T1A3 Terminal App

German Vocab Randomiser & Quiz Generator

Anna Cox October 2023

Overview:
Concept & Aims

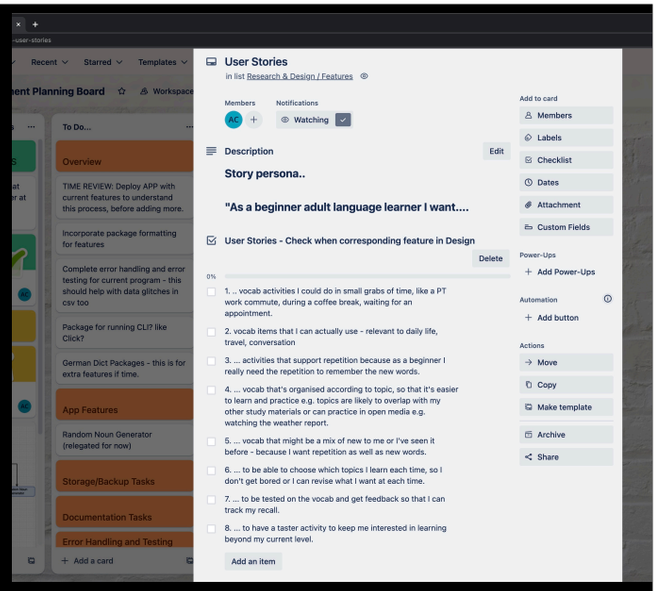
Structure and Features & how they achieve the aims

Software Development Plan and Implementation

Aims & Purpose

User Stories >> App features

- Begin with User Stories >> See screenshot here>>
  - Overall goal: a simple CLI app tool designed for the user who is a language learner to learn, recall and recycle vocabulary from a pool of lexical items targeted at their level.
  - Focus on learning and recycling rather than testing knowledge.
- Address needs of User with App feature concepts:
  - Choose a topic from a menu.
  - Get a randomised list of 10 items with a translation and example.
  - Generate on screen flashcards for each list (change direction?)
  - Generate a matching quiz to recall the items, possibly with spelling input.
  - Generate a gap-fill quiz to recall words and identify meaning from context.

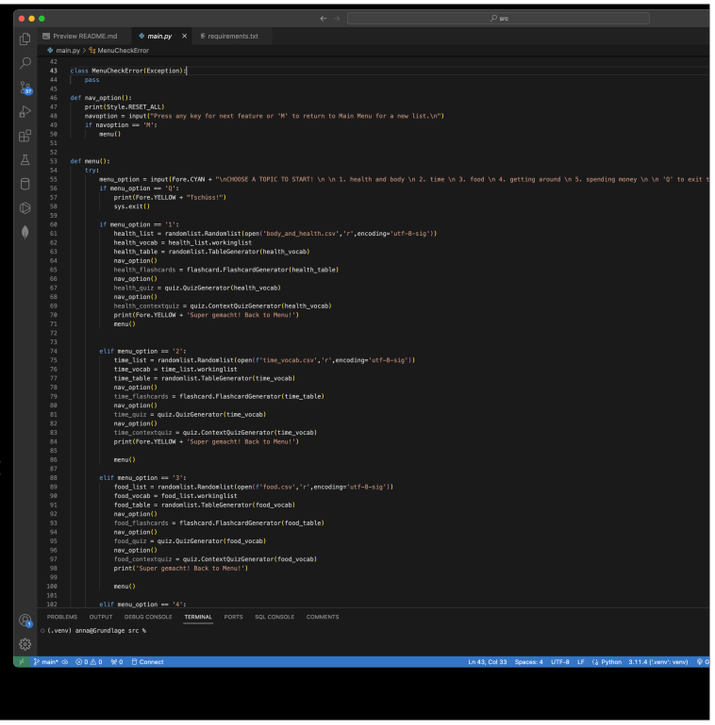


The Development Plan started with an idea for a language learning app for vocab, similar to that of flashcards, but with more features.
I considered User Stories for adult beginner language learners and what their preferences might be. I thought it might be important that they be able to scale their interaction with an app to very short or longer periods of time. I also think it's less important for adults to get marks like a test but to make progress with their learning through repetition at an appropriate level. The app focuses on accessible lists and learning features, with rewards for correct answers, freely given answers when prompted and unlimited guesses.

Features

Menu & App layout

- User chooses a topic from the vocab pool
- Can quit any time from the menu
- The menu functions as the structure for the other functions in the app to be called.
- Menu reflects app's sequential layout
- nav_option() allows user to return to main menu at end of each activity stage or exit the program.

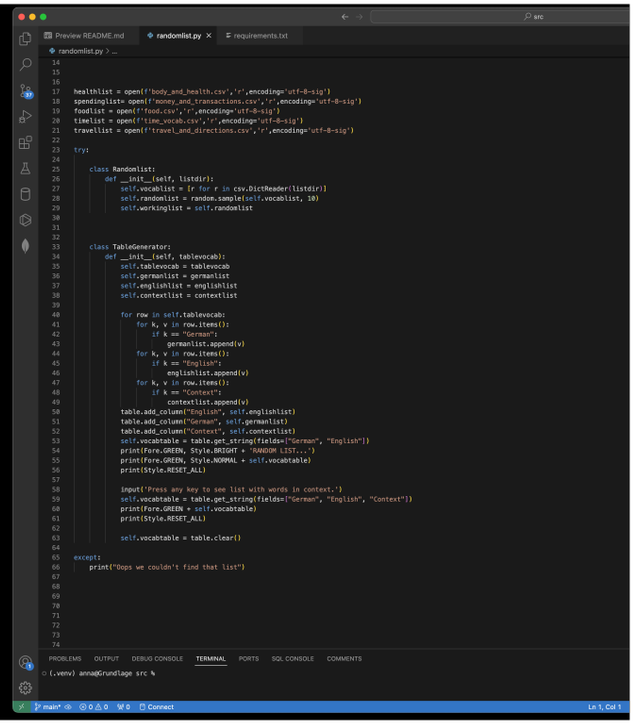


Each menu item leads to a sequence of functions that re-use in the same random list in a purposeful order, from the easiest or most passive learning exercises, to the more difficult productive and analytical tasks. The menu function itself is a simple group of conditionals linking the menu item number to the CSV file and then with the list information calling the activity functions to generate the list and run the flashcards and quiz. You can see the linear, sequential structure of the main function. I ended up deciding on this structure when it also became clear that I wanted to be able to go through the activities, but return to main at anytime. This allowed me to add a navigation function nav_option()notes to do this. I could also add an option then in the menu to Quit any time the user returns to the menu.

Features

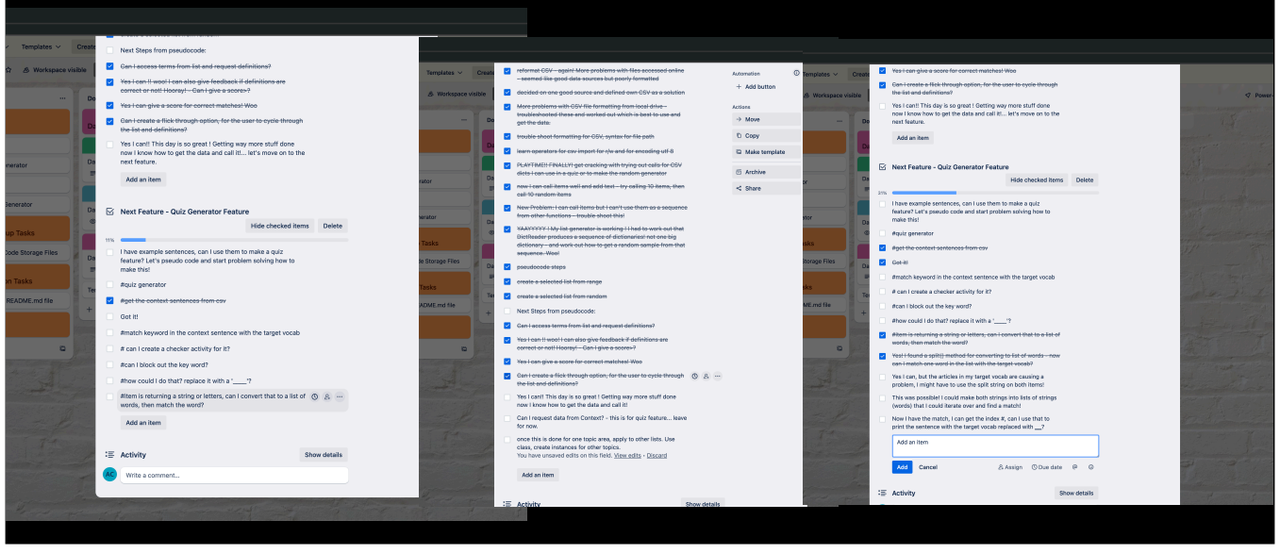
Random List Generator

- Get a randomised list of 10 items with a translation and example.
  - the following steps all practice this list.
- Uses class so that list can be generated in each instance for each topic list.
- The csv data is imported as a dictionary, so each item is read and iterated in dict form.
- It took me many tries at loop structures to make sure I was targeting the right part of each data structure and only adding that to the list I wanted to work with or print. I took a while to understand conceptually that the csv was delivering a list of dictionaries not one big dictionary and I would have to treat each item in the list as a dictionary in itself and iterating over the keys and values within it to extract the data I needed for the list.
- This had to be consistent and robust as I needed these values to remain the same for the rest of the features.
- Again, a more developed TDD skill would have helped me in that stage but I learned a lot from printing every valuable I was iterating
- I decided to use a table layout to make the list more robust and easier to read.



Working through it

As I figure out how to crack the list I get clues how to make the next features possible



You can see form the screen shots from last Wednesday that I finally figure out how to work with the CSV data effectively to get a consistent outcome. Once I have this I then figure out that I can call the list data to make flashcards. I then start to conceptualise how I could to the matching activity - possibly give a score or some kind of rewarding feedback and then finally imagine how I could print a sentence from the dictionary after matching a key word in it with a vocabulary item from the list and erasing that word on printing to make the gap-fill.

I took way too many screen shots this day for each new phase and idea. I was in effect testing, but not formally or with pytest. This was my favourite part of the process aside from running the program on another computer and it actually working! I could be completely engrossed with trying to solve a problem and then using the information I get from it to consider what else I could do with the program and trying things out. Hours and hours of time passed and I did completely wear myself out. If I never had to worry about time, I would just do this part of it!

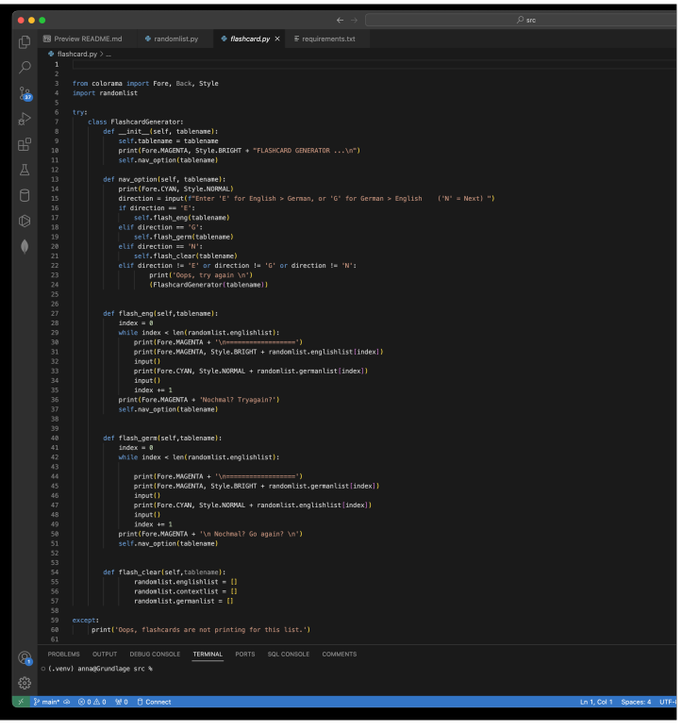
Features

Flashcard Generator

- Generate on screen flashcards for each list
  - Learn the vocab items in either language direction
  - Unlimited cycles available, move on at end of any cycle.

- Once I got the list I was able to make a flashcard generator, just calling each item from each language group at a time.
- Incorporating Pretty Table meant I had to change the format of the list structure because I needed to five PT clear row or column information that it could parse. The upside was it gave me a German and English list I could easily iterate over for the flashcard sequence.
- This meant that I could then also repeat the function and switch the order or the lists so that the user could choose to call a sequence German > English or English > German.
- Once I knew I needed two functions for that, it was clear I should start with a class instance to store the list information from random list and then create 2 objects to run it.

- This layout also was in harmony with the nav_option

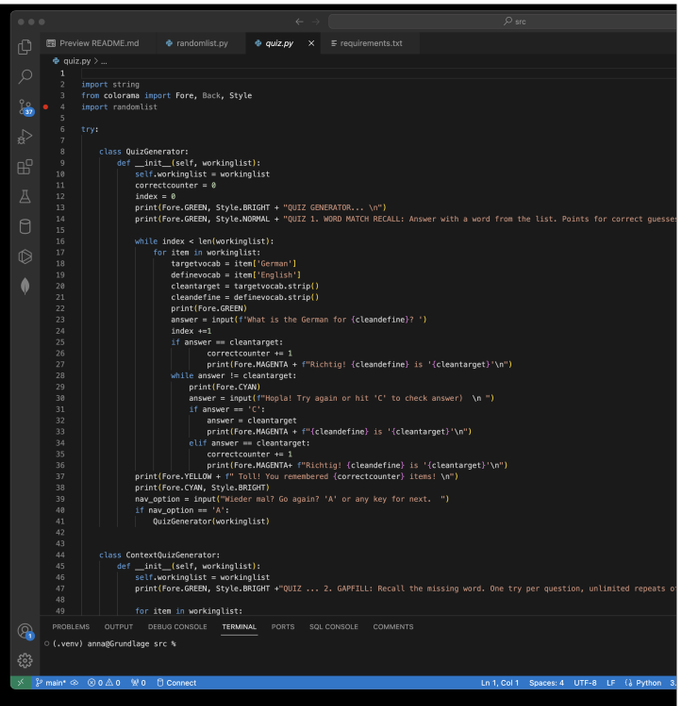


Features

Match Quiz Generator

- Generate a matching quiz to recall the items
    - Spelling also practiced here
    - No penalty for incorrect answers or spelling, unlimited guesses
    - Option to request answer
    - First-time guess score given!
    - Unlimited cycles available.

- The Match Quiz prints an English meaning (from the random list) with a question prompt and user input is checked with the German word in the list to produce a correct or incorrect answer.
- Dealing with language was so tricky here! If an item had been entered into the csv with a ‘ ‘ at the end or beginning of the word, a correct user input would yield and incorrect response from the function. I had to figure out how to clean up the language input before running it through the checks.
- User input also meant more possible conditional outcomes which all had to be accounted for. As the app’s purpose wasn’t to punish incorrect answers it was more functional to just allow the user to input as many times as they wanted. This meant adding a while loop around the conditionals negotiating the input, with a prompt to continue or leave the feature.
- Instead the user got feedback for the correct answer rather than all possible wrong answers or synonyms chosen.
- To help a user who is inputting but can’t get it, the option is there to check the answer with ‘C’.
- Correct answers also get the solution printed.



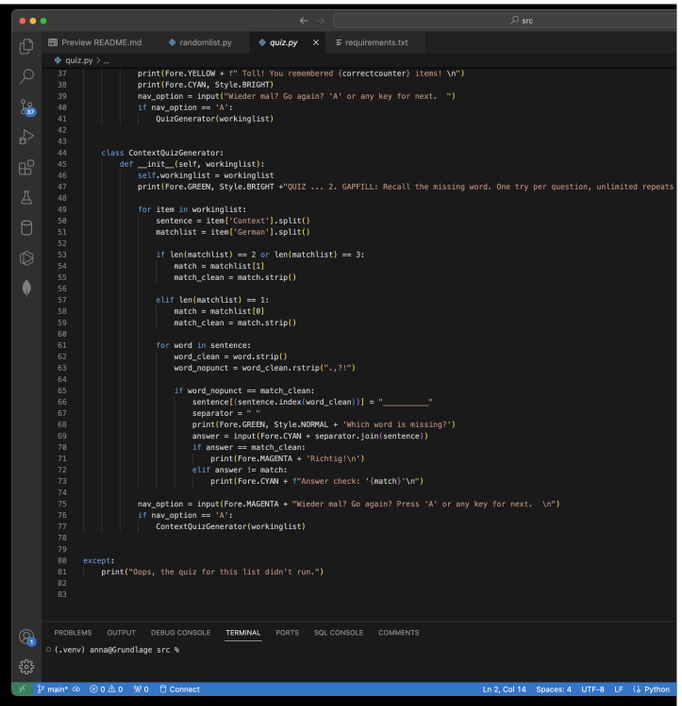
Also regarding user input multiple times. There were considerations outside of the code functionality. It’s possible that there are items in a list or in a topic pool with the same translation e.g. der Zug and die S-Bahn would both be called train in English. That would mean an incorrect response from the app could give a learner confusing feedback just because they didn’t choose that version of the meaning.

Features

Context Quiz Generator

- Generates a gap-fill quiz to recall words and identify meaning from context.
  - Spelling practiced.
  - One guess only, then answer given
  - Unlimited cycles through quiz available.
  - Exit or move on at end of a cycle.

- This feature takes the German Vocab items from the original csv random list of 10 and the corresponding Context sentence. It then matches the target vocab word with a word in the Context sentence and blanks it out to print to the user with a question prompt to fill the gap. The user's answer then has to be checked with the vocab item.
- This feature was the trickiest because I had 2 strings as dictionary values here rather than just 2 word items I was matching. I had to split the strings up into lists of individual words so that I could iterate over the words (Again it took me a while of printing each stage of the function to realise that at first I was just iterating over the letters in a string, then I finally split it.)
- Another problem is that even list items could come with spaces or punctuation attached to I had to figure out how clean them off in order to 1) get a match with each other and then 2) get a match with the user input.
- One I had that I had print the index location of the item as blank.
- In each case ensuring indentation was accurate so that I iterated through the vocab strings, the context strings and then repeated that process for all 10 dictionaries that made up the random list.
- Limitation - no lemmatisation (see slide notes)
- This was by far the most challenging feature for me to configure due to the language elements.



A further problem was the the vocab item was made of 2 words if if was a noun , the first being the definite article which is also likely to occur in the example sentence. I had to figure out how to exclude this word (der, die, das) from the matching process or I would end up with just 'der' blanked out of the final question.
One limitation here was that I wasn't able to account for variation in word form when vocab items appear in sentences, mostly this is conjugated verbs. In German there are also many separable verbs that are prefixed with prepositions, like in English, making it difficult to iterate to find the prefix and not a preposition.
As you might gather reading this, my mind did wander considerably at the possibilities for this. There are lemmetization packages mapping each lexical item in all its forms, or at least in present tense conjugations. However the complexity of this was just way beyond the scope of this project. I hope I become proficient enough at coding that it doesn't seem beyond the scope of my ability to make something like that. It was at this point that I did very much wish I'd chosen to make a game, rather than a vocabulary app. Knowing enough about the grammar to see what could be done and also be unsatisfied with something that wasn't accurate was very frustrating with my limited coding abilities at this stage. So I really tried to make something that did the job, benefitted the user and addressed their learning need, but stayed within the realm of what was reasonably accurate. The downside is that some of the context sentence won't run in the Context Quiz. Only the items that produce a question, print a question. I decided it was better to have 6 accurate questions and functioning code that to include errors.

Evaluation

Aims achieved…

- User stories have been addressed:
- Recyclable level appropriate vocab in topic pools, in a time-flexible manner.
- Random list generates and converts into 3-4 learning further learning activities.
- After wrestling with different CSV data options, I went with Goethe Institute's A1 vocab list (source in README.md) which gave appropriate input
-

Limitations…

- No lemmatisation included.
- A random noun generator from a pre-existing package or from an API wasn't included due to time restraints

Possibilities…

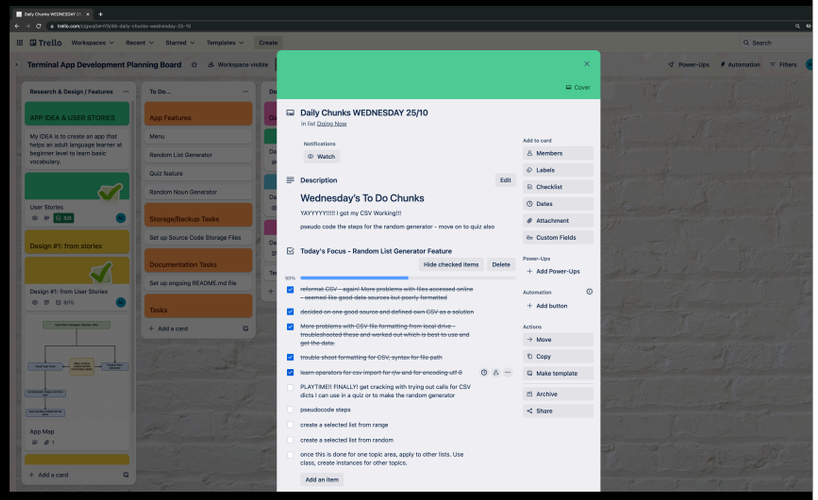
- More functionality could be expanded for linguistic accuracy as described at the Context Quiz.
- The app could also be added to or adapted easily by the user for suit their needs or grow with them. See notes below.

Possibilities!

The application has been loaded with csv files generated from Goethe Institute's A1 Vocabulary pool. However, the source code will draw the same activities from more csv files with the same column formatting - or the existing ones can be altered! This means that, for example, if you already have items organised into .csv on places like Anki (https://apps.ankiweb.net) or your purchased textbook materials, you could incorporate them all. Filepath updates in randomlist.py and main.py and adding menu items in main.py would be needed for new csv items. It also means the app could be adapted to other languages where 'German' or 'germ' in the source code, as well as the 'German' column in the csv files, was renamed. Or indeed, the English replaced with another first language.

Trello Board

- Daily to-do or doing list with ongoing checks.
- Each day a reflection on previous day to re-adjust.
- You can see in background move to Kansan style with To-Do steps, and behind there is also Done. This was much simpler for me than a running list of all the stages - less overwhelming, more focussed on little problem-solving chunks.
- That said, I've learned that tracking the big picture is weakness in my planning I need to improve



Development Arc

Ideally the development would have gone like this:

- User Stories
- Feature Concept
- Source Input Data (Vocabulary)
- Test & Develop
- Refactor and Compose OOP
- Error Handling
- Final testing
- Manual test run

In reality…

- All areas were followed except for the testing processes.
- I was unable to conceptualise how to receive, filter and process the input data in concise enough ways due to my inexperience. So I wasn't able to formulate tests for functions that weren't clear in my mind. I just ended up having to go for it because I could visualise the process I wanted the code to do and pseudocode it but I didn't know the best way to make it.

- I ended up making code I thought ran well and didn't break but I made it the hard way through trying different approaches I could figure from the code options and then choosing the one I thought was the driest.

- On reflection, a lot of my trial and error and running my program again and again performed a similar function to TDD but was less formal and less well planned.
- The next time app I make, I will have a better idea of how to balance testing, which is a kind of formal drafting I guess, with my source code development.

Development Plan

How effective was it? / What I learned

- Started with all areas listed
- Adapted layout to Kanban style with an active doing list I ran as a checklist in Trello.
- However, I was still late finishing this projected.
- I loved the problem solving however was extremely slow which led to high levels of anxiety further slowing the process. The Plan didn't help me progress more swiftly at that stage, although I knew what I had to do. I just had to keep going and eventually it get it done.

- Anxiety could cause significant delays and doubt to my process even if in the end I am capable of producing the task.
- Best approach for future would be to have smaller tasks earlier on to get my confidence up before assessment.
- I learned I would rather make code that doesn't break or make an inaccurate app, than be on time. Ideally, it's better to be both, but given the choice, I couldn't hand mine in without fixing it all.
- But also, learning is also about new things. Now I know how it goes, I'll feel less like feeling my way through the dark and be able to map the progress better. First time around, even the development map could only help me to a limited degree.

Please see the README.md for more extensive screen shots of Trello - or the link is also available there.