Bank OCR

Problem Description

User Story 1

You work for a bank, which has recently purchased an ingenious machine to assist in reading letters and faxes sent in by branch offices. The machine scans the paper documents, and produces a file with several entries which each look like this:

```
    _  _     _  _  _  _  _
  | _| _||_||_ |_   ||_||_|
  ||_  _|  | _||_|  ||_| _|
```

Each entry is 4 lines long, and each line has 27 characters. The first 3 lines of each entry contain an account number written using pipes and underscores, and the fourth line is blank. Each account number should have 9 digits, all of which should be in the range 0-9. A normal file contains around 500 entries.

Your first task is to write a program that can take this file and parse it into actual account numbers.

User Story 2

Having done that, you quickly realize that the ingenious machine is not in fact infallible. Sometimes it goes wrong in its scanning. The next step therefore is to validate that the numbers you read are in fact valid account numbers. A valid account number has a valid checksum. This can be calculated as follows:

```
account number:  3  4  5  8  8  2  8  6  5
position names:  d9 d8 d7 d6 d5 d4 d3 d2 d1

checksum calculation:
(d1+2*d2+3*d3+...+9*d9) mod 11 = 0
```

So now you should also write some code that calculates the checksum for a given number and identifies if it is a valid account number.

User Story 3

Your boss is keen to see your results. He asks you to write out a file of your findings, one for each input file, in this format:

```
457508000
664371495 ERR
86110??36 ILL
```

ie the file has one account number per row. If some characters are illegible, they are replaced by a ?. In the case of a wrong checksum, or illegible number, this is noted in a second column indicating status.

User Story 4

It turns out that often when a number comes back as ERR or ILL it is because the scanner has failed to pick up on one pipe or underscore for one of the figures. For example

```
    _  _  _  _  _  _     _
|_||_|| || ||_   |  |  ||_
  | _||_||_||_|  |  |  | _|
```

The 9 could be an 8 if the scanner had missed one |. Or the 0 could be an 8. Or the 1 could be a 7. The 5 could be a 9 or 6. So your next task is to look at numbers that have come back as ERR or ILL, and try to guess what they should be, by adding or removing just one pipe or underscore. If there is only one possible number with a valid checksum, then use that. If there are several options, the status should be AMB. If you still can't work out what it should be, the status should be reported ILL.

User Story 5

Fortunately, the bank is doing extremely well (thanks to your solution!) and many new accounts was opened recently. The bad news that we used up all available bank account numbers and decided to use hexadecimal numbers instead of decimals. Luckily the machine can read hexadecimal values as well. Your job is to extend the solution to parse the following hexadecimal digits:

```
 _  _   _   _ \  _   _
|_| |_\ |   | \ |_  |_
| | |_/ |_  |_/ |_  |
```

Make sure that the checksums and error correction functions are working with the new hexadecimal digits as well. Just like with decimal digits the machine can miss a '\' or '/' digit as well.

Clues

I recommend finding a way to write out 3x3 cells on 3 lines in your code, so they form an identifiable digit. Even if your code actually doesn't represent them like that internally. I'd much rather read

```
"    " +
"|_|" +
"  |"
```

than

```
"    |_|   |"
```

any day.

Some gotchas to avoid:

- Be very careful to read the definition of checksum correctly. It is not a simple dot product, the digits are reversed from what you expect.
- The spec does not list all the possible alternatives for valid digits when one pipe or underscore has been removed or added
- Don't forget to try to work out what a ? should have been by adding or removing one pipe or underscore.

How to solve the problem and submit your solution?

Please use python 3 only in your solution and try to **avoid** nonstandard library modules. (It is ok to use 3rd party library for testing, tooling and supporting your development environment.)

Please create your solution in a way that each User Story can be viewed & used by us independently of each other. please provide some guidance how to do so. Please provide a short description on how to use your solution, no further documentation is required.

Please submit all your code, test, documentation and auxiliary files so we can better understand your solution. If you used version controlling during development, please submit your version control history as well. The preferred way of submitting your solution is either in a tgz/zip file or through a publicly accessible github repository.

If your solution requires any install/setup/etc. steps, please indicate as well. If you used any testing/code-checking/etc. please indicate how we can use those as well.

The original exercise and description are coming from Coding Dojo (https://codingdojo.org/kata/BankOCR/). When working on your solution, please avoid reusing or repurposing solutions published by others online.

Have fun and we hope you will enjoy the challenge!