A New Kind Of String Manipulation

Python strings have many methods. You learned about some of those methods in the Strings chapter: `lower()`, `count()`, and `format()`. Now I want to introduce you to a powerful but little-known string manipulation technique: the `translate()` method.

```
translation_table = {ord('A'): ord('O')}          #①
print (translation_table)                         #②
#{65: 79}

print ('MARK'.translate(translation_table))       #③
#MORK
```

① String translation starts with a translation table, which is just a dictionary that maps one character to another. Actually, "character" is incorrect — the translation table really maps one *byte* to another.

② Remember, bytes in Python 3 are integers. The `ord()` function returns the ASCII value of a character, which, in the case of A–Z, is always a byte from 65 to 90.

③ The `translate()` method on a string takes a translation table and runs the string through it. That is, it replaces all occurrences of the keys of the translation table with the corresponding values. In this case, "translating" `MARK` to `MORK`.

Now you're getting to the really fun part. What does this have to do with solving alphametic puzzles? As it turns out, everything.

```
characters = tuple(ord(c) for c in 'SMEDONRY')       #①
print (characters)
#(83, 77, 69, 68, 79, 78, 82, 89)
```

```
guess = tuple(ord(c) for c in '91570682')            #②
print (guess)

#(57, 49, 53, 55, 48, 54, 56, 50)

translation_table = dict(zip(characters, guess))     #③
print (translation_table)
#{82: 56, 83: 57, 68: 55, 69: 53, 89: 50, 77: 49, 78: 54, 79: 48}

print ('SEND + MORE == MONEY'.translate(translation_table))  #④
#'9567 + 1085 == 10652'
```

① Using a generator expression, we quickly compute the byte values for each character in a string. `characters` is an example of the value of `sorted_characters` in the `alphametics.solve()` function.

② Using another generator expression, we quickly compute the byte values for each digit in this string. The result, `guess`, is of the form returned by the `itertools.permutations()` function in the `alphametics.solve()` function.

③ This translation table is generated by zipping `characters` and `guess` together and building a dictionary from the resulting sequence of pairs. This is exactly what the `alphametics.solve()` function does inside the for loop.

④ Finally, we pass this translation table to the `translate()` method of the original puzzle string. This converts each letter in the string to the corresponding digit (based on the letters in `characters` and the digits in `guess`). The result is a valid Python expression, as a string.

That's pretty impressive. But what can you do with a string that happens to be a valid Python expression?