Informatik 5. Klasse
Examensvorbereitung
Teil 2

OOP: Aufgabe 1

(a) Was ist die Grundidee der objektorientierten Programmierung (OOP)?

(b) Nenne zwei Konzepte, welche mit OOP eingeführt werden.

(c) Was ist eine Klasse?

(d) Was ist ein Objekt (eine Instanz)?

(e) Was sind Objektvariablen bzw. Klassenvariablen?

(f) Was sind Objektmethoden bzw. Klassenmethoden?

(g) Was ist ein Konstruktor?

OOP: Aufgabe 2

Erkläre möglichst genau mit Fachausdrücken, was das folgende Python-Programm macht.

```
class Example:

    a = 5

    def __init__(self, x):
        self.x = x

    def __str__(self):
        return('{0}'.format(self.x))

    def bar(self, y):
        self.x += y

    def foo(b):
        Example.a = b

e = Example(1)
e.bar(3)
print(e)
Example.foo(7)
print(Example.a)
```

OOP: Aufgabe 3

Erkläre möglichst genau mit Fachausdrücken, was das folgende Python-Programm macht.

```
1  class Stack:
2
3      def __init__(self):
4          self.items = []
5
6      def __str__(self):
7          return('{0} <top>'.format(self.items))
8
9      def __len__(self):
10         return len(self.items)
11
12     def push(self, x):
13         self.items.append(x)
14
15     def pop(self):
16         return self.items.pop()
17
18 s = Stack()
19 s.push(5)
20 s.push(3)
21 s.push(4)
22 print(len(s))
23 x = s.pop()
24 y = s.pop()
25 s.push(x+y)
26 print(s)
```

OOP: Aufgabe 4

Erkläre möglichst genau mit Fachausdrücken, was das folgende Python-Programm macht.

```
class Queue:

    def __init__(self):
        self.items = []

    def __str__(self):
        return 'in {0} out'.format(self.items)

    def __len__(self):
        return len(self.items)

    def enqueue(self, item):
        self.items.insert(0, item)

    def dequeue(self):
        return self.items.pop()

q = Queue()
q.enqueue(4)
q.enqueue(7)
x = q.dequeue()
print(x)
q.enqueue(1)
q.enqueue(2)
print(q)
print(len(q))
```

OOP: Aufgabe 5

Erkläre möglichst genau mit Fachausdrücken, was das folgende Python-Programm macht.

```
1  class Point:
2
3      def __init__(self, x, y):
4          self.x = x
5          self.y = y
6
7      def __str__(self): # überschreiben der str()-Funktion
8          return('({0}, {1})'.format(self.x, self.y))
9
10     def add(self, other):
11         x = self.x + other.x
12         y = self.y + other.y
13         return Point(x, y)
14
15     def swap(self):
16         self.x, self.y = self.y, self.x
17
18 P = Point(3, 4)
19 Q = Point(1, 2)
20 R = P.add(Q)
21 Q.swap()
22
23 print(P)
24 print(R)
25 print(Q)
```

OOP: Aufgabe 6

Erkläre möglichst genau mit Fachausdrücken, was das folgende Python-Programm macht und vervollständige den Code der Methode `umfang()` in Zeile 15.

```
1  class Rechteck:
2
3      def __init__(self, a, b):
4          self.a = a
5          self.b = b
6
7      def __str__(self):
8          txt = 'Länge: {0}, Breite {1}'.format(self.a, self.b)
9          return txt
10
11     def inhalt(self):
12         return self.a * self.b
13
14     def umfang(self):
15         ...
16
17 class Quadrat(Rechteck):
18
19     def __init__(self, a):
20         super().__init__(a, a)
21
22 r = Rechteck(3,4)
23 print(r.inhalt())
24 q = Quadrat(5)
25 print(q.inhalt())
```

Heiratsproblem: Aufgabe 1

Beantworte die folgenden Fragen zum Modell des Heiratsproblems.

(a) Zähle drei Voraussetzungen auf, welche die Kandidatenliste haben muss, damit sich das Heiratsproblem lösen lässt.

(b) Mit welcher Strategie kann das Heiratsproblem gelöst werden?

(c) Welche Eigenschaft hat die Lösung des Heiratsproblems?

Heiratsproblem: Aufgabe 2

Gib den Rang des Kandidaten an, der aufgrund „unserer“ Strategie ausgewählt werden muss, wenn die Begutachungsphase die Länge r hat.

(a) [8, 4, 10, 2, 5, 1, 3, 9, 7, 6], $r = 4$

(b) [2, 1, 6, 7, 4, 5, 10, 9, 3, 8], $r = 5$

(c) [2, 1, 4, 6, 5, 10, 9, 7, 8, 3], $r = 2$

(d) [7, 4, 1, 6, 2, 3, 10, 8, 9, 5], $r = 0$

Heiratsproblem: Aufgabe 3

Beschreibe die beiden für die partnersuchende Person ungünstigen Situationen, wenn sie die oben genannten Strategie befolgt.

Heiratsproblem: Aufgabe 4

(a) Die Elemente von $M = \{1, 2, 3\}$ stellen die Ränge potenzieller Partner dar. Notiere in der folgenden Tabelle alle Reihenfolgen dieser Ränge und bestimme für $r = 0, 1, 2$ den Wert des zu wählenden Partners mit der optimalen Strategie. Wenn es nicht möglich ist, eine Person zu bestimmen, wähle die letzte.

Permutation	$r = 0$	$r = 1$	$r = 2$

(b) Für welches r erhält man hier am häufigsten den optimalen Partner (also 3)?

(c) Wie gross muss man r bei optimaler Strategie für eine Liste der Länge n wählen?

Heiratsproblem: Aufgabe 5

Eine Liste von Kandiaten bestehe aus 200 Personen. Wie viele Kandidaten sollte man etwa ablehnen, wenn die optimale Strategie befolgt wird.

Heiratsproblem: Aufgabe 6

Auf wie viele Arten kann man 7 verschiedene Objekte in eine Reihenfolge bringen?

Heiratsproblem: Aufgabe 7

Welche Ausgabe macht das folgende Python-Codefragment?

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

Sortieren: Aufgabe 1

Der folgende Quellcode zeigt die Definition eines uns bekanntes Sortierverfahrens, das die Liste `L` $= [a_0, a_1, \ldots a_{n-1}]$ im Argument inplace sortiert.

```
def sort(L):
    n = len(L)
    i = 0
    while i < n:
        if i == 0 or L[i-1] < L[i]:
            i = i + 1
        else:
            L[i-1], L[i] = L[i], L[i-1]
            i = i - 1
```

(a) Um welchen Sortieralgorithmus handelt es sich? Woran erkennst du das?

(b) Zeige schrittweise, wie dieser Sortieralgorithmus die folgende Liste sortiert und gib die Anzahl der dafür nötigen Vergleiche an.

```
L = [3, 1, 4, 2]
```

(d) Gib die asymptotische Laufzeitklasse für den Best Case und den Worst Case von Listen der Länge n an. Wie sind die Listen dann aufgebaut?

Sortieren: Aufgabe 2

Der folgende Quellcode zeigt die Definition eines uns bekanntes Sortierverfahrens, das die Liste `L` $= [a_0, a_1, \ldots a_{n-1}]$ im Argument inplace sortiert.

```
def sort(L):
    n = len(L)
    for i in range(0, n-1):
        k = i
        for j in range(i+1, n):
            if L[j] < L[k]:
                k = j
        L[i], L[k] = L[k], L[i]
```

(a) Um welchen Sortieralgorithmus handelt es sich? Woran erkennst du das?

(b) Zeige schrittweise, wie dieser Sortieralgorithmus die folgende Liste sortiert und gib die Anzahl der dafür nötigen Vergleiche an.

```
L = [3, 1, 5, 2, 4]
```

(c) Gib die asymptotische Laufzeitklasse für den Best Case und den Worst Case von Listen der Länge n an. Wie sind die Listen dann aufgebaut?

Sortieren: Aufgabe 3

Der folgende Quellcode zeigt die Definition eines uns bekanntes Sortierverfahrens, das die Liste `L` $= [a_0, a_1, \ldots a_{n-1}]$ im Argument inplace sortiert.

```
def sort(L):
    n = len(L)
    for i in range(1, n):
        x = L[i]
        j = i-1
        while j >= 0 and L[j] > x:
            L[j+1] = L[j]
            j = j-1
        L[j+1] = x
```

(a) Um welchen Sortieralgorithmus handelt es sich? Woran erkennst du das?

(b) Zeige schrittweise, wie dieser Sortieralgorithmus die folgende Liste sortiert und gib die Anzahl dafür nötigen Vergleiche an.

```
L = [4, 2, 6, 5, 1]
```

(c) Gib die asymptotische Laufzeitklasse für den Best Case und den Worst Case von Listen der Länge n an. Wie sind die Listen dann aufgebaut?

Sortieren: Aufgabe 4

Der folgende Quellcode zeigt die Definition eines uns bekanntes Sortierverfahrens, das die Liste `L` $= [a_0, a_1, \ldots a_{n-1}]$ im Argument inplace sortiert.

```
def sort(L):
    n = len(L)
    for i in range(0, n-1):
        for j in range(0, n-i-1):
            if L[j] > L[j+1]:
                L[j], L[j+1] = L[j+1], L[j]
```

(a) Um welchen Sortieralgorithmus handelt es sich? Woran erkennst du das?

(b) Zeige schrittweise, wie dieser Sortieralgorithmus die folgende Liste sortiert und gib die Anzahl der dafür nötigen Vergleiche an.

```
L = [4, 8, 3, 6, 1]
```

(c) Gib die asymptotische Laufzeitklasse für den Best Case und den Worst Case von Listen der Länge n an. Wie sind die Listen dann aufgebaut?

Sortieren: Aufgabe 5

Schreibe eine Python-Funktion `is_sorted(L)`, die `True` zurückgibt, wenn die Liste `L` im Argument aufsteigend sortiert ist und `False` sonst.

Sortieren: Aufgabe 6

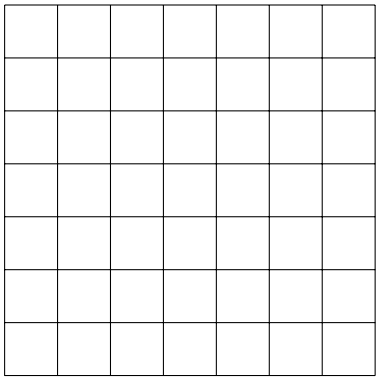
```
1  def quicksort(A):
2      quicksort_helper(A, 0, len(A))
3
4  def quicksort_helper(A, a, b):
5      if a < b:
6          m = partition(A, a, b)
7          quicksort_helper(A, a, m)
8          quicksort_helper(A, m+1, b)
9
10 def partition(A, a, b):
11     pivot = A[b-1]
12     i = a-1
13     for j in range(a, b-1):
14         if A[j] <= pivot:
15             i += 1
16             A[i], A[j] = A[j], A[i]
17     A[i+1], A[b-1] = A[b-1], A[i+1]
18     return i+1
```

(a) Zeige, schrittweise, wie der Partitionierungsfunktion von Quicksort die folgende Liste partitioniert.

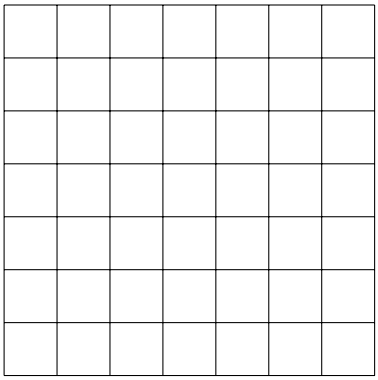
`L = [8, 9, 2, 1, 5, 4]`

(b) Zeige im dem folgenden Raster, wie eine Liste mit 7 Elementen im Best Case und im Worst Case von Quicksort partitioniert und sortiert wird. Notiere die Pivotposition und die Reihenfolge, in der die Teillisten sortiert werden.

Best Case



Worst Case



Sortieren: Aufgabe 7

```
1  def quicksort(A):
2      quicksort_helper(A, 0, len(A))
3
4  def quicksort_helper(A, a, b):
5      if a < b:
6          m = partition(A, a, b)
7          quicksort_helper(A, a, m)
8          quicksort_helper(A, m+1, b)
9
10 def partition(A, a, b):
11     pivot = A[b-1]
12     i = a-1
13     for j in range(a, b-1):
14         if A[j] <= pivot:
15             i += 1
16             A[i], A[j] = A[j], A[i]
17     A[i+1], A[b-1] = A[b-1], A[i+1]
18     return i+1
```

(a) Zeige, schrittweise, wie der Partitionierungsfunktion von Quicksort die folgende Liste partitioniert.

`L = [3, 6, 1, 7, 2, 9, 4]`

(b) Wie nennt man die Programmiertechnik, die in der Funktion `quicksort_helper(...)` angewendet wird?

(c) Beschreibe jeweils eine Typ von Liste, in welcher der Best Case bzw. der Worst Case auftritt und gib jeweils die asymptotische Laufzeitklasse für diese Fälle in Abhängigkeit der Listenlänge n an.

(d) Beschreibe zwei Modifikationen am Quicksort-Algorithmus, mit denen man den Worst Case (mit an Sicherheit grenzender Wahrscheinlichkeit) verhindern kann.