20 November 2015, 21 April 2017,15 April 2017

The function l33tify takes as inputs a string and a dictionary, and outputs the string translated to leet according to the dictionary

In [1]:
```
l33t_dict = {
'a': '4',
'b': '8',
'c': '(',
'e': '3',
'g': '[',
'i': '1',
'o': '0',
's': '5',
't': '7',
'x': '%',
'z': '2',
}
```

In [2]:
```
def l33tify(text, l33t_dict):

    textlist = list(text)

    for i in range(len(textlist)):

        if l33t_dict.get(textlist[i]) != None:

            textlist[i] = l33t_dict.get(textlist[i])

    return ''.join(textlist)
```

In [3]:
```
l33tify("pheer my leet skills", l33t_dict)
```

Out[3]:
```
'ph33r my l337 5k1ll5'
```

In [4]:
```
l33tify("python is cool", l33t_dict)
```

Out[4]:
```
'py7h0n 15 (00l'
```

In [5]:
```
adv_l33t_dict = {
'a': ['4', '/-\\', '/_\\', '@', '/\\'],
'b': ['8', '|3', '13', '|}', '|:', '|8', '18', '6', '|B'],
'c': ['<', '{', '[', '('],
'd': ['|)', '|}', '|]'],
'e': ['3'],
'f': ['|=', 'ph', '|#', '|"'],
'g': ['[', '-', '[+', '6'],
'h': ['|-|', '[-]', '{-}', '|=|', '[=]', '{=}'],
'i': ['1', '|'],
'j': ['_|', '_/', '_7', '_)'],
'k': ['|<', '1<'],
'l': ['|_', '|', '1'],
'm': ['|\\/|', '^^', '/\\/\\'],
'n': ['|\\|', '/\\/', '/V', '][\\\\]['],
'o': ['0', '()', '[]', '{}'],
'p': ['|o', '|O', '|>', '|*', '|Âˇr', '|D', '/o'],
```

```
'q': ['O_', '9', '(', ')', ''],
'r': ['|2', '12', '.-', '|^'],
's': ['5', '$', '§'],
't': ['7', '+', '7`', "'|'"],
'u': ['|_|', '\\_\\', '/_/', '\\_/', '(_)'],
'v': ['\\/'],
'w': ['\\/\\/', '(/\\)', '\\^/', '|/\\|'],
'x': ['%', '*', '><', '}{', ')('],
'y': ['`/', '¥'],
'z': ['2', '7_', '>_']
}
```

The l33tify function is not very leet because it always replaces with a fixed character. Since there are usually multiple possible replacements for a character, we can do better by having our function rotate through the possible replacements. The values of the leet dictionary will now be a list of strings, and the order of the replacement will follow the order of the list. The function advance_l33tify takes as inputs a string and a dictionary, and outputs the string translated to leet according to scheme described above.

In [6]:
```
def advance_l33tify(text, adv_l33t_dict):

    currentIndex = {}

    textList = list(text)

    for key in adv_l33t_dict.keys():

        currentIndex[key] = 0

    for i in range(len(textList)):

        if adv_l33t_dict.get(textList[i]) != None:

            length = len(adv_l33t_dict.get(textList[i]))
            index = currentIndex[textList[i]]
            currentIndex[textList[i]] = (index + 1)%length
            textList[i] = adv_l33t_dict.get(textList[i])[index]

    return ''.join(textList)
```

In [7]:
```
advance_l33tify("Bow b4 me 4 I am root!!!", adv_l33t_dict)
```

Out[7]:
```
'B0\\/\\/ 84 |\\/|3 4 I 4^^ |2()[]7!!!'
```

In [8]:
```
advance_l33tify("Mississippi", adv_l33t_dict)
```

Out[8]:
```
'M15$|§51|o|O|'
```

In [9]:
```
nus_matric = {
0: 'Y',
1: 'X',
2: 'W',
3: 'U',
4: 'R',
5: 'N',
6: 'M',
7: 'L',
8: 'J',
9: 'H',
10: 'E',
```

```
11: 'A',
12: 'B'
}
```

The function check_digit takes as inputs an identification number (which is a string of digits) and a lookup up table (which is a dictionary). It outputs the respective check digit (character) of the identification number.

In [10]:
```
def check_digit(num,matric):

    summation = 0
    length = len(matric)

    for character in num :

        summation = (summation + int(character))%(length)

    return matric[summation]
```

In [11]:
```
check_digit("0113093", nus_matric)
```

Out[11]:
```
'R'
```

In [12]:
```
check_digit("0129969", nus_matric)
```

Out[12]:
```
'E'
```

The function weighted_check_digit takes as inputs an identification number (which is a string of digits), a lookup up table (which is a dictionary) and the weights (which is a string of digits). It outputs the respective check digit (character) of the identification number.

In [13]:
```
sg_nric = dict(enumerate("JZIHGFEDCBA"))
sg_weights = "2765432"
```

In [14]:
```
def weighted_check_digit(num, sg_nric, sg_weights):

    summation = 0
    length = len(sg_nric)

    for i in range(len(num)) :

        summation = (summation + (int(num[i])*int(sg_weights[i])))%(length)

    return sg_nric[summation]
```

In [15]:
```
 weighted_check_digit("9702743", sg_nric, sg_weights)
```

Out[15]:
```
'I'
```

In [16]:
```
weighted_check_digit("9875133", sg_nric, sg_weights)
```

Out[16]:
```
'E'
```

Write a function char_at that take in as input, a character and a sentence, and outputs a list containing the positions (starting from 0) of the character in the sentence. The function should be case insensitive, i.e., uppercase and lowercase characters should be counted as the same.

In [17]:
```
def char_at(letter, sentence):

    positionList = []

    for i in range(len(sentence)):

        if sentence[i].lower() == letter.lower():

            positionList.append(i)

    return positionList
```

In [18]:
```
sentence = 'The quick brown fox jumps over the lazy dog.'
char_at('e', sentence)
```

Out[18]:
```
[2, 28, 33]
```

In [19]:
```
char_at('t', sentence)
```

Out[19]:
```
[0, 31]
```

In [20]:
```
char_at('7', sentence)
```

Out[20]:
```
[]
```

Write a function contains_duplicate that takes a string and returns True if the string contains any duplicate characters, and False otherwise. The function should be case insensitive, i.e., uppercase and lowercase characters should be counted as the same.

In [21]:
```
def contains_duplicate(text):

    flag = False

    for letter in text:

        repeats = char_at(letter, text)

        if len(repeats) > 1 :

            flag = True
            break

    return flag
```

In [22]:
```
 contains_duplicate("abcdefg")
```

Out[22]:
```
False
```

In [23]:
```
contains_duplicate("ambidextrously")
```

Out[23]:
```
False
```

In [24]:
```
contains_duplicate("Double-dating")
```

Out[24]:
```
True
```

In [25]:
```
contains_duplicate("cs1010s")
```

Out[25]:
```
True
```

Write a function duplicate_within that takes in a string and an integer i, and returns True if the there exists any duplicated characters in the string that are at most i characters apart, and False otherwise.The function should be case insensitive, i.e., uppercase and lowercase characters should be counted as the same.

In [26]:
```
def duplicate_within(text, num):

    flag = False

    for letter in text:

        repeats = char_at(letter, text)

        if len(repeats) > 1:


            for i in range(len(repeats) - 1):

                    value = repeats[i+1] - repeats[i]
                    if value == num:

                        return True

    return False
```

In [27]:
```
duplicate_within("Double-dating", 6)
```

Out[27]:
```
False
```

In [28]:
```
duplicate_within("Double-dating", 7)
```

Out[28]:
```
True
```

In [29]:
```
duplicate_within("ballooning", 1)
```

Out[29]:
```
True
```

In [30]:
```
duplicate_within("ballooning", 0)
```

Out[30]:
```
False
```

In [31]:
```
duplicate_within("ambidextrously", 1)
```

Out[31]:
```
False
```