A File Of Patterns

You've factored out all the duplicate code and added enough abstractions so that the pluralization rules are defined in a list of strings. The next logical step is to take these strings and put them in a separate file, where they can be maintained separately from the code that uses them.

First, let's create a text file that contains the rules you want. No fancy data structures, just whitespace-delimited strings in three columns. Let's call it `plural4-rules.txt`.

```
[sxz]$                $    es
[^aeioudgkprt]h$      $    es
[^aeiou]y$           y$    ies
$                     $    s
```

Now let's see how you can use this rules file.

```
import re

def build_match_and_apply_functions(pattern, search, replace):     #①
    def matches_rule(word):
        return re.search(pattern, word)
    def apply_rule(word):
        return re.sub(search, replace, word)
    return (matches_rule, apply_rule)

rules = []
with open('plural4-rules.txt', encoding='utf-8') as pattern_file:  #②
    for line in pattern_file:                                      #③
        pattern, search, replace = line.split(None, 3)             #④
        rules.append(build_match_and_apply_functions(              #⑤
                pattern, search, replace))
```

① The `build_match_and_apply_functions()` function has not changed. You're still using closures to build two functions dynamically that use variables defined in the outer function.

② The global `open()` function opens a file and returns a file object. In this case, the file we're opening contains the pattern strings for pluralizing nouns. The `with` statement creates what's called a *context*: when the `with` block ends, Python will automatically close the file, even if an exception is raised inside the with block. You'll learn more about with blocks and file objects in the Files chapter.

③ The `for line in <fileobject>` idiom reads data from the open file, one line at a time, and assigns the text to the line variable. You'll learn more about reading from files in the Files chapter.

④ Each line in the file really has three values, but they're separated by whitespace (tabs or spaces, it makes no difference). To split it out, use the `split()` string method. The first argument to the `split()` method is `None`, which means "split on any whitespace (tabs or spaces, it makes no difference)." The second argument is `3`, which means "split on whitespace 3 times, then leave the rest of the line alone." A line like `[sxz]$ $ es` will be broken up into the list `['[sxz]$', '$', 'es']`, which means that pattern will get `'[sxz]$'`, search will get `'$'`, and replace will get `'es'`. That's a lot of power in one little line of code.

⑤ Finally, you pass `pattern`, `search`, and `replace` to the `build_match_and_apply_functions()` function, which returns a tuple of functions. You append this tuple to the `rules` list, and `rules` ends up storing the list of match and apply functions that the `plural()` function expects.

The improvement here is that you've completely separated the pluralization rules into an external file, so it can be maintained separately from the code that uses it. Code is code, data is data, and life is good.