

Altklausuren

18/19

1

1

Realitätsausschnitt, Abstraktion, dienen einem Zweck

2

Ist ein Paar $G(V, E)$ wobei V eine endliche Menge von Knoten und E ist eine Menge von Geordnete Knoten Paaren.

3

Bestimmt durch zwei Merkmale:

1. Eine Menge von gültigen Werten
2. Durchführbare Operation zu den Werten

4

Vorteil:

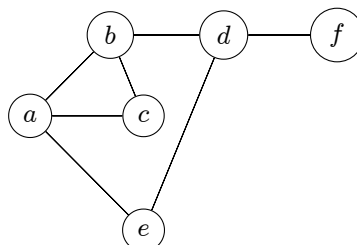
Einfach zu beschreiben

Nachteil:

Sehr Ineffizient

2

1



2

```
0 1 1 0 1 0
1 0 1 1 0 0
1 1 0 0 0 0
0 1 0 0 1 1
1 0 0 1 0 0
0 0 0 1 0 0
```

3

Ja, da man von jedem beliebigen Knoten jeden anderen erreichen kann.

4

$K = \{\{a, b\}, \{b, d\}, \{d, e\}, \{e, a\}\}$

5

3

6

b, a, c, d, e, f

7

b, a, c, e, d, f

3

1

```
def repFirstElm(L):
    first_elm = L[0]
    count = 0

    for i in range(len(L)):
        if L[i] == first_elm:
            count += 1
```

```

        return count

L = [1,2,1,4,2,6,1]
print(repFirstElm(L))

```

2

n ist die Länge der Liste L

```

def repFirstElm(L):
    first_elm = L[0]          # 1
    count = 0                 # 1

    for i in range(len(L)):   # n
        if L[i] == first_elm: # 2
            count += 1         # 2

    return count              # 1

L = [1,2,1,4,2,6,1]          # 1
print(repFirstElm(L))        # 1

```

$(4n + 5) \in O(n)$

4

1

1. Richtig
2. Falsch
3. Richtig
4. Falsch
5. Richtig
6. Richtig
7. Falsch

2

A

5

1

```
kontonummer = 12345678
kontostand = 0 # Euro

class Konto:
    def __init__(self, kn, kohle): # Kohle in Euro
        self.kontonummer = kn
        self.kontostand = kohle
    def einzahlen (self, betrag): # betrag in Euro
        self.kontostand += betrag
    def __str__(self):
        return "Kontonummer: " + str(self.kontonummer) + ",
Kontostand: " + str(self.kontostand)

myKonto = Konto(7654321, 100)
print(myKonto.kontostand)
myKonto.einzahlen(60)
print(myKonto.kontostand)

print(myKonto)
```

2

Kontonummer, Kontostand

3

siehe 1

6

1

```
s=1
i=2
while i < 12:
    s *= i
```

```
        i += 2

print(s)

s=1
for i in range(2,12,2):
    s*=i
print(s)
```

2

```
def f(x):                # x=5
    y=x//2               # y=2
    return g(y, y+1)    # 4.0

def g(x,y):              # x=2, y=3
    z=x**y               # z=8
    return z/x           # 8/2=4.0

x=1
x=f(5)                   # x=4.0
print(x)
```

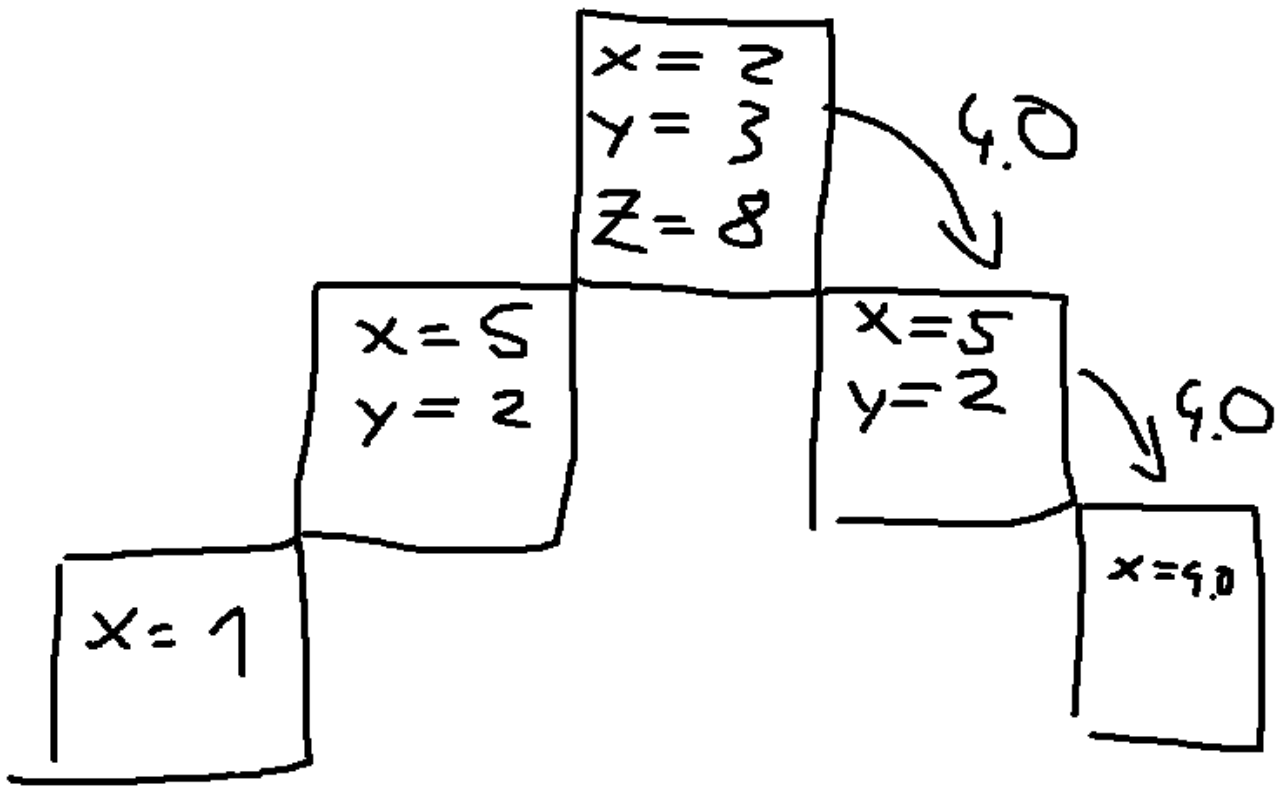
a)

aktuell: 5, formal: x

b)

4.0

c)



7

1

Rekursion, Funktionen höherer Ordnung, Keine Seiteneffekte, Bestehen nur aus Funktionsaufrufen und Definitionen

2

```

def explist(n):
    print("Aufruf von", n)
    e = []
    if (n==0):
        e = [1]
    else:
        e = explist(n-1) + [2**n]

    print("n=", n, e)
    return e

print(explist(5))

```

[1, 2, 4, 8, 16, 32]

3

```
from functools import reduce
le=reduce(lambda x,y: x+len(y),[[1,1],[1],[1,1,1],[1,1]],0)
```

$(((((0), 2), 1), 3), 2) \rightarrow 8$

4

a)

```
f=lambda x: x//2
l=list(map(f, [10,12,14,17,19]))

print(l)
```

b)

```
f=lambda x: 5<=x<=9
l=list(filter(f, [1,2,3,4,5,6,7,8,9,10,11,12,13]))

print(l)
```

5

Linkssequenz:

$(1, (8, (2, (12, ())))))$

Rechtssequenz:

$((((((), 1), 8), 2), 12)$

6

```
def rechtsAddFirst(xs, x):
    if(xs == ()):
        return ((), x)

    first = xs[0]
    last = xs[1]
```

```

        return (rechtsAddFirst(first, x), last)

xs = ((((((),1),8),2),12)
x = 5
print(rechtsAddFirst(xs,x))

```

```

def linksAddFirst(xs, x):
    if(xs == ()):
        return (x, ())

    first = xs[0]
    rest = xs[1]

    return (first, linksAddFirst(rest, x))

xs = (1,(8,(2,(12,()))))
x = 5
print(linksAddFirst(xs,x))

```

8

1

Werkzeuge um Quellcode in für Computer ausführbaren Code zu "übersetzen"

2

Interpreter:

Übersetzung Zeilenweise

Unmittelbare Ausführung, Zeile für Zeile

Compiler:

Übersetzung des gesamten Programmes vor der Ausführung

Ausführung des gesamten Programmes danach

3

x	y
4	8

```

1. LOAD r1 [4]    ; Lade den Wert von x in r1
2. LOAD r2 [8]    ; Lade den Wert von y in r2
3. GOLS r1 r2 6   ; Wenn r1 < r2 Gehe zu 6
4. STORE r1 [8]   ; Speicher den Wert in r1 in [8]
5. GOTO 7
6. STORE r2 [4]   ; Speicher den Wert in r2 in [4]
7. STOP

```

9

1

Eingabe:

Programm und Funktion

Ausgabe:

1 Falls das Programm die Funktion berechnet, sonst 0

2

Eingabe:

$n \in \mathbb{N}$

Ausgabe:

1 falls n Primzahl, sonst 0

3

Eine Menge M ist **abzählbar unendlich**, wenn sie gleichmächtig zur Menge \mathbb{N} der natürlichen Zahlen ist

Feb. 23

1

a)

Beschränkung auf (gerade) Verbindungen zwischen Kreuzungen (kein Verlauf, kein Material, kein Zustand)

b)

Abstraktion von Kreuzung als Knoten und Straßen als Pfade.

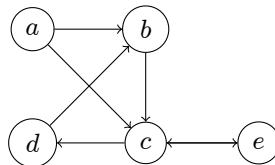
2

$$G = (V, E), V = \{a, b, c, d, e\}$$

a)

	a	b	c	d	e
a	0	1	1	0	0
b	0	0	1	0	0
c	0	0	0	1	1
d	0	1	0	0	0
e	0	0	1	0	0

b)



c)

$$K_1 = \{(b, c), (c, d), (d, b)\}$$

$$K_2 = \{(c, e), (e, c)\}$$

d)

$$Z_1 = \{(b, c), (c, e), (e, c), (c, d), (d, b)\}$$

e)

a, b, c, d, e

f)

a, b, c, d, e

3

$d(u, v) = \min(l \mid \text{es gibt ein Pfad } g \text{ der Länge } l)$

2

1

Algorithm Anzahl der Kanten

Input: Adjazenzmatrix L eines Gerichteten Graphen

Output: Anzahl der Kanten L

```
function ZÄHLEKANTEN( $L$ )  
     $z \leftarrow 0$   
    for all  $i$  in  $L$  do  
        if  $i$  is equal to 1 then  
             $z \leftarrow z + 1$   
        end if  
    end for  
    return  $z$   
end function
```

2

Algorithm Check for Duplicate Elements

Input: A list L of positive integers

Output: 1 if there are no duplicates, 0 if duplicates are found

```
function CHECKFORDUPLICATES( $L$ )  
    for  $i \leftarrow 0$  to length of  $L - 1$  do  
        for  $j \leftarrow 0$  to length of  $L - 1$  do  
            if  $i \neq j$  and  $L[i] == L[j]$  then  
                return 0  
            end if  
        end for  
    end for  
    return 1  
end function
```

3

1

```

n = int(input("Geben sie eine ganze positive Zahl an: ")) # 3
x = 0 # 1
z = 0 # 1

while (x < n): # n
    y = n # 1
    while y > 0: # n
        z += 1 # 2
        y = y - 1 # 2
    x += 1 # 2

```

$$(4n^2 + 3n + 5) \in O(n^2)$$

2

Obere Schranke:

$$2n^4 + 9n^2 \dots$$

4

```

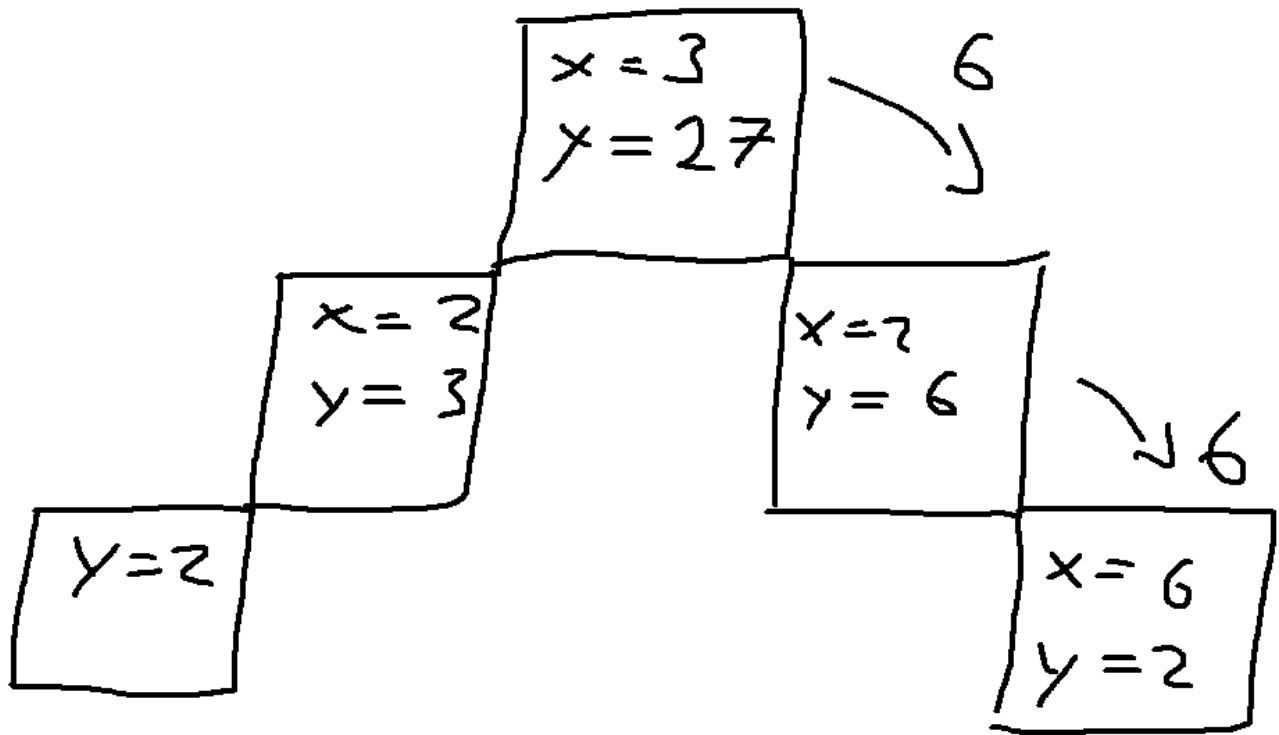
def f(x):
    y = x + 1
    y = g(y)
    return y

def g(x):
    y = 3 ** x
    return y // 4

y = 2
x = f(y)
print(x)

```

3



5

1+3

```
class Hund:
    def __init__(self, hungrig):
        self.hungrig = hungrig

    def fressen(self, menge):
        self.hungrig = self.hungrig - menge

    def bellen(self):
        for i in range(self.hungrig):
            print("wuf")

hund = Hund(10)
hund.fressen(4) # oder hund.fressen(hund.hungrig - 6)
hund.bellen()
```

6

2

```
def func(n):  
    x = 1  
    for i in range(n):  
        x = 3 * x  
    return(x)  
  
def funcRec(x):  
    if x == 0:  
        return 1  
    return funcRec(x-1)*3  
  
n = 10  
print(func(n))  
print(funcRec(n))
```

8

4

Unabhängig von externen Eingabewerten deshalb lösbar.