

SLP_3e_chp2

August 1, 2021

1 Chapter 2 - Regular Expression - Exercises

1.1 from Speech and Language Processing by Martin and Jurafsky, 3e the freely available draft dated December 2020

1.2 code by Vaibhav Mittal on 31st July, 2021

```
[1]: from re import finditer, compile
import numpy as np
# from https://regexone.com/references/python
```

1.3 RegEx to find the set of all alphabetic strings

```
[12]: regex = compile(r'[A-Za-z]+')
target = 'abc012!@#abc!@#a bcb      1'

print('Target string:- ' + target)
print('\n')
for i in finditer(regex, target):
    print(i)
```

Target string:- abc012!@#abc!@#a bcb 1

```
<re.Match object; span=(0, 3), match='abc'>
<re.Match object; span=(9, 12), match='abc'>
<re.Match object; span=(15, 16), match='a'>
<re.Match object; span=(17, 20), match='bcb'>
```

1.4 Find the set of all lowercase alphabetic strings ending in a 'b'

```
[13]: regex = compile(r'[a-z]+b')
target = 'abc012!@#abcb!@#a bcb      1abb#bbba AB Ab'

print('Target string:- ' + target)
print('\n')
for i in finditer(regex, target):
    print(i)
```

```
<re.Match object; span=(0, 2), match='ab'>  
<re.Match object; span=(9, 13), match='abcb'>  
<re.Match object; span=(18, 21), match='bcb'>  
<re.Match object; span=(23, 26), match='abb'>  
<re.Match object; span=(27, 30), match='bbb'>
```

[illegible]

```
<re.Match object; span=(0, 3), match='aab'>
<re.Match object; span=(16, 19), match='aab'>
<re.Match object; span=(21, 24), match='aab'>
<re.Match object; span=(28, 31), match='aab'>
<re.Match object; span=(31, 34), match='aab'>
<re.Match object; span=(34, 37), match='aab'>
```

```
[15]: regex = compile(r'(\w+)\s+\1\b')
      target = "Humbert Humbert the Humbert big the the big bug"

      print('Target string:- ' + target)
      print('\n')
      for i in finditer(regex, target):
          print(i)
```

```
<re.Match object; span=(0, 15), match='Humbert Humbert'>
<re.Match object; span=(32, 39), match='the the'>
```

1.7 All strings that start at the beginning of the line with an integer and end at the end of the line with a word

```
[17]: regex = compile(r'^\d.*\w$')
target = r"12Humbert Humbert the Humbert big the the big bug\nHumbert Humbert_
↳the Humbert big the the big bug\nHumber humbert is"

print('Target text:- ' + target)
print('\n')
for i in finditer(regex, target):
    print(i)
```

Target text:- 12Humbert Humbert the Humbert big the the big bug\nHumbert Humbert
the Humbert big the the big bug\nHumber humbert is

```
<re.Match object; span=(0, 117), match='12Humbert Humbert the Humbert big the
the big bug>
```

2 ELIZA-like RegEx program

An ELIZA-like program is implemented here. This program acts like a young, curious intern asking their supervisor on how to get things done. It uses simple regex substitutions and engages in lot of simple repetition

```
[ ]: 
```

```
[ ]: 
```

3 Minimum Edit Distance

```
[5]: def min_edit_distance(source, target, del_cost = 1, ins_cost = 1, sub_cost = 2):
    """
    A function which takes a source and target (string) and returns the minimum_
    ↳edit distance (integer)
    """

    n = len(source)
    m = len(target)
    D = np.zeros((n+1, m+1))

    for i in range(1, n+1):
        D[i, 0] = D[i-1, 0] + del_cost
    for j in range(1, m+1):
        D[0, j] = D[0, j-1] + ins_cost

    for i in range(1, n+1):
```

```

        for j in range(1, m+1):
            deletion = D[i-1, j] + del_cost
            insertion = D[i, j-1] + ins_cost
            substitution = D[i-1, j-1] + calculate_sub_cost(source[i-1],
↪target[j-1], sub_cost)
            D[i, j] = min(deletion, insertion, substitution)
        return D[n, m]

```

```

[6]: def calculate_sub_cost(source, target, sub_cost = 2):
      """
      A function to calculate substitution costs taking the substitution or
↪non-substitution into account
      """

      if source == target:
          return 0
      else:
          return sub_cost

```

```

[7]: med = min_edit_distance("intention", "execution", del_cost = 1, ins_cost = 1,
↪sub_cost = 2)
      print(med)

```

8.0

3.1 Edit Distance between “leda” and “deal” with each cost as 1

```

[8]: print(min_edit_distance("lead", "deal", sub_cost = 1))

```

2.0

3.2 Edit Distance between “drive” and “brief” and between “drive” and “divers”

```

[10]: print("The minimum edit distance between 'drive' and 'brief' is " +
↪str(min_edit_distance("drive", "brief")))
      print("The minimum edit distance between 'drive' and 'divers' is " +
↪str(min_edit_distance("drive", "divers")))

```

The minimum edit distance between 'drive' and 'brief' is 4.0

The minimum edit distance between 'drive' and 'divers' is 3.0

3.3 Output an alignment from the minimum edit distance algorithm

```

[19]: def min_edit_distance_with_alignment(source, target, del_cost = 1, ins_cost =
↪1, sub_cost = 2):
      """

```

A function which takes a source and target (string) and returns the alignment between the two (integer)

```

"""

n = len(source)
m = len(target)
D = np.zeros((n+1, m+1))

for i in range(1, n+1):
    D[i, 0] = D[i-1, 0] + del_cost
for j in range(1, m+1):
    D[0, j] = D[0, j-1] + ins_cost

backtrace = []
for i in range(1, n+1):
    for j in range(1, m+1):
        deletion = D[i-1, j] + del_cost
        insertion = D[i, j-1] + ins_cost
        substitution = D[i-1, j-1] + calculate_sub_cost(source[i-1],
↪target[j-1], sub_cost)
        D[i, j] = min(deletion, insertion, substitution)

        if substitution == D[i, j]:
            backtrace.append('diagonal')
        if insertion == D[i, j]:
            backtrace.append('up')
        if deletion == D[i, j]:
            backtrace.append('left')

print(backtrace)

return D[n, m]

```

[20]: med = min_edit_distance_with_alignment("intention", "execution", del_cost = 1,
↪ins_cost = 1, sub_cost = 2)

```

['diagonal', 'up', 'left', 'diagonal', 'up', 'left', 'diagonal', 'up', 'left',
'diagonal', 'up', 'left', 'diagonal', 'up', 'left', 'diagonal', 'up', 'left',
'diagonal', 'up', 'up', 'diagonal', 'up', 'left', 'diagonal', 'up', 'left',
'diagonal', 'up', 'left', 'diagonal', 'up', 'left', 'diagonal', 'up', 'left',
'diagonal', 'up', 'left', 'left', 'diagonal', 'up', 'left', 'diagonal',
'diagonal', 'up', 'left', 'diagonal', 'up', 'left', 'diagonal', 'up', 'left',
'diagonal', 'up', 'left', 'diagonal', 'up', 'left', 'diagonal', 'up', 'left',
'diagonal', 'up', 'left', 'left', 'diagonal', 'up', 'diagonal', 'up', 'up',
'up', 'up', 'left', 'diagonal', 'up', 'left', 'diagonal', 'up', 'left', 'left',
'left', 'diagonal', 'up', 'left', 'diagonal', 'up', 'left', 'diagonal', 'up',
'left', 'diagonal', 'up', 'left', 'diagonal', 'up', 'left', 'diagonal', 'up',

```

```
'left', 'diagonal', 'up', 'left', 'diagonal', 'left', 'left', 'diagonal', 'up',  
'left', 'diagonal', 'up', 'left', 'diagonal', 'up', 'left', 'diagonal', 'up',  
'left', 'diagonal', 'up', 'up', 'up', 'left', 'left', 'diagonal', 'up', 'left',  
'diagonal', 'up', 'left', 'diagonal', 'up', 'left', 'diagonal', 'up', 'left',  
'left', 'diagonal', 'up', 'up', 'left', 'diagonal', 'up', 'left', 'diagonal',  
'up', 'left', 'diagonal', 'up', 'left', 'diagonal', 'up', 'left', 'left',  
'left', 'diagonal', 'up', 'left', 'diagonal', 'up', 'left', 'diagonal', 'up',  
'left', 'diagonal', 'up', 'left', 'diagonal', 'up', 'left', 'left', 'left',  
'left', 'diagonal']
```

[]: