Language Learner 3

A Flashcard Style Application for Acquiring Vocabulary!

Table of Contents

1 Brief Introduction

Way back in September 2002 I was 4 years into a new career writing and testing software for a company called AMT-Sybex in Dublin, Ireland. Right at the peak of my programming prowess, I decided to quit (temporarily as it turned out) that job and start up a new life in France. I had studied French at school but had never been much good, so I decided to write an application using the Delphi programming language called “Language Learner 2”. The idea was simple enough – you buy a reasonably small French-English dictionary, and you transcribe all the translations to the application, which stored them on file. In “Testing” mode, the application then gives you each of the saved words back (in random order – why else would we bother!) and asks you to translate them.

If that seems like a boring way to pick up a new language, well it is...until you realise how effective it can be! It really depends on “how” you learn. At school in Ireland the focus was very often (too often!) on “rote” learning. “Here’s a list, please have it memorised by Monday as there will be an exam!” English Literature, History, Geography, Science...they were all taught that way to some extent. The technique suited me, and so I decided to revive the “old ways” when I moved to France and had the terrifying realisation that I would not be able to get by in a rural French town with English alone (since the natives spoke little English outside of the phrase “Rosbif”)! I speak fluent French now, thanks in part to the vocabulary-bashing nature of this application!

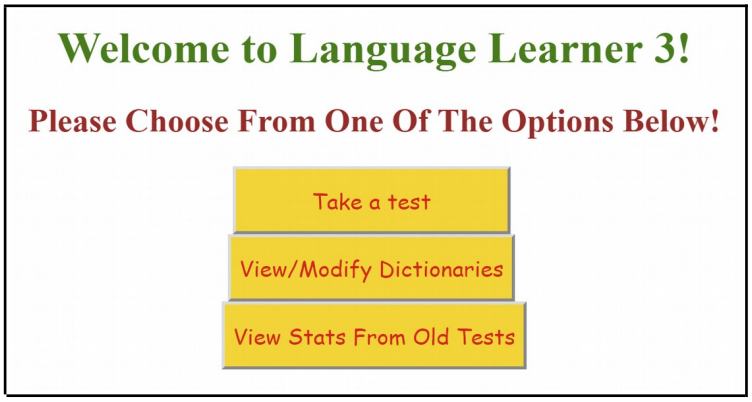
“Hey, but whatever happened to Language Learner 1?”, I hear nobody cry! Well, that was written in a programming language called AMOS, a really amazing coding tool for the Commodore Amiga, even further back in the mists of time in the summer of 1995. I had just completed my first year of University in Cork (studying English Literature and Applied Psychology), and I rewarded myself for passing those exams with the purchase of AMOS. I had been coding as a hobby since Christmas of 1991, when I got my first computer, the legendary Commodore 64. I still dream about “Register 56320” and “Register 56321” to this day, and I continually promise myself that I will spend my last year as a professional coder only writing new software for that wonderful machine. By the way I intend to retire from the coding world at age 100, so if in a few years from now you hear about a 99-year-old writing new joystick-bashing games for newly refurbished C64s, you’ll know it’s me!

2 Technical Notes

- This application has been written in a typical Client/Server architecture, using JavaScript/HTML/CSS for the Front End, and Python for the Backend.
- The framework I've used is Flask. I looked into Django and Flask when starting out, but I chose Flask as it's so quick and easy to get a bare-bones application off the ground and "working".
- This is the first application I have written using any of these languages. The writing of this application is intended as a self-training exercise for me. As such there are no doubt a myriad of code improvements that could be made.
- My intention was to lean on my almost 20 years of good coding practises in a professional setting, as I started out with these new (for me) technologies. As such I have written the code in a tidy manner, keeping the algorithms simple and using comments extensively throughout.
- The database I used is SQLite. I'm sure I could have used others, but this one seemed simple to implement and get up and running with minimal fuss. I've used Oracle and DB2 throughout my professional career, and the SQLite commands are pretty much the same. Plus there's a very handy application called "DB Browser for SQLite" which lets you create, query and update the database quite easily so it was an easy pick in the end.
- Language Learner 3 is still being updated so not all the options on-screen are available (a message should pop up to inform the user when this is the case) and there are a few bugs/areas lacking validation.
- It does the essential tasks properly however: you can enter new/modify existing words for language dictionaries, and run tests against those languages.

3 Functionality Walk-through: Taking a Test

The opening (intro) screen looks like this:



Let’s dive right in and select “Take a test”!

Notes:

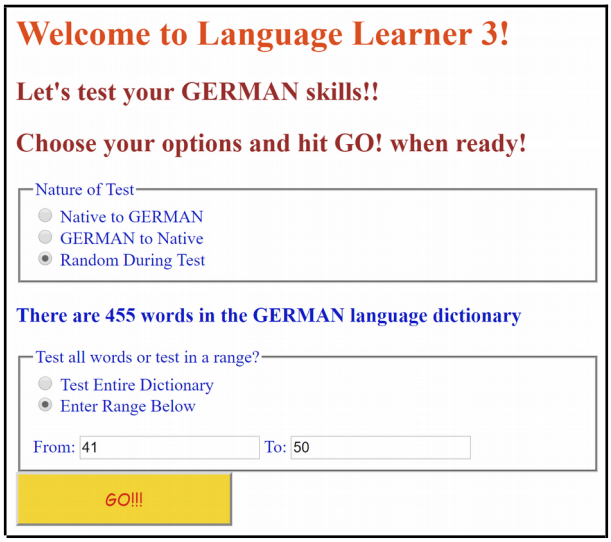
- The option “View Stats From Old Tests” has not yet been coded.

We are then presented with a screen containing a button for each distinct language which currently exists on the database:

These buttons are added to the screen dynamically (using JavaScript) so theoretically there is no limit as to how many different languages can be stored on the database at the same time. Useful for when humans venture beyond the solar system and out towards new suns and new languages!

Let’s stick to Earth languages for now, and choose German.

Once we have chosen a language, we are presented with a screen asking us to choose some optional parameters to enhance our learning experience!



From here the user can choose whether to have all words displayed in their native language (and hence give the German equivalent), the reverse of this, or a random mix during the test. Let's pick "Random During Test" as this makes the test more interesting.

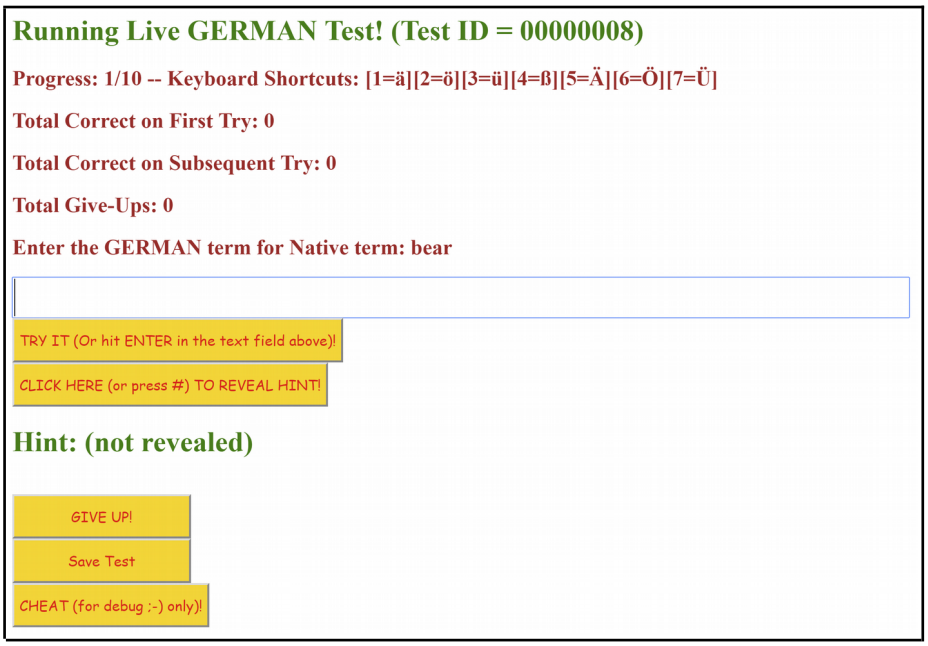
It would take a couple of hours to test the full German dictionary, so let's restrict the word range to just ten words for the purpose of this demonstration.

Click "GO!!!" and we'll begin!

Notes:

- There will be a future option to allow the user to be tested only on the words that they got wrong in previous test(s)
- There is no validation to prevent the user from choosing a "To" value greater than the number of words in the dictionary, so the Python code will crash if the user does so. For now, the user will have to comply with a certain lack of robustness until code has been completed.

We are now brought to the “Running Live Test” screen:

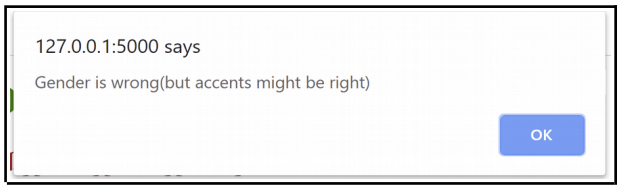


- Line 1 shows the Test ID. This is used to store the results of each test on the database.
- Line 2 shows us how many words we are into the test, plus a sequence of language-specific keyboard shortcuts for characters not available on a regular English keyboard. These keyboard shortcuts are held on the database and can be configured there for each language. In this case, if the user presses “1”, the character “ä” appears instead.
- Lines 3, 4 and 5 keep running totals of how the user is performing on the test!
- Line 6 is prompting the user to enter a GERMAN word for the native (English) word “bear” (the animal – press the HINT button to reveal potential hints!).
- If the prompted word was in GERMAN and the user is asked for the native word, then if the GERMAN word has a plural stored on the database, the plural will appear in parentheses on this line.

The correct answer is “**der bär**” but we will enter “**das bar**” (so the gender is wrong and the word has an incorrect/missing accent).

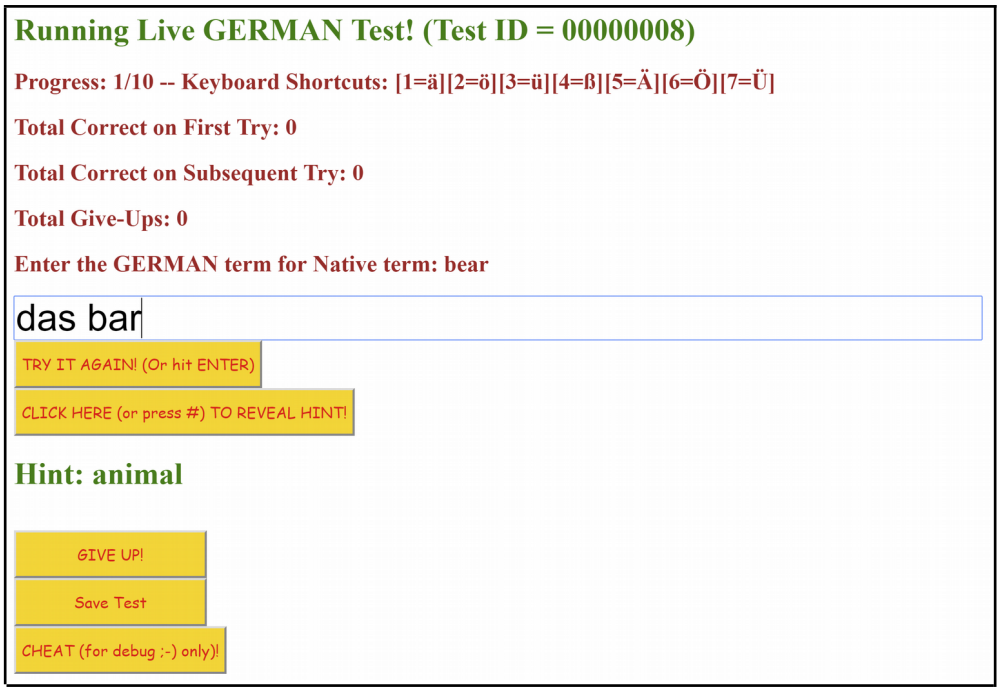
Press ENTER or click the button “TRY IT!” to submit “**das bar**” to see what happens!

We are shown two prompts (alerts) on the basis of what we have entered. They are:



and:

...and the screen is now updated to look like this:



Note that the “Try It” button has updated to “Try It Again”. The “Hint: animal” was revealed after I clicked the “REVEAL HINT” button and the incorrect guess “das bar” is shown in the text field.

The test results are recorded on a word-by-word basis, so will be entered on the database as one of:

- Right First Time
- Right after retry
- Give Up

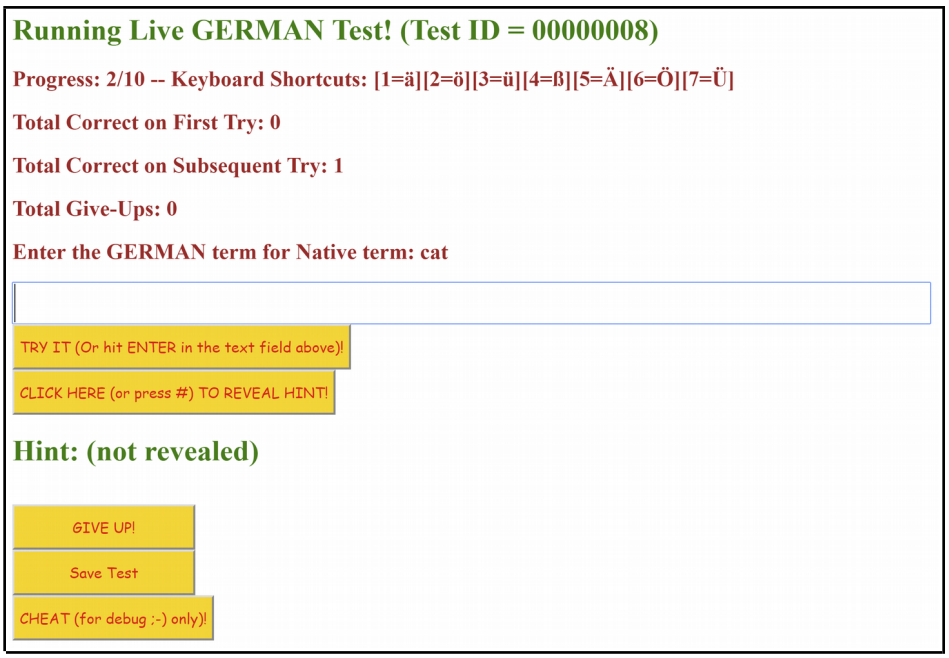
The two pop-up messages informed the user that:

1. They got the gender of the noun wrong (das instead of der)

2. At least one accent was incorrect (“a” instead of “ä”)

Let’s enter the correct value now. When we do so, the result is accepted and the screen updates to

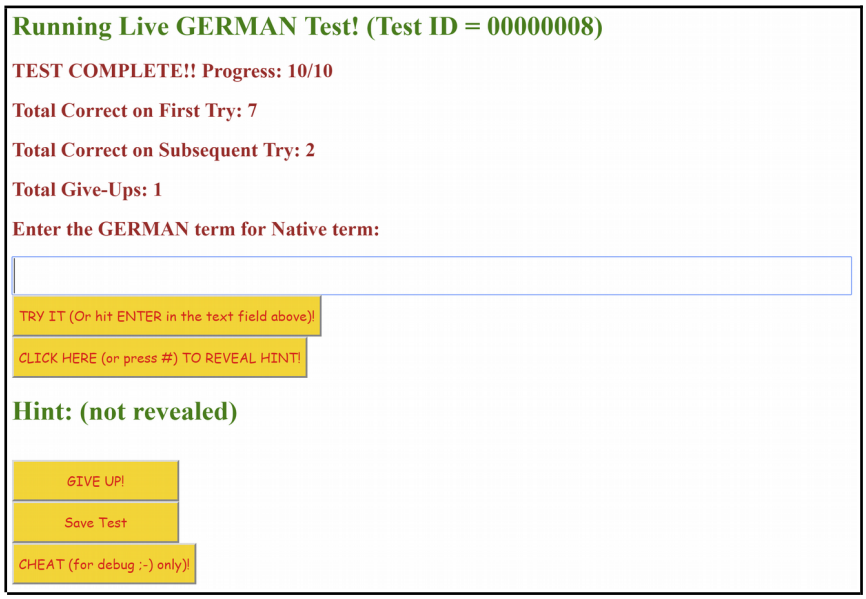
- show the new totals (Progress 2/10, Total Correct on Subsequent Try: 1)
- restore the captions on the buttons
- hide the hint
- prompt for the next word



Notes:

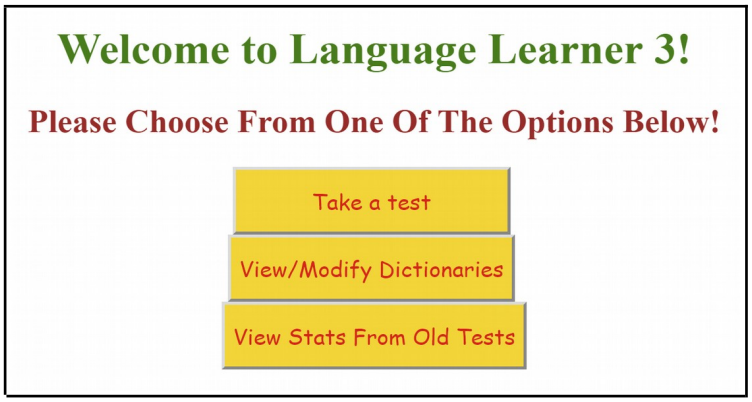
- The “Save Test” functionality has not yet been coded. It will allow the user to save the test at the current point and resume later
- The “CHEAT” button reveals the answer and as suggested, is for debug only! It has no bearing on the “Totals” above and so relies on the honesty of the user :-)

Once all the words have been tested (ten in this case) the final screen is shown and the test is over. We are one step closer to being fluent in our target language!

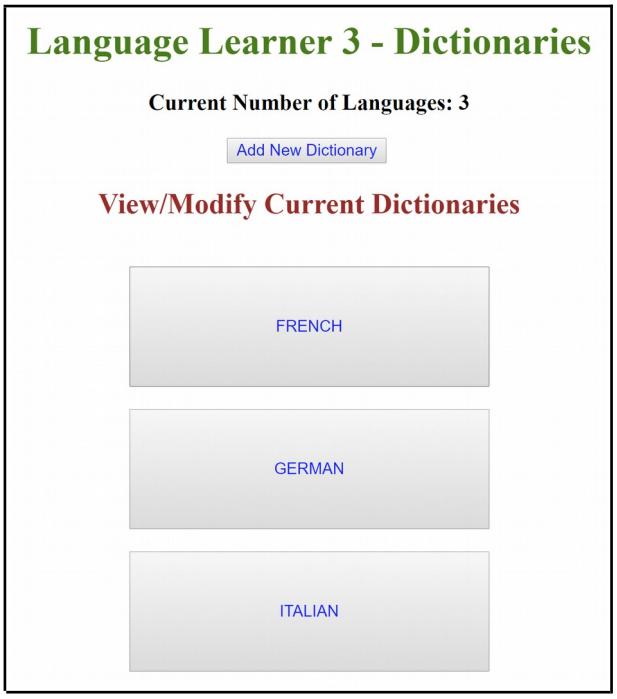


4 Viewing and Modifying Dictionaries

The second option on the opening (intro) screen allows the user to view and/or modify existing dictionaries.



Selecting this option takes us to the following screen:

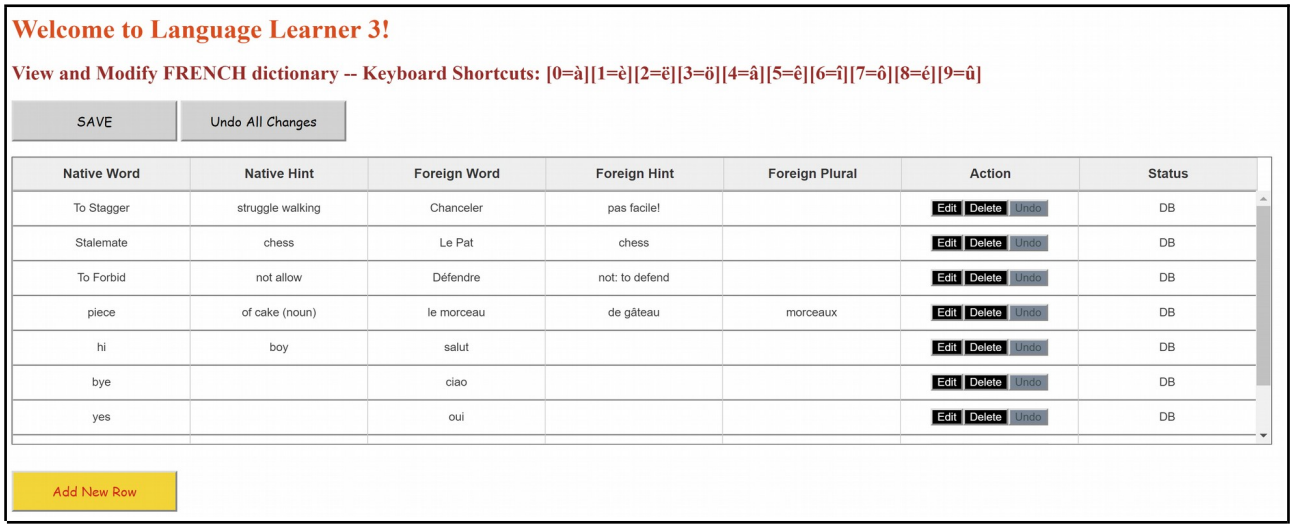


The language buttons are added dynamically by the JavaScript code, so in theory we can have as many languages as we like.

Notes:

- The “Add New Dictionary” functionality is not currently coded. To add a new dictionary the user must create a single entry on the database for the new language, after which a button will appear for the new language.

Let’s choose French. We are brought to this screen:

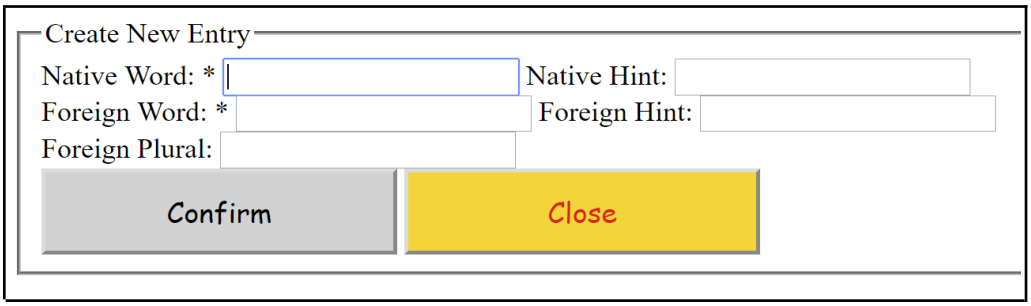


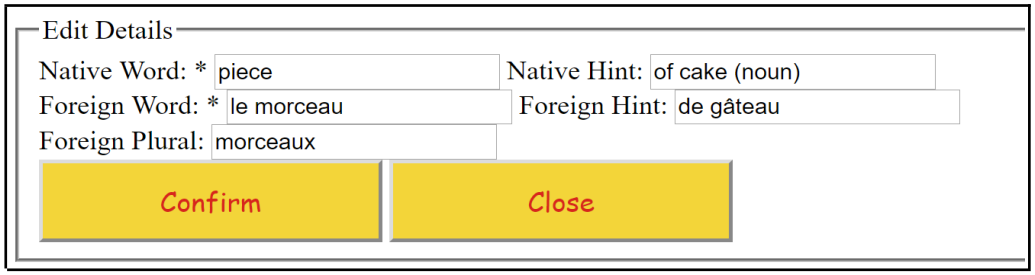
From here we can see a scrollable table which holds all of the words currently in the French dictionary.

Certain buttons are disabled at this point, because we haven’t made any changes yet:

- SAVE
- Undo All Changes
- Undo buttons on the individual words in the table

If we click “Add New Row” or “Edit” (against an individual word on the table) some extra fields open and we can create/modify words from here.





Once the details have been updated we can SAVE them to the database (or undo to cancel).

Note:

- The “Delete” option is not currently coded – it will most likely deactivate words rather than delete them.

5 Database Structure

Four files currently exist on the database. They are listed here with their fields.

1. DICT_RECORD (holds all the words for all the languages).
    - NativeWord
    - NativeHint
    - ForeignWord
    - ForeignHint
    - ForeignPlural
    - ForeignLanguage (Key Field)
    - UniqueNum (Key Field)
2. DIRECT_OBJECT_GENDERS (e.g. “Der”, “Das”, “Die” in German)
    - ForeignLanguage
    - DirectObject
3. SPECIAL_CHARS (e.g. “FRENCH”, “1”, “è”, “e”)
    - ForeignLanguage
    - Key
    - SpecialChar
    - CompareChar
4. TEST_HISTORY
    - ForeignLanguage
    - TestID (unique on a per-language basis)
    - UniqueNum
    - WordSeqInTest (“when” the word was prompted for in the test)
    - WordDirection (e.g. ForeignToNative)
    - NativeWord
    - ForeignWord
    - Result (“RightFirstTime”, “GiveUp”, “RightSubsequentTime”)