Programmieren mit Python

Teil 11: Ein Quizspiel programmieren

Dr. Aaron Kunert
aaron.kunert@salemkolleg.de

09. November 2022

Dictionaries

Problemstellung

Eine Variable soll nicht nur die Namen von Ländern enthalten, sondern auch noch deren Hauptstadt.

Wie macht man das?

Lösung

```
capitals = {"Deutschland": "Berlin", "Spanien": "Madrid", "Italien": "Rom"}

country = input("Von welchem Land möchtest Du die Hauptstadt wissen?")

print(f"Die Hauptstadt von { country } ist { capitals[country] } Punkte")
```

Struktur eines *Dictionaries*

`my_dict = {key_1:value_1, key_2:value_2, ..., key_n:value_n}`

Das *Dictionary* `my_dict` enthält Schlüssel-Wert-Paare (*key-value-pairs*). Die Schlüssel müssen eindeutig und unveränderlich sein (z.B. vom Typ `string` oder `int`). Die Werte dürfen beliebige Datentypen sein.

Good to know

- Zur besseren Übersichtlichkeit werden Dictionaries oftmals wie folgt formatiert:

```
capitals = {
  "Deutschland": "Berlin",
  "Spanien": "Madrid",
  "Italien": "Rom"
}
```

- Dictionaries sind mutable, können also verändert werden.
- Dictionaries besitzen keine vernünftige Anordnung und können nicht geordnet werden.
- Ein Dictionary kann leer sein.

- Oftmals bietet es sich an, statt einem Dictionary eine Liste von Dictionaries zu verwenden:

```
countries = [
  {
    "name": "Deutschland",
    "capital": "Berlin",
    "pop": 82000000,
    "is_eu_member": True
  },
  # ...
  {
    "name": "Italien",
    "capital": "Rom",
    "pop": 65000000,
    "is_eu_member": True
  }
]
```

Auf Dictionary-Elemente zugreifen

Sei `my_dict = {"a": 5, "b": 8}`.

Mit der Syntax `my_dict["a"]` kann man den Wert an der Stelle `"a"` auslesen.

Mit der Syntax `my_dict["a"] = 12` kann man einzelne Werte des Dictionaries verändern.

Auf diese Weise können auch ganz neue Paare hinzugefügt werden. Zum Beispiel:
`my_dict["c"] = -2`.

Übung

Dictionary manipulieren

Gegeben sei das folgende Dictionary:

```
grades = {"Mathe": 8, "Bio": 11, "Sport": 13}
```

Bestimme die Durchschnittsnote dieser drei Fächer. Verbessere danach Deine Mathenote um einen Punkt und füge noch eine weitere Note für Englisch hinzu (Abfrage über Konsole). Gib danach erneut den Durchschnitt an.

Lösung

Dictionary manipulieren

```
grades = {"Mathe": 8, "Bio": 11, "Sport": 13}

grades_sum = grades["Mathe"] + grades["Bio"] + grades["Sport"]
average = grades_sum/len(grades)
print(f"Der Durchschnitt ist {average} Punkte")

grades["Mathe"] += 1

eng_grade = input("Welche Note hast Du in Englisch? ")
eng_grade = int(eng_grade)
grades["Englisch"] = eng_grade

grades_sum += grades["Englisch"]
average = grades_sum/len(grades)
print(f"Der Durchschnitt ist {average} Punkte")
```

Einen Eintrag aus einem Dictionary entfernen

Wie bei Listen, kann man mittels `del`-Statement einen Eintrag aus einem Dictionary entfernen:

```
del eu_countries["united_kingdom"]
```

Was wird hier passieren?

```
old_capitals = {"Deutschland": "Bonn", "Norwegen": "Oslo"}
new_capitals = old_capitals

new_capitals["Deutschland"] = "Berlin"

print(old_capitals)
print(new_capitals)
```

Erklärung

Da Dictionaries mutable sind, findet bei ihnen der Aufruf mittels *Call by Reference* statt. Das heißt, dass in der Variable `old_capitals` bzw. `new_capitals` nicht die Länder gespeichert sind, sondern nur die Speicheradresse, wo die Länder zu finden sind. Ändert man die zugrundeliegenden Daten an einer Stelle, so ändern sie sich daher auch an der anderen Stelle.

Eine Kopie von einem Dictionary erstellen

Mit der Funktion `dict()` kann man eine Kopie von einem Dictionary erstellen.

Beispiel: `dict(my_dict)` erstellt eine Kopie von `my_dict`.

Schleife über Dictionary I

Ähnlich wie bei Listen kann man Schleifen auch über ein Dictionary laufen lassen.

Beispiel

```
capitals = {"Litauen": "Vilnius", "Lettland": "Riga", "Estland": "Tallin"}

for item in capitals:
  print(item)

# Litauen
# Lettland
# Estland
```

Schleife über Dictionary II

Möchte man in der Schleife nicht nur die Schlüssel, sondern auch die Werte des Dictionaries zur Verfügung haben, so muss man die Methode `.items()` auf das Dictionary anwenden.

Beispiel

```
capitals = {"Litauen": "Vilnius", "Lettland": "Riga", "Estland": "Tallin"}

for key, value in capitals.items():
  print(f"Hauptstadt von {key}: {value}")

# Hauptstadt von Litauen: Vilnius
# Hauptstadt von Lettland: Riga
# Hauptstadt von Estland: Tallin
```

Übungen

Zwei Dictionaries kombinieren

Gegeben seien zwei Dictionaries, z.B.

```
eu = {"Deutschland": "Berlin", "Frankreich": "Paris" }
```

und

```
non_eu = {"Russland": "Moskau", "China": "Peking" }
```

Füge die Einträge des zweiten Dictionaries zum ersten Dictionary hinzu.

Ein Dictionary „filtern“

Sei ein beliebiges Dictionary mit Noten gegeben. Entferne alle Einträge, deren Note schlechter als 5 Punkte ist.

Lösung

Zwei Dictionaries kombinieren

```
eu = {"Deutschland": "Berlin", "Frankreich": "Paris" }
non_eu = {"Russland": "Moskau", "China": "Peking" }

for key,value in non_eu.items():
  eu[key] = value
print(eu)
```

Ein Dictionary „filtern“

```
grades = {"Deutsch": 11, "Mathe": 3, "Sport": 14, "Geschichte": 1}
#  Man darf die Länge eines Dictionaries in einer Schleife nicht verändern, deshalb machen wir eine
result = dict(grades)
for key, value in grades.items():
  if value < 5:
    del result[key]
print(result)
```

Ein Dictionary zerlegen

Mit der Methode `.keys()` erhält man eine Liste aller Schlüssel eines Dictionaries.

Mit der Methode `.values()` erhält man eine Liste aller Werte eines Dictionaries.

In beiden Fällen, muss das Ergebnis mittels der Funktion `list()` in eine Liste umgewandelt werden.

Beispiel

```
my_dictionary = {"China": "Peking", "Japan": "Tokio", "Korea": "Seoul"}

countries = my_dictionary.keys()
countries = list(countries)

capitals = my_dictionary.values()
capitals = list(capitals)

print(countries)   # ["China", "Japan", "Korea"]
print(capitals)    # ["Peking", "Tokio", "Seoul"]
```

Comprehensions

Typische Manipulationen

Sehr häufig möchte man eine Datenstruktur (d.h. eine Liste oder ein Dictionary) basierend auf den Werten manipulieren. Dabei werden vor allem zwei Aspekte immer wieder gebraucht: Transformationen und Filter.

Transformation

Ersetzt man jedes Element einer Liste durch ein aus dem ursprünglich berechnetem Element, so spricht man von einer *Transformation* (engl. map/mapping).

Beispiel

Gegeben ist die Liste `my_list = [2, 5, 3, 12, 7]`. Die Liste soll so manipuliert werden, dass alle Einträge durch ihren doppelten Wert ersetzt werden.

Traditionelle Lösung

```
my_list = [2, 5, 3, 12, 7]
result = []
for k in my_list:
  result.append(2 * k)
print(result)
```

The Pythonian Way

```
my_list = [2, 5, 3, 12, 7]
result = [2 * k for k in my_list]
print(result)
```

Filter

Streicht man Elemente entsprechend ihres Wertes aus einer Liste, so spricht man von einem *Filter*.

Beispiel

Gegeben ist die Liste `my_list = [2, 5, 3, 12, 7]`. Aus der Liste sollen alle ungeraden Einträge gestrichen werden.

Traditionelle Lösung

```
my_list = [2, 5, 3, 12, 7]
result = []
for k in my_list:
  if k % 2 == 0:
    result.append(k)
print(result)
```

The Pythonian Way

```
my_list = [2, 5, 3, 12, 7]
result = [k for k in my_list if k % 2 == 0]
print(result)
```

Kombination aus Transformation und Filter

Selbstverständlich können Transformationen und Filter auch kombiniert werden.

Beispiel

Lösche alle ungeraden Zahlen und verdopple dann alle Zahlen:

```
my_list = [2, 5, 3, 12, 7]
result = [2 * k for k in my_list if k % 2 == 0]
print(result)
```

Struktur List Comprehension

```
result = [ Wunscheintrag(k)  for k in my_list if Bedingung(k) ]
```

Dictionary Comprehension

Man kann das gleiche Verfahren auch auf Dictionaries anwenden. Dabei können jeweils key und value für die Transformationen und Filter verwendet werden.

Beispiel

```
my_dict = {"a": 2, "b": 3}
result = {key: value for (key, value) in my_dict.items()}
print(result)
```

Struktur Dictionary Comprehension

```
result = { Wunsch-schlüssel(k,v) : Wunsch-Wert(k,v)
                for (k,v) in my_dict.items() if Bedingung(k,v) }
```

Übung

List Comprehension

Gegeben sei eine beliebige Liste von ganzen Zahlen. Streiche alle Zahlen, die ungerade oder negativ sind. Ersetze die übrigen Zahlen durch ihre Hälfte.

Lösung

```
my_list = [2, -3, 40, 15, 7, 8]
result = [k//2 for k in my_list if k % 2 == 0 and k >= 0]
```

Dictionary Comprehension

Gegeben sei das Dictionary `{"Mathe": 9, "Sport": 13, "Physik": 4, "Bio": 12}`. Lösche nun daraus alle Noten unter 5 Punkte sowie die Sportnote. Zusätzlich soll das Dictionary danach wie folgt aussehen: `{"In Mathe": "9 Punkte", "In Bio": "12 Punkte"}`.

Lösung

```
grades = {"Mathe": 9, "Sport": 13, "Physik": 4, "Bio": 12}
result = {
    "In " + key: f"{value} Punkte"
    for (key, value) in grades.items()
    if value >= 5 and key != "Sport"
}
```

Funktionen

Wie man Code wiederverwerten kann

Problemstellung

Es sei eine Liste mit Temperaturen gegeben:

```
temperatures = [22, 18, 20, 15, 12, 7, 5, -2, 4]
```

Es sollen zunächst Durchschnitte von jeweils der folgenden Grundgesamtheit genommen werden:

- Alle Werte
- Die ersten drei Werte
- Jeder zweite Wert

Statt durch eine Zahl, soll das Ergebnis jedoch mit den Worten

- `"Mild"` für Durchschnitte $\geq$ 15 Grad
- `"Zu kalt"` für Durschnitte $<$ 15 Grad

abgespeichert werden. Wie macht man das *elegant*?

Lösung (Hauptsache es funktioniert)

```
average = sum(temperatures) / len(temperatures)
if average >= 5:
  average = "Mild"
else:
  average = "Zu kalt"

average_2 = sum(temperatures[:3]) / len(temperatures[:3])
if average_2 >= 5:
  average_2 = "Mild"
else:
  average_3 = "Zu kalt"

average_3 = sum(temperatures[::2]) / len(temperatures[::2])
if average_3 < 5:
  average_3 = "Zu kalt"
else:
  average_3 = "Mild"
#
```

Nachteile dieser Lösung

- Viel Schreibarbeit, viel Wiederholung
- Der Code ist schwierig zu lesen. Man sieht vor lauter Wiederholungen nicht, was passiert.
- Jedes Mal, wenn man diese „Berechnungslogik“ verwendet, könnte man einen (Tipp-)Fehler machen.
- Wenn man das Anforderungsprofil minimal ändert, muss diese „Logik“ bei *jedem* Auftreten im Code geändert werden (z.B. statt `"Mild"` soll das Ergebnis `"Warm"` heißen). In echten Projekten, kann das schnell ein paar Hundert Male sein.

Bessere Lösung

```
def compute_average(temp_list):
  result = sum(temp_list) / len(temp_list)
  if result >= 5:
    result = "Mild"
  else:
    result = "Zu kalt"
  return result

average = compute_average(temperatures)
average_2 = compute_average(temperatures[:3])
average_3 = compute_average(temperatures[::2])
```

Definition: Funktion

Eine Funktion ist ein Codeblock, der nur ausgeführt wird, wenn die Funktion *aufgerufen* wird. Man kann der Funktion Werte als *Parameter* übergeben. Sie kann auch einen Wert als Ergebnis *zurückgeben*.

Man kann sich eine Funktion wie eine Maschine vorstellen, wo man oben Dinge (=Parameter) hineinfüllt und unten ein Ergebnis (=Rückgabewert) herausbekommt. Unabhängig von dem Eingabe-Ausgabe-Prinzip, kann solch eine Maschine auch Nebeneffekte (z.B. Krach) produzieren.

Man unterscheidet zwischen *Definition* und *Ausführung* einer Funktion.

Struktur der Funktions-Definition

`def` *Funktionsname*(*Parameter_0*, *Parameter_1*, ..., *Parameter_n*):

␣␣ *Codezeile1*

␣␣ *Codezeile2*

⋮

␣␣ `return` *Ergebnis*

Struktur eines Funktionsaufrufs

result = *Funktionsname*(*Argument_0*, *Argument_1*, ..., *Argument_n*)

Good to know

- Eine Funktion muss schon *vor* dem ersten Aufruf definiert worden sein (das ist nicht in allen Sprachen so).
- Die Eingabwerte nennt man in der Funktionsdefintion *Parameter*, beim Aufruf der Funktion nennt man sie jedoch *Argumente*.
- Nicht jede Funktion braucht Eingangsdaten. Die Liste von Parametern einer Funktion kann daher leer sein.
- Beim Aufruf spielt die Reihenfolge der angegebenen Argumente eine entscheidene Rolle. Sie werden entsprechend der Reihenfolge den Parametern in der Definition zugeordnet.
- Eine Funktion muss nicht unbedingt etwas zurückgeben, d.h. das `return`-Statement ist optional.
- Das `return`-Statement muss nicht unbedingt am Schluss der Funktion stehen. Jedoch wird Code, der nach dem `return`-Statement kommt, nicht mehr ausgeführt.

Übungen

Funktion ohne Parameter

Schreibe eine Funktion, die Deinen Namen auf der Konsole ausgibt.

Funktion mit einem Parameter

Schreibe eine Funktion, die die übergebene Zahl verdoppelt und das Ergebnis zurückgibt.

Funktion mit zwei Parametern

Schreibe eine Funktion, die die beiden übergebenen Zahlen multipliziert und das Ergebnis zurückgibt.

Funktion ohne Rückgabewert

Was gibt eine Funktion zurück, die kein `return`-Statement enthält?

Lösungen

Funktion ohne Parameter

```
def my_name():
  print("Aaron Kunert")
```

Funktion mit einem Parameter

```
def double(number):
  return number * 2
```

Funktion mit zwei Parametern

```
def multiply(number1,number2):
  return number1 * number2
```

Übung

Aggregatzustand von Wasser

Schreibe eine Funktion, die die Temperatur als Paramter erwartet und abhängig von der Temperatur den Aggregatzustand von Wasser (`"fest"`,`"flüssig"`,`"gasförmig"`) als String zurückgibt.

Zusatz: Schaffst Du es ohne die Schlüsselwörter `elif` und `else`?

Lösung

Aggregatzustand von Wasser

```
def get_state(temp):
  if temp < 0:
    return "fest"
  if temp > 100:
    return "gasförmig"
  return "flüssig"
```

Komplexere Übung

Gewichtete Durschnittsnote

Schreibe eine Funktion, die eine Liste der folgenden Struktur erwartet:

```
grades = [
  {
    "subject": "Deutsch",
    "grade": 14,
    "is_major": True
  },
  # ...
  {
    "subject": "Sport",
    "grade": 11,
    "is_major": False
  }
]
```

Berechne die Durchschnittsnote, wobei Hauptfächer doppelt gewichtet werden sollen.

Lösung

Gewichtete Durchschnittsnote

```
def weighted_average(grades):
  weighted_sum = 0
  weighted_length = 0
  for grade in grades:
    if grade["is_major"]:
      weighted_sum += 2 * grade["grade"]
      weighted_length += 2
    else:
      weighted_sum += grade["grade"]
      weighted_length += 1
  result = weighted_sum/weighted_length
  return result
```

Übung

Zinsrechner

Angenommen, Du hast 1000€ so angelegt, dass es darauf jeden Monat 0,5% Zinsen gibt (Haha – als ob!). Schreibe eine Funktion, die einen Geldbetrag erwartet und zurückgibt, nach wie vielen Monaten, dieser Geldbetrag erreicht wurde.

Lösung

Zinsrechner

```
def months_until_rich(target):
    month = 0
    balance = 1000
    while balance < target:
        balance *= 1.005
        month += 1
    return month
```

Zinsrechner mit Sparrate

Angenommen, Du hast wieder 1000€ so angelegt, dass es darauf jeden Monat 0,5% Zinsen gibt (Haha – als ob!). Zusätzlich gibt es nun jedoch noch eine monatliche Sparrate von 25€, die ebenfalls auf das Konto eingezahlt wird. Schreibe eine Funktion, die einen Geldbetrag erwartet und zurückgibt, nach wie vielen Monaten, dieser Geldbetrag erreicht wurde.

Lösung

Zinsrechner mit Sparrate

```
def months_until_rich(target):
    month = 0
    balance = 1000
    while balance < target:
        balance += 25
        balance *= 1.005
        month += 1
    return month
```

Optionale Parameter

Manchmal wirst Du bei Funktionen bemerken, dass einige der Parameter fast immer den gleichen Wert haben. In diesem Fall, möchtest Du diese Parameter nicht bei jedem Aufruf immer hinschreiben, sondern nur dort, wo er vom Standardfall abweicht. Dies ist möglich, wenn man den Standardwert (*default value*) bei der Definition mit angibt.

Wichtig: Bei der Definition müssen die optionalen Parameter immer hinter den Pflichtparametern stehen.

Beispiel

```
def double(number, factor=2):
  return number * factor
```

Diese Funktion ist sehr vielseitig: Im einfachen Fall verdoppelt sie die eingegebene Zahl. Optional lässt sich der Faktor aber beliebig verändern.

Typischer Einsatzbereich

Oftmals merkt man im Verlauf eines Projektes, dass eine gegebene Funktion nicht flexibel genug ist, dann kann man sie um einen optionalen Parameter erweitern, ohne den bisherigen Code verändern zu müssen.

Fiktives Beispiel

Stell Dir vor, Du baust einen Zinsrechner wie oben, den auch schon einige Deiner Freund*innen, die den gleichen Sparplan haben, ebenfalls verwenden. Eine Deiner Freundinnen hat jedoch eine bessere Bank gefunden, die ihr 0,6% Zinsen gibt. Also erweiterst Du die Funktion, so dass auch die Höhe der Zinsen anpassbar ist.

Jedoch möchtest Du den bisherigen Code nicht verändern. Daher definierst Du den Zinssatz als optionalen Parameter, so dass die Funktion „abwärtskompatibel“ zu ihrer bisherigen Verwendung ist.

Die Definition startet dann mit `def months_until_rich(target, interest=0.5):`

Übung

Flexibler Zinsrechner

Erweitere den Zinsrechner, so dass optional der Zinssatz angegeben werden kann.

Lösung

Flexibler Zinsrechner

```
def months_until_rich(target, interest=0.5):
    month = 0
    balance = 1000
    while balance < target:
        balance += 25
        balance *= (1 + interest/100)
        month += 1
    return month
```

Schwierige Übung

Flexibler Durchschnittsrechner

Erweitere die Funktion zur Berechnung von gewichteten Durchschnittsnoten so, dass optional der Gewichtungsfaktor angegeben werden kann.

Lösung

Flexibler Durchschnittsrechner

```
def weighted_average(grades, weight=2):
  weighted_sum = 0
  weighted_length = 0
  for grade in grades:
    if grade["is_major"]:
      weighted_sum += weight * grade["grade"]
      weighted_length += weight
    else:
      weighted_sum += grade["grade"]
      weighted_length += 1
  result = weighted_sum/weighted_length
  return result
```

Named Parameters

Hat eine Funktion viele Parameter, von denen etliche optional sind, so kann man einen Parameter statt über die Reihenfolge auch über den Namen übergeben.

Beispiel

```
def my_function(parameter1, parameter2=0, parameter3="x", parameter4=-17):
  # ...
```

Möchte man jetzt die Funktion mit einem eigenen Wert `parameter1` und `parameter4` aufrufen aber alles andere auf Standard lassen, so geht das wie folgt:

```
my_function(15, parameter4=-20)
```

Übung

Ganz flexibler Zinsrechner

Erweitere den Zinsrechner, so dass zusätzlich optional der Startbetrag und die monatliche Sparrate angepasst werden können.

Welchen der Parameter muss man verdoppeln, um am schnellsten 10.000€ zu erreichen?

Lösung

Ganz flexibler Zinsrechner

```
def months_until_rich(target, interest=0.5, initial_amount=1000, savings_rate=25):
    month = 0
    balance = initial_amount
    while balance < target:
        balance += savings_rate
        balance *= (1 + interest/100)
        month += 1
    return month
```

Scope

Wo Variablen gültig sind

Problemstellung

Sei `my_variable` eine Variable mit Wert 1. Schreibe eine Funktion, die bei Aufruf die Variable `my_variable` um 1 erhöht. Schreibe eine Funktion, die bei Aufruf die Variable `my_variable` um 1 erhöht.

Wie macht man das?

Das Problem

```
my_variable = 1

def increment():
  my_variable = my_variable + 1

increment()
print(my_variable)
```

Die offensichtliche Lösung funktioniert nicht. Warum nicht?

Experiment I

```
global_variable = 1

def my_function():
  local_variable = 5

my_function()
print(global_variable)
print(local_variable)
```

Beobachtung

Eine Variable, die innerhalb einer Funktion definiert wurde, ist auch nur innerhalb der Funktion sichtbar.

Experiment II

```
global_variable = 1

def my_function():
  print(global_variable)


my_function()
print(global_variable)
```

Beobachtung

Eine *globale* Variable ist auch innerhalb einer Funktion definiert.

Experiment III

```
global_variable = 1

def my_function():
  global_variable = 5
  print(global_variable)


my_function()
print(global_variable)
```

Beobachtung

Eine Variable innerhalb einer Funktion kann den gleichen Namen wie eine Variable außerhalb haben, allerdings ist die innere Variable nur innerhalb der Funktion sichtbar.

Experiment IV

```
global_variable = 1

def my_function():
  print(global_variable)
  global_variable = 5


my_function()
print(global_variable)
```

Beobachtung/Erklärung

Python entscheidet anhand des Kontexts ob `global_variable` eine globale Variable ist, oder eine lokale Variable, die zufällig den gleichen Namen wie eine globale Variable trägt.

Falls Python denkt, dass es sich um eine globale Variable handelt, so kann diese nur gelesen, nicht aber geschrieben (d.h. neu definiert) werden.

Das Eingangsbeispiel

```
my_variable = 1

def increment():
  my_variable = my_variable + 1

increment()
print(my_variable)
```

Erklärung

Da `my_variable` rechts vom Gleichheitszeichen steht, denkt Python, dass es sich um die globale Variable `my_variable` handelt. Da `my_variable` aber auch links vom Gleichheitszeichen steht, wird auch schreibend auf die Variable zugegriffen. Das ist nicht erlaubt.

Mögliche Lösung

```
my_variable = 1

def increment(var):
  return var + 1

my_variable = increment(my_variable)
print(my_variable)
```

Definition

Der Gültigkeitsbereich einer Variable wird *Scope* genannt.

Scope in Python

In Python unterscheidet man zwischen *global Scope* und *local Scope*. Im local Scope hat man nur Lesezugriff auf den global Scope.

Achtung Ausnahme

```
my_list = [1, 2, 3]

def append(item):
  my_list.append(item)

append(4)
print(my_list)
```

Erklärung

Da die Variable `my_list` nicht überschrieben wird, sondern nur das referenzierte Objekt verändert wird, erkennt Python dies nicht als Schreibzugriff und erlaubt dieses Vorgehen.

Warum ist der Zugriff auf den Global Scope eingeschränkt?

- Funktionen sollen möglichst wenige Nebeneffekte haben. Wenn eine Funktion den global Scope verändern kann, ist dies ein großer Nebeneffekt.
- Wenn man eine Funktion schreibt, muss man sich keine Gedanken machen, ob ein Variablenname schon vergeben ist.
- Wenn man sich innerhalb einer Funktion den Kontakt zum global Scope reduziert, so ist die Funktion besser zu verstehen, zu warten und zu testen.
- ...

Persistenz

Lesen und Schreiben von Dateien

Grundprinzip

Um mit Dateien zu arbeiten, geht man immer in 3 Schritten vor:

1. Datei öffnen
2. Datei bearbeiten (d.h. z.B. lesen, überschreiben, etwas anhängen)
3. Datei schließen

Das Schließen von Dateien ist relativ wichtig, kann aber schnell mal vergessen werden. Daher bietet Python eine spezielle Syntax mithilfe des Keywords `with` an.

Gesamten Text einer Datei einlesen

```
with open("some_file.txt") as my_file:
  my_text = my_file.read()
  print(my_text)
```

Erklärung

- Die Funktion `open` öffnet die angegebene Datei (Python geht per se davon aus, dass die Datei im gleichen Ordner wie das ausgeführte Skript liegt).
- Ein *Dateiobjekt* wird in der Variable `my_file` gespeichert (der Variablenname ist beliebig)
- Die Methode `.read()` liest den Text-Inhalt der Datei, so dass er in einer Variable gespeichert werden kann
- Sobald der eingerückte Block verlassen wird, wird die Datei automatisch geschlossen

Den Text einer Datei zeilenweise einlesen

```
with open("some_file.txt") as my_file:
  my_lines = my_file.readlines()
  for line in my_lines:
    print(f"The line reads: {line}")
```

Erklärung

- Die Methode `.readlines()` gibt eine *Liste* der Zeilen des Inhalts der Datei `"some_file.txt"` zurück.
- Durch diese Liste kann man mittels einer `for`-Schleife durchiterieren.

Übungen

Text einlesen

Lade Dir aus dem FirstClass die Datei `"tf.txt"` herunter und kopiere sie in Dein Python-Projekt. Gib den Text auf der Konsole aus.

Zeilen zählen

Lade Dir aus dem FirstClass die Datei `"vs.txt"` herunter und kopiere sie in Dein Python-Projekt. Gib auf der Konsole aus, aus wievielen Zeilen der Text besteht.

Zählfunktion

Schreibe eine Funktion, die zu dem übergebenen Dateinamen die Anzahl an Zeilen zurückgibt.

Lösungen

Text einlesen

```
with open("tf.txt") as my_file:
  my_text = my_file.read()
  print(my_text)
```

Zeilen zählen

```
with open("vs.txt") as my_file:
    my_lines = my_file.readlines()
    length = len(my_lines)
    print(length)
```

Zählfunktion

```
def count_lines(filename):
    with open(filename) as my_file:
        my_lines = my_file.readlines()
        return len(my_lines)
```

Text in eine Datei schreiben

```
with open("some_file.txt", "w") as my_file:
  my_file.write("Hello everybody")
```

Erklärung

- Ruft man `open` mit dem zweiten Parameter `"w"` auf, so wird die Datei im Schreibmodus geöffnet.
- Existierte die Datei zuvor noch nicht, so wird sie erzeugt.
- Mit der Methode `.write("Inhalt")` lässt sich Text in eine Datei schreiben.
- Achtung: Öffnet man eine Datei im Schreibmodus, so wird der bisherige Inhalt überschrieben.

Text an eine Datei anhängen

```
with open("some_file.txt", "a") as my_file:
  my_file.write("Some text to append")
```

Erklärung

- Ruft man `open` mit dem zweiten Parameter `"a"` auf, so wird die Datei im *Append*-Modus geöffnet.
- Existierte die Datei zuvor noch nicht, so wird sie erzeugt.
- Mit der Methode `.write("Inhalt")` lässt sich Text an die Datei anhängen.
- Der bis dahin in der Datei vorhandene Inhalt wird nicht verändert oder gelöscht.
- Der einzige Unterschied zum letzten Punkt ist der Modus (`"a"` statt `"w"`).

Achtung Umlaute

Hat man eine etwas ältere Version von Python und möchte man Dateien, die Umlauten und andere Sonderzeichen enthalten, bearbeiten, so muss man beim Öffnen der Datei noch den Parameter `encoding="utf-8"` übergeben.

Beispiel

```
with open("some_file.txt", "a", encoding="utf-8") as my_file:
  my_file.write("Hier ein Text mit Umlauten: äöüß")
```

JSON

Ein universelles Datenformat

Definition: JSON

JSON (Java Script Object Notation) ist ein Daten-Format, um Verschachtelungen von Listen und Dictionaries darzustellen, zu speichern und auszutauschen. Die Syntax entspricht (fast) der üblichen Python-Syntax und wird von den meisten Programmiersprachen „verstanden“.

Beispiel: Eine Liste von Ländern

```
[
  {
    "name": "Germany",
    "capital": "Berlin",
    "population": 83190556,
    "cities": ["Berlin", "Hamburg","München","Köln"]
  },
  {
    "name": "France",
    "capital": "Paris",
    "population": 67422000
    "cities": ["Paris","Marseilles", "Lyon", "Toulouse"]
  },
  ...
]
```

Eigenschaften

- Dictionaries und Listen dürfen beliebig verschachtelt werden.
- Die äußerste Ebene kann ein Dictionary oder eine Liste sein.
- Es müssen doppelte Anführungsstriche verwendet werden.
- Neben Dictionaries und Listen können folgende Datentypen verwendet werden:
  - Integer
  - String
  - Float
  - Boolean (`true` bzw. `false`)
  - `null` (entspricht `None`)

Python's JSON-Modul

Um in Python Daten im JSON-Format einzulesen und zu speichern, benötigt man das mitgelieferte JSON-*Modul*. Dazu einfach die folgende Zeile am Beginn des Python-Skripts anfügen:

```
import json

...
```

Daten als JSON-Datei abspeichern

```
import json

data = {"a": 1, "b": 2}  # some dummy data

with open("some_file_name.json","w") as my_file:
  json.dump(my_data, my_file)
```

Erklärung

- Zunächst wird die Datei `some_file_name.json` im Schreibmodus geöffnet.
- Die Funktion `json.dump` erwartet die Daten und eine Datei. Die Daten werden im JSON-Format in der Datei abgespeichert.
- Achtung: Der bisherige Inhalt von `some_file_name.json` wird überschrieben.

Daten aus einer JSON-Datei importieren

```
import json

with open("some_file_name.json") as my_file:
  data = json.load(my_file)

print(data)
```

Erklärung

- Zunächst wird die Datei `some_file_name.json` im Lesemodus geöffnet.
- Die Funktion `json.load` erwartet eine JSON-Datei und gibt die eingelesenen Daten als Liste bzw. Dictionary zurück.

Übung

Userdaten

Lade Dir aus dem FirstClass die Datei `"player.json"` herunter und kopiere sie in Dein Python-Projekt. Öffne die Datei und gib den Namen, der Spielerin, sowie das Level und den Punktestand auf der Konsole aus.

Lösung

```
import json

with open("player.json") as my_file:
    data = json.load(my_file)

print(f"Name: { data["name"] } ")
print(f"Punktestand: { data["score"] } ")
print(f"Level: { data["level"] } ")
```

Übung

Levelfortschritt speichern

Verwende wieder die Datei `"player.json"`. Implementiere die Funktion `levelup()`, die das Userprofil einliest, das Level um 1 und den Punktestand um 100 erhöht und die neuen Daten wieder in der Datei `"player.json"` abspeichert.

Lösung

```
import json

def levelup():
    with open("player.json") as my_file:
        data = json.load(my_file)
    data["score"] += 100
    data["level"] += 1
    with open("player.json","w") as my_file:
        json.dump(data, my_file)
```

Web APIs

Wie Python im Netz surfen kann

Definition: API

Eine *API* (Application Programming Interface) ist im Allgemeinen eine klar definierte Schnittstelle zwischen Programmen.

Im Kontext des Internets geht es dabei meist um Webanwendungen, die statt einer für menschen lesbaren Seite, ein maschinenlesbares JSON ausliefern. Auf diese Weise ist ein Datenaustausch zwischen Deinem Programm und einer Webanwendung in beide Richtungen möglich.

Eine Web-Api besteht meist aus einer (oder mehreren) Webadressen und einer Anleitung, wie sie zu benutzen ist.

Beispiele für APIs

- **Instagram Api**: Erhalte Infos, wer Dir folgt und welche Reaktionen Du auf Deine Bilder erhältst.
- **GoogleMaps Api**: Sende Koordinaten an Google-Maps und erhalte die Adresse.
- **REST Countries**: Erhalte Daten von Ländern dieser Erde.
- **Stock Market Apis**: Erhalte live Daten zu Börsenkursen.

Anwendungsbeispiele

- **Instagram**: Schreibe Dir einen Python-Scheduler, wann welche Bilder von Dir veröffentlicht werden sollen.
- **REST Countries**: Nutze die Daten, um ein Quiz zu schreiben.
- **Stock Market Apis**: Trading Apps, SMS-Warnung bei einbrechenden Preisen.

Wie macht man einen API-Call?

Im Folgenden gehen wir durch die Schritte, die man ausführen muss, um einen lesenden API-Call auszuführen. Wir verwenden dafür die **REST Countries Api**. Die Dokumentation (*Docs*) der Api findet sich unter `https://restcountries.com/`.

Schritt 1: Finde die richtige URL (d.h. Webadresse)

Zunächst benötigt man eine URL (den sogennante *Endpoint*). Diese kann völlig statisch sein, oftmals aber sind Teile der URL dynamisch bzw. hängen von Deiner genauen Anfrage ab.

Beispiel:

`https://restcountries.com/v3.1/all`

oder

`https://restcountries.com/v3.1/name/france`

Schritt 2: Schicke einen Request an den Endpoint

Verwende folgenden Code, um einen Request zu schicken:

```
import requests
...

response = requests.get("https://restcountries.com/v3.1/name/france")
data = response.json()
# print(data)
```

Schritt 3: Erweitere den Request ggf. um *Query-Parameter*

```
import requests
...

payload = {"fields": ["capital","population","continents"]}
response = requests.get("https://restcountries.com/v3.1/name/france", params=payload)
data = response.json()
# print(data)
```

Schritt 4: Transformiere, filtere und speichere die Daten je nach Bedarf

```
import requests
import json
...

payload = {"fields": ["capital","population","continents"]}
response = requests.get("https://restcountries.com/v3.1/name/france", params=payload)
data = response.json()

with open("france.json","w") as my_file:
  json.dump(data, my_file)
```

Übung

Daten eines Landes extrahieren

Stelle einen Request zu Italien an die Countries-Api. Bereite die zurückgegebenen Daten so auf, dass folgendes auf der Konsole erscheint:

```
Hauptstadt: Rome
Bevölkerung: 59554023
Kontinent: Europe
```

Lösung

Daten eines Landes extrahieren

```python
import requests

response = requests.get("https://restcountries.com/v3.1/name/italy")
data = response.json()

data = data[0]
capital = data["capital"][0]
population = data["population"]
continent = data["continents"][0]

print(f"Hauptstadt: {capital}")
print(f"Bevölkerung: {population}")
print(f"Kontinent: {continent}")
```

Übung

Hauptstadt-Orakel

Schreibe eine Funktion, die den (englischsprachigen) Namen eines Landes erwartet und den Namen der Hauptstadt zurückgibt.

Lösung

Hauptstadt-Orakel

```python
import requests

def get_capital(country):
    response = requests.get("https://restcountries.com/v3.1/name/" + country)
    data = response.json()
    data = data[0]
    capital = data["capital"][0]
    return capital
```

Übung

Datenaufbereitung

Erstelle eine Liste aller Länder. Jedes Land soll dabei als Dictionary der folgenden Form abgespeichert werden:

```
{
  "name": "Italy",
  "capital": "Rome",
  "continent": "Europe",
  "population": 59554023
}
```

Speichere diese Liste als JSON in der Datei `countries.json`.

Lösung

Datenaufbereitung

```python
import requests
import json

response = requests.get("https://restcountries.com/v3.1/all")
data = response.json()

countries = []
for country in data:
    if "capital" in country.keys():
        countries.append({
            "name": country["name"]["common"],
            "capital": country["capital"][0],
            "population": country["population"],
            "continent": country["continents"][0]
        })

with open("countries.json", "w") as my_file:
    json.dump(countries,my_file, indent=4)
```

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.

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.

Lösung

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.

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.

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.

Lösung

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.