Piotr Kojałowicz

Personal Info

Adres

Zachodnia 20/67

53-644 Wrocław

Telefon

889 29 59 75

E-mail

piotr.kojalowicz@gmail.com

Wrocław, 06.02.2018

Hacking 101

I.

The solution sent is in section II. In section I is presented the process of creating application.

1. I started my work by writing a function that counts the power of the letter "a" in any word.

```
def count_a(word):
    count_a = word.count("a")          ### > counts how many "a" is in the word <
    i=0
    a=0
    x=1
    while i < (sum_a)*x:                ### > then counts full power of "a" in word <
        i=i+x                           ### > each subsequent repeating letter is getting stronger <
        a=a+i
    return a
```

2. Then I did the same for the letters "b" and "c.

3. Later I created a function that counts power for the phrases "baa" and "ba".

```
def count_baa_ba(word):                          ### > counts how many "baa" is in the word
    sum_baa=word.count("baa")*20                 ### and it multiplies by 20 <
    sum_ba=((word.count("ba")) - (word.count("baa")))*10   ### > then counts how many single
    return sum_baa+sum_ba                                  ### "ba" is in the word and
                                                           ### it multiplies by 10 <
```

4. At the end of this solutions I created two functions, the first to count the sum of power, the second to print the result on screen.

```
def sum(word):
    if (word.count("a") + word.count("b") + word.count("c")) == len(word):
        return (count_a(word) + count_b(word) + count_c(word) + count_baa_ba(word))
    else:                   ### > condition that the hack consists only of letters that have
        return 0            ### power if yes count the power
                            ### if not return 0 <

def hack_calculator(hack):
    print ('Hack "%s" is worth %s power.' %(hack, sum(hack))
```

CONCLUSIONS

"+":

- quickly written code,
- fast working application.

"-":

- limited number of letters and constant power,
- difficult to modify and scale the code,
- change of parameters: letters, phrases and power possible only by programmers.

II.

The second solution was to increase the functionality of the application. In this section, changes will be described in relation to the first section.

1. I started the work by modifying the function to count the power of a single letter so that it can count any letter of any power.

```
def count_letter(hack=str, letter=vars, power=int):      ### > three obiecty are taken:
    count_letter = hack.count(letter)                    ### hack, letter and power <
    i=0                                                  ### > vars "a" is converted into
    sum=0                                                ### a variable "letter" <
    while i < (count_letter)*power:                      ### > constant x is replaced with
        i=i+power                                        ### by parameter "power" <
        sum+=i
    return sum
```

2. I added here functions counting the power of a single phrase.

```
def count_phrase(hack=str, phrase=str, power=int):          ### > three obiecty are taken:
    sum=hack.count(phrase)*power                            ### hack, letter and power <
    return sum                              ### > the phrases do not increase their power
                                            ### at repetitions so there is just a multiplication <
```

3. This time, counting the sum of the power of letters and phrases turned out to be complicated, that I put them into two functions. The first simpler sums the power of letters.

```
def sum_letters(hack, letters=dict()):              ### > the function takes two obiecty:
    sum = 0                                         ### hack and dictionary of letters <
    for i in letters:
        sum += count_letter(hack, i, letters[i])    ### > a function is "count_letter "called for
    return sum                                      ### each letter, after which the powers
                                                    ### of subsequent letters are added <
```

4. In this case, the phrase turned out to be more difficult because:
a) the phrase may contain a different phrase and the greater phrase is counted, it follows from the example,
b) and additional similar phrases may appear separately.

```
def sum_phrases(hack, phrases=dict()):              ### > the function takes two obiecty:
    phrases_copy = phrases.copy()                   ### hack and dictionary of phrases <
    for i in phrases:                               ### > a) - loop that allows to calculate the power
        for j in phrases:                           ### of the longest phrases that are not part of
            if j!=i and j in i and i in hack:       ### the others, to give value 0 a short phrase <
                phrases_copy[j]=0
    sum=0
    for i in phrases_copy:
        sum+=count_phrase(hack, i, phrases_copy[i])      ### > sum of the longest phrases <
```

```
    for i in phrases:                ### > b) - loop that allows you to calculate the power
        for j in phrases:            ### of shorter phrases if they occur separately <
            if j!=i and j in i and i in hack:
                if hack.count(j)>hack.count(i):
                    sum+=phrases[j]
    return sum
```

5. At the end it was necessary to create a function which, as in the first solution, will check if the hack consists of letters with power only, if so, its power will count, if not it will grant him 0.

```
def full_power(hack=str, letters=dict(), phrases=dict()):   ### > the function takes three obiecty:
    count_letters_form_dict=0            ### hack, dictionary of letters and dictionary of phrases <
    for i in letters:                                ### > this loop is count how meny letters
        count_letters_form_dict+=hack.count(i)       ### form dictionary is in hack <
    if count_letters_form_dict == len(hack):         ### > this condition is chacking that
        return sum_letters(hack, letters) + sum_phrases(hack, phrases)    ### ...
    else:                                            ### are diffrent letters in hack that
        return 0                                     ### in letters dictionary <
```

6. The last function is again the function that sends the result to the screen.

```
def hack_calculator(hack=str, letters=dict(), phrases=dict()):
    print(' Hack %s is worth %s power.' %(hack, full_power(hack, letters, phrases)))
                                        ### > the function takes three obiecty:
                                        ### hack, dictionary of letters and dictionary of phrases <
```

<u>CONCLUSIONS</u>

"+":

- greater satisfaction of the author,
- large functionality of the application,
- easier to expand the application.

"-":

- slower work,
- more time spent on work,
- big complicated code.