1. **Einlesen des Daten**

Das Programm wird in Python implementiert.

Z.4: Die Daten, die ich hier angewendet habe, sind die Text-Daten, die sich in dem Git Repository des Wettbewerbs befinden.

- Hier gibt man den absoluten file path ein, zB.“C:/Users/UserName/Desktop/raetsel0.txt”. Codecs wird hier aufgrund der Lesbarkeit der Buchstaben wie ü, ä, ö usw benutzt.
- *list1* gibt die abgelesene Text-Datei in String-Format in einer Liste zurück. Von *raesel0.txt* wird zB. `['_h __, _a_ __r ___e __b___!\n', 'arbeit eine für je oh was\n']` wiedergegeben.

2. **Deklaration wichtiger Variablen**

Z.6 - 14:

- *list1* hat zwei Elemente. *blank* gibt eine Liste von unausgefüllten Wörter zurück und *words* die vollständigen Wörter.

  z.B: `blank = ['_h', '__', '_a_', '__r', '___e', '__b___!']`

  `words = ['arbeit', 'eine', 'für', 'je', 'oh', 'was']`
- jedoch sollen Satzzeichen in *blank* nicht berücksichtigt werden, daher die Methode *re.sub()*

3. **Methoden**

Z.16:

- hier wird jedes Element in *blank* mit jedem Element in *words* vergleicht.
- wenn zwei Elemente gleich lang sind, wird spontan eine Liste *same_length* wiedergegeben, da *same_length = [i, j]* mit *i* in *blank* und *j* in *words*.
- die Methode *same_character* weist nach, ob *i* nur aus ‘_’ besteht, zB. in *['__', 'je'],* wenn es der Fall ist, wird dieses *same_length* geprinted.
- eine weitere Schleife wird durchgeführt, um in *same_length* zu prüfen, ob ein Buchstabe in der gleichen Position auftaucht. Wenn ja, wird dieses geprinted.
- letztendlich bekommt jedes i (Rätselwort) eine oder mehrere Spekulation.

Anmerkung

Die Methode possible_match kann leider keine eindeutige Lösung für die Aufgabe darstellen, sondern nur rechnerischen Vermutungen oder Ähnlichkeiten zurückgeben. Hier fehlt nämlich das Sprachverständnis des Computers. Trotzdem ist jede Vermutung in Reihenfolge des Rätsels angezeigt, welches die Spekulation des Spielers deutlich vereinfacht.