



Regular Expressions

Objectives

To become familiar with some of Python's regular expression tools, and continuing practice with Python syntax and string handling.

Reference Material

Primarily Chapter 6 Regular Expressions, but you may also need to refer to Chapter 4 String Handling. The online documentation for the **re** module may also be useful.

Questions

1. Write a Python script that will perform *basic* validation and formatting of UK postcodes.

The postcodes are read from file `postcodes.txt`. A skeleton script, **Ex6.py**, contains the necessary file handling code - you fill in the gaps at the **TODO (a)** comments. This script is also used for the second exercise (below), so ignore the **TODO(b)** comments for the time being.

Blank lines should be ignored.

The following formatting is to be done:

- Remove all new-lines, tabs and spaces

- Convert to uppercase

- Insert a space before the final digit and 2 letters

Print out all the reformatted postcodes

Hints:

Keep it simple; don't try to do all the formatting on one line of code!

Read the **TODO (a)** comments in the code skeleton - ignore those for part (b) for the time being.

There are several ways to insert the space, the simplest is to use `re.sub` and a back-reference.

Put regular expressions into raw strings.



2. Now extend your script so that it performs *basic* pattern validation of each postcode (**TODO (b)** comments)

The input lines are only to contain a postcode, with no other text.

The format of a UK postcode is as follows:

One or two uppercase alphabetic characters

followed by: between one and two digits

followed by: an optional single uppercase alphabetic character

followed by: a single space

followed by: a single digit

followed by: two uppercase alphabetic characters

Alphabetic characters are those in the range A-Z.

Print out all the reformatted postcodes, indicating which are in error, and a count of valid and invalid codes at the end.

Hints:

Do the formatting first, and then test the resulting pattern

Read the TODO (b) comments in the code skeleton

Use a raw string for your regular expression

The test file has 25 valid and 5 invalid postcodes.

If time allows...

3. The area and district part of the postcode (the part before the space) can be validated by looking in file **validpc.txt**. This also records the country the area is in. We have not done the File IO chapter yet, but this is how you read an entire file into a list, one line per element:

```
valid = open('validpc.txt').read().splitlines()
```

Modify your code to search for the area-district captured from your regular expression, and output which country the postcode belongs to. We suggest the following steps:

- a) Read **validpc.txt** and store it into a dictionary, where the key is the area-district and the value is the country. So, using the statement given above, you need to write a `for` loop to generate the dictionary:

```
for txt in valid:
    # Split up the line around a comma, etc...
```

- b) Alter your main regular expression to capture the relevant part of the postcode.
- c) If the postcode matches the RE, look it up in the dictionary and report which country it belongs to, or an error message if it is not there.



Solutions

```
import re

infile = open('postcodes.txt', 'r')

valid    = 0
invalid  = 0

for postcode in infile:
    # Ignore empty lines
    if postcode.isspace(): continue

    # (a): Remove newlines, tabs and spaces
    postcode = re.sub('[ \t\n]', '', postcode)

    # (a): Convert to uppercase
    postcode = postcode.upper()

    # (a): Insert a space before the final digit and 2 letters
    postcode = re.sub('([0-9][A-Z]{2})$', r' \1', postcode)

    # Print the reformatted postcode
    print postcode

    # (b) Validate the postcode, returning a match object
    m = re.match(
        r"[A-Z]{1,2}[0-9]{1,2}[A-Z]? [0-9][A-Z]{2}",
        postcode)
    if m:
        valid = valid + 1
    else:
        invalid = invalid + 1

infile.close()

# (b) Print the valid and invalid totals
print valid, "valid codes and", invalid, "invalid codes"
```

If time allows:



Here is the additional code:

```
# Part c
valid_dict = {}
valid = open('validpc.txt').read().splitlines()
for txt in valid:
    line=txt.split(',')
    valid_dict[line[0]] = line[1]
valid=None
# end of valid_dict initialisation
...
# Note the extra parentheses
m = re.match(
    r"([A-Z]{1,2}[0-9]{1,2}[A-Z]?) [0-9][A-Z]{2}",
    postcode)
if m:
    valid = valid + 1
    # Part c
    area_district = m.groups()[0]

    if area_district in valid_dict:
        print postcode,"is in",valid_dict[area_district]
    else:
        print postcode,"not found"
else:
    invalid = invalid + 1
```