Aufgabenblatt 02

Die folgenden Aufgaben sind dafür ausgelegt im Terminal bearbeitet zu werden. Recherchiere im Internet, falls du nicht weiterkommst oder mehr über etwas wissen willst. Achte jedoch darauf Erklärungen nicht nur hinzunehmen. Hinterfrage, warum etwas funktioniert und wie es funktioniert. Experimentiere mit unterschiedlichen Herangehensweisen und probiere neues aus!

1. Hello, Bash!

Im Folgenden werden wir ein kleines Programm als `bash`-Skript implementieren. Das Programm wird am Start eine Zufallszahl zwischen 0 und 100 bestimmen. Danach wird euch auffordern eine Zahl einzugeben, wenn ihr richtig geraten habt, wird euch das Programm gratulieren und sich beenden. Falls ihr aber zu tief, oder zu hoch geraten habt, wird euch das Programm sagen, ob ihr zu tief, oder hoch geraten habt und euch so lange erneut fragen, bis ihr die Zahl gefunden habt.

Überlegt euch bevor ihr mit der Aufgabe fortfahrt, welche Funktionalitäten ihr dafür benötigt. Versucht diese einzelnen Funktionalitäten in der Shell zu reproduzieren.

1.1. Projekt erstellen

1. Erstellt zuerst ein neues Verzeichnis für eure zukünftigen Projekte.
2. Legt ein neues `Hello, Bash!` Projekt an.
3. Erstellt eine Datei in diesem Projekt, welche als Entwicklungslogbuch dienen soll, schreibt dort hinein wie ihr das Projekt über Zeit verändert und warum ihr diese Änderung vorgenommen habt.
4. Erstellte in diesem Projekt eine neue Datei mit dem Namen `mein_programm.sh`, oder wie ihr es auch nennen wollt.
5. Der Inhalt von `mein_programm.sh` sollte wie folgt aussehen:

```
#!/usr/bin/env bash

echo "Hello, world!"
```

6. Stellt sicher, dass die Datei ausführbar ist.
7. Führt die Datei aus, indem ihr das Script, bzw. einen Pfad zu dieser, in eure Shell eingebt.

Eines der einfachsten Programme, die ihr als ProgrammiererInnen schreiben, werdet ist wohl das `Hello, World!` Programm. Auch wenn es in der Regel nicht sonderlich spektakulär ist, erlaubt es euch zu prüfen, ob ihr eure Entwicklungsumgebung richtig eingerichtet habt.

1.2. User Input & Variablen

Nun sollt ihr euer Skript erweitern, sodass dieses erst nach dem Namen einer Person fragt und nach der Eingabe, diese Person dann grüßt.

- Wie genau stellt ihr euch den Ablauf dieses Programms vor?
- Soll dem User vor der Namenseingabe noch etwas angezeigt werden?
- Wie genau wollt ihr den Namen speichern / an den Teil vom Programm übergeben, der dann einen Gruß ausgibt?
- Wie gebt ihr vordefinierten Text zusammen mit der Eingabe aus?

Es ist beim Entwickeln von Programmen oft hilfreich über die Teilfunktionalitäten/Use Cases nachzudenken, und wie diese miteinander interagieren.

1.3. If-Bedingungen

Nachdem ihr euer Programm interaktiv gemacht habt, solltet ihr dem ganzen eine Logik geben. Hierfür werden wir das Programm anpassen, sodass es sich wie folgt verhält:

1. Ihr fragt am Anfang nach der Eingabe einer Zahl
2. Ihr prüft, ob die eingegebene Zahl größer, kleiner oder gleich einer von euch im Voraus festgelegten Zahl ist.
3. Abhängig davon, ob die eingegebene Zahl größer, kleiner oder gleich war, gebt ihr einen kurzen Text aus, welche die User darüber informiert, wie sie geraten haben.

Damit habt ihr schon die Kernfunktionalität von der ursprünglichen Programmvision implementiert! Jetzt geht es darum die User-Experience beim Verwenden eures Programms zu verbessern.

1.4. Mehrere Versuche

Damit die User das Skript nicht mehrfach ausführen müssen, bis sie die Zahl gefunden haben, werden wir nun eine Schleife implementieren, welche dafür sorgt, dass das Programm so lange fragt, bis die Zahl gefunden wurde.

- Überlegt euch, welche Teile von eurem Programm bei falscher Eingabe wiederholt werden sollen, und welche nicht.
- Wann ist die Abbruchbedingung der Schleife erreicht?

1.5. Generieren von Zufallszahlen

Euch wird sicher schon aufgefallen sein, dass es nicht sehr spannend ist das Spiel zu spielen, wenn ihr die gesuchte Zahl schon kennt. Daher ist es euer Ziel die gesuchte Zahl am Anfang des Programms zufällig auszuwählen (und in einer Variable abzuspeichern).

- Welche Möglichkeiten habt ihr unter `Bash` eine Zahl zwischen 0 und 100 zu generieren?
- Wie speichert ihr die Zahl für die Weiterbenutzung im Rest des Programms?

Wenn ihr diese Aufgabe abgeschlossen habt, dann habt ihr das Programm erfolgreich implementiert!

1.6. Gewinnstrategie

Überlegt euch eine Strategie, wie ihr das Spiel mit den wenigsten Versuchen gewinnen könnt. Notiert eure Gewinnstrategie.

- Gibt es Eigenschaften der Zufallszahlengenerierung, um vermuten, oder sogar bestimmen zu können, was die Zufallszahl sein wird?
- Wenn die Zufallszahl nicht zu erraten ist, was ist dann die beste erste Eingabe, die ihr machen könnt?
- Welche Eingabe ist die beste zweite Eingabe, wenn ihr wisst, dass die gesuchte Zahl über oder unter eurer vorherigen Eingabe liegt?

Spielt das Spiel 10-mal und schreibt euch auf wie viele Versuche ihr insgesamt gebraucht hat. Ist das bedeutsam mehr, oder weniger als die Versuche von anderen Gruppen? Wenn ja, woran könnte das liegen?

1.7. Fehlerbehandlung (Bonusaufgabe)

Macht euch nun darüber Gedanken inwiefern euer Programm sich unerwartet verhalten könnte.

- Ruft euer Programm Befehle auf, welche fehlschlagen können? Wenn ja, wie wirken sich diese Fehler auf euer Programm aus?
- Gibt es Eingaben, welche User machen können, mit welchen euer Programm nicht umgehen kann?
- Gibt es noch andere potenzielle Probleme?

Wenn ihr mögliche Probleme in eurem Programm identifiziert habt, wie würdet ihr vorgehen, um diese zu beheben? Was könnt ihr an eurem Programm verändern, damit diese Probleme nicht mehr auftreten können?

- Implementiert eure Fehlerbehebungen, testet diese und fragt dann eine andere Gruppe, ob die euer Programm „falsch verwenden“ kann.

1.8. Abstraktion (Bonusaufgabe)

Ziel dieser Aufgabe ist es das Lesen vom Programmcode einfacher zu machen, indem ihr repetitive Teile eures Programms an eine einzelne Stelle auslagert und diese dann vom Rest des Programm referenziert. Findet hierfür Teile von eurem Programm, welche gute Kandidaten für Abstraktion sind:

- Gibt es teile vom Programm, welche eine klar benennbare Aufgabe erfüllen, sich aber wiederholen?
- Gibt es Teile in eurem Programm, welche sich von der Funktionalität in ein bis zwei Wörter zusammenfassen lassen, aber kompliziert zu verstehen sind?

Sobald ihr die entsprechenden Teile identifiziert habt, könnt ihr am Anfang von eurem Programm `Bash`-Funktionen erstellen, die zu abstrahierende Funktionalität implementieren. Nachdem ihr die Funktionen erstellt habt, könnt ihr die komplexen Teile von eurem Programm durch Funktionsaufrufe ersetzen.

2. Hello, Python!

Nachdem ihr nun gelernt habt, wie ihr mehrere `Bash` Befehle miteinander in einem Script kombiniert und den Anwendungsfluss lenkt, werden wir die gleichen Prinzipien auf eine neue Programmiersprache anwenden: Python.

Warum eine neue Programmiersprache? Ziel dieser Aufgabe ist es euch natürlich zu vermitteln wie ihr in Python programmiert. Darüber hinaus ist aber auch das Ziel euch zu zeigen, dass es viele Konzepte gibt, die euch nicht nur Domänenspezifisch sind, sondern über viele Sprachen anwendbar sind. Und wenn ihr diese versteht und mit einem analytischen Denken an Probleme herangeht, ihr die meisten Probleme relativ einfach gelöst kriegt.

2.1. Python REPL

Ähnlich zu deiner Shell besitzt Python auch einen REPL Modus. Diesen kannst du starten, indem du `python` direkt in deiner Shell ausführst, ohne weitere Argumente zu übergeben.

1. Öffne das Python REPL.
2. Verwende die `print` Funktion von Python, um Text auszugeben.
3. Berechne das Ergebnis von $365 \cdot (((600 \cdot 5.6 + 10) \cdot 0.9 \cdot 0.08) \cdot 0.01)$
4. Berechne das Ergebnis von $20 \cdot 19 \cdot 18 \cdot 17 \cdot 16 \cdot 15 \cdot 14 \cdot 13 \cdot 12 \cdot 11 \cdot 10 \cdot 9 \cdot 8 \cdot 7 \cdot 6 \cdot 5 \cdot 4 \cdot 3 \cdot 2 \cdot 1$
5. Berechne $\frac{4}{3}$. Erhälst du das erwartete Ergebnis? Was sind die Verwendeten Datentypen? Ändert sich der Datentyp? Wenn ja, was würde passieren, wenn sich der Datentyp nicht ändert?
6. Verwende die `input` Funktion, um User Input zu bekommen.
7. Addiere zwei Strings. Was erwartest du was passiert? Was ist passiert?
8. Erstelle eine Liste, welche 8 unterschiedliche Zahlen enthält.
9. Finde heraus was ein Python `dictionary` ist, und wofür dieses gut ist.
10. Informiere dich wie du mithilfe von Pythons `random` Modul Zufallszahlen generierst. Wie kannst du eine Zahl generieren, die größer 1 ist und kleiner gleich 100 ist?
11. „Printe“ 1000 Zufallszahlen mithilfe einer Schleife.
12. Generiere 100-mal eine Zufallszahl und speichere die größte. Tipp: Hier willst du wahrscheinlich eine if-Bedingung innerhalb von einer Schleife verwenden.
13. Verlasse das Python REPL mit `STRG-D` oder indem du `exit()` tippst.

Was hat besonders gut geklappt, im Vergleich, zu der Shell die du bereits kennengelernt hast? Was war neu? Was hat mit der Shell besser funktioniert?

2.2. Python Skripte

Nachdem du dich nun ein bisschen mit Python vertraut gemacht hast, ist es nun an der Zeit das Wissen anzuwenden, um das Skript aus Abschnitt 1 zu reproduzieren.

1. Legt ein neues `Hello, Bash!` Projekt an.
2. Erstelle eine Datei `mein_programm.py` mit folgendem Inhalt:

```
#!/usr/bin/env python

print("Hello, world!")
```

3. Stellt sicher, dass die Datei ausführbar ist.
4. Führt die Datei aus, indem du das Script, bzw. einen Pfad zu dieser, in deine Shell eingibst. Du könntest die Datei auch ausführen, indem du diese dem Python Programm direkt übergibst: `python mein_programm.py`.

Verwende nun das was Wissen, das du bereits aus dem Python REPL kennst und passe die Datei so an, dass diese sich wie die `Bash`-Variante verhält.

2.3. Python Code Lesen

Kannst du herausfinden, was das folgende Pythonprogramm ausgeben wird, ohne dieses vorher ausgeführt zu haben?

Tipp: Das `%`-Symbol ist hier der Module-Operator, das heißt es wird der Rest zurückgegeben, der bei der Division zweier Zahlen übrig bleibt. Zum Beispiel ist `10 % 3 = 1`.

```python
def fizz_buzz(n):
  result = []
  for i in range(1, n + 1):
    if i % 3 == 0 and i % 5 == 0:
      result.append("FizzBuzz")
    elif i % 3 == 0:
      result.append("Fizz")
    elif i % 5 == 0:
      result.append("Buzz")
    else:
      result.append(str(i))
  return result

for value in fizz_buzz(10):
  print(value)
```

2.4. Textabenteuer! (Bonusaufgabe)

Erinnerst du dich an die Bücher, welche eine interaktive Geschichte erzählt haben? Die haben so funktioniert, dass du bei der ersten Seite angefangen hast zu lesen, und nach ein paar Seiten wurdest du vor eine Entscheidung gestellt: Wenn du A machst, dann lese weiter auf Seite 41 und wenn du dich für B entscheidest, dann lies weiter auf Seite 84, und so weiter. Genau sowas sollst du als interaktives Spiel implementieren! Fange hierbei mit folgendem unvollständigem Programm an und erweiter es um die Funktionen, die du haben willst und die Geschichte, die du erzählen willst.

(Code auf der nächsten Seite)

```
#!/usr/bin/env python

geschichte = {
    "anfang": {
        "text": "Du wachst morgen in deinem Bett auf. Es sind Propädeutika und du hast deinen Wecker nicht gehört. Was machst
du?",
        "optionen": {
            "Zähne putzen": "Ich putze als erstes meine Zähne, ich will ja nicht Gefahr laufen auf meine Klamotten Zahnpasta
zu bekommen.",
            "umziehen": "Ich ziehe mich als erstes um.",
            "ende": "Mein Schlafanzug ist so schick, dass der auch Straßentauglich ist, und Mundgeruch habe ich sowieso
keinen. Ich gehe direkt zur Uni."
        }
    },
    "Zähne putzen": {
        "text": "Du springst aus dem Bett und eilst ins Bad. In Rekordzeit sind deine Zähne geputzt. Jetzt musst du dich
entscheiden: Ziehst du dich noch um oder rennst du direkt zur Uni?",
        "optionen": {
            "umziehen": "Ein Schlafanzug ist vielleicht doch nicht das passende Outfit für die Uni. Ich ziehe mich schnell
um.",
            "ende": "Fürs Umziehen bleibt keine Zeit mehr. Ich schnappe meine Sachen und renne direkt zur Uni."
        }
    },
    "umziehen": {
        "text": "Du wirfst dir schnell ein paar Klamotten über. Ein Blick auf die Uhr zeigt: Viel Zeit bleibt dir nicht mehr.
Putzt du noch deine Zähne oder gehst du sofort los?",
        "optionen": {
            "Zähne putzen": "Ein bisschen Zeit für frischen Atem muss sein. Ich putze noch schnell meine Zähne.",
            "ende": "Keine Zeit mehr! Ich schnappe meine Tasche und mache mich sofort auf den Weg zur Uni."
        }
    },
    "ende": {
        "text": "Du hast es noch rechtzeitig zu den Propädeutika geschafft und niemand hat etwas bemerkt. Puh. Das war
knapp.",
        "optionen": { }
    }
}

def optionen_verfuegbar():
    len(geschichte[aktuelles_segment]['optionen'].keys()) > 0

aktuelles_segment = "anfang"

while optionen_verfuegbar():
    print(geschichte[aktuelles_segment]['text'])
    print()

    for segment, option in geschichte[aktuelles_segment]['optionen'].items():
        print("Option: " + segment)
        print(option)
        print()

    # TODO: User sollen nur vorgeschlagene Segmente eintippen dürfen.
    aktuelles_segment = input("Mit welchem Segment machst du weiter?\n")

# TODO: Wenn ein Segment ohne Optionen erreicht wird soll das Programm zwar beenden, es soll vorher aber noch der passende
Text abgespielt werden.
```