S2-M-LÖS-Statistik-1

August 27, 2024

1 Aufgabe - Ganzzahlige Division - Variablen - Operationen

a) Ganzzahlige Division

[ ]:
```
a = int(input("Gib eine natürliche Zahl ein: "))
b = int(input("Gib eine natürliche Zahl ein: "))

wert = a // b

print("Der ganzzahlige Wert von", a, "/", b, "beträgt:", wert)
```

```
Der ganzzahlige Wert von 13 / 7 beträgt: 1
```

b) Ganzzahlige Division, Rest

[ ]:
```
a = int(input("Gib eine natürliche Zahl ein: "))
b = int(input("Gib eine natürliche Zahl ein: "))

wert = a // b
rest = a - b * wert

print("Der Rest von", a, "/", b, "beträgt:", rest)
```

```
Der Rest von 96 / 45 beträgt: 6
```

2 Aufgabe - Gerade / Ungerade - Variablen - Operationen

Ohne eingebaute Funktion.

[ ]:
```
zahl = int(input("Gib eine natürliche Zahl ein: "))
if (zahl // 2) * 2 == zahl:
    print(zahl, " ist gerade")
else:
    print(zahl, " ist ungerade")
```

```
123  ist ungerade
```

Mit Hilfe einer eingebauten Funktion 'divmod'.

[ ]:
```
zahl = int(input("Gib eine natürliche Zahl ein: "))

# Daes Resultat von divmod ist ein Tupel mit dem ganzzahligen Wert und dem Rest␣
 ↪bei der Division
resultat = divmod(zahl, 2)

if resultat[1] == 0:
    print(zahl, " ist gerade")
else:
    print(zahl, " ist ungerade")
```

```
144  ist gerade
```

3 Aufgabe - Kreiskegelstumpf - Variablen - Operationen

[ ]:
```
import math as math

R = float(input("Gib den Radius (R) der Grundfläche ein:"))
r = float(input("Gib den Radius (r) der Deckfläche ein:"))
h = float(input("Gib die Höhe (h) des geraden Kreiskegelstumpfes ein:"))

# mittlerer Umfang: (2*R*pi + 2*r*pi) / 2
mittlerer_umfang = math.pi * (R + r)

# Mantellinie: sqrt((R-r)^2 + h^2) mit Pythagoras
mantel_linie = math.sqrt( (R - r)**2 + h**2 )

# "Rechtecks-Fläche" = Mantelfläche
mantel_flaeche = mittlerer_umfang * mantel_linie

# Ausgabe der Eingangs- und Ausgangsgrösse
print("Radius (R) der Grundfläche:", R)
print("Radius (r) der Grundfläche:", r)
print("Radius (h) der Grundfläche:", h)

# gerundet auf 3 Nachkommastellen
print("Die Mantelfläche des geraden Kreiskegelstumpfes beträgt:", "%.3f " %␣
 ↪mantel_flaeche)
```

```
Radius (R) der Grundfläche: 6.0
Radius (r) der Grundfläche: 4.0
Radius (h) der Grundfläche: 5.0
Die Mantelfläche des geraden Kreiskegelstumpfes beträgt: 169.180
```

4 Aufgabe - Zeichenketten

a) Einzelnes Zeichen

[ ]:
```
zeichen = input("Gib ein einzelnes Zeichen ein:")

zeichenkette = 'Sesam oeffne dich!'

if zeichen in zeichenkette:
    print("das Zeichen:", zeichen, "kommt in:", zeichenkette, "vor.")
else:
    print("das Zeichen:", zeichen, "kommt in:", zeichenkette, "nicht vor.")
```

```
das Zeichen: p kommt in: Sesam oeffne dich! nicht vor.
```

b) Position dieses einzelnen Zeichens, erstes Auftreten

[ ]:
```
zeichen = input("Gib ein einzelnes Zeichen ein:")

zeichenkette = 'Sesam oeffne dich!'

# falls das letzte Auftreten gefragt wäre, so würde man dies mit "rfind"␣
 ↪eruieren
position = zeichenkette.find(zeichen)

print("das Zeichen:", zeichen, "kommt in: *", zeichenkette, "* an der Position:
 ↪", position, "vor.")
```

```
das Zeichen: e kommt in: * Sesam oeffne dich! * an der Position: 1 vor.
```

5 Aufgabe - Mengen / Sets

[ ]:
```
U = {1, 3, 5, 7, 9, 11, 13}
G = {2, 4, 6, 8, 10, 12, 14}
P = {1, 3, 5, 7, 11, 13, 17, 19, 23}

# Operationen auf Mengen
V = U.union(G)              # Vereinigungsmenge
S = U.intersection(P)       # Schnittmenge
D = P.difference(U)         # Differenzmenge

print("U =", U)
print("G =", G)
print("P =", P)

print("Vereinigungsmenge \nU \u222A G =", V)        # Operator für die␣
 ↪Vereinigungsmenge \u222A
print("Schnittmenge \nU \u2229 P =", S)             # Operator für die␣
 ↪Schnittmenge \u2229
print("Differenzmenge \nP \u2216 U =", D)           # Operator für die␣
 ↪Differenzmenge \u2216
```

```
U = {1, 3, 5, 7, 9, 11, 13}
G = {2, 4, 6, 8, 10, 12, 14}
P = {1, 3, 5, 7, 11, 13, 17, 19, 23}
Vereinigungsmenge
U  G = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14}
Schnittmenge
U  P = {1, 3, 5, 7, 11, 13}
Differenzmenge
P  U = {17, 19, 23}
```

6 Aufgabe - Bruchrechnen

[ ]:
```
from fractions import Fraction

print(Fraction(5, 6) + Fraction(1, 3))      # 5/6 + 1/3
print(Fraction(1, 3) + Fraction(2, 3))      # 1/3 + 2/3
print(Fraction(3, 4) - Fraction(2, 3))      # 3/4 - 2/3
```

```
7/6
1
1/12
```

7 Aufgabe - Verzweigungen

[ ]:
```
# für die Berechnung der Quadratwurzel benötigen wir das Modul "math"
import math as math

a = float(input("Gib die den Koeffizienten a ein:"))
b = float(input("Gib die den Koeffizienten b ein:"))
c = float(input("Gib die den Koeffizienten c ein:"))

diskriminante = b ** 2 - 4 * a * c

if diskriminante >= 0:
    if diskriminante == 0:
        loesung = - b / (2 * a)
        print("Die Doppellösung der quad. Gleichung (a=", a, "b=", b, "c=", c,␣
 ↪") lautet:", loesung)
    else:
        loesung_1 = (- b + math.sqrt(diskriminante)) / (2 * a)
        loesung_2 = (- b - math.sqrt(diskriminante)) / (2 * a)
        print("Die beiden Lösungen der quad. Gleichung (a=", a, "b=", b, "c=",␣
 ↪c, ") lauten:")
        print("x1=", loesung_1)
        print("x2=", loesung_2)
else:
```

```
    print("Die Gleichung (a=", a, "b=", b, "c=", c, ") besitzt keine reellen␣
 ↪Lösungen!")
```

```
Die Gleichung (a= 1.0 b= 1.0 c= 12.0 ) besitzt keine reellen Lösungen!
```

8 Aufgabe - Additionen, Schleifen

Mit Hilfe der Formel von Gauss.

[ ]:
```
# Via Console eine Zahl auffordern (ohne Plausibilisierung)!
n = int(input("Gin eine natürliche Zahl (n) ein:"))

# Formel von Gauss, Herleitung durch geeignete Anordnung der Summanden
# Beweis durch vollständige Induktion
summe = (n / 2)*(n + 1)

# Ausgabe, falls ohne Nachkommastellen ("%.0f " %)
print("Summe der ersten", n, "natürlichen Zahlen ist:", "%.0f " % summe)
```

```
Summe der ersten 55 natürlichen Zahlen ist: 1540
```

Mit Hilfe einer Schleife.

[ ]:
```
# Via Console eine Zahl auffordern (ohne Plausibilisierung)!
n = int(input("Gin eine natürliche Zahl (n) ein:"))

# die Summe auf den Wert 0 initialisieren:
sum = 0

# durch wiederholtes Addidieren die Summe berechnen
# da der Index bei 0 beginnt, muss bei der Endzahl 1 addiert werden
for i in range(n+1):
    sum = sum + i

# Ausgabe
print("Summe der ersten", n, "natürlichen Zahlen ist:", sum)
```

```
Summe der ersten 55 natürlichen Zahlen ist: 1540
```

9 Aufgabe - Zufallszahlen, Schleifen

a) Liste von 50 Zufallszahlen im Intervall I = [0; 10].

[ ]:
```
from random import randint

zufalls_zahlen = []
for i in range(50):
    zufalls_zahlen.append(randint(0 , 10))
```

```
print(zufalls_zahlen)
```

```
[10, 2, 10, 8, 4, 7, 9, 2, 7, 10, 10, 8, 5, 2, 7, 9, 5, 2, 5, 4, 8, 4, 4, 6, 5,
10, 5, 9, 8, 0, 8, 0, 0, 6, 10, 3, 6, 7, 6, 1, 3, 10, 1, 10, 2, 10, 0, 6, 6, 2]
```

b) Summe der 50 Zufallszahlen.

[ ]:
```
from random import randint

zufalls_zahlen = []
for i in range(50):
    zufalls_zahlen.append(randint(0 , 10))

print(zufalls_zahlen)

summe = 0
for zahl in zufalls_zahlen:
    summe = summe + zahl
    # summe = summe + zahl

print("Die Summe der 50 Zufallszahlen:", summe)
```

```
[0, 9, 9, 8, 8, 3, 7, 3, 7, 9, 0, 7, 10, 6, 0, 8, 4, 7, 6, 2, 7, 8, 1, 6, 4, 5,
0, 8, 2, 0, 0, 2, 1, 7, 4, 6, 10, 1, 8, 0, 10, 5, 6, 2, 0, 7, 6, 10, 1, 7]
Die Summe der 50 Zufallszahlen: 247
```

c) Wie oft kommt die 0 vor?

[ ]:
```
from random import randint

zufalls_zahlen = []
for i in range(50):
    zufalls_zahlen.append(randint(0 , 10))

# Ausgabe der Zufallszahlen und - in der neuen Zeile
# die Anzahl Nullen mit der eingebauten Methode count
print(zufalls_zahlen, "\nDie Anzahl Nullen:", zufalls_zahlen.count(0))
```

```
[7, 6, 4, 1, 8, 6, 10, 4, 5, 1, 2, 8, 0, 5, 4, 8, 2, 3, 6, 8, 10, 9, 6, 9, 0, 7,
2, 5, 5, 2, 7, 4, 1, 8, 10, 2, 6, 3, 10, 0, 8, 1, 5, 8, 6, 3, 0, 8, 0, 10]
Die Anzahl Nullen: 5
```

d) An welcher Position erscheint das erste Mal die 0?

[ ]:
```
from random import randint

zufalls_zahlen = []
for i in range(50):
    zufalls_zahlen.append(randint(0 , 10))
```

```
# Ausgabe der Zufallszahlen und - in der neuen Zeile
# die Position der ersten Null mit der eingebauten Methode index
# Achtung: Die Positionsausgabe erfolgt in Pythonzählweise, das heißt, 0
# entspräche dem ersten Element, 1 dem zweiten Element usw.
print(zufalls_zahlen, "\nDie Position der ersten Null", zufalls_zahlen.index(0))
```

```
[2, 10, 9, 7, 7, 6, 3, 8, 5, 5, 0, 4, 1, 5, 1, 0, 3, 2, 8, 3, 5, 10, 4, 3, 2, 1,
4, 3, 7, 2, 9, 2, 9, 4, 3, 5, 5, 0, 3, 4, 1, 5, 8, 9, 2, 10, 7, 3, 3, 1]
Die Position der ersten Null 10
```

10 Aufgabe - Verzweigungen, Schleifen

a) Quersumme einer Zahl berechnen.

[ ]:
```
zahl = input("Gib eine natürliche Zahl ein: ")

# eingegebene Zahl in einen String umwandeln
str_zahl = zahl
quersumme = 0
for ziffer in str_zahl:
    # jedes Zeichen in eine Zahl umwandeln und addieren
    quersumme = quersumme + int(ziffer)

print("Die Quersumme der Zahl", zahl, "lautet: ", quersumme)
```

```
Die Quersumme der Zahl 25 lautet:  7
```

b) **Variante 1**: Den ggT (grösster gemeinsamer Teiler) zweier Zahlen berechnen.

[ ]:
```
# mit dem importierten Befehl "gcd" kann es wie folgt gelöst werden:
# es wird nicht die ganze Bibliothek importiert, sondern nur der Befehl "gcd"
from math import gcd

zahl_1 = int(input("Gib die erste Zahl ein: "))
zahl_2 = int(input("Gib die zweite Zahl ein: "))

resultat = gcd(zahl_1, zahl_2)

print("der grösste gemeinsame Teiler von: ", zahl_1, " und ", zahl_2, " lautet:␣
↪", resultat)
```

```
der grösste gemeinsame Teiler von:  5  und  9  lautet:  1
```

b) **Variante 2**: Den ggT (grösster gemeinsamer Teiler) zweier Zahlen berechnen. Mit dem **Euklidischen Algorithmus**. Als Beispiel dienen die beiden Zahlen 174 und 78: $>$ 174 : 78 = 2 Rest 18 $>$ 78 : 18 = 4 Rest 6 $>$ 18 : 6 = 3 Rest 0

[ ]:
```
zahl_1 = int(input("Gib die erste Zahl ein: "))
zahl_2 = int(input("Gib die zweite Zahl ein: "))
```

```
a, b = zahl_1, zahl_2

# Division mit Rest, wobei a % b = Rest der Division: a / b
while b != 0:
    a, b = b, a % b

print("der grösste gemeinsame Teiler von: ", zahl_1, " und ", zahl_2, " lautet:␣
 ↪", a)
```

```
der grösste gemeinsame Teiler von:  4  und  8  lautet:  4
```

c) **Variante 1 - Testen**: Primzahl oder nicht?

[ ]:
```
# aus der Biliothek "math" benötigen wir die beiden Befehle "sqrt" und "ceil"
from math import sqrt, ceil

# die nachstehende Funktion liefert uns zwei Rückgabewerte: entweder die Zahl␣
 ↪und True, falls es sich um eine
# Primzahl handelt. Sonst wird ein Faktor und False zurückgegeben:
def primzahl(n):
    # bis zu Wurzel(n) müssen mögliche Faktoren getestet wwerden; mit "ceil"␣
 ↪wird die ganzahlige Zahl bestimmt
    wurzel_n = ceil(sqrt(n))
    for faktor in range(2, wurzel_n):
        if n % faktor == 0:             # teilt "faktor" die Zahl n ?
            return faktor, False        # keine Primzahl
    return n, True                      # ist Primzahl

zahl = int(input("Gib eine natürliche Zahl ein: "))

resultat = primzahl(zahl)

if resultat[1]:
    print(zahl, " ist Primzahl.")
else:
    print(zahl, " ist keine Primzahl. Ein Faktor lautet: ", resultat[0])
```

```
120  ist keine Primzahl. Ein Faktor lautet:  2
```

c) **Variante 2 - mit Bibliotheksfunktion: SymPy**: Primzahl oder nicht?

[ ]:
```
# in der Bibliothek "sympy" gibt es bereits einen Befehl "isprime", mit welchem␣
 ↪wir die Frage beantworten können:
from sympy import isprime
zahl = int(input("Gib eine natürliche Zahl ein: "))

if isprime(zahl):
    print(zahl, " ist eine Primzahl.")
```

```
else:
    print(zahl, " ist keine Primzahl.")
```

```
96  ist keine Primzahl.
```

d) Primzahlen - Sieb des Eratosthenes

[ ]:
```
from math import sqrt,ceil

obere_grenze = int(input("Bis zu welcher Zahl willst du die Primzahlen␣
 ↪berechnen?"))

wurzel_n = ceil(sqrt(obere_grenze))              # nur bis zur Quadratwurzel der␣
 ↪oberen Grenzen
                                                 # relevant

prim = []                                        # leere Liste
gestrichen = [False] * obere_grenze              # Liste mit lauter␣
 ↪"False"-Werten bis zur
                                                 # oberen Grenze füllen
for i in range(2, wurzel_n):
    if not gestrichen[i]:
        for j in range(i**2, obere_grenze, i):   # alle Vielfachen von i␣
 ↪streichen
            gestrichen[j]=True
            print(i,j)                           # kann auskommentiert werden,␣
 ↪dient nur der
                                                 # Überprüfung...
for i in range(2, obere_grenze):                 # alle nicht gestrichenen␣
 ↪Zahlen als
    if not gestrichen[i]:                        # Primzahlen in die Liste␣
 ↪aufnehmen
        prim.append(i)

print(prim)
```

```
2 4
2 6
2 8
2 10
2 12
2 14
2 16
2 18
2 20
2 22
2 24
2 26
```

2 28
2 30
2 32
2 34
2 36
2 38
2 40
2 42
2 44
2 46
2 48
2 50
2 52
2 54
2 56
2 58
2 60
2 62
2 64
2 66
2 68
2 70
2 72
2 74
2 76
2 78
2 80
2 82
2 84
2 86
2 88
2 90
2 92
2 94
2 96
2 98
3 9
3 12
3 15
3 18
3 21
3 24
3 27
3 30
3 33
3 36
3 39
3 42

```
3 45
3 48
3 51
3 54
3 57
3 60
3 63
3 66
3 69
3 72
3 75
3 78
3 81
3 84
3 87
3 90
3 93
3 96
3 99
5 25
5 30
5 35
5 40
5 45
5 50
5 55
5 60
5 65
5 70
5 75
5 80
5 85
5 90
5 95
7 49
7 56
7 63
7 70
7 77
7 84
7 91
7 98
[2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73,
79, 83, 89, 97]
```

e) **Variante 1**: Primfaktorenzerlegung

[ ]:
```
from sympy import primefactors

zahl = int(input("Zerlege die folgende Zahl in Primfaktoren:"))

print("Primfaktoren von",zahl," sind:")
print(primefactors(zahl))
```

```
Primfaktoren von 123457  sind:
[123457]
```

e) **Variante 2**: Primfaktorenzerlegung

[ ]:
```
# ohne weitere Hilfsmittel
def primfaktoren(n):
    faktoren = []
    teiler = 2
    while teiler ** 2 <= n:
        if n % teiler == 0:
            faktoren.append(teiler)
            n = n // teiler
        else:
            teiler = teiler + 1
    faktoren.append(n)
    return faktoren

zahl = int(input("Zerlege die folgende Zahl in Primfaktoren:"))

print("Primfaktoren von",zahl," sind:")
print(primfaktoren(zahl))
```

```
Primfaktoren von 96325  sind:
[5, 5, 3853]
```

[ ]:
```

```