

```
1  from __future__ import print_function
2
3  american_number_system = {
4      'zero': 0,
5      'one': 1,
6      'two': 2,
7      'three': 3,
8      'four': 4,
9      'five': 5,
10     'six': 6,
11     'seven': 7,
12     'eight': 8,
13     'nine': 9,
14     'ten': 10,
15     'eleven': 11,
16     'twelve': 12,
17     'thirteen': 13,
18     'fourteen': 14,
19     'fifteen': 15,
20     'sixteen': 16,
21     'seventeen': 17,
22     'eighteen': 18,
23     'nineteen': 19,
24     'twenty': 20,
25     'thirty': 30,
26     'forty': 40,
27     'fifty': 50,
28     'sixty': 60,
29     'seventy': 70,
30     'eighty': 80,
31     'ninety': 90,
32     'hundred': 100,
33     'thousand': 1000,
34     'million': 1000000,
35     'billion': 1000000000,
36     'point': '.'
37 }
38
39 decimal_words = ['zero', 'one', 'two', 'three', 'four', 'five', 'six', 'seven',
40                  'eight', 'nine']
41
42 """
43 #TODO
44 indian_number_system = {
45     'zero': 0,
46     'one': 1,
47     'two': 2,
48     'three': 3,
49     'four': 4,
50     'five': 5,
51     'six': 6,
52     'seven': 7,
53     'eight': 8,
54     'nine': 9,
55     'ten': 10,
56     'eleven': 11,
57     'twelve': 12,
58     'thirteen': 13,
59     'fourteen': 14,
60     'fifteen': 15,
61     'sixteen': 16,
62     'seventeen': 17,
63     'eighteen': 18,
64     'nineteen': 19,
65     'twenty': 20,
66     'thirty': 30,
67     'forty': 40,
68     'fifty': 50,
69     'sixty': 60,
70     'seventy': 70,
71     'eighty': 80,
72     'ninety': 90,
```

```

72     'hundred': 100,
73     'thousand': 1000,
74     'lac': 100000,
75     'lakh': 100000,
76     'crore': 10000000
77 }
78 """
79
80
81 """
82 function to form numeric multipliers for million, billion, thousand etc.
83 input: list of strings
84 return value: integer
85 """
86
87
88 def number_formation(number_words):
89     numbers = []
90     for number_word in number_words:
91         numbers.append(american_number_system[number_word])
92     if len(numbers) == 4:
93         return (numbers[0] * numbers[1]) + numbers[2] + numbers[3]
94     elif len(numbers) == 3:
95         return numbers[0] * numbers[1] + numbers[2]
96     elif len(numbers) == 2:
97         if 100 in numbers:
98             return numbers[0] * numbers[1]
99         else:
100             return numbers[0] + numbers[1]
101     else:
102         return numbers[0]
103
104
105 """
106 function to convert post decimal digit words to numeral digits
107 input: list of strings
108 output: double
109 """
110
111
112 def get_decimal_sum(decimal_digit_words):
113     decimal_number_str = []
114     for dec_word in decimal_digit_words:
115         if (dec_word not in decimal_words):
116             return 0
117         else:
118             decimal_number_str.append(american_number_system[dec_word])
119     final_decimal_string = '0.' + ''.join(map(str, decimal_number_str))
120     return float(final_decimal_string)
121
122
123 """
124 function to return integer for an input `number_sentence` string
125 input: string
126 output: int or double or None
127 """
128
129
130 def word_to_num(number_sentence):
131     if type(number_sentence) is not str:
132         raise ValueError("Type of input is not string! Please enter a valid number
133                             word (eg. \'two million twenty three thousand and forty nine\')")
134
135     number_sentence = number_sentence.replace('-', ' ')
136     number_sentence = number_sentence.lower() # converting input to lowercase
137
138     if (number_sentence.isdigit()): # return the number if user enters a number string
139         return int(number_sentence)
140
141     split_words = number_sentence.strip().split() # strip extra spaces and split
142     sentence into words

```

```

142 clean_numbers = []
143 clean_decimal_numbers = []
144
145 # removing and, & etc.
146 for word in split_words:
147     if word in american_number_system:
148         clean_numbers.append(word)
149
150 # Error message if the user enters invalid input!
151 if len(clean_numbers) == 0:
152     raise ValueError("No valid number words found! Please enter a valid number
153 word (eg. two million twenty three thousand and forty nine)")
154
155 # Error if user enters million, billion, thousand or decimal point twice
156 if clean_numbers.count('thousand') > 1 or clean_numbers.count('million') > 1 or
157 clean_numbers.count('billion') > 1 or clean_numbers.count('point') > 1:
158     raise ValueError("Redundant number word! Please enter a valid number word
159 (eg. two million twenty three thousand and forty nine)")
160
161 # separate decimal part of number (if exists)
162 if clean_numbers.count('point') == 1:
163     clean_decimal_numbers = clean_numbers[clean_numbers.index('point')+1:]
164     clean_numbers = clean_numbers[:clean_numbers.index('point')]
165
166 billion_index = clean_numbers.index('billion') if 'billion' in clean_numbers
167 else -1
168 million_index = clean_numbers.index('million') if 'million' in clean_numbers
169 else -1
170 thousand_index = clean_numbers.index('thousand') if 'thousand' in clean_numbers
171 else -1
172
173 if (thousand_index > -1 and (thousand_index < million_index or thousand_index <
174 billion_index)) or (million_index > -1 and million_index < billion_index):
175     raise ValueError("Malformed number! Please enter a valid number word (eg.
176 two million twenty three thousand and forty nine)")
177
178 total_sum = 0 # storing the number to be returned
179
180 if len(clean_numbers) > 0:
181     # hack for now, better way TODO
182     if len(clean_numbers) == 1:
183         total_sum += american_number_system[clean_numbers[0]]
184
185     else:
186         if billion_index > -1:
187             billion_multiplier = number_formation(clean_numbers[0:billion_index])
188             total_sum += billion_multiplier * 1000000000
189
190         if million_index > -1:
191             if billion_index > -1:
192                 million_multiplier =
193                 number_formation(clean_numbers[billion_index+1:million_index])
194             else:
195                 million_multiplier =
196                 number_formation(clean_numbers[0:million_index])
197             total_sum += million_multiplier * 1000000
198
199         if thousand_index > -1:
200             if million_index > -1:
201                 thousand_multiplier =
202                 number_formation(clean_numbers[million_index+1:thousand_index])
203             elif billion_index > -1 and million_index == -1:
204                 thousand_multiplier =
205                 number_formation(clean_numbers[billion_index+1:thousand_index])
206             else:
207                 thousand_multiplier =
208                 number_formation(clean_numbers[0:thousand_index])
209             total_sum += thousand_multiplier * 1000
210
211         if thousand_index > -1 and thousand_index != len(clean_numbers)-1:
212             hundreds = number_formation(clean_numbers[thousand_index+1:])
213         elif million_index > -1 and million_index != len(clean_numbers)-1:

```

```
201         hundreds = number_formation(clean_numbers[million_index+1:])
202     elif billion_index > -1 and billion_index != len(clean_numbers)-1:
203         hundreds = number_formation(clean_numbers[billion_index+1:])
204     elif thousand_index == -1 and million_index == -1 and billion_index == -1:
205         hundreds = number_formation(clean_numbers)
206     else:
207         hundreds = 0
208     total_sum += hundreds
209
210 # adding decimal part to total_sum (if exists)
211 if len(clean_decimal_numbers) > 0:
212     decimal_sum = get_decimal_sum(clean_decimal_numbers)
213     total_sum += decimal_sum
214
215 return total_sum
```