Python Lösungen+ Prüfungsvorbereitung

Aufgabe 1.1

Was für eine Art von Programm braucht es, um ein Python-Programm zu schreiben?

Man benötigt einen Texteditor. Das ist ein Programm mit dem man Texte schreiben und bearbeiten kann.

Aufgabe 1.2

Was für eine Art von Programm braucht es, um ein Python-Programm auszuführen?

einen Python-Intepreter (also ein Programm, das wiederum ein Programm ausführt)

Aufgabe 1.3

Zähle die drei grundsätzlichen Typen von Programmierfehlern auf und nenne jeweils ein Beispiel dazu.

- Syntax-Fehler
  - eine Variable falsch geschrieben,
  - Code nicht 4 Zeichen eingerückt,
  - Doppelpunkt vergessen,
  - ...
- Laufzeit-Fehler
  - Division durch Null,
  - eine Datei, die eingelesen werden soll, ist nicht vorhanden,
  - ...
- Logische Fehler
  - Fehler in einer Berechnung ,
  - Bedingungen nicht korrekt formuliert,
  - falsche Reihenfolge von Code,
  - ...

Aufgabe 1.4

Welche Dateiendung haben Python-Programme normalerweise?

Die Endung `.py`

Aufgabe 2.1

```
print(3+2*9-1)
```

```
20
```

Aufgabe 2.2

```
print(8+9*4-6)
```

```
38
```

Aufgabe 2.3

```
print(7+4*1-8)
```

```
3
```

Aufgabe 2.4

```
print(32 / 8)
```

```
4.0
```

Aufgabe 2.5

```
print(26 % 7)
```

```
5
```

Aufgabe 2.6

```
print(20 // 8)
```

```
2
```

Aufgabe 2.7

```
1 print(2**2**3)
```

```
256
```

Aufgabe 3.1

```
a = 4
a = a + 6
a = a * 2
a = a - 9
print(a)
```

```
11
```

Aufgabe 3.2

```
a = 3
b = a + 9
a = b + a - 8
print(a)
```

```
7
```

Aufgabe 3.3

```
a, b, c = 9, 2, 8
b, c, a = a, b, c
print(c)
```

```
2
```

Aufgabe 3.4

```
b = 5
b += 4
b *= -1
print(b)
```

```
-9
```

Aufgabe 3.5

```
x = input('Eingabe: ')
y = 5*int(x)
print(y)
```

Die Ziffer 2 wird als Zeichenkette (`'2'`) in der Variablen mit dem Namen `x` gespeichert. Vor der Multiplikation mit der ganzen Zahl 5 wird die Zeichenkette `'2'` in die entsprechende Zahl `2` umgewandelt. Somit hat das Produkt den Wert `10`, der in der Variablen mit dem Namen `y` gespeichert wird. Dieser Wert `10` wird in Zeile 3 auf der Shell ausgegeben.

Aufgabe 3.6

(a) `anzahl_personen` ja

(b) `4you` nein (Ziffer an erster Stelle verboten)

(c) `_1234` ja

(d) `lieferung-mai-2022` nein (enthält Sonderzeichen `-`)

(e) `else` nein (Python-Schlüsselwort)

(f) `Or` ja (`Or` $\neq$ `or`)

Aufgabe 3.7

```
1 print('{}{}{}{}'.format(3, 7, 'a', 'x'))
```

```
37ax
```

Aufgabe 4.1

```
a=4
b=3
print(a != b)
```

```
True
```

Aufgabe 4.2

```
print(False and True)
```

```
False
```

Aufgabe 4.3

```
print(False or False)
```

```
False
```

Aufgabe 4.4

```
print(True or False and True)
```

```
True
```

Aufgabe 4.5

```
print(not(False or True) or False)
```

```
False
```

Aufgabe 4.6

```
print(not(3 < 2) or 4 == 2)
```

```
True
```

Aufgabe 4.7

```
print(7 < 4 <= 7)
```

```
False
```

Aufgabe 5.1

```
x = 7
if x != 1:
    x = x + 3
x = x + 1
print(x)
```

```
11
```

Aufgabe 5.2

```
x = 7
if x != 5:
    x = x + 4
else:
    x = x + 3
x = x + 1
print(x)
```

```
12
```

Aufgabe 5.3

```
z = 9
if z < 2:
    z = z + 1
elif z < 4:
    z = z + 2
elif z < 6:
    z = z + 3
else:
    z = z + 4
print(z)
```

```
13
```

Aufgabe 5.4

```
b = 7
if b == 7:
    if b > 5:
        b = b + 1
    else:
        b = b + 2
else:
    if b >= 4:
        b = b + 3
    else:
        b = b + 4
print(b)
```

8

Aufgabe 6.1

```
for i in range(3, 5):
    print(2*i)
```

```
6
8
```

Aufgabe 6.2

```
for k in range(1, 10, 4):
    print(k)
```

```
1
5
9
```

Aufgabe 6.3

```
for j in range(10, 7, -1):
    print(j)
```

```
10
9
8
```

Aufgabe 6.4

```
for e in [3, -8, 7, -4, 0]:
    if e > 0:
        print(e)
```

```
3
7
```

Aufgabe 6.5

```
i = 0
while (i < 10):
    i = 2*i+1
    print(i)
```

```
1
3
7
15
```

Aufgabe 6.6

```
i = 0
while (i < 10):
    i = 2*i+1
print(i)
```

```
15
```

Aufgabe 6.7

```
i = 4
s = 0
while (s < 20):
    s = s + i
    i = i + 3
    print(s)
```

```
4
11
21
```

Aufgabe 6.8

```
i = 0
while (i < 10):
    print(i)
```

Ja, denn die die Abbruchbedingung kann nicht erreicht werden.

Aufgabe 6.9

```
i = 10
while i != 0:
    i = i - 2
```

Nein, denn die Abbruchbedingung `i=0` wird nach 5 Schleifendurchläufen erreicht.

Aufgabe 6.10

```
for e in [4, -7 , 6, 8]:
    if e > 5:
        break
    else:
        print(e)
```

```
4
-7
```

Aufgabe 6.11

```
for i in range(1, 11):
    if i % 3 == 0:
        continue
    else:
        print(i)
```

```
1
2
4
5
7
8
10
```

Aufgabe 6.12

```
for i in range(2, 4):
    for j in range(3, 5):
        print(i+j)
```

```
5
6
6
7
```

Aufgabe 7.1

```
L = [3, -7, 1, 5, 8]
print(L[2])
```

```
1
```

Aufgabe 7.2

```
L = [[5, 3, 2], [1, 7], [9, 4, 8]]
print(L[2][0])
```

```
9
```

Aufgabe 7.3

```
L = [9, 3, 4, 2, 7, 5]
print(L[-2])
```

```
7
```

Aufgabe 7.4

```
L = [9, 3, 4, 2, 7, 5]
print(len(L))
```

```
6
```

Aufgabe 7.5

```
L = [[5, 3, 2], [1, 7], [9, 4, 8]]
print(len(L))
```

```
3
```

Aufgabe 7.6

```
L = [8,1,4,9]
L[2] = 7
print(L)
```

```
[8, 1, 7, 9]
```

Aufgabe 7.7

```
L = [5, 2, 3]
L.append(7)
print(L)
```

```
[5, 2, 3, 7]
```

Aufgabe 7.8

```
L = [5, 9, 8]
L.insert(1, 2)
print(L)
```

```
[5, 2, 9, 8]
```

Aufgabe 7.9

```
L = [7, 9, 5, 8, 2]
L.pop()
print(L)
```

```
[7, 9, 5, 8]
```

Aufgabe 7.10

```
L = [7, 9, 5, 8, 2]
x = L.pop()
print(x)
```

```
2
```

Aufgabe 7.11

```
L = [7, 9, 5, 8, 2]
x = L.pop(2)
print(x)
```

```
5
```

Aufgabe 7.12

```
A = [9, 3, 2]
B = [4, 1]
print(A + B)
```

```
[9, 3, 2, 4, 1]
```

Aufgabe 7.13

```
print(3 * [7,2])
```

```
[7, 2, 7, 2, 7, 2]
```

Aufgabe 7.14

```
L = [7, 9, 5, 8, 2, 1, 4, 3]
print(L[2:5])
```

```
[5, 8, 2]
```

Aufgabe 7.15

```
L = [7, 9, 5, 8, 2, 1, 4, 3]
print(L[:3])
```

```
[7, 9, 5]
```

Aufgabe 7.16

```
L = [7, 9, 5, 8, 2, 1, 4, 3]
print(L[6:])
```

```
[4, 3]
```

Aufgabe 7.17

```
A = [6, 7, 2]
B = A
B[1] = 9
print(A)
```

```
[6, 9, 2]
```

Aufgabe 7.18

```
A = [6, 7, 2]
B = A[:]
B[1] = 9
print(A)
```

```
[6, 7, 2]
```

Aufgabe 7.19

```
L = [6, 1, 2, 7, 9]
L.reverse()
print(L)
```

```
[9, 7, 2, 1, 6]
```

Aufgabe 7.20

```
L = [6, 1, 2, 7, 9]
L.sort()
print(L)
```

```
[1, 2, 6, 7, 9]
```

Aufgabe 7.21

```
L = [6, 4, 2, 8]
print(sum(L))
```

```
20
```

Aufgabe 7.22

```
[y, x, z] = [5, 3, 1]
print(x)
```

```
3
```

Aufgabe 7.23

```
L = [9, 2, 4, 1, 8]
s = 0
for e in L:
    s += e
print(s)
```

```
24
```

Aufgabe 7.24

```
L = [9, 2, 4, 1, 8]
s = 0
for i in range(0, len(L)):
    s += L[i]
print(s)
```

```
24
```

Aufgabe 7.25

```
T = (9, 3, 4, 2, 7, 5)
print(T[3])
```

```
2
```

Aufgabe 7.26

```
L = (9, 3, 4, 2, 7, 5)
L[3] = 8
print(L)
```

```
...
    L[3] = 8
    ~^^^
TypeError: 'tuple' object does not support item assignment
```

Aufgabe 7.27

```
L = (9, 3, 4, 2, 7, 5)
print(L[1:4])
```

```
(3, 4, 2)
```

Aufgabe 7.28

```
L = [a for a in range(2, 5)]
print(L)
```

```
[2, 3, 4]
```

Aufgabe 7.29

```
A = [1, 2, 3]
B = [2*x for x in A]
print(B)
```

```
[2, 4, 6]
```

Aufgabe 7.30

```
Z = [[0 for i in range(2)] for j in range(3)]
print(Z)
```

```
[[0, 0], [0, 0], [0, 0]]
```

Aufgabe 7.31

```
L = [0 if i % 2 == 0 else 1 for i in range(7)]
print(L)
```

```
[0, 1, 0, 1, 0, 1, 0]
```

Aufgabe 8.1

```
22
```

Aufgabe 8.2

```
19
```

Aufgabe 8.3

```
None
```

Aufgabe 8.4

```
17
```

Aufgabe 8.5

```
19
```

Aufgabe 8.6

```
15
```

Aufgabe 8.7

```
18
```

Aufgabe 8.8

```
9
```

Aufgabe 8.9

```
0
```

Aufgabe 8.10

```
9
3
```

Variablen, die in einer Funktion definiert sind, sind nur im betreffenden Funktionsrumpf (eingerückten Bereich) gültig. Gleichzeitig „überschatten“ sie globale Variablen mit gleichem Namen. Nach der Ausführung werden die lokalen Variablen wieder gelöscht und globale Variablen mit gelichem Namen werden wieder sichtbar.

Aufgabe 8.11

```
1 def dreieckUmfang(a, b, c):
2     return a + b + c
```

Aufgabe 8.12

```
1 def trapezInhalt(a, c, h):
2     m = (a+c)/2
3     return m*h
```

Aufgabe 8.13

```
None
```

Aufgabe 8.14

```
9
```

Aufgabe 8.15

```
7
```

Aufgabe 8.16

```
5
```

Aufgabe 8.17

```
15
```

Aufgabe 8.18

8

Aufgabe 8.19

[1, 2, 3, 4]

Aufgabe 8.20

```
1 def f(n):
2     if n == 0:
3         return 1
4     else:
5         return f(n-1)+n
6
7 print(f(4))
```

$$
\begin{aligned}
f(4) &= f(3) + 4 \\
&= (f(2) + 3) + 4 \\
&= ((f(1) + 2) + 3) + 4 \\
&= (((\mathbf{f(0)} + 1 + 2) + 3) + 4 \quad (\textbf{Base Case erreicht}) \\
&= (((\mathbf{1} + 1) + 2) + 3) + 4 \\
&= 11
\end{aligned}
$$

11

Aufgabe 8.21

```
1 def f(n):
2     if n < 2:
3         return n
4     else:
5         return 2*f(n-2) + 1
6
7 print(f(5))
```

$$
\begin{aligned}
f(5) &= 2 * f(3) + 1 \\
&= 2 * (2 * \mathbf{f(1)} + 1) + 1 \quad (\textbf{Base Case erreicht}) \\
&= 2 * (2 * \mathbf{1} + 1) + 1 \\
&= 2 * 3 + 1 = 7
\end{aligned}
$$

7

Aufgabe 8.22

12

Aufgabe 9.1

```
s = '''Das ist
ein Text'''
print(s)
```

```
Das ist
ein Text
```

Aufgabe 9.2

```
s = 'Das ist \nWahnsinn'
print(s)
```

```
Das ist
Wahnsinn
```

Aufgabe 9.3

```
s = 'C\'est la vie!'
print(s)
```

```
C'est la vie!
```

Aufgabe 9.4

```
s = 'A\\B'
print(s)
```

```
A\B
```

Da der Backslash zum Maskieren der Stringbegrenzungszeichen (`"..."` und `'...'`) sowie für die Bildung von Steuerzeichen (`\n`) verwendet wird, kann er nicht direkt in einer Zeichenkette auftreten und muss seinerseits durch einen zweiten Backslash maskiert werden.

Aufgabe 9.5

```
s = 'Hund'
print(s[1:])
```

```
und
```

Aufgabe 9.6

```
print('A' + 2*'B' + 'A')
```

```
ABBA
```

Aufgabe 9.7

```
s = 'Was ist das?'
print(len(s))
```

```
12
```

Jedes Zeichen (auch Leerzeichen, Zeichenschaltungen und Tabuloren) wird gezählt.

Aufgabe 9.8

```
s = "hallo"
print(list(s))
```

```
['h', 'a', 'l', 'l', 'o']
```

Die `list()`-Methode zerlegt eine Zeichenkette in eine Liste von Einzelzeichen (*characters*).

Aufgabe 9.9

```
print(ord('A'))
```

```
65
```

An Prüfungen zu diesem Thema steht eine ASCII-Tabelle zur Verfügung, so dass der Wert (die „Ordnungszahl“ des Zeichens) dort abgelesen werden kann.

Aufgabe 9.10

```
print(chr(66))
```

```
B
```

Aufgabe 9.11

```
s = "Ananas"
print(s.count('a'))
```

```
2
```

Die String-Methode `str.count(<zeichenkette>)` zählt, wie oft `<zeichenkette>)` in `str` vorkommt. Man beachte, dass Gross- und Kleinschreibung unterschieden wird.

Aufgabe 9.12

```
s = 'Hallo'
s = s.lower()
print(s)
```

```
hallo
```

Aufgabe 9.13

```
s = 'ch'
s.upper()
print(s)
```

```
ch
```

Achtung: Da Zeichenketten unveränderlich (*immutable*) sind, können die nicht durch die String-Methoden verändert werden. Dafür liefern die Methoden einen Rückgabewert und es liegt in der Verantwortung des Programmierers, diesen Rückgabewert in einer Variablen zu speichern.

Aufgabe 9.14

```
s = 'abcde'
s = s.replace('a', 'b')
print(s)
```

```
bbcde
```

Die Methode str.replace() ersetzt die Zeichenkette im ersten Parameter durch die Zeichenkette im zweiten Parameter. Man mann die Anzahl der Ersetzungen durch einen dritten Parameter beschränken aber das ist kein Prüfungsstoff.

Aufgabe 9.15

```
L = ['x', 'y', 'z']
s = '+'.join(L)
print(s)
```

```
x+y+z
```

Aufgabe 9.16

```
s = 'teller'
print(s.split('e'))
```

```
['t', 'll', 'r']
```

Aufgabe 9.17

```
s = 'abcdxyz'
s = s.strip('abyz')
print(s)
```

```
cdx
```

`str.strip(<zeichenkette>)` entfernt links und rechts von `str` die in `<zeichenkette>` vorkommenden Zeichen. Die Reihenfolge ist dabei unwichtig. Fehlt der Parameter, so werden automatisch alle Formen von „Whitspaces“ (Leerzeichen, Tabulatoren, Zeilenschaltungen) entfernt.

`str.lstrip(<zeichenkette>)` und `str.rstrip(<zeichenkette>)` funktionieren analog, nur dass sie auf jeweils einer Seite (*left*, *right*) wirken.

Aufgabe 9.18

```
s = 'abcxyz'
s = s.lstrip('abyz')
print(s)
```

```
cxyz
```

Aufgabe 9.19

```
s = 'abcxyz'
s = s.rstrip('abyz')
print(s)
```

```
abcx
```

Aufgabe 9.20

```
s = 'a{}pb{}m'.format(3, 'e')
print(s)
```

```
a3pbem
```

Aufgabe 9.21

```
s = 't{1}k{0}w{1}'.format(3, 'e')
print(s)
```

```
tek3we
```

Aufgabe 9.22

```
print(int("15" + "4") + 3)
```

```
157
```

Aufgabe 9.23

```
18.0
```

Aufgabe 9.24

```
2
```

Aufgabe 10.1

```
3/4
```

Aufgabe 10.2

```
(2,7)+(7,2)
```

Aufgabe 10.3

```
3,1,5
```

Aufgabe 10.4

```
7,3
```

Aufgabe 10.5

```
+123456
-123456
 123456
-123456
```

Aufgabe 10.6

```
******a
b******
***c***
```

Aufgabe 10.7

```
100,000,000
100_000_000
```

Aufgabe 10.8

```
1101;0b1101
15;0o15
13
d;0xd
D;0XD
```

Aufgabe 10.9

```
0.667
0.6667
6.667e-01
6.6667e-01
6.6667e+02
```

Aufgabe 10.10

```
079-999-99-99
```

Aufgabe 10.11

```
abc*uvw
```

Aufgabe 10.12

```
abcxyz
```

Zeilenschaltungen müssen hier explizit mit dem Newline-Character `'\n'` geschrieben werden.

Aufgabe 10.13

```
[5, 8, 7, 9]
```

Aufgabe 11.1

```
3
```

Aufgabe 11.2

```
b
```

Aufgabe 11.3

```
4
```

Aufgabe 11.4

```
-3
```

Aufgabe 11.5

```
-2
-1
3
9
```

Aufgabe 11.6

```
2
```

Aufgabe 11.7

```
65000
```

Aufgabe 11.8

```
8
```

Aufgabe 11.9

```
99
```

Aufgabe 11.10

```
c
m
u
```

Aufgabe 11.11

```
[5, 7]
```

Aufgabe 11.12

```
6
```

Aufgabe 11.13

```
3
```

Aufgabe 12.1

```
5
```

Aufgabe 12.2

```
False
```

Aufgabe 12.3

```
{1, 2, 3, 4, 5, 7}
```

Aufgabe 12.4

```
{2}
```

Aufgabe 12.5

```
{1, 4, 7}
```

Aufgabe 12.6

```
True
```

Aufgabe 12.7

```
{1, 4, 5, 7}
```

Aufgabe 12.8

```
{1, 7}
```

Aufgabe 12.9

```
13
```

Aufgabe 12.10

```
[1, 2, 3, 7]
```

Aufgabe 12.11

```
42
```

Aufgabe 12.12

```
[1, 3]
```

Aufgabe 12.13

```
['H', 'N', 'O', 'P', 'T', 'Y']
```

Aufgabe 13.1

```
Traceback (most recent call last):
  File "python-13-pvor-01.py", line 3, in <module>
    print(a)
NameError: name 'a' is not defined
```

Aufgabe 13.2

```
Traceback (most recent call last):
  File "python-13-pvor-02.py", line 1, in <module>
    import xyz.py
ModuleNotFoundError: No module named 'xyz.py'; 'xyz' is not a package
```

Aufgabe 13.3

```
30
```

Aufgabe 13.4

```
Traceback (most recent call last):
  File "python-13-pvor-04.py", line 3, in <module>
    print(f(5))
NameError: name 'f' is not defined
```

Aufgabe 13.5

```
10
```

Aufgabe 13.6

```
40
```

Aufgabe 13.7

```
6
```

Aufgabe 13.8

```
5.0
```

Aufgabe 13.9

```
3.14
```

Aufgabe 15.1

(a) Ein Bauplan für Objekte (ein abstrakter Datentyp)

(b) Ein Objekt, das zur Laufzeit aus einer Klasse erzeugt wird

(c) Eine Variable, die zu einem Objekt gehört

(d) Eine Funktion, die zu einem Objekt gehört

(e) Eine Variable, die zu einer Klasse gehört

(f) Eine Funktion, die zu einer Klasse gehört

(g) Eine spezielle Methode, mit der ein Objekt initialisiert wird.

Aufgabe 15.2

```
12
```

Aufgabe 15.3

```
(4, 2)
```

Aufgabe 15.4

```
1.0
```

Aufgabe 15.5

```
25
```

Aufgabe 15.6

```
class Lampe:

    def __init__(self):
        self.zustand = False

    def __str__(self):
        if self.zustand == True:
            return 'Lampe an'
        else:
            return 'Lampe aus'

    def schalten(self):
        if self.zustand == False:
            self.zustand = True
        else:
            self.zustand = False
# Test-Client: (nur zur Illustration)
L = Lampe()  # erzeugt einen neue (ausgeschaltete) Lampe
L.schalten() # Schaltet die Lampe ein
L.schalten() # Schaltet die Lampe aus
L.schalten() # Schaltet die Lampe ein
print(L)     # => 'Lampe ein'
```

Aufgabe 15.7

```
class Car:

    max_speed = 120

    def __init__(self):
        self.speed = 0

    def accelerate(self, delta_v):
        self.speed = min(self.speed+delta_v, Car.max_speed)

    def brake(self, delta_v):
        self.speed = max(self.speed-delta_v, 0)

    def get_speed(self):
        return self.speed

c1 = Car()
c2 = Car()

c1.accelerate(50)
c1.accelerate(30)
c1.brake(40)

c2.accelerate(40)
c2.brake(50)

print(c1.get_speed())
print(c2.get_speed())
```

```
40
0
```

Aufgabe 15.8

```
1  class Image:
2
3      def __init__(self, width, height):
4          self.w = width
5          self.h = height
6          self.img = [[0 for i in range(width)] for j in range(height)]
7
8      def set_pixel(self, x, y, color=1):
9          self.img[y][x] = color
10
11     def write(self, filename):
12         fd = open(filename, mode='w')
13         fd.write('P1\n')
14         fd.write('{0} {1}\n'.format(self.w, self.h))
15         for i in range(0, self.h):
16             for j in range(self.w):
17                 fd.write('{0} '.format(self.img[i][j]))
18         fd.close()
19
20 I = Image(3, 3)
21 I.set_pixel(0, 0)
22 I.set_pixel(1, 1)
23 I.set_pixel(2, 2)
24 I.write('test.pbm')
```

(a) Das Modul definiert eine Klasse `Image`

- Der Konstruktor `__init__(self, width, height)` initialisiert die Attribute `self.w` und `self.h` mit den Parameterwerten `width` und `height` Dem Attribut `self.img` wird die Nullmatrix mit `height` Zeilen und `width` Spalten zugewiesen.
- Die Methode `set_pixel(...)` setzt in der Zeile `y` und der Kolonne `x` ein Pixel mit der „Farbe“ `color`. Wird dieser Parameter beim Aufruf der Methode weggelassen, so wird `1` verwendet.
- Die Methode `write(...)` öffnet eine neue Datei mit dem angegebenen Dateinamen, und schreibt die Bildinformationen (Bildformat, Dimension, Pixelwerte) in diese Datei.

(b)
```
P1
3 3
1 0 0 0 1 0 0 0 1
```

Aufgabe 15.9

```
from math import gcd # greatest common divisor

class Bruch:

    def __init__(self, z, n):
        t = gcd(z, n)
        self.z = z // t
        self.n = n // t
        if self.n < 0:
            self.z = -self.z
            self.n = -self.n

    def __str__(self):
        if self.n == 1:
            return '{0.z}'.format(self)
        else:
            return '{0.z}/{0.n}'.format(self)

    def __add__(self, other):
        z = self.z * other.n + self.n + other.z
        n = self.n * other.n
        return Bruch(z, n)

a = Bruch(3,9)
b = Bruch(1,6)
c = a + b

print(a)
print(c)
```

(a) Das Modul definiert eine Klasse `Image`

- Der Konstruktor `__init__(self, width, height)` initialisiert die Attribute `self.w` und `self.h` mit den Parameterwerten `width` und `height` Dem Attribut `self.img` wird die Nullmatrix mit `height` Zeilen und `width` Spalten zugewiesen.
- Die Methode `set_pixel(...)` setzt in der Zeile `y` und der Kolonne `x` ein Pixel mit der „Farbe“ `color`. Wird dieser Parameter beim Aufruf der Methode weggelassen, so wird `1` verwendet.
- Die Methode `write(...)` öffnet eine neue Datei mit dem angegebenen Dateinamen, und schreibt die Bildinformationen (Bildformat, Dimension, Pixelwerte) in diese Datei.

(b)
```
P1
3 3
1 0 0 0 1 0 0 0 1
```

Aufgabe 15.10

```
1  class Rechteck:
2
3      anzahl = 0
4
5      def __init__(self, laenge, breite):
6          self.a = laenge
7          self.b = breite
8          Rechteck.anzahl += 1
9
10     def umfang(self):
11         return 2*(self.a + self.b)
12
13     def inhalt(self):
14         return  self.a * self.b
15
16     def add(self, other):
17         return Rechteck(self.a+other.a, self.b+other.b)
18
19     def get_anzahl():
20         return Rechteck.anzahl
21
22 r1 = Rechteck(2,5)
23 r2 = Rechteck(4,3)
24 r3 = r1.add(r2)
25
26 print(r1.umfang())
27 print(r2.inhalt())
28 print(r3.b)
29 print(Rechteck.get_anzahl())
```

(a)

Zeile	Ausgabe	Erklärung
26:	`14`	Umfang des Rechtecks `r1`
27:	`12`	Flächeninhalt des Rechtecks `r2`
28:	`8`	Summe der Breiten der Rechtecke `r1` und `r2`
29:	`3`	Anzahl der erzeugten Rechtecke

(b)

- Eine *Klasse* ist ein Bauplan für Objekte eines bestimmten Typs. Hier wird die Klasse in den Zeilen 1–20 definiert.
- Eine *Instanz* ist ein zur Laufzeit erzeugtes Objekt, das die in der Klasse definierten Attribute und Methoden besitzt. In Python werden Instanzen erzeugt, indem der Klassenname als Funktion mit allfälligen Parametern aufgerufen wird. Dies ruft den Konstruktor auf, der die Instanz mit den übergebenen Parametern initialisiert. Im Beispielcode werden drei Instanzen in den Zeilen 22–24 definiert. Beachte, dass Instanz `r3` aus der „Summe“ der Instanzen `r1` und `r2` entsteht.
- Eine *Objekteigenschaft* ist ein Attribut, das spezifisch für ein Objekt ist. Hier z. B. die Länge 2 und die Breite 5 des Rechtecks `r1`. Um in Python auf die Eigenschaft einer Instanz zuzugreifen, muss man ihr wie in Zeile 28 mit der Punkt-Schreibweise den Instanznamen voranstellen.

- Eine *Objektmethode* ist eine Funktion, die auf eine spezifische Instanz einer Klasse angewendet wird. Hier sind das die Methoden `umfang()` und `inhalt()`. Um in Python eine Methode auf eine Instanz anzuwenden, muss man ihr wie in den Zeilen 24, 26 und 27 mit der Punkt-Schreibeweise den Instanznamen voranstellen.
- Eine *Klasseneigenschaft* (auch Klassenattribut genannt) ist ein Attribut, das für die gesamte Klasse gilt und nicht spezifisch für eine Instanz. Hier ist das die Variable `anzahl`, welche immer dann inkrementiert (um 1 erhöht) wird, wenn der Konstruktor (siehe unten) ein neues Objekt erzeugt. Um in Python auf eine Klasseneigenschaft zuzugreifen, muss man ihr mit der Punkt-Schreibweise den Klassennamen voranstellen.
- Eine *Klassenmethode* ist eine Methode, die auf die Klasse selbst und nicht auf eine spezifische Instanz angewendet wird. Sie kann auf Klasseneigenschaften zugreifen und wird häufig verwendet, um Funktionen zu implementieren, die für die gesamte Klasse relevant sind. Um in Python eine Methode auf eine Klasse anzuwenden muss man ihr wie in Zeile 29 mit der Punkt-Schreibeweise den Klassennamen voranstellen.
- Der *Konstruktor* einer Klasse ist eine spezielle Methode, die automatisch aufgerufen wird, wenn eine neue Instanz (ein Objekt) der Klasse erstellt wird. Der Konstruktor wird verwendet, um die Anfangswerte der Objekteigenschaften festzulegen und um notwendige Initialisierungen durchzuführen. In Python ist der Konstruktor die Spezialmethode `__init__(self, ...)`, die hier in den Zeilen 5–8 definiert wird. Der erste Parameter, der üblicherweise `self` genannt wird, repräsentiert die Speicheradresse der zu erzeugenden Instanz, die sowohl dem Variablennamen der Instanz zugewiesen wird als auch die Information über den Ort darstellt, wo die Eigenschaften der Instanz abgelegt sind.