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

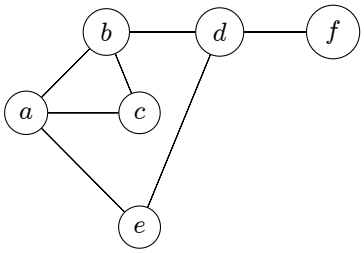
4

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

```python
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)
```

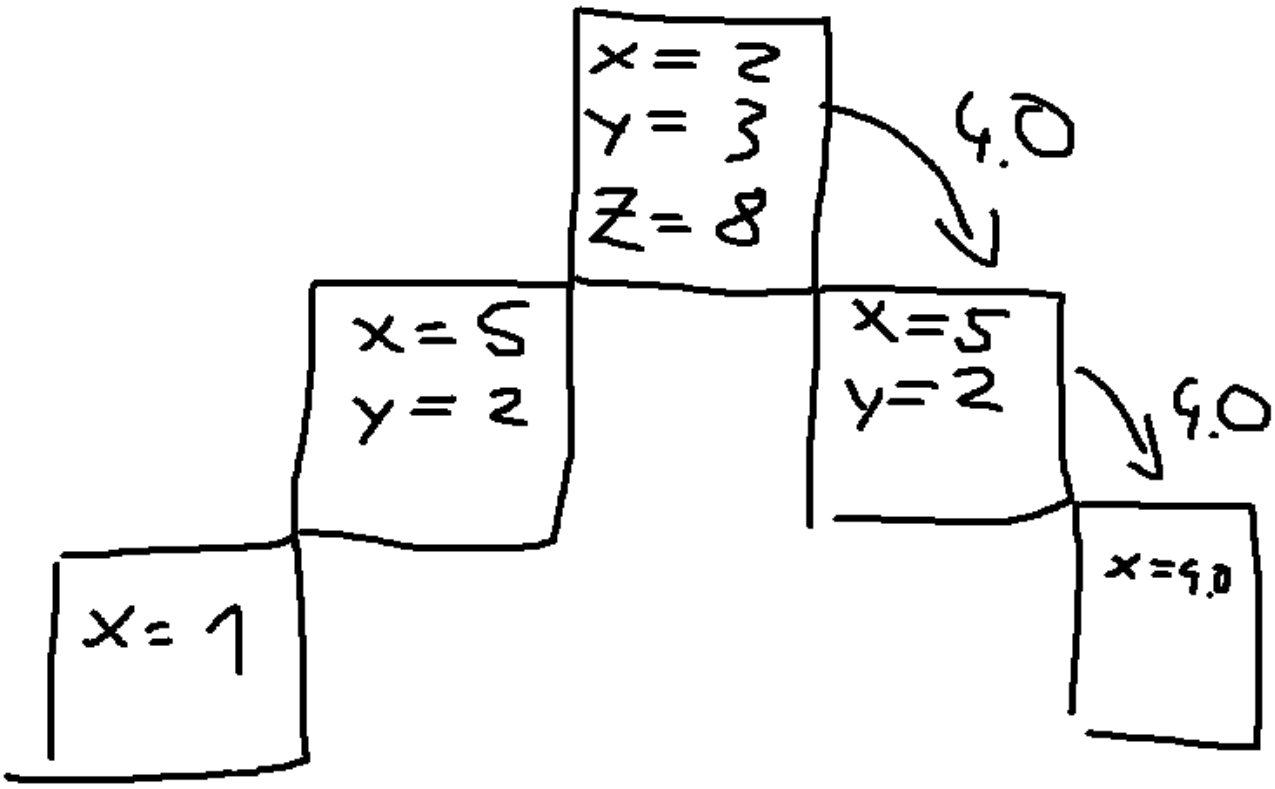
a)

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 "übersetzten"

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

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

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

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 j **in** L **do**
    for all i **in** j **do**
      if i **is equal to** 1 **then**
        $z \leftarrow z + 1$
      end if
    end for
  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
```

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

$3 + 1 + 1 + n(1 + 2 + n(2 + 2)) = 5 + n(3 + 4n) = 5 + 3n + 4n^2$

2

Obere Schranke:
$2n^4 + 9n^2$....

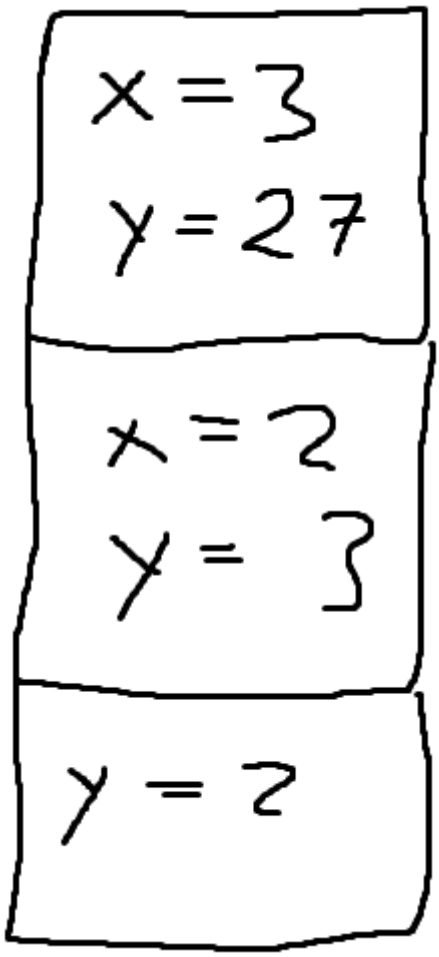
4

```
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

1

```
x = abs(x)
for i in range(1, x): # keine Schleifen in funktional
        x = x + 4
print(bin(x))
```

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))
```

3

```
L = [2,4,1,6,7]

# a)
L1 = list(map(lambda x: x+1, L))

# b)
L2 = list(filter(lambda x: x%2==0, L))
L3 = list(filter(lambda x: (x>=2 and x<=6), L))
L4 = list(filter(lambda x: x in [2,4,6], L))

print("a", L1)
print("b", L2)
print("b2", L3)
print("b3", L4)
```

4

```
# (2, (1, ()))

def occurs(xs, x):
        if xs == ():
                return False
        elif xs[0] == x:
                return True
        return occurs(xs[1], x)

print(occurs((2, (1, ())), 5))
print(occurs((2, (1, ())), 1))
```

7

1

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

2

Zeile für Zeile Interpretation beim Interpreter, gesamter Code kompiliert/übersetzt beim Compiler.

3

Übersetzt Assembler code in Maschinen Code (für Computer lesbar/verständlich)

4

Zeile	r1	r2	r3	x	y	z
0	-	-	-	1	1	1
1	8	-	-	1	1	1
2	8	1	-	1	1	1
3	8	1	1	1	1	1
4	8	2	1	1	1	1
5				1	1	1
8	8	2	1	1	1	1

Zeile	r1	r2	r3	x	y	z
9	3	2	1	1	1	1
10	3	2	1	1	1	3
11	3	2	1	1	1	2

x=1, y=1, z=2

8

4

Unabhängig von externen Eingabewerten deshalb lösbar.

Feb. 21

Funktional

ii

```
L = [8,80,800,8000,80000,800000] # map -> [8,8,8,8,8,8]

f = lambda x: int(x/(10**L.index(x)))
print(list(map(f, L)))

f = lambda x: L[0]
print(list(map(f, L)))
```

Nachsch. 23

1 Graphen

1

a)

Verzicht auf exakte Daten (Temperatur), Einordnung in Kategorien

b)

Verzicht auf andere Wetterlagen (Wind, Luftfeuchtigkeit, Regen)

2

Ein Gerichteter Graph ist ein Paar $G(V, E)$, wobei V ist eine endlich Menge an Knoten und E eine Menge an geordneten Knotenpaaren.

3

1

2

```
  m n o p q
m 0 1 1 0 0
n 0 0 0 1 1
o 0 1 0 0 0
p 0 0 1 0 0
q 0 1 0 0 0
```

3

$K_1 = \{(n, p), (p, o), (o, n)\}$
$K_2 = \{(q, n), (n, q)\}$

4

$Z = \{(q, n), (n, p), (p, o), (o, n), (n, q)\}$

5

$n \to p \to q \to o$
$m \to n \to o \to p \to q$

6

$m \to n \to p \to o \to q$

2. Pseudocode

3. Ressourcen

2

$2n^4 + n^3 - 11 \in \Theta(n^4)$

für $O(n^4)$
$\exists n_0, c_{>0} \forall n : n \geq n_0 : f(n) \leq c \cdot g(n) \in O(n^4)$

$$\begin{array}{rcll} 2n^4 + n^3 - 11 & \leq & c \cdot n^4 & | \quad \text{c=11} \\ 2n^4 + n^3 - 11 & \leq & 11 \cdot n^4 & | \quad \text{n=0} \\ 2 \cdot 0^4 + 0^3 - 11 & \leq & 11 \cdot 0^4 & \\ -11 & \leq & 0 & \end{array}$$

für $\Omega(n^4)$
$\exists n_0, c_{>0} \forall n : n \geq n_0 : f(n) \geq c \cdot g(n) \in \Omega(n^4)$

$$\begin{array}{rcll} 2 \cdot n^4 + n^3 - 11 & \geq & c \cdot n^4 & | \quad c = 1 \\ 2 \cdot n^4 + n^3 - 11 & \geq & 1 \cdot n^4 & | \quad n = 2 \\ 2 \cdot 2^4 + 2^3 - 11 & \geq & 1 \cdot 2^4 & \\ 37 & \geq & 32 & \end{array}$$

3

b)

t_Q

4 Imperative

```
def f(x):
        y=x-2
        return g(y)

def g(x):
        y=3*x
        return y//5

y=5
x=f(y)
print(x)
```

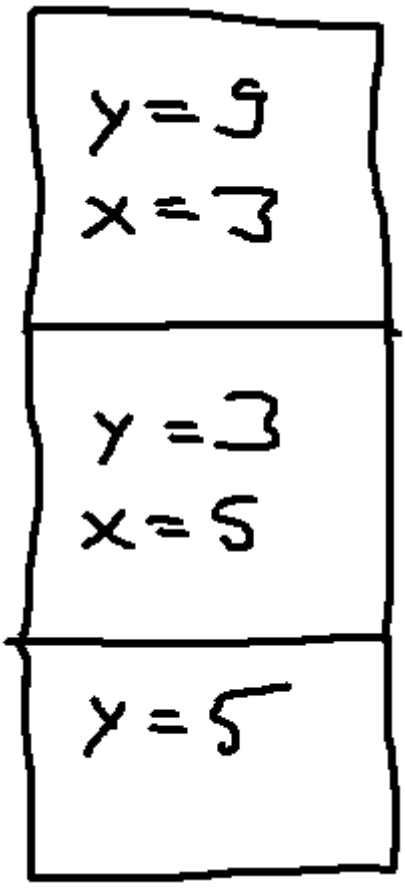
1

aktuell: 5
formal: x

2

```
1
```

3



5 Objekt

```
class Konto:
        def __init__(self, kontostand):
                self.kontostand = kontostand

        def abheben(self, betrag):
                if self.kontostand >= betrag:
                        self.kontostand -= betrag

        def __str__(self):
                return "Kontostand: " + str(self.kontostand)

meinKonto = Konto(10000)
print(meinKonto)
meinKonto.abheben(60000)
```

```
print(meinKonto)
meinKonto.abheben(6000)
print(meinKonto)
```

2

Datenelement: kontostand

6. Funktional

```
def g(x): # 1. Ordnung
        return x + 1
print(g(5))

def h(f, x, y, z, a): # 2. Ordnung, wenn f 1. Ordnung ist
        return f(x)

print(h(g, 5))
```

2

```
def func(n):
        x = 1
        for i in range(1, n+1):
                x = 3*x
        return x

def rec_func(n):
        if n == 0:
                return 1
        return rec_func(n-1) * 3

print(func(5))
print(rec_func(5))
```

3

```
L=[5,7,8,2,0]

# a)
f=lambda x: x-2
```

```
print(list(map(f, L)))

# b)
f=lambda x: x%2 != 0
print(list(filter(f, L)))
```

4

```
# ((), 1), 2)

def occurs(xs, x):
        if xs == ():
                return False
        elif xs[1] == x:
                return True
        return occurs(xs[0], x)

print(occurs(((((), 4), 2), 5), 5))
print(occurs(((((), 4), 2), 5), 7))
```

7 Comiler

3

```
1. LOAD r1 [4]   ; Lade x in r1
2. LOAD r2 [8]   ; Lade y in r2
3. GOGR r1 r2 7  ; Gehe zu 7 wenn r1 > r2
4. ADD r2 r2 1   ; r2 = r2 + 1
5. STORE r2 [8]  ; Speicher r2 in y
6. GOTO 9        ;
7. MUL r1 r1 3   ; r1 = r1 * 3
8. STORE r1 [4]  ; Speicher r1 in x
9. STOP          ;
```

8 Unentscheidbarkeit

1

Ausgabe immer `0` oder `1` (Ja oder nein, wahr oder falsch)

2

Eingabe: Funktion und Programm
Ausgabe: 1 falls das Programm die Funktion berechnet, 0 falls nicht

3

Eingabe: $n \in \mathbb{N}$
Ausgabe: 1 wenn n eine Primzahl, sonst 0

4

Unabhängig von externen Eingabewerten, deshalb lösbar.

Eine Menge M ist abzählbar unendlich wenn eine bijektive Funktion von $\mathbb{N} \to M$ gibt