ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ

Факультет программной инженерии и компьютерной техники

ЛАБОРАТОРНАЯ РАБОТА № 1, 2

ПО ДИСЦИПЛИНЕ «Технологии программирования»

Выполнили:

Юсюмбели В. И.

Плохотнюк В. С.

Группа: Р3411

Преподаватель:

Оголюк А. А.

Санкт-Петербург

2019

```
import re


def test(result, expected):
    print(result, expected, result ==
expected)


# list1.py

# 1.
# Вх: список строк, Возвр: кол-во строк
# где строка > 2 символов и первый символ ==
последнему


def me(words):
    return sum(map(lambda s: len(s) > 2 and
s[0] == s[-1], words))


print("\n1. me")
test(me(['tixt', 'xyzx', 'apple', 'xacadu',
'aabbbccca', 'a', 'aa']), 3)
test(me(['tix', 'xyz', '', 'apple',
'xacadu', 'aabbbccc', 'a', 'aa']), 0)


# 2.
# Вх: список строк, Возвр: список со строками
(упорядочено)
# за искл всех строк начинающихся с 'x', которые
попадают в начало списка.
# ['tix', 'xyz', 'apple', 'xacadu',
'aabbbccc'] -> ['xacadu', 'xyz', 'aabbbccc',
'apple', 'tix']


def fx(words):
    return sorted(filter(lambda w: w[0] ==
'x', words)) + sorted(filter(lambda w: w[0]
!= 'x', words))


print("\n2. fx")
test(fx(['tix', 'xyz', 'apple', 'xacadu',
'aabbbccc']), ['xacadu', 'xyz', 'aabbbccc',
'apple', 'tix'])
test(fx(['tix', 'apple', 'aabbbccc']),
['aabbbccc', 'apple', 'tix'])


# 3.
# Вх: список непустых кортежей,
# Возвр: список сортир по возрастанию последнего
элемента в каждом корт.
# [(1, 7), (1, 3), (3, 4, 5), (2, 2)] ->
[(2, 2), (1, 3), (3, 4, 5), (1, 7)]


def sort(lists):
    return sorted(lists, key=lambda l: l[-
1])


print("\n3. sort")
test(sort([(1, 7), (1, 3), (3, 4, 5), (2,
2)]), [(2, 2), (1, 3), (3, 4, 5), (1, 7)])
test(sort([(1,), (3,), (3, 4, 5, 3), (2, 2,
-2)]), [(2, 2, -2), (1,), (3,), (3, 4, 5,
3)])


# list2.py

# 1.
# Вх: список чисел, Возвр: список чисел, где
# повторяющиеся числа урезаны до одного
# пример [0, 2, 2, 3] returns [0, 2, 3].
def rm_adj(nums):
    return list(set(nums))


print("\n1. rm_adj")
test(rm_adj([0, 2, 2, 3]), [0, 2, 3])


# 2. Вх: Два списка упорядоченных по
возрастанию, Возвр: новый отсортированный
объединенный список


def sort_concat(list1, list2):
    lists = [list1, list2]
    result = []
    current_positions = [0, 0]
    result_length = len(list1) + len(list2)
    for i in range(1, result_length):
        choose_list =
int(lists[0][current_positions[0]] >
lists[1][current_positions[1]])


result.append(lists[choose_list][current_pos
itions[choose_list]])
        current_positions[choose_list] += 1

        if len(lists[choose_list]) <=
current_positions[choose_list]:
            result += lists[int(not
choose_list)][current_positions[int(not
choose_list)]:]
            break
    return result


print("\n2. sort_concat")
test(sort_concat([1, 2, 3], [4, 5, 6]), [1,
2, 3, 4, 5, 6])
test(sort_concat([1, 2, 3], [1, 2, 3]), [1,
1, 2, 2, 3, 3])


# string1.py

# 1.
# Входящие параметры: int <count> ,
# Результат: string в форме
# "Number of: <count>", где <count> число из
вход.парам.
#  Если число равно 10 или более, напечатать
"many"
#  вместо <count>
#  Пример: (5) -> "Number of: 5"
#  (23) -> 'Number of: many'
def num_of_items(count):
    return "Number of: " + ("many" if
int(count) >= 10 else str(count))


print("\n1. num_of_items")
test(num_of_items(10), "Number of: many")
test(num_of_items(11), "Number of: many")
test(num_of_items(2), "Number of: 2")


# 2.
# Входящие параметры: string s,
# Результат: string из 2х первых и 2х последних
символов s
# Пример 'welcome' -> 'weme'.
def start_end_symbols(s):
    return s[:2] + s[-2:]


print("\n2. start_end_symbols")
test(start_end_symbols("welcome"), "weme")
test(start_end_symbols("di"), "didi")
test(start_end_symbols("l"), "ll")
test(start_end_symbols(""), "")
```

```
# 3.
# Входящие параметры: string s,
# Результат: string где все вхождения 1го символа заменяются на '*'
# (кроме самого 1го символа)
# Пример: 'bibble' -> 'bi**le'
# s.replace(stra, strb)
def replace_char(s):
    if len(s) < 1:
        return ""
    return s.replace(s[0], '*').replace('*', s[0], 1)


print("\n3. replace_char")
test(replace_char("bibble"), "bi**le")
test(replace_char("s"), "s")
test(replace_char(""), "")


# 4
# Входящие параметры: string a и b,
# Результат: string где <a> и <b> разделены пробелом
# а превые 2 симв обоих строк заменены друг на друга
# T.e. 'max', pid' -> 'pix mad'
# 'dog', 'dinner' -> 'dig donner'
def str_mix(a, b):
    return a.replace(a[:2], b[:2], 1) + ' ' + b.replace(b[:2], a[:2], 1)


print("\n4. str_mix")
test(str_mix("max", "pid"), "pix mad")
test(str_mix("dog", "dinner"), "dig donner")
test(str_mix("d", "pi"), "pi d")


# string2.py
```

```
# 1.
# Вх: строка. Если длина > 3, добавить в конец "ing",
# если в конце нет уже "ing", иначе добавить "ly".
def v(s):
    if len(s) <= 3:
        return s

    if str(s).endswith("ing"):
        return str(s) + "ly"

    return s + "ing"


print("\n1. v")
test(v('starting'), 'startingly')
test(v('start'), 'starting')
test(v('sta'), 'sta')


# 2.
# Вх: строка. Заменить подстроку от 'not' до 'bad'. ('bad' после 'not')
# на 'good'.
# Пример: So 'This music is not so bad!' -> This music is good!

def nb(s):
    return re.sub('not.*bad', 'good', s)


print("\n2. nb")
test(nb('This music is not so bad!'), 'This music is good!')
test(nb('This music is not so bad bad!'), 'This music is good!')
test(nb('This music is not not so bad bad!'), 'This music is good!')
test(nb('This music is not  bad not so bad bad!'), 'This music is good!')
```