Programmieren mit Python

Teil 11: Ein Quizspiel programmieren

Dr. Aaron Kunert
aaron.kunert@salemkolleg.de

14. Dezember 2022

Projekt: Geografie-Quiz

Das *minimal viable product* (MVP)

Anforderungen:

- Es werden Länder aus einem JSON gelesen.
- Zu einem Land aus der Liste wird die Hauptstadt abgefragt.
- Es erscheinen 4 Lösungsmöglichkeiten (Multiple Choice).
- Durch Eingabe einer Zahl zwischen 1 bis 4 kann getippt werden.
- Nach Eingabe erscheint ein kurzes Feedback (Richtig/Falsch).

Beispielausgabe

```
Was ist die Hauptstadt von Frankreich?
(1) Bratislava
(2) Berlin
(3) Paris
(4) Stockholm


Antwort:
```

Beispielausgabe

```
Was ist die Hauptstadt von Frankreich?
(1) Bratislava
(2) Berlin
(3) Paris
(4) Stockholm


Antwort: 3
```

Beispielausgabe

```
Was ist die Hauptstadt von Frankreich?
(1) Bratislava
(2) Berlin
(3) Paris
(4) Stockholm


Antwort: 3


Das war korrekt!
```

Übung

Multiple Choice Frage

Lies die Datei `"countries.json"` ein. Nimm das erste Land aus der Liste (Afghanistan) und stelle die Frage nach der Hauptstadt mit 4 Antwortmöglichkeiten wie oben auf der Konsole dar.

Tip: Aufgabe 4 vom letzten Übungsblatt kann helfen

Lösung

Multiple Choice Frage

```
import random
import json
with open("countries.json") as file:
    countries = json.load(file)

capitals = []
for country in countries:
    capitals.append(country["capital"])

capital = capitals[0]
random.shuffle(capitals)
answer_options = capitals[0:3]
answer_options.append(capital)
random.shuffle(answer_options)
country = countries[0]

print(f"Was ist die Hauptstadt von { country['name'] }?\n")
for index,option in enumerate(answer_options):
    print(f"({ index + 1 }) { option }")
```

Welche Verbesserungen sind dringend nötig?

Schritte nach Priorität:

1. Antwort einlesen und Feedback geben (Antwort war richtig/falsch)
2. Statt Afghanistan soll ein zufälliges Land abgefragt werden
3. Sicherstellen, dass die Antwort-Optionen stets paarweise unterschiedlich sind

Übungen

Antwort einlesen

Frage eine Antwort ab.

Antwort prüfen

Entscheide, ob die Antwort richtig oder falsch ist, und gib das Ergebnis auf der Konsole aus.

Land zufällig auswählen

Sorge dafür, dass das abgefragte Land zufällig ausgewählt wird.

Lösungen

Antwort einlesen

```
...
for index, option in enumerate(answer_options):
  print(f"({index + 1}) {option}")
answer = input("Antwort: ")
```

Antwort prüfen

```
...
answer = input("\nAntwort: ")
index = int(answer) - 1
if answer_options[index] == country["capital"]:
  print("Das war korrekt.")
else:
  print("Das war leider falsch.")
```

Land zufällig auswählen

```
...
with open("countries.json") as my_file:
  countries = json.load(my_file)
random.shuffle(countries)
country = countries[0]
...
```

Möglicher Bug

Um die Lösungsmöglichkeiten zu generieren, werden 3 Haupstädte zufällig ausgewählt und dann die korrekte Hauptstadt hinzugefügt. Theoeretisch ist es möglich, dass die korrekte Hauptstadt schon bei den 3 zufälligen Haupstädten dabei war.

Ist nur ein kleines Problem

Das ist nur ein kosmetisches Problem. Die Funktionalität geht davon nicht kaputt.

Mögliche Lösung

Mittels `.remove()` kann man die korrekte Hauptstadt aus der Liste `capitals` entfernen.

Bessere Lösung

```
...
for current_country in countries:
  if current_country["capital"] != country["capital"]:
    capitals.append(current_country["capital"])
...
```

Warum ist die zweite Lösung besser?

Übung

Feature Request

Es sollen (zunächst) nur Hauptstädte aus Europa abgefragt werden.

Feature Request: Nur europäische Hauptstädte

```
...
with open("countries.json") as my_file:
  countries = json.load(my_file)

filtered_countries = []
for country in countries:
  if country["continent"] == "Europa":
    filtered_countries.append(country)

countries = filtered_countries

random.shuffle(countries)
...
```

Refactoring

Wie man immer mehr Feature Requests unterbringt

Mehrere Fragen

Ein Quiz mit nur einer Frage ist schon sehr lame. Daher sollen ab jetzt nacheinander 10 Fragen gestellt werden.

Problem

Das ganze wird langsam unübersichtlich.

Lösung

Refactoring

Definition

Als *Refactoring* von Code wird das Aufräumen bzw. Umstrukturieren von Code genannt, um ihn übersichtlicher und leichter erweiterbar zu machen. Die Funktionalität des Codes wird dabei nicht verändert.

Typische Maßnahmen sind:

- Hinzufügen von Kommentaren
- Struktur durch Leerzeilen
- Umbennung von Variablen/Funktionen
- Zerlegung des Codes in kleinere Schritte
- Extraktion von gleichen Werten in eine Variable
- Extraktion von Arbeitsschritten in Funktionen

Arbeitsschritte in unserem Projekt

1. Einlesen aller Länder
2. Länder filtern (nur Europa)
3. Ein Land (`country`) zufällig wählen
4. Liste der Lösungsmöglichkeiten anlegen (in Abhängigkeit von `country`)
5. Frage stellen
6. Antwort auswerten/Feedback geben

Mögliche Maßnahmen

- Umbenennung von `country` in `active_country`.
- Extraktion des Einlesens und Filterns in Funktionen.
- Extraktion der Erstellung der Lösungsmöglichkeiten in eine Funktion
- Extraktion der Anzeige der Frage in eine Funktion
- Extraktion der Auswertung der Antwort in eine Funktion

Einlesen

```
def get_countries():
  with open("countries.json") as my_file:
    return json.load(my_file)
```

Filtern

```
def filter_countries(countries, continent):
  result = []
  for country in countries:
    if country["continent"] == continent:
      result.append(country)
  return result
```

Erstellung der Antwort-Optionen

```
def get_answer_options(active_country, countries):
  capitals = []
  for country in countries:
    if country["capital"] != active_country["capital"]:
      capitals.append(country["capital"])
  random.shuffle(capitals)
  answer_options = capitals[0:3]
  answer_options.append(active_country["capital"])
  random.shuffle(answer_options)
  return answer_options
```

Anzeige der Frage

```
def display_question(active_country, answer_options):
  print("\n\n")
  print(f"Was ist die Hauptstadt von {active_country['name']}?\n")
  for index, option in enumerate(answer_options):
    print(f"({index + 1}) {option}")
```

Auswertung der Antwort

```
def check_answer(active_country, answer_options):
  answer = input("\nAntwort: ")
  index = int(answer) - 1
  if answer_options[index] == active_country["capital"]:
    print("Das war korrekt.")
  else:
    print("Das war leider falsch.")
```

Programmablauf

```
...
def get_countries():
...
def filter_countries(countries, continent):
...
def get_answer_options(active_country, countries):
...
def display_question(active_country, answer_options):
...
def check_answer(active_country, answer_options):
...
countries = get_countries()
countries = filter_countries(countries, "Europa")
random.shuffle(countries)
active_country = countries[0]
answer_options = get_answer_options(active_country, countries)
display_question(active_country, answer_options)
check_answer(active_country, answer_options)
```

Mehrere Fragen

Schreibe das Programm so um, dass nicht nur eine Frage, sondern 10 (verschiedene) Fragen gestellt werden.

Lösung

```
...
random.shuffle(countries)
for index in range(0,10):
  active_country = countries[index]
  answer_options = get_answer_options(active_country, countries)
  display_question(active_country, answer_options)
  check_answer(active_country, answer_options)
```

Ein bisschen Kosmetik

Je nach Größe der Konsole sieht man noch Teile der alten Frage, oder aber man sieht das Feedback nicht mehr richtig. Dies lässt sich durch eine schlaue Verteilung von Leerzeilen und einem Bestätigungsdialog lösen:

```
...
  print("\n"*100)
  print(f"Was ist die Hauptstadt von {country['name']}?\n")
...
...
  score = score + check_answer(active_country, answer_options)
  input("\nWeiter mit beliebiger Taste")
```

Übung

Feature Request: Korrektur

Statt nur dem Hinweis `"Das war leider falsch"` soll nun auch noch zusätzlich die richtige Antwort ausgegeben werden.

Beispiel:

```
Das war leider falsch. Die richtige Antwort wäre "Athen" gewesen.
```

Lösung

```
...
def check_answer(active_country, answer_options):
  answer = input("\nAntwort: ")
  index = int(answer) - 1
  correct_answer = active_country["capital"]
  if answer_options[index] == correct_answer:
    print("Das war korrekt.")
    return 1
  else:
    print(f"Das war leider falsch. Die richtige Antwort wäre \"{correct_answer}\" gewesen.")
    return 0
...
```

Übung

Feature Request: Punktzahl

Am Ende des Spiels soll angezeigt werden, wie viele Antworten richtig waren.

Hinweis

Führe eine globale Variable `score` ein und verwende entsprechende Rückgabewerte in der Funktion `check_answer`.

Lösung

```
score = 0
...
def check_answer(active_country, answer_options):
  answer = input("\nAntwort: ")
  index = int(answer) - 1
  if answer_options[index] == active_country["capital"]:
    print("Das war korrekt.")
    return 1
  else:
    print("Das war leider falsch.")
    return 0
...
for index in range(0,10):
  ...
  score = score + check_answer(active_country, answer_options)
  ...
print(f"\n\nDu hast {score}/10 Fragen richtig")
```

Übung

Feature Request: Mehr Kontinente

Wie muss man den Code umschreiben, damit Länder aus Europa *und* Asien abgefragt werden.

Hinweis

Schreibe die Funktion `filter_countries` so um, dass sie statt einem String `continent` eine Liste `continents` erwartet.

Lösung

```
...
def filter_countries(countries, continents):
  result = []
  for country in countries:
    if country["continent"] in continents:
      result.append(country)
  return result
...
countries = get_countries()
countries = filter_countries(countries, ["Europa", "Asien"])
...
```

Weitere Verbesserungsideen

- Besseres Fehlerhandling (was ist, wenn die Antwort nicht zwischen 1 und 4 ist?)
- Anzahl der Fragen variabel machen.
- Schwierigkeitsgrad variabel (d.h. Anzahl der Antwortmöglichkeiten) variabel machen
- Spiel vorzeitig beendbar machen.
- Menü zu Beginn, wo man die Kontinente angeben kann (Mehrfachauswahl)
- Neuer Fragentyp: Anzahl der Einwohner. Hier kann man das Mutliple-Choice anders aufbauen, oder durch explizite Antworten ersetzen.
- Falsch beantwortete Fragen in einer Trainingsrunde wiederholen.
- Den Punktestand unter einem Username abspeichern.
- Punktzahl von der Reaktionszeit abhängig machen.