CIS 210, Fall 2016
Introduction to Computer Science

Project 1, Part 1

This assignment is due Friday, September 30 at 5pm. Save your Python program in a file called alphacode.py and turn that file in using Canvas.

Purpose

Practice sequential programming. Understand numeric representation and modular arithmetic.

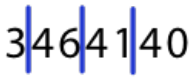
Too many PINs

We have to remember too many numbers: credit card PINs, student ID number, codes to open doors, and on and on. We're warned that we should remember these numbers instead of writing them down, but really the human brain is not well-suited to remembering a lot of meaningless numbers. I'll admit: I write them down. But could I remember more?

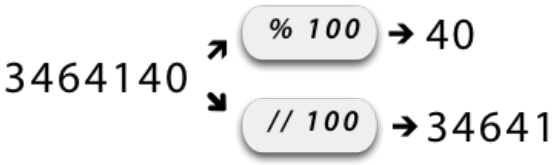
People aren't any better at remembering random sequences of letters than remembering random sequences of digits. However, we are much better at remembering sounds we can pronounce. The word “fesi” doesn't mean anything to me, but it's easier to remember than the number 2354 or the harder-to-pronounce word “iksf”. It turns out to be pretty easy to convert numbers like 2354 into pronounceable nonsense words like “fesi”

To create words that are easy to pronounce, we can build them out of consonant-vowel pairs like “ka” or “te”. As it turns out, if we omit ‘x’ and treat ‘y’ as a consonant, the English alphabet has 20 consonants (bcdfghjklmnpqrstvwyz) and 5 vowels (aeiou). 20×5=10×10=100, so one consonant and a vowel (20×5 combinations) is just enough to represent a pair of decimal digits (10×10 combinations).

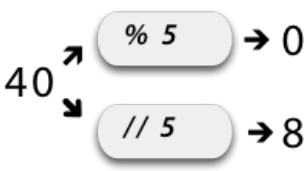
To convert a number (like a phone number, 346-4140) into a pronounceable string, we'll need to divide it into two-digit chunks.



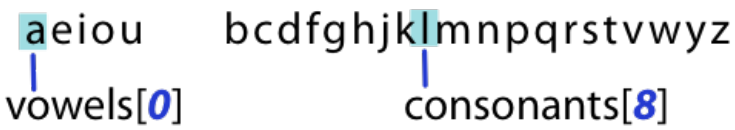
We're dividing it up in base 10; the underlying representation in base 2 is not relevant to us in this program. To get the last two (low order) digits in base 10, we can take the remainder when divided by 100, using the ‘%’ operator. To get the rest of the digits, we'll use integer division ‘//’, because we want an integer result (34641, not 34641.40).



Now we want to convert those last two decimal digits, 40, into letters. We have 20 consonants and 5 vowels. Suppose we divide a number between 0 and 99 by 5. The quotient will be a number in the range 0..19, and the remainder will be in the range 0..4:



Just right for picking one of 20 consonants and one of 5 vowels!



If I do this again for the next two digits (41), and the next (46), and then the highest digit (3, treated as 03), the phone number can be converted to the word “bomelela”. It has a nice ring to it.

Requirements

You will create a Python program to automate the process described above. To keep it simple, start by converting only 4-digit numbers, like credit card PINs. The input to your program is a 4-digit decimal number on the command line, and the output should be formatted *exactly* as shown in these examples:

```
$ python3 alphacode.py 4327
Encoding of 4327 is lohi
$ python3 alphacode.py 1298
Encoding of 1298 is dizo
```

After your program works correctly for four digit numbers, make it work for positive integers of any length. Don't do it by making your program long and repetitious. Use a loop instead. Each time through the loop, in addition to converting the last two digits of the PIN code to letters, chop those digits off the PIN code using integer division. If you keep chopping off digits, eventually what is left is zero, so your loop could be controlled by

```
while pin > 0:
    # your code in the loop
```

If you have trouble designing the loop, use Piazza to ask questions or see us in lab or office hours.

Getting started

This starter code may be helpful. You will also need the test_harness.py library module from the here.

Challenge yourself

If you are learning Python for the first time, I think learning to express this algorithm in Python and make your program work correctly will be enough of a challenge to get started. But if you've just finished CIS 122, or have other experience programming in Python, perhaps this assignment is not challenging enough. Here is something you can do to make it a little more interesting: These alphabetic codes don't really help unless we also have a way to convert a word back into a number. Make your program convert either way: If the input is an integer, convert it to a word. If the input is a word of the correct form (consonants alternating with vowels), convert it back to an integer. If the input cannot be encoded or decoded (e.g., abba123 cannot be translated either direction), print a useful error message.

There are no points for this extra work. Do it only if it makes the project more interesting and challenging. If you do part or all of the extra work, the header comment of your file should include a note describing what you implemented, and how.

Grading

35 points possible

- 15: Program runs and consistently produces correct output for positive integers of any length. 10 points if it works only for four-digit integers. 5 points if it produces some consistent alphabetic code (unique for each integer input) but doesn't match the specified output.
- 10: Follows CIS 210 coding guidelines, including author identification and header .
- 10: Clarity. The program should not only be consistent with the requirements and approach described here, but it should be very easy to read the program and verify its consistency with the spec.