([[0, 0], [1, 0], [1, 1], [0, 1]])  
  
# Number of simulations  
N = 10000  
  
# Simulate and plot  
prob\_one\_obtuse, prob\_two\_obtuse = simulate\_and\_plot(N, square\_corners)  
print(f"Probability of Exactly 1 Obtuse Angle: {prob\_one\_obtuse}")  
print(f"Probability of Exactly 2 Obtuse Angles: {prob\_two\_obtuse}")

**BILD MIT N = 500** ROTEN ZUFALLIGEN PUNKTEN GENERIERT WIRD + WKT. EIN ZUFALLIG AUSGEW. PUNKT AUS QUADRAT IM INNEREN DES UNTEREN ODER OBEREN DREIECKES BEFINDET.

import numpy as np  
import matplotlib.pyplot as plt  
  
  
def is\_inside\_triangle(p, triangle):  
 *"""Check if point p is inside the triangle."""* def sign(p1, p2, p3):  
 return (p1[0] - p3[0]) \* (p2[1] - p3[1]) - (p2[0] - p3[0]) \* (p1[1] - p3[1])  
  
 d1 = sign(p, triangle[0], triangle[1])  
 d2 = sign(p, triangle[1], triangle[2])  
 d3 = sign(p, triangle[2], triangle[0])  
  
 has\_neg = (d1 < 0) or (d2 < 0) or (d3 < 0)  
 has\_pos = (d1 > 0) or (d2 > 0) or (d3 > 0)  
  
 return not (has\_neg and has\_pos)  
  
  
def simulate\_and\_plot\_with\_triangles\_and\_square(N):  
 # Triangles' vertices  
 lower\_triangle = np.array([[0, 0], [1, 0], [0.5, 0.5]])  
 upper\_triangle = np.array([[0, 1], [1, 1], [0.5, 0.5]])  
  
 points = np.random.rand(N, 2)  
 inside\_triangle = [is\_inside\_triangle(point, lower\_triangle) or is\_inside\_triangle(point, upper\_triangle) for point  
 in points]  
  
 plt.figure(figsize=(6, 6))  
 # Points inside triangles  
 plt.scatter(points[inside\_triangle, 0], points[inside\_triangle, 1], color='blue', label='Inside Triangles')  
 # Points outside triangles  
 plt.scatter(points[~np.array(inside\_triangle), 0], points[~np.array(inside\_triangle), 1], color='red',  
 label='Outside Triangles')  
  
 # Drawing the square boundary  
 square = plt.Rectangle((0, 0), 1, 1, fill=False, edgecolor='black', linewidth=1)  
 plt.gca().add\_patch(square)  
  
 # Drawing the triangles  
 plt.plot(\*zip(\*np.append(lower\_triangle, lower\_triangle[0:1], axis=0)), color='green', linestyle='dashed')  
 plt.plot(\*zip(\*np.append(upper\_triangle, upper\_triangle[0:1], axis=0)), color='green', linestyle='dashed')  
  
 plt.title(f'Random Points in Square with Triangles (N = {N})')  
 plt.xlabel('x')  
 plt.ylabel('y')  
 plt.legend()  
 plt.axis('equal')  
 plt.show()  
  
 probability = sum(inside\_triangle) / N  
 print(f"Probability of being inside either triangle: {probability}")  
  
  
# Number of points  
N = 5000  
  
# Calculate the probability and plot with triangles and square  
simulate\_and\_plot\_with\_triangles\_and\_square(N)

**LABOR 3**

**URNE-SIMULATIONEN UND TH. WKT.**

Urne = 6 rote K, 4 blaue K und 6 grune K. Man zieht 3 Kugeln hintereinander ohne Zurucklegen. Ereignisse:  
A:**mindestens** eine rote Kugel entnommen  
B:**alle** entnommenen Kugeln = dieselbe Farbe

1) Anhand Simulationen Wkt. fur: P (A), P (B), P (A ∩ B), P (B|A)

2) Theoretische Wkt. fur P (A), P (B), P (A ∩ B), P (B|A)

import random  
  
N = 10000 # Number of simulations  
  
count\_A = 0  
count\_B = 0  
count\_A\_and\_B = 0  
  
for \_ in range(N):  
 # Correct distribution of the balls: 6 red, 4 blue, 6 green  
 balls = random.sample(['r'] \* 6 + ['b'] \* 4 + ['g'] \* 6, k=3)  
  
 if 'r' in balls:  
 count\_A += 1  
  
 if len(set(balls)) == 1:  
 count\_B += 1  
  
 if len(set(balls)) == 1 and balls[0] == 'r':  
 count\_A\_and\_B += 1  
  
P\_A = count\_A / N  
P\_B = count\_B / N  
P\_A\_and\_B = count\_A\_and\_B / N  
P\_B\_ohne\_A = P\_A\_and\_B / P\_A  
  
# Theoretical probabilities  
total\_balls = 6 + 4 + 6 # Total number of balls  
  
# P(A): Probability of drawing at least one red ball  
prob\_no\_red = (10 / total\_balls) \* (9 / (total\_balls - 1)) \* (8 / (total\_balls - 2))  
prob\_red = 1 - prob\_no\_red

# P(B): Probability that all drawn balls are of the same color  
prob\_same\_color = (6 / total\_balls) \* (5 / (total\_balls - 1)) \* (4 / (total\_balls - 2)) + \  
 (4 / total\_balls) \* (3 / (total\_balls - 1)) \* (2 / (total\_balls - 2)) + \  
 (6 / total\_balls) \* (5 / (total\_balls - 1)) \* (4 / (total\_balls - 2))  
  
# P(A ∩ B): Probability of both events happening (drawing three red balls)  
prob\_red\_and\_same\_color = (6 / total\_balls) \* (5 / (total\_balls - 1)) \* (4 / (total\_balls - 2))  
  
# P(B|A): Probability of B given A  
prob\_same\_color\_given\_red = prob\_red\_and\_same\_color / prob\_red  
  
print("Simulated P\_A:", P\_A)  
print("Theoretical P\_A:", prob\_red)  
print("Simulated P\_B:", P\_B)  
print("Theoretical P\_B:", prob\_same\_color)  
print("Simulated P\_A\_and\_B:", P\_A\_and\_B)  
print("Theoretical P\_A\_and\_B:", prob\_red\_and\_same\_color)  
print("Simulated P\_B|A:", P\_B\_ohne\_A)  
print("Theoretical P\_B|A:", prob\_same\_color\_given\_red)

**HISTOGRAM ZEICHNEN -RELATIVE HAUFIGKEIT**

Zeichne ein **Histogramm der relativen Haufigkeiten** der Zahlen die beim 500-maligen Wurfeln erhalten wurden. Was stellt das blau gezeichnete Histogramm dar? Wie verandert sich das Bild wenn N = 2000?

import numpy  
from random import randrange  
from matplotlib.pyplot import bar, show, hist, grid, legend, xticks  
N=2000  
daten = [randrange(1,7) for \_ in range(N)]  
#print(daten)  
z, count = numpy.unique(daten, return\_counts=True)  
d=dict([(z[i],count[i]/N) for i in range(0,6)])  
print("Zahl und relative Haufigkeit:",d)  
bar(z, count/N, width=0.9,color="red", edgecolor="black", label="relative Haufigkeiten")  
bar(z,1/6, width=0.7,color="blue", edgecolor="black", label="theoretische Wahrscheinlichkeit")  
legend(loc="lower left")  
xticks(range(0,7))  
grid ()  
show ()

Wurfel geworfen. Das Spiel gewinnt derjenige, der die Summe der drei aufgetauchten Zahlen vorhersagt.  
(1) Man simuliere dieses Spiel N -mal (z.B. 1000), + Histogramm der rel. H. Zeichne auch die Balken fur die th. Wkt. Man vergleiche die mit den erhaltenen Werten aus den Simulationen.  
(2) Auf welche Zahl (oder Zahlen) muss man wetten, um die großten Gewinnchancen zu haben?  
(3) Welche ist die th. Wkt., dass diese Zahl (oder Zahlen) auftaucht? Man vergleiche mit den Ergebnissen der Simulationen.

import random  
import matplotlib.pyplot as plt  
import numpy as np  
  
# Corrected simulation  
N = 500 # Number of simulations  
sums = []  
  
for \_ in range(N):  
 die1 = random.randint(1, 6)  
 die2 = random.randint(1, 6)  
 die3 = random.randint(1, 6)  
 sums.append(die1 + die2 + die3)  
  
# Calculating theoretical probabilities  
theoretical\_probabilities = {}  
for i in range(1, 7):  
 for j in range(1, 7):  
 for k in range(1, 7):  
 dice\_sum = i + j + k  
 if dice\_sum in theoretical\_probabilities:  
 theoretical\_probabilities[dice\_sum] += 1  
 else:  
 theoretical\_probabilities[dice\_sum] = 1  
  
# Convert counts to probabilities  
for key in theoretical\_probabilities:  
 theoretical\_probabilities[key] /= 6 \*\* 3  
  
# Plotting the results  
plt.figure(figsize=(10, 6))  
  
# Calculating relative frequencies for the simulation  
simulated\_counts = {i: sums.count(i) / N for i in range(3, 19)}  
  
# Bars for simulated relative frequencies  
plt.bar(simulated\_counts.keys(), simulated\_counts.values(), color="red", alpha=0.5, label='Simulated Relative Frequencies', width=0.4)  
  
# Bars for theoretical probabilities  
plt.bar(theoretical\_probabilities.keys(), theoretical\_probabilities.values(), alpha=0.5, color="blue", label='Theoretical Probabilities', width=0.4)  
  
plt.xlabel('Sum of Dice')  
plt.ylabel('Probability')  
plt.title('Dice Roll Sum Probabilities: Simulation vs Theoretical')  
plt.xticks(np.arange(3, 19)) # Set x-ticks to show every possible sum  
plt.legend()  
plt.show()

Wkt. p1, p2, p3 schatzt folgendes Programm? Welche sind die theoretischen Werte fur p1, p2, p3?

|  |
| --- |
| % Labor 3 - A4 import random; import numpy c1,c2,a1,a2=0,0,0,0 |
| N=10000 A= list(range(1,21)) for \_ in range(N):  i=numpy.random.randint(len(A)) v=A[i] c1=c1+(v%2) |
| c2=c2+((v%2)==0) a1=a1+(v%2)\*((v%3)==0); a2=a2+ ((v%2)==0)\*(6<=v and v<=10)  p1=a1/c1 p2=a2/c2 p3=c1/N |
| print("Aus den Simulationen:") print(f"p1={p1:.6f}") print(f"p2={p2:.6f}") print(f"p3={p3:.6f}") |

import random;  
import numpy  
  
c1, c2, a1, a2 = 0, 0, 0, 0  
N = 10000  
A = list(range(1, 21))  
  
for i in range(N):  
 i = numpy.random.randint(len(A))  
 v = A[i]  
 c1 = c1 + (v % 2)  
 c2 = c2 + ((v % 2) == 0)  
 a1 = a1 + (v % 2) \* ((v % 3) == 0); # zahlt ob v ungerade ist und teilbar durch 2  
 a2 = a2 + ((v % 2) == 0) \* (6 <= v and v <= 10) # zahlt v gerade und zw 6 und 10  
  
p1 = a1 / c1  
p2 = a2 / c2  
p3 = c1 / N  
  
print("Aus den Simulationen:")  
print(f"p1={p1:.6f}")  
# p1 bedingte Wkt. eine zufallige Zahl aus A ist teilbar durch 3, wenn man weiss die Zahl ist ungerade  
print(f"p2={p2:.6f}")  
# p2 bedingte Wkt. eine zufallige Zahl aus A ist zwischen 6 und 10, wenn man weiss die Zahl ist gerade  
print(f"p3={p3:.6f}")  
# p3 bedingte Wkt. eine zufallige Zahl aus A ist ungerade  
  
  
w1 = 3/10  
w2 = 3/10  
w3 = 10/20  
print("Theoretische Wahrscheinlichkeiten:")  
print(f"w1={w1:.2f}")  
# p1 bedingte Wkt. eine zufallige Zahl aus A ist teilbar durch 3, wenn man weiss die Zahl ist ungerade  
print(f"w2={w2:.2f}")  
# p2 bedingte Wkt. eine zufallige Zahl aus A ist zwischen 6 und 10, wenn man weiss die Zahl ist gerade  
print(f"w3={w3:.2f}")

**ESTIMARI DACA SE ARUNCA UN ZAR/ ZILE DE NASTERE**

Wkt. dass in 1 Gruppe von 5 Personen **genau** 2 Geburtstag im selben Monat haben und die anderen 3 Personen verschiedene Geburtstage haben?

a) Anhand Simulationen.

b) Th. Wahrscheinlichkeit an.

import random  
from itertools import combinations  
  
# a) Simulation Approach  
N = 10000 # Number of simulations  
success\_count = 0  
  
for \_ in range(N):  
 birthdays = [random.randint(1, 12) for \_ in range(5)]  
 birthday\_counts = [birthdays.count(month) for month in set(birthdays)]  
 if sorted(birthday\_counts) == [1, 1, 1, 2]:  
 success\_count += 1  
  
simulated\_probability = success\_count / N  
  
# b) Theoretical Probability Calculation  
# Choose 2 out of 5 people  
choose\_two\_people = len(list(combinations(range(5), 2)))  
  
# Choose 1 month for these two people, and 3 different months for the remaining  
assign\_months = 12 \* 11 \* 10 \* 9  
  
# Total possible combinations of birthdays  
total\_combinations = 12 \*\* 5  
  
theoretical\_probability = (choose\_two\_people \* assign\_months) / total\_combinations  
  
# Output the results  
print(f"Simulated Probability: {simulated\_probability}")  
print(f"Theoretical Probability: {theoretical\_probability}")

Man schatze anhand Simulationen die Wkt. dass beim zweimaligen Werfen einesWurfels Summe ≥ 7,  
a) wenn, dass **beim ersten Wurf** eine 4 erhalten wurde;  
b) wenn, dass **beim zweiten Wurf** eine gerade Zahl erhalten wurde.

c) Welche sind die th. Wkt.en bei a), bzw. b) ?

import random  
  
# Number of simulations  
N = 10000  
  
count\_A = 0 # Count of sum being at least 7  
count\_B\_a = 0 # Count of first die being 4 when sum is at least 7 (for part a)  
count\_B\_b = 0 # Count of second die being even when sum is at least 7 (for part b)  
  
# Simulation loop  
for \_ in range(N):  
 die1 = random.randint(1, 6)  
 die2 = random.randint(1, 6)  
  
 # Condition A: Sum of dice being at least 7  
 if die1 + die2 >= 7:  
 count\_A += 1  
  
 # Condition B for part (a): First die is 4  
 if die1 == 4:  
 count\_B\_a += 1  
  
 # Condition B for part (b): Second die is even  
 if die2 % 2 == 0:  
 count\_B\_b += 1  
  
# Calculating the simulated conditional probabilities  
simulated\_prob\_a = count\_B\_a / count\_A if count\_A > 0 else 0  
simulated\_prob\_b = count\_B\_b / count\_A if count\_A > 0 else 0  
  
# Theoretical probabilities  
# a) For a given sum >= 7, the first die is 4 in 1 out of 6 possible outcomes (for die 1)  
theoretical\_prob\_a = 1 / 6

# b) The second die being even (2, 4, 6) and resulting in a sum >= 7  
# There are 15 total outcomes where the sum is >= 7 (6+1, 5+2, ..., 1+6 twice for symmetry)  
# Out of these, 9 outcomes have an even second die (2, 4, 6 in various combinations)  
theoretical\_prob\_b = 9 / 15  
  
# Output the results  
print(f"Simulated Conditional Probability for (a): {simulated\_prob\_a}")  
print(f"Theoretical Conditional Probability for (a): {theoretical\_prob\_a}")  
print(f"Simulated Conditional Probability for (b): {simulated\_prob\_b}")  
print(f"Theoretical Conditional Probability for (b): {theoretical\_prob\_b}")

**LABOR 4**

**Zufallige Werte der ZG**

Generieren von zufalligen Werten der ZG: X ∼ 0.4 0.1 0.3 0.2 . Simulation von zufalligen Werten fur X + Histogramm fur 1000 Werte + Balnken fur die th. Wkt. zeichnen:

import numpy  
from random import randrange  
from matplotlib.pyplot import bar, show, hist, grid, legend, xticks  
  
N = 1000  
x = [0, 1, 3, 5]  
P = [0.4, 0.1, 0.3, 0.2]  
rng = numpy.random.default\_rng()  
r = rng.choice(x, size=N, replace=True, p=P)  
x, count = numpy.unique(r, return\_counts=True)  
bar(x, count / N, width=0.9, color="red", edgecolor="black", label="relative Haufigkeiten")  
bar(x, P, width=0.7, color="blue", edgecolor="black", label="theoretische Wkt.")  
legend(loc="lower left")  
xticks(range(1, 7))  
grid()  
show()

**ZG = Wert, generiere Werte + Schatzungen**

Die Zufallsgroße X= Anzahl von Fehlern in der Zeitung: in 25% - keine Tippfehler, in 35% - 1 Tippfehler, in 25% - 2 , in 10% 3 und auf dem Rest 4 Tippfehler.

1. Man generiere zufallige Werte fur X.

2. Man schatze anhand der Simulationen die Wkt. dass hochstens 1 Tippfehler in einem zufallig ausgewahlten Artikel auftaucht.

3. Wie viele Tippfehler sind *durchschnittlich* in einem online Artikel dieser Zeitung zu erwarten, d.h. man verlangt die Schatzung von dem Erwartungswert E(X). Man berechne den theoretischen Erwartungswert.

# ▶ Man generiere zufa ̈llige Werte fu ̈r X.  
import numpy  
from random import randrange  
from matplotlib.pyplot import bar, show, hist, grid, legend, xticks  
  
N = 1000  
x = [0, 1, 2, 3, 4]  
P = [0.25, 0.35, 0.25, 0.1, 0.05]  
rng = numpy.random.default\_rng()  
r = rng.choice(x, size=N, replace=True, p=P)  
x, count = numpy.unique(r, return\_counts=True)  
bar(x, count / N, width=0.9, color="red", edgecolor="black", label="relative Haufigkeiten")  
bar(x, P, width=0.7, color="blue", edgecolor="black", label="theoretische Wkt.")  
legend(loc="lower left")  
xticks(range(1, 5))  
grid()  
show()  
  
# ▶ Man scha ̈tze anhand der Simulationen die Wahrscheinlichkeit, dass ho ̈chstens 1 Tippfehler in einem zufa ̈llig gewa ̈hlten Artikel auftaucht.  
print("Anhand der Simulationen Wkt. hochstens 1 Trippfehler", (count[0] + count[1]) / N)  
print("Theoretische Wkt. hochstens 1 Trippfehler", P[0] + P[1])  
  
# ▶ Wie viele Tippfehler sind durchschnittlich (im Mittel) in einem online Artikel dieser Zeitung zu erwarten, d.h. man verlangt die Scha ̈tzung von dem Erwartungswert E(X). Man berechne den theoretischen Erwartungswert.  
E = numpy.mean(r)  
Et = sum([i] \* P[i] for i in range(len(x)))  
print("Aus Simulationen sind dirchschnittlich ", E, "Trippfehler")  
print("Theoretisch sind dirchschnittlich ", Et, "Trippfehler") # E(x) = x[0]\*P[0]+...+x[4]\*P[4]

**ZG mit binomialer Verteilung**

Gegeben sind n, N ∈ N∗, p ∈ (0, 1).Die Zufallsgroße X hat binomiale Verteilung X ∼ Bino(n, p), wenn P(X=k)=Cnkpk(1−p)n−k, k∈{0,...,n}.

▶ Man generiere N (z.B. 500,1000,...) Werte der Zufallsgroße X mit binomialer Verteilung X ∼ Bino(n, p) mit n=8,p=0.5.Man benutze hier fur scipy.stats.binom.rvs.  
▶ Man erstelle das Histogramm der relativen Haufigkeiten der zufalligen Werten von X. Auf demselben Bild zeichne man auch die Balken fur die th. Wkt. fur diese benutze man scipy.stats.binom.pmf.

from scipy.stats import binom  
import matplotlib.pyplot as plt  
import numpy as np  
  
# Given parameters  
n = 8 # number of trials  
p = 0.5 # probability of success on each trial  
N = 1000 # number of random values to generate  
  
# Generate random values from the binomial distribution  
X = binom.rvs(n, p, size=N)  
  
# Calculate the probability mass function (pmf) for each value of k  
k\_values = np.arange(0, n+1)  
theoretical\_probabilities = binom.pmf(k\_values, n, p)  
  
# Plot the histogram of the simulated values  
plt.hist(X, bins=np.arange(-0.5, n+1.5, 1), density=True, alpha=0.5, label='Simulated Relative Frequencies')  
  
# Overlay the theoretical probabilities  
plt.bar(k\_values, theoretical\_probabilities, width=0.1, alpha=0.8, color='red', label='Theoretical Probability')  
  
# Add labels and title  
plt.xlabel('Number of Successes (k)')  
plt.ylabel('Probability')  
plt.title('Histogram of Binomial Distribution (n=8, p=0.5)')  
plt.xticks(k\_values) # Set x-ticks to show each value of k  
plt.legend()  
  
# Show the plot  
plt.show()  
  
# Calculate the probability of getting exactly k=5 successes  
k = 5  
probability\_k = binom.pmf(k, n, p)  
print(f"Man berechnet P(X = {k}) = {probability\_k:.6f}")

**Computer + Rechner mit Virus (avem Wkt. dat deja)**

In einem Computerpool sind 7 Rechner. Die Wahrscheinlichkeit, dass ein neuer Virus einen Rechner angreift ist 0.4, unabhangig von anderen Rechnern.  
▶ Welche ist die Wahrscheinlichkeit, dass der Virus:  
a) hochstens 3 Rechner;

b) mindestens 4 Rechner;

c) genau 4 Rechner angreift?

Man gebe die Antworten anhand Simulationen (binom.rvs) und vergleiche diese mit den theoretischen Wahrscheinlichkeiten (benutze man binom.cdf, binom.pmf).

import numpy as np  
import matplotlib.pyplot as plt  
from scipy.stats import binom  
  
  
n\_recomputers = 7  
p\_virus\_attack = 0.4  
  
N = 10000  
simulated\_values = binom.rvs(n\_recomputers, p\_virus\_attack, size=N)  
  
simulated\_prob\_a = np.mean(simulated\_values <= 3)  
theoretical\_prob\_a = binom.cdf(3, n\_recomputers, p\_virus\_attack)  
  
simulated\_prob\_b = np.mean(simulated\_values >= 4)  
theoretical\_prob\_b = 1 - binom.cdf(3, n\_recomputers, p\_virus\_attack)  
  
simulated\_prob\_c = np.mean(simulated\_values == 4)  
theoretical\_prob\_c = binom.pmf(4, n\_recomputers, p\_virus\_attack)  
  
  
print(f"a) Simulated Probability (at most 3 computers): {simulated\_prob\_a:.4f}, Theoretical Probability: {theoretical\_prob\_a:.4f}")  
print(f"b) Simulated Probability (at least 4 computers): {simulated\_prob\_b:.4f}, Theoretical Probability: {theoretical\_prob\_b:.4f}")  
print(f"c) Simulated Probability (exactly 4 computers): {simulated\_prob\_c:.4f}, Theoretical Probability: {theoretical\_prob\_c:.4f}")  
  
plt.hist(simulated\_values, bins=np.arange(-0.5, n\_recomputers+1.5, 1), density=True, alpha=0.7, label='Simulated Histogram')  
  
x = np.arange(0, n\_recomputers+1)  
theoretical\_probs = binom.pmf(x, n\_recomputers, p\_virus\_attack)  
plt.stem(x, theoretical\_probs, linefmt='r-', markerfmt='ro', label='Theoretical Probabilities')  
  
plt.xlabel('Number of Computers Attacked')  
plt.ylabel('Probability')  
plt.legend()  
  
plt.show()

**Zufallsgenerator fur Verteilung**

A math equations on a white background

Description automatically generated

import numpy as np  
from matplotlib import pyplot as plt  
from scipy.stats import geom  
  
# Parameters  
p\_success = 1/5 # Probability of picking a 5  
N = 1000 # Number of simulations  
  
# Simulate the number of trials until the first 5 appears  
counts\_until\_five = []  
for \_ in range(N):  
 count = 0  
 while np.random.randint(1, 6) != 5:  
 count += 1  
 counts\_until\_five.append(count)  
  
# Convert the list to a numpy array for easier calculations  
counts\_until\_five = np.array(counts\_until\_five)  
  
# Calculate the probabilities and expected value  
probability\_X\_leq\_3 = np.mean(counts\_until\_five <= 3)  
probability\_X\_gt\_3 = np.mean(counts\_until\_five > 3)  
expected\_value\_X = np.mean(counts\_until\_five)  
  
# Calculate theoretical probabilities using the geometric distribution  
k\_values = np.arange(0, max(counts\_until\_five) + 1)  
theoretical\_probabilities = geom.pmf(k\_values, p\_success)  
  
# Plot the histogram of the counts  
plt.hist(counts\_until\_five, bins=np.arange(0.5, max(counts\_until\_five)+1.5, 1), density=True, alpha=0.7, label='Empirical Histogram')  
  
# Overlay the theoretical probabilities  
plt.plot(k\_values, theoretical\_probabilities, 'ro-', label='Theoretical Probabilities')  
# Add labels and title  
plt.xlabel('Number of Trials Until First 5')  
plt.ylabel('Relative Frequency / Probability')  
plt.title('Geometric Distribution of Trials Until First Success')  
plt.legend()  
  
# Show the plot  
plt.show()  
  
# Print the results  
print(f'Probability that X <= 3: {probability\_X\_leq\_3:.4f}')  
print(f'Probability that X > 3: {probability\_X\_gt\_3:.4f}')  
print(f'Expected Value of X (Simulated): {expected\_value\_X:.4f}')  
print(f'Expected Value of X (Theoretical): {1/p\_success - 1:.4f}') # E(X) for geometric distribution

**Erwartungswert + Summe Kugeln + rel. Hauf.**

Eine Urne enthalt 1 Kugel mit der Ziffer 1, 2 mit 2, 3 mit 3. Aus der Urne werden 2 Kugeln ohne Zurucklegen gezogen. S sei die Summe der beiden Kugeln.  
▶ Man generiere N (z.B. 500,1000,...) zufallige Werte fur S und zeichne das Histogramm der relativen Haufigkeiten. Auf demselben Bild zeichne man die Balken fur die th. Wkt.

▶ Man schatze zusatzlich den Erwartungswert E(S) und berechne den theoretischen Erwartungswert von S.

from itertools import combinations  
import numpy as np  
import matplotlib.pyplot as plt  
  
# Define the balls in the urn  
urn = [1] \* 5 + [2] \* 6 + [3] \* 9  
  
# Calculate all possible unique combinations of two balls  
unique\_combinations = list(combinations(urn, 2))  
  
# Calculate all possible sums and their counts  
sums\_counts = {}  
for combo in unique\_combinations:  
 combo\_sum = sum(combo)  
 if combo\_sum in sums\_counts:  
 sums\_counts[combo\_sum] += 1  
 else:  
 sums\_counts[combo\_sum] = 1  
  
# Calculate theoretical probabilities  
theoretical\_probs = {k: v / len(unique\_combinations) for k, v in sums\_counts.items()}  
theoretical\_sums, theoretical\_freqs = zip(\*theoretical\_probs.items())  
  
# Theoretical expected value  
theoretical\_expected\_value = sum(k \* v for k, v in theoretical\_probs.items())  
  
# Number of simulations  
N = 1000  
  
# Simulate the drawing of 2 balls without replacement N times and calculate the sum  
simulated\_sums = [sum(np.random.choice(urn, size=2, replace=False)) for \_ in range(N)]  
  
# Plot the histogram of the simulated sums  
plt.hist(simulated\_sums, bins=range(min(simulated\_sums), max(simulated\_sums)+2), density=True, alpha=0.5, label='Simulated Relative Frequencies')  
  
# Overlay the theoretical probabilities  
plt.bar(theoretical\_sums, theoretical\_freqs, width=0.1, alpha=0.8, color='red', label='Theoretical Probability')  
  
# Add labels and title  
plt.xlabel('Sum of Balls')  
plt.ylabel('Relative Frequency / Probability')  
plt.title('Histogram of Sums from Drawing 2 Balls Without Replacement')  
plt.legend()  
  
# Show the plot  
plt.show()  
  
# Estimated expected value from the simulation  
empirical\_expected\_value = np.mean(simulated\_sums)  
  
# Print the estimated and theoretical expected values  
print(f"Estimated Expected Value of X (Simulated): {empirical\_expected\_value:.4f}")  
print(f"Theoretical Expected Value of X: {theoretical\_expected\_value:.4f}")

**LABOR 5**

**Standabweichung + Erwrtungswert + Dichtefunktion**

Teepackungen sollten mit jeweils 200 g Inhalt abgefullt werden. Die abgefullte Menge Tee X in einer Packung ist normal verteilt X ∼ N(μ,σ2); die dafur zustandige Abfullmachine hat eine Standardabweichung von σ = 3 g und ist auf einen Erwartungswert μ = 199 g eingestellt.

1. Anhand 1000 simulierten Daten, welche ist im Mittel die abgefullte Menge Tee in einer Packung? => scipy.stats.norm.rvs(μ,σ,1000) fur die Generierung von Daten, danach numpy.mean.
2. Mit welcher Wkt. werden in einer Packung weniger als 195 g Tee abgefullt? Mit welcher Wkt. werden in einer Packung zw 195g und198g Tee abgefullt? Mit welcher Wkt. werden in einer Packung mehr als 195 g Tee abgefullt? Man vergleiche das Ergebnis mit den th. Wkt. (th. Wkt. => scipy.stats.norm.cdf.)
3. Die generierten Daten der Stichprobe sollen in 16 Klassen (Intervallen) eingeteilt und man zeichne das entsprechende Histogramm der relativen Haufigkeiten mit matplotlib.pyplot.hist(Daten,bins=16,density=True,edgecolor="black",label="rel. Hfg.") Auf demselben Bild zeichne man auf dem Intervall [min(Daten),max(Daten)] auch die Dichtefunktion der N(μ, σ2) Verteilung (μ = 199, σ = 3). Hinweis: Man benutze norm.pdf(x, μ, σ) und plot.
4. Mit Hfg, Klasse=numpy.histogram(Daten, bins=16) zahle man wie viele Daten in jeder Klasse sind (ausdrucken mit print).

# a)  
import numpy  
import matplotlib.pyplot as plt  
from scipy.stats import norm  
  
mu = 199  
sigma = 3  
N = 1000  
Daten = norm.rvs(mu, sigma, N) # Beispiel Generieren von 1000 Daten  
print(Daten)  
print("Im Mittel die abgefullte Menge Tee in einer Packung ist", numpy.mean(Daten), "g")  
print("Theoretischer Erwartungswert fur die abgefullte Menge Tee in einer Packung ist", mu, "g")  
  
# b)  
# p1 = Wkt. in einer Packung werden weniger als 195 g Tee abgefullt  
test1 = numpy.mean(Daten <= 195)  
print("(Aus Simulationen) Wkt. in einer Packung werden weniger als 195 g Tee abgefullt", test1)  
print("(Theoretisch) P(X<195) = P (X<=195)=", norm.cdf(195, mu, sigma))  
  
# p2 = Wkt. in einer Packung werden zwischen 195 g und 198 g Tee abgefullt  
test2 = numpy.mean((Daten > 195) & (Daten < 198))  
print("(Aus Simulationen) Wkt. in einer Packung werden zwischen 195 g und 198 g Tee abgefullt", test2)  
print("(Theoretisch) P(<195X<198) = P (195<X<=198)=", norm.cdf(198, mu, sigma) - norm.cdf(195, mu, sigma))  
  
# p3 = Wkt. in einer Packung werden mehr als 195 g Tee abgefullt  
test3 = numpy.mean(Daten > 195)  
print("(Aus Simulationen) Wkt. in einer Packung werden mehr als 195 g Tee abgefullt", test3)  
print("(Theoretisch) P(X>195) = P (X>=195)=", 1 - norm.cdf(195, mu, sigma))  
  
plt.hist(Daten, bins=16, density=True, edgecolor="black", label="rel.Hfg.")  
# plt.show()  
  
x = numpy.linspace(min(Daten), max(Daten), 101)  
y = norm.pdf(x, mu, sigma) #Dichtefunktion von X -> f(x)  
plt.plot(x, y, "y-", label = "Dichtefunktion")  
plt.legend(loc = "upper left")  
plt.show()  
  
# d)  
Hfg, Klasse = numpy.histogram(Daten, bins = 16)  
#(1) Absolute Hfg. 3 der Klasse [189.4102 , 190.5905]  
for i in range(16):  
 print(f"({i+1:2d}) absolute Hfg. ist {Hfg[i]:3d} fur die Klasse[{Klasse[i]:3.3f}, {Klasse[i+1]:3.3f}]")  
  
print(Hfg)  
print(Klasse)

**Exponentialverteilung**

Die Zeit T (in Sekunden), die ein Drucker benotigt, um ein Werbeplakat zu drucken, folgt einer Exponentialverteilung Exp(λ) mit dem Parameter λ = 1/12 .

(a) Man simuliere N = 1000 Daten fur eine Stichprobe.

Welche ist die durchschnittliche Druckzeit fur das Drucken eines Plakats?  
(b) Man zeichne ein Histogramm mit 15 Klassen fu ̈r die simulierten Daten und auf demselben Bild zeichne man die Dichtefunktion expon.pdf(x,0,1/λ).  
(c) Man schatze danach die Wahrscheinlichkeiten P (T < 20), P (T > 10), P (10 < T < 30). Man vergleiche das Ergebnis mit den theoretischen Wahrscheinlichkeiten, welche man mit expon.cdf(x,0,1/λ) berechnet.

import numpy  
from matplotlib import pyplot as plt  
from scipy.stats import expon, norm  
  
L=1/12 # X ~Exp(1/12) L = lambda  
N=1000  
Daten = expon.rvs(loc=0,scale=1/L,size=N)  
# print(Daten)  
  
# a  
print("Druckzeit fur das Drucken eines Plakats", numpy.mean(Daten))  
print("Theoretische Wkt fur das Drucken eines Plakats", 1/L, "Sekunden")  
  
# b  
plt.hist(Daten,bins=15,density=True,edgecolor="black",label="rel.Hfg.")  
  
x = numpy.linspace(min(Daten), max(Daten), 100)  
y = expon.pdf(x, 0, 1/L)  
plt.plot(x, y, "g-")  
plt.legend()  
plt.show()  
  
# c  
print("P(T<20)", numpy.mean(Daten < 20))  
print("P(T>10)", 1 - numpy.mean(Daten < 10))  
# print("P(10<T<30)", numpy.mean(Daten > 10 and Daten < 30))  
  
print("Theoretische Wkt (T<20)", norm.cdf(20, 0, 1/L))  
print("Theoretische Wkt (T>10)", 1 - norm.cdf(10, 0, 1/L))  
print("Theoretische Wkt (10<T<30)", norm.cdf(30, 0, 1/L) - norm.cdf(10, 0, 1/L))

**Anhand Simulationen schatzen**

Jedesmal, wenn Professor X eine Gruppe von 6 Personen trifft, wettet er 6 euro, dass mindestens zwei von diesen 6 Personen im gleichen Monat Geburtstag haben. Anhand Simulationen schatze man: den durchschnittlichen Gewinn oder Verlust bei dieser Wette, bzw. die Wahrscheinlichkeit p, mit welcher Professor X eine Wette gewinnt.

import random  
  
  
def birthday\_simulation(num\_simulations):  
 total\_winnings = 0  
 successful\_bets = 0  
  
 for \_ in range(num\_simulations):  
 birthdays = [random.randint(1, 12) for \_ in range(6)]  
 if len(birthdays) != len(set(birthdays)):  
 successful\_bets += 1  
 total\_winnings += 6  
 else:  
 total\_winnings -= 6  
  
 probability = successful\_bets / num\_simulations  
 average\_gain\_loss = total\_winnings / num\_simulations  
  
 return probability, average\_gain\_loss  
  
  
def theoretical\_probability():  
 no\_shared\_birthday\_prob = 1.0  
 for i in range(6):  
 no\_shared\_birthday\_prob \*= (12 - i) / 12  
  
 probability = 1 - no\_shared\_birthday\_prob  
 average\_gain\_loss = -6 \* no\_shared\_birthday\_prob + 6 \* probability  
  
 return probability, average\_gain\_loss  
  
  
num\_simulations = 100000  
simulated\_probability, simulated\_avg\_gain\_loss = birthday\_simulation(num\_simulations)  
theoretical\_prob, theoretical\_avg\_gain\_loss = theoretical\_probability()  
print(f"Theoretical Probability of Winning: {theoretical\_prob}")  
print(f"Simulated Probability of Winning: {simulated\_probability}")  
print(f"Theoretical Average Gain/Loss per Bet: {theoretical\_avg\_gain\_loss} euros")  
print(f"Simulated Average Gain/Loss per Bet: {simulated\_avg\_gain\_loss} euros")

**Dichtefunktion + Verteilungsfunktion**

Man stelle die Dichtefunktion bzw. die Verteilungsfunktion fur a) X ∼ Unif[−2,2];  
b) X ∼ Exp(2); grafisch dar auf den Intervallen: fur a) [-3,3]; fur b) [0,4]!

2) Man schatze in jedem Fall anhand Simulationen P (1 < X < 1.5). Man vergleiche den geschatzen Wert mit dem theoretischen Wert indem man spezifische Python Befehle benutzt!  
3) Man schatze in jedem Fall anhand Simulationen den Erwartungswert E(X) und die Varianz V (X).

# Man stelle die Dichtefunktion bzw. die Verteilungsfunktion fur  
# 1)  
  
import matplotlib.pyplot as plt  
import numpy  
from matplotlib.pyplot import show, hist, grid, legend, plot,title, figure  
from scipy.stats import uniform, expon  
import numpy as np  
N = 1000  
  
x = np.linspace(-3, 3, 101)  
y = uniform.pdf(x, -2, 4) #? 4 = 2 - (-2) = b - a ==> X ∼ Unif[−2, 2]  
plot(x, y, "r-", label="Dichtefunktion Unif[-2, 2]")  
legend(loc="lower center")  
grid()  
show()  
  
x = np.linspace(0, 4, 101)  
L = 2 #? X ∼ Exp(2)  
y = expon.pdf(x, 0, 1/L)  
plot(x, y, "r-", label="Dichtefunktion Exp(2)")  
legend(loc="upper center")  
grid()  
show()  
  
  
x = np.linspace(-3, 3, 101)  
y = uniform.cdf(x, -2, 4) #? 4 = 2 - (-2) = b - a ==> X ∼ Unif[−2, 2]  
plot(x, y, "bo", label="Verteilungsfunktion Unif[-2, 2]")  
legend(loc="lower center")  
grid()  
show()  
x = np.linspace(0, 4, 101)  
L = 2 #? X ∼ Exp(2)  
y = expon.cdf(x, 0, 1/L)  
plot(x, y, "bo", label="Verteilungsfunktion Exp(2)")  
legend(loc="upper center")  
grid()  
show()  
  
#2)  
u = uniform.rvs(-2,4,N) #zufallige Daten Unif[-2,2]  
L = 2 #Exp(L) L = 2  
e = expon.rvs(0,1/L,N) #zufallige Daten Exp(2)  
#anhand Simulationen P(1<x<1.5). Man vergleiche de geschatzten Wert mit dem theoretischen Wert  
print("(aus Simulationen) Unif[-2, 2]: P(1 < X < 1.5) = ", np.mean((1 < u) & (u < 1.5)))  
print("(theoretisch) Unif[-2, 2]: P(1 < X < 1.5) = ", uniform.cdf(1.5, -2, 4) - uniform.cdf(1, -2, 4))  
  
print("(aus Simulationen) Exp(2): P(1 < X < 1.5) = ", np.mean((1 < e) & (e < 1.5)))  
print("(theoretisch) Exp(2): P(1 < X < 1.5) = ", expon.cdf(1.5, 0, 1/L) - expon.cdf(1, 0, 1/L))  
  
#! c)  
print(f"(Simuationen) X ∼ Unif[-2, 2]: E(X) = {np.mean(u):1.4f}") #? Erwartungswert  
print(f"(Simuationen) X ∼ Unif[-2, 2]: V(X) = {np.mean(u):1.4f}") #? Varianz  
print(f"(Simuationen) X ∼ Exp(2): E(X) = {np.mean(e):1.4f}") #? Erwartungswert  
print(f"(Simuationen) X ∼ Exo(2): V(X) = {np.mean(e):1.4f}") #? Varianz