בס"ד

**UnicodeTranslator:**

This exercise is a bonus, it includes usage of Unicode, command line arguments, the os module, usage of external modules, researching online solutions to technological problems.

This exercise is concerned with translating Hebrew letters to English.

Throughout the exercise, you will have to look online for solutions to different problems. I recommend as a first stage, write to yourself what you want the program to do, what the technological difficulties you have, and then search online for solutions to each specific requirement.

Also, in each section – be sure to read the entire section before starting work, but don’t read ahead to the other sections.

Background:

\*Character encodings:

The computer represents characters according to binary codes: behind the scenes, when a character is saved, its binary is actually a number. These numbers have meanings, representing different characters according to **character encodings** – codes of translating these binaries to character representations.

There are different encodings available. The most basic encoding is the ASCII encoding – it has 128 different characters – represented by an 8 bit number. ASCII works for English, but once you want to represent more character sets, the need for larger numbers arises.

* You can view the character codes with the ord and chr methods in Python

Unicode is the standard for larger character sets. Unicode is a mapping from characters to numbers. It includes almost all languages, with a unique number representing each character.

\*UTF8 and so on – Unicode is the representation in numbers. There are different schemes for representing the Unicode numbers in binary. UTF8 is one way to represent the numbers in binary. (There exists a problem when translating, because you want to use the minimum number of bytes possible, so there are different ways to do this – one Unicode character does not necessarily need all the bytes it can use).

\* Use Git for source control.

Part 1: (3 hours) Not all programs support Unicode. The difference between Unicode and ASCII can cause problems, where not all programs can understand input that is in Unicode. In the first stage, write a method in Python that receives as input a Unicode string with Hebrew letters, and returns the same string with the representation of the Hebrew letters in English.

* Your method should be simple. The problem can be very complicated, it can be very difficult to create an exact translatation. For this stage, a simple innacurate translation is enough. For example, translating the letter ‘ל’ to ‘l’ is not complicated, but what about "א"? Think about this. A rough translation is enough, for example a substitution of one letter for one letter. ‘h’ for "ה", for example, which will not be 100% accurate (consider the word "האמת"), but a substitution of one letter is quite simple to implement.
* Read about how to handle Unicode strings in Python 3
* **Use library modules!!** Many of these features, for string handling especially, are already implemented somewhere. Read online, search for the features you want. It is very likely that there exist methods and modules that also implement this behavior.

Part 2: (1 hour) Create a program that receives a root directory, walks over the directory and changes the filenames in Hebrew to their English letter equivalents. Your program should receive the path of the root directory from the command line. Use the method from Part 1.

* Read about how to read arguments from the command line. Hint: argparse module
* Think about the design – should this program be in the same file as the method from the first part? (the answer can be yes, it is not a leading question, but a design choice you need to consider)
* Your program should change only filenames, and not directory names.
* The program needs to iterate over **all** the files under the root directory. Hint: os.walk method

Part 3: (1 hour) Change your program to support the feature of a mock run – during a mock run, it generates output of the changes that will happen in the directory, but does not actually change anything. The output can be printing to thee screen the original filenames and the names they will be replaced with, or to a file with this information (the choice is yours). The program does a mock run when run with the command line argument –mock.

* Correct design – where should the mock argument be handled? Think about how the script from before changes for this.

Part 4: (2 hours) Upgrade the translation algorithm from the first stage. Figure out some end cases it does not treat (remember "האמת"?) and modify it to offer a readable representation.

* This should still be simple – don’t overdo it..

Part 5:(Bonus): (3 hours) A file can have additional details, metadata (for example, title, authors, and so on). Add a feature to translate the additional details as well.

* Think about which details you are going to change – there are a lot of details a file can have
* Think about how you are going to modify and read these details? Read online, search for a solution. Also, feel free to install external packages if you need to.
* If you do install external package, you have to document somewhere in your project that it requires these packages to work correctly.

Part 6: (Bonus): (2 hours)Change your program to accept a command line argument telling it whether to modify additional details or not. The way this is going to work – your program is going to have default behavior, and a flag telling it to behave differently.

* Think about what you want the default behavior to be. This can be: by default, the script changes file details, unless a flag called –dont-translate-details appears, or vice versa: by default the script does not change file details, but if run with the flag –translate-details it changes them.
* Think about your program’s behavior when this flag is run with –mock – what happens? What do you want its behavior to be?