Prof. Dr. Heinz Schmitz
Fachbereich Informatik
Hochschule Trier

Einführung in die Programmierung
Wintersemester 2024/2025
15. Januar 2025

Aufgabensammlung

Inhaltsverzeichnis

1. Programme entwickeln

1.3 Berechnen Sie die Summe der ersten $n = 28$ aufeinanderfolgenden Quadratzahlen mit Hilfe der folgenden Formel:

$$\sum_{k=1}^{n} k^2 = \frac{1}{6} n(n+1)(2n+1)$$

1.4 In einem Modul werden im Laufe eines Semester 84 Aufgaben gestellt. Für jede erfolgreich bearbeitete Aufgabe erhält man einen Punkt. Um die Prüfungsvorleistung (PV) zu erhalten müssen mindestens 45% der maximalen Punktzahl erreicht werden. Es gibt 14 Übungstermine.

1. Wieviele Punkte muss man im Durchschnitt pro Übungstermin erreichen, um die PV zu erhalten?
2. In den letzten drei Übungsterminen werden insgesamt 17 Aufgaben gestellt. Wieviele Punkte muss man vorher erreicht haben, um die PV noch erhalten zu können?
3. Angenommen, Studentin A hat bisher 12 Punkte erreicht und es verbleiben noch 8 Übungstermine. Wieviele Punkte muss A im Durchschnitt pro verbleibendem Übungstermin noch erzielen, um die PV zu erhalten?
4. Angenommen, es werden jede Woche genau 6 Aufgaben gestellt. Student B kommt wegen eines Auslandspraktikums erst später zurück an die Hochschule. Was ist der späteste Übungstermin, an dem B noch einsteigen und die PV erreichen kann?

1.5 Berechnen Sie die $17.$ Zahl der Fibonacci-Folge mit Hilfe der Formel von Moivre-Binet ($n = 17$):

$$fib(n) = \frac{1}{\sqrt{5}} \left(\left(\frac{1+\sqrt{5}}{2} \right)^n - \left(\frac{1-\sqrt{5}}{2} \right)^n \right)$$

2. Variablen, Ausdrücke, Anweisungen

2.3 Ändern Sie Ihre Lösung zu Aufgabe 1.3 so ab, dass Sie im Ausdruck eine Variable `n` verwenden, der Sie vorher 28 oder jede andere Zahl zuweisen können. Welches Ergebnis erhalten Sie für -5 und 3.5? In welchen Fällen stimmt die Gleichung aus Aufgabe 1.3 und in welchen nicht?

2.4 Ändern Sie Ihre Lösung zu Aufgabe 1.4 so ab, dass Sie Variablen für die Anzahl Aufgaben, den Prozentsatz für die Mindestpunktzahl und die Anzahl der Übungstermine einführen. Gehen Sie bei den zusätzlich vorgegebenen Zahlen in den Teilaufgaben entsprechend vor. Achten Sie auf die Wahl passender Namen und Kommentare.

2.5 Ändern Sie Ihre Lösung zu Aufgabe 1.5 so ab, dass Sie im Ausdruck eine Variable `n` verwenden. Welche Werte ergeben sich für $n = 9, 17, 26, 30$?

2.6 Geben Sie Lösungen einer quadratischen Gleichung in Normalform mittels p-q-Formel als Ausdruck an und verwenden Sie für p und q Variablen `p` bzw. `q`, die Sie vor dem Auswerten des Ausdrucks auf die gewünschten Werte setzen können.

Siehe 3.4.2 auf der Seite https://de.wikipedia.org/wiki/Quadratische_Gleichung.

Welche beiden numerischen Lösungen haben die folgenden Gleichungen jeweils?

$$x^2 - 3x - 10 = 0 \tag{1}$$
$$x^2 - x - 1 = 0 \tag{2}$$
$$3x^2 - 2x = 0 \tag{3}$$
$$3x^2 - 15x + 18 = 0 \tag{4}$$

2.7 Geben Sie einen Ausdruck an, der eine Zeichenkette so erzeugt, dass in `'Informatik'` alle Vokale n-fach erscheinen. Für $n = 3$ ergibt sich beispielsweise `'IIInfooormaaatiiik'`. Was erhalten Sie bei $n = -2$ und $n = 0.5$?

Ändern Sie Ihren Ausdruck so, dass zusätzlich auch aufeinanderfolgende Konsonaten n-fach wiederholt werden. Für $n = 2$ ergibt sich beispielsweise `'IInfnfoormrmaattiikk'`.

3. Funktionen

3.4 Betrachten Sie die Aufgabe 2.3 bzw. 1.3 und schreiben Sie eine Funktion `q_summe` mit Parameter `n`, die das Ergebnis mit **`print`** ausgibt. Rufen Sie Ihre Funktion `q_summe` mit den Daten aus Aufgabe 2.3 auf und machen Sie weitere eigene Tests.

3.5 Betrachten Sie die Aufgabe 2.4 bzw. 1.4 und schreiben Sie eine Funktion `vorleistung` mit drei Parametern für die Anzahl der Aufgaben, den Mindestprozentsatz für das Erreichen der PV und die Anzahl der Übungstermine. Die Funktion soll die Ergebnisse der Teilaufgaben aus 1.4 nacheinander mit **`print`** ausgeben.

Rufen Sie Ihre Funktion `vorleistung` mit den Daten aus Aufgabe 1.4 auf, und auch mit den folgenden Angaben (jeweils in der Reihenfolge Anzahl Aufgaben, Prozentsatz, Anzahl Übungstermine):

(a) $64, 51, 11$ (b) $100, 30, 15$ (c) $10, 60, 1$ (d) $1000, 1, 0$

3.6 Betrachten Sie die Aufgabe 2.5 bzw. 1.5 und schreiben Sie eine Funktion `binet` mit Parameter `n`, die das Ergebnis mit **`print`** ausgibt. Rufen Sie Ihre Funktion `binet` mit den Daten aus Aufgabe 2.5 auf und machen Sie weitere eigene Tests.

3.7 Betrachten Sie die Aufgabe 2.6 und schreiben Sie eine Funktion `pqFormel` mit den Parametern `a,b,c` (Koeffzienten der quadratischen Gleichung), die die Lösungen x_1 und x_2 mit **`print`** ausgibt. Verwenden Sie bei den Aufrufen Ihrer Funktion `pqFormel` die Daten aus Aufgabe 2.6.

3.8 Die folgenden Funktionen haben alle den Parameter `x` und erwarten ein Argument von Typ **`int`**:

1. Schreiben Sie eine Funktion `f1`, die `x` um 1 erhöht, den Wert von `x` mit **`print`** ausgibt und anschließend einen Funktionsaufruf `f2(x)` macht.
2. Schreiben Sie eine Funktion `f2`, die `x` um 2 erhöht, den Wert von `x` mit **`print`** ausgibt und anschließend zwei Funktionsaufrufe `f3(x)` macht.
3. Schreiben Sie eine Funktion `f3`, die `x` um 3 erhöht und den Wert von `x` mit **`print`** ausgibt.

Welche Ausgabe liefert der Aufruf `f1(0)`? Zeichnen Sie analog zur Vorlesung das zugehörige Stapeldiagramm.

4. Gestaltung von Schnittstellen

4.1 1. Siehe Buch.

   Verwenden Sie die Datei `EiP_04_polygon6.py` aus der Vorlesung und den Aufruf `kreis(bob,100)`.

4.2 Siehe Buch.

Verwenden Sie die Funktion `bogen` aus der Datei `EiP_04_polygon6.py` aus der Vorlesung. Beginnen Sie mit einer Funktion, die ein einzelnes Blütenblatt zeichnet.

4.3 Siehe Buch. Beginnen Sie mit einer Funktion, die ein einzelnes 'Kuchenstück' zeichnet. Welche geometrische Figur ist das und durch welche Parameter kann man diese repräsentieren?

4.4 Siehe Buch.

4.5 Siehe Buch.

4.6 Betrachten Sie die Aufgabe 3.7 und schreiben Sie passende Funktionen `pqFormel` und `abcFormel`, jeweils inkl. Docstring. Wie können Sie ausnutzen, dass die Formeln zur Berechnung beider Lösungen sehr ähnlich sind?

5. Bedingungen und Rekursion

5.7 Überlegen Sie, wie Sie die Summe der ersten $n \in \mathbb{N}$ Quadratzahlen rekursiv berechnen können. Geben Sie diese Summe und ebenso die Werte $n^2, (n-1)^2, \ldots, 1^2$ aus.

Was passiert beim Aufruf Ihrer Funktion mit dem Argument -1 und wie könnten Sie in Ihrer Funktion darauf reagieren?

Importieren Sie Ihre Lösung aus Aufgabe 3.4 und vergleichen Sie die Ergebnisse bei verschiedenen Eingaben.

5.8

1. Schreiben Sie eine Funktion `teiler_bis_9(n)`, die alle Zahlen $1 \leq a \leq 9$ dahingehend prüft, ob a ein Teiler von n ist. Geben Sie a aus, falls diese Bedingung erfüllt ist, ansonsten nicht. Richten Sie es so ein, dass n in der Konsole eingegeben werden kann.
2. Schreiben Sie eine Funktion `alle_teiler(n)`, die unter Verwendung einer rekursiven Funktion alle Teiler von n ausgibt.
3. Rufen Sie `alle_teiler(n)` nach einer Benutzereingabe von n auf. Probieren Sie aus, welches maximale n möglich ist, bevor die maximale Rekursionstiefe erreicht wird.
4. Recherchieren Sie, wie Sie die maximale Rekursionstiefe ermitteln können.

6. Funktionen mit Rückgabewert

6.1 Siehe Buch. Zeichnen Sie das Stapeldiagramm für den Moment, wenn die Ausführung die **`return`**-Zeile der Funktion `a` erreicht. Was wird am Ende mit **`print`** ausgegeben?

6.2 Siehe Buch.

6.3 Siehe Buch. Bestimmen Sie vor Ihrer Implementierung der zweiten Teilaufgabe mindestens 7 Testfälle mit Sollergebnissen, die u.a. die Eingaben aus der ersten Teilaufgabe umfassen.

6.4 Siehe Buch. Die Rekursion läuft über den Parameter a. Betrachten Sie die Fälle $a = 0$, $a = 1$ und $a \geq 2$ separat.

6.5 Siehe Buch. Bestimmen Sie vor Ihrer Implementierung mindestens 7 Testfälle mit Sollergebnissen.

6.6 Zerlegen Sie die Funktion `vorleistung` aus Aufgabe 3.5 in einzelne Funktionen mit Rückgabewert und passen Sie die Parameter entsprechend an. Lagern Sie mehrfache Berechnungen derselben Zwischenwerte in weitere Funktionen aus. Verwenden Sie für das Testen die Daten aus der vorherigen Aufgabe.

6.7 Ändern Sie Ihre Lösung aus Aufgabe 4.6 so, dass Sie anhand der Diskriminate die Fälle unterscheiden, dass es eine, keine oder zwei reelle Lösungen gibt.

7. Iteration

7.4 Ändern Sie die Funktion `sequenz(n)` für die Collatz-Folge aus der Vorlesung so zu einer Funktion `sequenz_laenge(n)` ohne **`print`**-Ausgaben ab, dass sie die Länge der Folge zurückliefert. Bei den Startwerten $n = 1, 5, 10$ ergeben sich beispielsweise die Längen 1, 6 bzw. 7.

Bei welchem Startwert unter 100000 ergibt sich die längste Collatz-Folge und wie lang ist diese?

7.5 Schreiben Sie eine Funktion `primtest(n)`, die für $n \geq 2$ testet, ob n eine Primzahl ist. Verwenden Sie eine **`while`**-Schleife über die möglichen Teiler von $2, \ldots, \sqrt{n}$ und überlegen Sie, wann Sie die Funktion vorzeitig beenden können. Ist 3203431780337 eine Primzahl?

7.6 Für die *Eulersche Zahl* e gilt die Gleichung

$$e = 1 + \frac{1}{1} + \frac{1}{1 \cdot 2} + \frac{1}{1 \cdot 2 \cdot 3} + \frac{1}{1 \cdot 2 \cdot 3 \cdot 4} + \cdots = \sum_{k=0}^{\infty} \frac{1}{k!}$$

Dabei werden die Summanden für größere k immer kleiner. Schreiben Sie eine Funktion `my_euler()`, die e annähert bis der letzte Summand kleiner 10^{-6} (in PYTHON: `1e-6`) ist. Können Sie eine Implementierung erreichen ohne die Fakultätsfunktion aufzurufen?

Vergleichen Sie die von Ihnen berechnete Zahl mit `math.e`.

8. Strings

8.6 Schreiben Sie eine Funktion `common_prefix(v, w)`, die für zwei Zeichenketten das gemeinsame Präfix maximaler Länge zurückliefert.

Für `v="abcdef"` und `a="ab12"` ist dies beispielsweise `"ab"`.

8.7 Eine Palindromzahl besteht vorwärts und rückwärts gelesen aus der gleichen Ziffernfolge. Die größte Palindromzahl, die aus dem Produkt zweier zweistelliger Zahlen gebildet werden kann, ist $9009 = 91 \cdot 99$. Was ist die größte Palindromzahl, die aus dem Produkt zweier dreistelliger Zahlen gebildet werden kann?

9. Wortspiele

9.10 Schreiben Sie eine Funktion `verwendet_genau(wort, buchstaben)` nach dem Muster *Suche*, die ein Wort und einen String mit Zeichen erwartet und `True` zurückliefert, falls das Wort aus genau diesen Zeichen besteht. Wieviele Wörter gibt es in *wortliste.txt*, die genau aus den Buchstaben g, e, t, n und r bestehen?

Geben Sie eine zweite Version Ihrer Funktion an, die die Aufgabe durch Reduktion löst.

9.11 Schreiben Sie eine Funktion `datei_oeffnen(datei)`, die einen Dateinamen inkl. Endung als String erwartet und das Dateiobjekt zurückliefert. Die Funktion soll die Anzahl der Zeilen in der Datei per **`print`** ausgeben und die Leseposition wieder zurück auf den Anfang setzen. Wieviele Zeilen enthält *buddenbrooks.txt*?

9.12 Schreiben Sie eine Funktion `verboten_zaehlen(fin, verboten)`, die ein Dateiobjekt und einen String mit Kleinbuchstaben erwartet. Die Funktion soll die Anzahl der Worte in der Datei zählen, die keinen der verbotenen Buchstaben enthalten. Mit der Angabe der Kleinbuchstaben in `verboten` sind auch der entsprechenden Großbuchstaben verboten.

Wieviele Wörter aus *wortliste.txt* bleiben übrig, wenn a, e, i, und u verboten werden?

Bestimmen Sie drei verschiedene verbotene Zeichen, bei denen die Funktion `verboten_zaehlen` garantiert die maximale Anzahl Wörter ohne diese Zeichen liefert.

9.13 Wieviele der Palindrome in *wortliste.txt* enthalten mindestens ein o und ein u, aber weder s noch r?

9.14 Betrachten Sie nochmal die Aufgabe 6.6 und Vorgängeraufgaben. Historische Daten zeigen, dass die durchschnittliche Punktzahl pro Aufgabe bei 0.518 liegt. Wieviele der 121 Studierenden, die in diesem Semester das Modul belegt haben, werden voraussichtlich die PV erhalten?

Betrachten Sie dabei die aufeinanderfolgenden Abgaben der Aufgaben als Bernoulli-Kette, siehe

`https://de.wikipedia.org/wiki/Bernoulli-Prozess`

und berechnen Sie zunächst die Erfolgswahrscheinlichkeit für genügend viele erfolgreich abgegebene Aufgaben.

10. Listen

10.1 Siehe Buch. Achten Sie drauf, dass die Argument-Liste `t` an der aufrufenden Stelle unverändert bleibt. Schreiben Sie ergänzend einige Testfälle u.a. mit leeren verschachtelten Listen, negativen Werte oder Datentyp **`float`**.

10.2 Siehe Buch (an der i-Stelle steht die Summe der erste i Elemente). Schreiben Sie eine Version, bei der das Argument nicht verändert wird, und eine zweite Version ohne **`return`**-Anweisung, bei der das Resultat an der aufrufenden Stelle hinterher im Argument ablesbar ist.

10.3 Siehe Buch. Ergänzen Sie Testfälle für Listen mit 0, 1, 2 und mehr Elemente und verschiedenen Datentypen. Schreiben Sie jeweils vorher die Sollergebnisse auf. Zeichen Sie ein Zustandsdiagramm für einen Aufruf von `middle(t)`

10.4 Siehe Buch. Ergänzen Sie Testfälle für Listen mit 0, 1, 2 und mehr Elemente und verschiedenen Datentypen. Schreiben Sie jeweils vorher die Sollergebnisse auf. Zeichen Sie ein Zustandsdiagramm für einen Aufruf von `chop(t)`.

10.5 Siehe Buch.

10.6 Siehe Buch. Welches Ergebnis erhalten Sie, wenn Sie die Funktion mit zwei Listen aufrufen, die jeweils die gleichen Zahlen enthalten?

10.7 Siehe Buch. Funktioniert Ihre Funktion auch für Strings?

10.8 Siehe Buch.

10.9 Siehe Buch.

10.10 Siehe Buch. Spielen Sie vorher auf Papier vier Fälle mit einer Wortliste mit einer geraden und mit einer ungeraden Anzahl an Elementen durch, in denen jeweils das gesuchte Wort enthalten bzw. nicht enthalten ist.

Verwenden Sie zum Erstellen der Wortliste eine Funktion aus der vorherigen Aufgabe und recherchieren Sie die Bedeutung der Anweisung

`from` `...` **`import`** `...` `as` `...`

10.11 Siehe Buch.

10.12 Siehe Buch.

11. Dictionaries

11.1 Siehe Buch. Nennen Sie Ihre Funktion `erstelle_dictionary()` und verwenden Sie `None` als Werte.

Schreiben Sie eine zweite Funktion `erstelle_liste()` für den Geschwindigkeitsvergleich, die eine Liste aller Wörter liefert. Messen Sie die Zeit, wie lange die 25000-fache Suche eines Worten im Dictionary, in der Liste bzw. per Bisektion in der Liste dauert.

11.2 Siehe Buch.

11.3 Siehe Buch. Implementieren Sie zunächst die Ackermann-Funktion rekursiv. Verwenden Sie für die globale Lookup-Tabelle die Parameter `(m, n)` als Schlüsselwerte im Dictionary. Vergleichen Sie die Laufzeiten mit und ohne *Memoization* von m und n.

11.4 Siehe Buch.

11.5 Siehe Buch.

11.6 Schreiben Sie eine Funktion `add_number(contacts, name, number)`, wobei gilt:

- `contacts` ist ein Dictionary mit Schlüsse-Wert-Paaren vom Typ **`str:list`**,
- `name` ist von Typ **`str`**, und
- `number` ist von Typ **`int`**.

Ihre Funktion fügt `number` der Liste `contacts[name]` hinzu und vermeidet dabei doppelte Einträge. Gibt es noch keine solche Liste in `contacts`, wird sie neu angelegt.

Schreiben Sie eine weitere Funktion `same_numbers(contacts)`, die ein Dictionary zurückliefert, dessen Schlüssel alle Nummern sind, die in `contacts` vorkommen. Der zugehörige Wert ist jeweils eine Liste aller Namen, die diese Nummer in den Kontakten haben. Geben Sie angemessene Testfälle an.

11.7 In der Kaffeeküche Ihrer Firma ist eine Getränkeliste zu verwalten: Die Beschäftigten können dort Getränke entnehmen und vermerken Art und Anzahl in der Liste. Die Bezahlung erfolgt monatlich im Sekretariat. Aktuell sind folgende Preise festgelegt: Becher Kaffee €0,70, Becher Tee €0,60, kleine Wasserflasche €0,90. Hier ist ein beispielhafter Verlauf:

Person A: ein Kaffee

Person B: ein Tee

Person A: zwei Kaffee

Person C: ein Wasser

Person B: ein Wasser

Sekretariat: Person B bezahlt

Person D: ein Wasser, zwei Tee

Sekretariat: Wieviel muss Person A bezahlen?

Person A: zwei Kaffee

Person C: ein Wasser

Sekretariat: Wie groß ist die Summe aller noch offenen Beträge?

Person B: ein Wasser

Person C: ein Kaffee

Person B: zwei Tee Wasser

Person C: Welche Getränke habe ich noch nicht bezahlt?

Person D: ein Wasser, zwei Tee

Person B: ein Wasser, zwei Tee

Sekretariat: Person A bezahlt

Person D: ein Tee

Person E: drei Kaffee

Sekretariat: Person A hat die Firma verlassen

Person C: ein Kaffee

Person B: ein Wasser

Sekretariat: Preisänderung, Becher Kaffee kostet ab jetzt €0,80

Person C: ein Kaffee

Sekretariat: Wieviele Getränke sind insgesamt noch nicht bezahlt?

Person B: ein Kaffee

Sekretariat: Wie sieht die Gesamtliste aus?

Person D: ein Kaffee

Sekretariat: Es gibt neuerdings auch Apfelschorle in kleinen Flaschen zum Preis von €1,10

Person B: ein Wasser, zwei Tee

Person B: eine Apfelschorle

Person C: ein Wasser, zwei Apfelschorle

Person C bezahlt

Setzen Sie die 'digitale Getränkeliste' um, indem Sie ein globales Dictionary führen, in dem für jede Person wiederum ein Dictionary verwaltet wird, das zu jeder Getränkeart die Anzahl dieser Getränke enthält. Verwenden Sie zusätzlich ein globales Dictionary, das jeder Getränkeart den jeweiligen Preis zuordnet.

Schreiben Sie für die einzelnen Vorgänge passende Funktionen, so dass u.a. der obige Verlauf durch einzelne Funktionsaufrufe abgebildet werden kann. Berechnen Sie vor Ihrer Implementierung die Sollwerte für alle Bezahlvorgänge.

12. Tupel

12.1 Siehe Buch. Wählen Sie als Beispiel die `wortliste.txt`.

12.2 Siehe Buch.

12.3 Siehe Buch.

12.4 Siehe Buch.

12.5 Zwei Spieler wählen gleichzeitig genau eine der Möglichkeiten 'Papier', 'Stein' oder 'Schere'.

1. Schreiben Sie eine Funktion `kette(n)`, die eine Liste mit n vielen zufällig gewählten Einträgen 'Papier', 'Stein' oder 'Schere' enthält.
2. Schreiben Sie eine Funktion `spielverlauf(n)`, die eine Liste aus n vielen Paaren aus je zwei Werten 'Papier', 'Stein' oder 'Schere' liefert. Dabei bestehen die ersten Komponenten aus einer zufälligen Kette für Spieler 1 und die zweiten Komponenten aus einer zufälligen Kette für Spieler 2.
3. Wie oft gewinnt Spieler 1 bei $n = 10, 30, 50, 100$? Wiederholen Sie das Experiment für jeden Wert von n mindestens 1000-mal. Schreiben Sie unter anderem eine Funktion `auswertung(v)`, die für einen Spielverlauf `v` zurückgibt, wie oft Spieler 1 bei diesem Spielverlauf gewinnt.

13. Datenstrukturen

13.1

13.2

14. Dateien

14.1

14.2