OF Informatik **Klasse 5d** **Examensstoff**

1. *Repetition Python*

   (a) Arithmetische und logische Operatoren, einfache Datentypen, Casting
   `+`, `-`, `*`, `/`, `**`, `//`, `%`, `not`, `and`, `or`, `int(obj)`, `float(obj)`, `str(obj)`, `bool(obj)`

   (b) Variablen, Zuweisungen, erweiterte Zuweisungen und Mehrfachzuweisungen

   (c) Bedingte Anweisungen, Verzweigungen und Vergleichsoperatoren
   `if`, `if/else`, `if/elif/.../elif/else`, `==`, `=!`, `<`, `<=`, `>`, `>=`,

   (d) Schleifen
   `for`, `while`, `break`, `continue`, `range(start, end[, step])`

   (e) Listen
   `+`, `*`, `len(L)`, `L[i]`, `L[-i]`, `L[i:j]`, `L[i:]`, `L[:j]`, `L[:]`, `L.append(item)`, `L.pop()`, `L.pop(i)`, `L.insert(pos,item)`, `L.max()`, `L.min()`, `L.sum()`, `L.count(item)`, `L.sorted()`, `L.reverse()`

   (f) Funktionen (mit und ohne Parameter, mit und ohne `return`, Rekursion)

   (g) Zeichenketten
   `+`, `*`, `len(s)`, `int(s)`, `float(s)`, `ord(s)`, `chr(s)`, `s[i]`, `s[i:]`, `s[:j]`, `s[i:j]`, `s[:]`, `s.upper()`, `s.lower()`, `s.replace(x, y)`, `s.split(sep)`, `s.join(list_of_str)`, `s.count(t)`, `s.format(...)`

   (h) Dictionaries
   `d=dict()` (oder `d={}`), `d[key]`, `d[key] = value`, `del d[key]` (oder `d.pop(key)`), `len(dict)`)

   (j) Erkennen von Programmfehlern (Syntaxfehler, Laufzeitfehler, logische Fehler)

   (k) Schreiben kleinerer Programme: Verarbeiten der Werte von Listen (Summe, Maximum, Minimum, ...), einfache Berechnungen

2. *Lösung des Heiratsproblems durch Computersimulation*

   (a) Problemstellung (Regeln)

   (b) Optimale Strategie

   (c) Simulation mit zufälligen Permutationen

3. *Objektorientierte Programmierung (OOP) mit Python*

   (a) Zentrale Idee

   (b) Klassen

   (c) Objekt- und Klassenvariablen

   (d) Objekt- und Klassenmethoden

   (e) Vererbung

   (f) Vorteile der OOP

4. *Algorithmen (allgemein)*

   (a) Was ist ein Algorithmus?

   (b) Laufzeitkomplexitäten

   (c) Grenzen der Berechenbarkeit

   (d) Euklidscher Algorithmus zur Berechnung des ggT's (klassisch und modern)

5. *Sortieralgorithmen*

   (a) Du kannst die folgende Verfahren an ihrem Python-Code erkennen, auf konkrete Beispiellisten anwenden und ihre Laufkeitkomplexität im Best- und im Worst-Case angeben.

   - Gnomesort
   - Selectionsort
   - Insertionsort
   - Bubblesort

   (b) Du kannst die Funktionsweise von Quicksort (insbesondere den Partitionierungsschritt) an einer einfachen Beispielliste aufzeigen.

   (c) Du kannst die Laufkeitkomplexität von Quicksort im Best- und im Worst-Case angeben sowie die Form der Listen für den Best- und den Worst-Case beschreiben.

   (d) Du kannst zwei Pivotstrategien von Quicksort angeben, mit denen der Worst Case verhindert werden kann.

6. *Halbordnungen* (*partielle Ordnungen*)

   (a) Darstellungen von Halbordnungen durch Hasse-Diagramme

   (b) gerichtete (azyklische) Graphen

   (c) Topologische Sortierung mit Anwendungen

7. *Suchalgorithmen*

   (a) Lineare Suche

   (b) Binäre Suche

   (d) „Naives“ Pattern Matchting

   (e) Algorithmus von Boyer-Moore-Horspool

8. *Das Travelling Salesman Problem (TSP)*

   (a) Problemstellung und Anwendungen

   (b) Vollständige kantengewichtete Graphen

   (c) Darstellung von kantengewichteten Graphen als Matrizen

   (d) Permutationen

   (e) Brute-Force-Lösung (Laufzeitkomplexität)

   (f) Nearest Neighbor-Heuristik (Laufzeitkomplexität)

   (g) Vor- und Nachteile von (e) und (f)