

# Einführung in die wissenschaftliche Programmierung

## 1 Allgemeine Hinweise zur Klausur

- Kennzeichnen Sie bitte jedes Blatt (insg. 7 Seiten) lesbar mit Ihrem Namen und die erste Seite zusätzlich mit Matrikelnummer und Unterschrift; trennen Sie die Heftung nicht auf.
- Bearbeiten Sie die Aufgaben auf den Angabenblättern. Sollte der Platz für einen Aufgabenteil nicht reichen, benutzen Sie bitte die Rückseite und machen Sie einen entsprechenden Vermerk bei der Aufgabe.
- Maximal können Sie 40 Punkte erreichen, bei jeder Aufgabe sind die für die jeweiligen Teilaufgaben vorgesehenen Punkte angegeben. Zum Bestehen müssen 17 Punkte erreicht werden.
- Als Hilfsmittel ist nur diese Gedächtnisstütze, die in der Klausur verteilt wird, zugelassen (mitgebrachte Ausdrucke sind nicht zugelassen).
- Die Bearbeitungszeit beträgt 60 Minuten.

## 2 Python kurzgefasst

### 2.1 Datentypen und Operationen

- Einfache Datentypen: Ganze Zahlen, Fließkommazahlen, Wahrheitswerte (Literele: `True` und `False`)
- Container: Strings (Literele: `'Text'`, Zeilenumbruch mit `\n`), Tupel  $x_0, x_1, \dots, x_{n-1}$  (ggf. mit `()` begrenzen), Listen  $[x_0, x_1, \dots, x_{n-1}]$  und Dictionaries  $\{i_0:x_0, i_1:x_1, \dots\}$ .
  - Zugriff auf Komponenten mittels `container[Index]`, Länge: `len(container)`.
  - Anfügen an Liste: `liste.append(neu)`; List comprehensions: z.B. `[ausdruck for name in liste]`
  - Slice einer Liste: `liste[start:stop]` bezeichnet Kopie der Elemente `start, \dots, stop-1` (Ersatzwerte, falls weggelassen, 0 bzw. `len(liste)`). Auf der linken Seite einer Zuweisung: Listenbereich ersetzen.
  - Liste  $[a, a+1, \dots, b-1]$  mit `range(a, b)`
  - Schlüssel/Werte/Einträge eines Dictionaries `d` als Liste: `d.keys()`, `d.values()`, `d.items()`
- Arithmetische Operationen: `+` (bei Strings, Tupeln und Listen: zusammenfügen), `-`, `*`, `/`, `**` (Potenzieren), `%` (Modulo; `%` zur Stringformatierung s.u.)
- Vergleichsoperatoren: `>`, `>=`, `<`, `<=`, `==`, `!=`, logische Verknüpfungen: `and`, `or`, `not`
- Umwandlung in String: `str(x)` / in ganze Zahl: `int(x)` / in Fließkommazahl: `float(x)`
- Zuweisung mittels `=`

### 2.2 Schleifen und Fallunterscheidungen

- for-Schleife: `for name in liste: block`
- while-Schleife: `while bedingung: block`
- Fallunterscheidung: `if bedingung: block`
- Fallunterscheidung mit `else`-Zweig: `if bedingung: block else: block` (Mehr Zweige mit `elif`)

## 2.3 Ein-/Ausgabe, Formatierung

- Ausgabe auf den Bildschirm: `print x, y, ...` (Komma am Ende unterdrückt den Zeilenumbruch)
- Formatierungsoperator `string % ausdruck bzw. tupel`: Stringdarstellung des Wertes bzw. der Werte ersetzen den/die Platzhalter im String. Wichtige Platzhalter: `%d` (ganze Zahl), `%g` (Fließkommazahl), `%s` (String). Zwischen `%` und dem Formatbuchstaben ggf. minimale Feldbreite (ganze Zahl) und/oder Fließkomma-Genauigkeit (Dezimalpunkt und ganze Zahl)
- Datei öffnen: `datei = open(Dateiname, Zugriff)` – Zugriff `'w'` (Schreiben) oder `'r'` (Lesen)
- Schreiben in Datei: `print >> datei, ...` oder `datei.write(string)`
- Lesen aus Datei: `datei.readline()` (eine Zeile lesen und als String zurückliefern, am Dateiende leerer String) oder `datei.readlines()` (Ganze Datei als Liste von Strings)
- Datei schließen: `datei.close()`

## 2.4 Funktionsdefinitionen

- Funktionen definieren: `def funktion(formalparameter) : block`
- Funktionsaufruf `funktion(aktualparameter)`
- Funktionsergebnis mittels `return ergebnis`

## 2.5 Klassendefinitionen

- Klassen definieren: `class klasse(basisklasse) : block`
- Funktionen in der Klassendefinition werden zu Methoden der Objekte. Erster Parameter ist der Bezug auf das Objekt selbst (`self`); wird beim Methodenaufruf automatisch hinzugefügt.
- Zugriff auf Attribute (und Methoden): `objekt.attribut` (ggf. mit `self` als Objektbezug)
- Objekte instanziiieren: `klasse(konstruktorparameter)`
- Methoden mit Spezialfunktion: Konstruktor `__init__(self, ...)` (beim Instanziiieren aufgerufen), `__str__(self)` (Darstellung als String bei `print` und `str`), `__add__(self, other)` und `__mul__(self, other)` (Addition/Multiplikation)

## 2.6 Reguläre Ausdrücke

<code>.</code>	beliebiges Zeichen außer newline	<code>\number</code>	n-ter zuvor gefundener Text
<code>^</code>	Beginn eines strings	<code>\d</code>	entspricht <code>[0-9]</code>
<code>\$</code>	Ende eines strings	<code>\D</code>	entspricht <code>[^0-9]</code>
<code>*</code>	Ausdruck beliebig oft	<code>\s</code>	entspricht beliebigem whitespace (= <code>[\t\n\r\f\v]</code> )
<code>+</code>	Ausdruck beliebig oft	<code>\S</code>	alles außer whitespace
<code>?</code>	Ausdruck Null- oder einmal	<code>\w</code>	beliebiges alphanumerisches Zeichen
<code>{m}</code>	Ausdruck genau m Mal	<code>\W</code>	beliebiges nicht-alphanumerisches Zeichen
<code>{m,n}</code>	Ausdruck m bis n Mal	<code>\A</code>	entspricht dem Beginn eines strings
<code>&lt;...&gt;?</code>	nicht gierig	<code>\Z</code>	entspricht dem Ende eines strings
<code>[...]</code>	Beliebiges Zeichen aus der Menge	<code>A B</code>	A oder B
<code>[^...]</code>	Zeichen nicht aus der Menge	<code>(...)</code>	Speichert den gefundenen Inhalt