Übungsblatt zur Programmierung: Kontrollstrukturen

Übungsaufgaben zur Programmierung

Prof. Dr. Nicolas Meseth

Reflektionsfragen

1. Was passiert, wenn mehrere `if`-Bedingungen hintereinander erfüllt sind? Welche Auswirkungen hat es, ob du `if`–`if`–`if` oder `if`–`elif`–`else` verwendest?
2. Warum ist die Reihenfolge der Bedingungen wichtig? Was passiert, wenn du zuerst `if x > 0:` und danach `elif x > 10:` schreibst?
3. Was ist deiner Meinung nach besser lesbar: viele verschachtelte `if`-Anweisungen oder wenige klar strukturierte Bedingungen? Wann wird Code durch viele Bedingungen unübersichtlich – und wie kann man das vermeiden?
4. In welchen Fällen ist es sinnvoller, ein Dictionary zur Umsetzung einer Fallunterscheidung zu verwenden, anstatt `if`–`elif`–`else` zu schreiben? Welche Vorteile hat das Dictionary bei festen Zuordnungen – und welche Nachteile, wenn die Bedingungen komplizierter werden?

Programmieraufgaben

1. Schreibe ein Programm für die Umwandlung von Punkten (0–100) in das übliche Notenschema der Hochschule Osnabrück. Die Bestehensgrenze liegt bei 50% der Punkte. Das Programm soll eine Punktzahl als Eingabe akzeptieren und die entsprechende Note ausgeben.
2. Schreibe ein Programm, das prüft, ob ein eingegebenes Jahr ein Schaltjahr ist. Ein Jahr ist Schaltjahr, wenn es durch 4 teilbar ist **und** nicht durch 100, **außer** es ist durch 400 teilbar.

3. Schreibe ein Spiel, bei dem der Computer eine zufällige Zahl zwischen 1 und 100 auswählt. Die spielende Person darf mehrmals raten, wobei der Computer nach jedem Versuch das Feedback „zu hoch" oder „zu niedrig" gibt. Das Spiel läuft weiter, bis die richtige Zahl erraten wurde. Zum Schluss zeigt der Computer die Anzahl der benötigten Versuche an. **Hinweis**: Ihr benötigt hierfür eine WHILE-Schleife!

4. Viele Webseiten verlangen sichere Passwörter, um deine Daten zu schützen. Schreibe ein Programm, das ein Passwort entgegennimmt und prüft, ob es bestimmten Sicherheitskriterien entspricht. Ein Passwort gilt als sicher, wenn es **mindestens 8 Zeichen lang ist**, **mindestens einen Großbuchstaben** und **mindestens eine Ziffer** enthält. Gib eine passende Rückmeldung aus: entweder „Passwort ist sicher." oder eine Liste der fehlenden Anforderungen wie „Passwort ist zu kurz und enthält keine Zahl."

5. Nicht jeder Tag kommt in jedem Monat vor – zum Beispiel gibt es keinen 31. Februar. Schreibe ein Programm, das **Tag** und **Monat** als Zahl einliest und prüft, ob das eingegebene Datum existieren kann. Berücksichtige dabei: Der Februar hat **höchstens 28 Tage**, die Monate **April, Juni, September und November** jeweils **30 Tage**, alle anderen **31 Tage**. Dein Programm soll ausgeben, ob das Datum gültig ist oder nicht – zum Beispiel „Gültiges Datum" oder „Ungültiges Datum: Der 31.4. existiert nicht."

6. Eine Nutzerin oder ein Nutzer gibt eine Zahl zwischen 1 und 7 ein. Dein Programm soll den passenden Wochentag ausgeben – also 1 → Montag, 2 → Dienstag, … 7 → Sonntag. Welche Methoden fallen dir ein, um die Zahlen den Wochentagen zuzuordnen? Gib bei ungültiger Eingabe (z. B. 0 oder 8) eine passende Fehlermeldung aus.