#1

Напишіть функцію real\_len, яка підраховує та повертає довжину рядка без наступних керівних символів: [\n, \f, \r, \t, \v]

Для перевірки правильності роботи функції real\_len їй будуть передані наступні рядки:

* 'Alex\nKdfe23\t\f\v.\r'
* 'Al\nKdfe23\t\v.\r'

def real\_len(text):

real\_len = ""

for char in text:

if char == "\n":

real\_len += ""

elif char == "\f":

real\_len += ""

elif char == "\r":

real\_len += ""

elif char == "\v":

real\_len += ""

elif char == "\t":

real\_len += ""

else:

real\_len += char

sup\_real\_len = len(real\_len)

return sup\_real\_len

print(real\_len('Alex\nKdfe23\t\f\v.\r'))

#2

Ваша компанія веде блог. Треба реалізувати функцію find\_articles для пошуку за статтями цього блогу. Є список articles\_dict, в якому міститься опис статей блогу. Кожен елемент цього списку є словником з наступними ключами: прізвища авторів - ключ 'author', назва статті - ключ 'title', рік видання - ключ 'year'.

Реалізуйте функцію find\_articles,

Параметр key функції визначає поєднання букв для пошуку. Наприклад, при key="Python" функція шукає, чи є у списку articles\_dict статті, у назві чи іменах авторів яких зустрічається це поєднання літер. Якщо такі елементи списку були знайдені, треба повернути новий список зі словників, що містять прізвища авторів, назву та рік видання всіх таких статей.

Другий ключовий параметр функції letter\_case визначає, чи треба враховувати під час пошуку регістр літер. За умовчанням він дорівнює False і регістр немає значення тобто пошук в тексті "Python" і "python" це те ж саме. Інакше потрібно шукати повний збіг.

*def* find\_articles(*key*, *letter\_case*=False):

    dict\_num = 0

    new\_list = []

    for list in articles\_dict:

        for value in list.values():

            if *letter\_case*:

                value = str(value)

                index = value.find(*key*)

                if index >= 0:

                    new\_list.append(articles\_dict[dict\_num])

            elif *letter\_case* == False:

                value = str(value)

                index = value.find(*key*)

                value\_low = value.lower()

                index\_mix = value\_low.find(*key*)

                key\_lower = *key*.lower()

                index\_low = value\_low.find(key\_lower)

                if index\_low >= 0 or index >= 0 or index\_mix >= 0:

                    new\_list.append(articles\_dict[dict\_num])

        dict\_num += 1

    print(new\_list)

find\_articles("ocean", True)

#3

Ваша компанія проводить маркетингові кампанії за допомогою SMS-розсилок. Автоматичний збір телефонних номерів із бази даних формує певний перелік. Але клієнти залишають свої номери у довільному вигляді:

" +38(050)123-32-34",

" 0503451234",

"(050)8889900",

"38050-111-22-22",

"38050 111 22 11 ",

Сервіс розсилок чудово розуміє і може відправити SMS клієнту, тільки якщо телефонний номер містить правильні цифри. Необхідно реалізувати функцію sanitize\_phone\_number, яка прийматиме рядок з телефонним номером та буде нормалізувати його, тобто. буде прибирати символи (, -, ), + та пробіли.

Результат:

380501233234

0503451234

0508889900

380501112222

38050111221

*def* sanitize\_phone\_number(*phone*):

    clear\_phone = ""

    for ch in *phone*:

        if 48 <= ord(ch) <= 57:

            clear\_phone = clear\_phone + ch

        else:

            continue

    return clear\_phone

print(sanitize\_phone\_number("    +38(050)123-32-34"))

#4

У модулі роботи з функціями ми писали функцію get\_fullname для складання повного імені користувача. Виконаємо невелике продовження для цього завдання та напишемо функцію is\_check\_name, яка приймає два параметри (fullname, first\_name) і повертає логічне значення True або False. Це результат перевірки, чи є рядок first\_name префіксом рядка fullname. Функція is\_check\_name чутлива до регістру літер, тобто "Sam" і "sam" для неї різні імена.

*def* is\_check\_name(*fullname*, *first\_name*):

    parts = *fullname*.split(" ")

    if parts[0] == *first\_name*:

        return True

    else:

        return False

#5

Напишіть функцію get\_phone\_numbers\_for\_сountries, яка буде:

1. Приймати список телефонних номерів.
2. Санітизувати (нормалізувати) отриманий список телефонів клієнтів за допомогою нашої функції sanitize\_phone\_number.
3. Сортувати телефонні номери за вказаними у таблиці країнами.
4. Повертати словник зі списками телефонних номерів для кожної країни у такому вигляді:

{

"UA": [<тут список телефонів>],

"JP": [<тут список телефонів>],

"TW": [<тут список телефонів>],

"SG": [<тут список телефонів>]

}

1. Якщо не вдалося порівняти код телефону з відомими, цей телефон повинен бути доданий до списку словника з ключем 'UA'.

*def* sanitize\_phone\_number(*phone*):

    new\_phone = (

*phone*.strip()

        .removeprefix("+")

        .replace("(", "")

        .replace(")", "")

        .replace("-", "")

        .replace(" ", "")

    )

    return new\_phone

*def* get\_phone\_numbers\_for\_countries(*list\_phones*):

    country\_list = {

        "UA": [],

        "JP": [],

        "TW": [],

        "SG": []

    }

    for numbers in *list\_phones*:

        new\_phone = str(sanitize\_phone\_number(numbers))

        if new\_phone.startswith("81"):

            country\_list["JP"].append(new\_phone)

        elif new\_phone.startswith("65"):

            country\_list["SG"].append(new\_phone)

        elif new\_phone.startswith("886"):

            country\_list["TW"].append(new\_phone)

        else:

            country\_list["UA"].append(new\_phone)

    return country\_list

#6

Розглянемо завдання на перевірку спаму в електронному листі або фільтрацію заборонених слів у повідомленні.

Нехай функція is\_spam\_words приймає рядок (параметр text), перевіряє його на вміст заборонених слів зі списку (параметр spam\_words), і повертає результат перевірки: True, якщо є хоч одне слово присутнє зі списку, та False, якщо в тексті стоп-слів не виявлено.

Слова у параметрі text можуть бути у довільному регістрі, а значить функція is\_spam\_words, при пошуку заборонених слів, [регістру незалежна](https://en.wikipedia.org/wiki/Case_sensitivity) і весь текст має зводитися до нижнього регістру. Для спрощення, будемо вважати, що в рядку є тільки одне заборонене слово.

Передбачити третій параметр функції space\_around, який за замовчуванням дорівнює False. Він відповідатиме за те, що функція шукатиме окреме слово чи ні. Слово вважати, що стоїть окремо, якщо ліворуч від слова знаходиться символ пробілу або воно розташоване на початку тексту, а праворуч від слова знаходиться пробіл або символ крапки.

Наприклад, у тексті ми шукаємо слово "лох". То в слові "Молох" виклик та результат буде наступним:

print(is\_spam\_words("Молох", ["лох"])) # True

print(is\_spam\_words("Молох", ["лох"], True)) # False

Тобто у другому випадку слово не окреме і є частиною іншого.

У цьому прикладі функція поверне True в обох випадках.

print(is\_spam\_words("Ты лох.", ["лох"])) # True

print(is\_spam\_words("Ты лох.", ["лох"], True)) # True

import re

def is\_spam\_words(text, spam\_words, space\_around=False):

for spam in spam\_words:

text = text.lower()

if space\_around == False:

dirty = re.findall(spam, text)

if dirty == None:

return False

else:

return True

if space\_around == True:

dirty = re.match(spam, text)

dirty\_0 = re.match(spam, text)

dirty\_1 = re.match(spam + " ", text)

dirty\_2 = re.match(" " + spam, text)

dirty\_3 = re.match(" " + spam + " ", text)

dirty\_4 = re.search(" " + spam + ".", text)

print(f"dirty\_0= {dirty\_0} dirty = {dirty}, dirty\_1= {dirty\_1}, dirty\_2= {dirty\_2}, dirty\_3= {dirty\_3},dirty\_4= {dirty\_4}")

if dirty == None and dirty\_0 == None and dirty\_1 == None and dirty\_2 == None and dirty\_3 == None and dirty\_4 == None:

return False

else:

return True

print(is\_spam\_words('Молох бог ужасен.', ['лох'], True))

#7

Ви вже навчилися працювати з рядками глибше і тепер у вас є повний набір інструментів для обробки рядків за допомогою Python.

За допомогою функції zip, за аналогією прикладу теорії, створіть словник TRANS для транслітерації. Створюйте словник TRANS поза функцією translate

Напишіть функцію translate, яка проводить транслітерацію кириличного алфавіту на латинську.

Функція translate:

* приймає на вхід рядок та повертає рядок;
* проводить транслітерацію кириличних символів на латиницю;

Приклад виконання:

print(translate("Дмитро Короб")) # Dmitro Korob

print(translate("Олекса Івасюк")) # Oleksa Ivasyuk

Примітка: У задачі, при створенні словника TRANS, код TRANS[ord(c.upper())] = l.title() буде вважатися неправильним, а TRANS[ord(c.upper())] = l.upper() — правильним. Це компроміс, тому що в першому випадку ми враховуємо великі літери, а в другому — правильно, якщо ім'я буде все великими літерами. Щоб не ускладнювати завдання, прийнято як у документах — все ім'я друкується великими.

CYRILLIC\_SYMBOLS = "абвгдеёжзийклмнопрстуфхцчшщъыьэюяєіїґ"

CYRILLIC\_SYMBOLS = tuple(CYRILLIC\_SYMBOLS)

TRANSLATION = ("a", "b", "v", "g", "d", "e", "e", "j", "z", "i", "j", "k", "l", "m", "n", "o", "p", "r", "s", "t", "u",

"f", "h", "ts", "ch", "sh", "sch", "", "y", "", "e", "yu", "ya", "je", "i", "ji", "g")

TRANS = {}

for c, l in zip(CYRILLIC\_SYMBOLS, TRANSLATION):

TRANS[ord(c)] = l

TRANS[ord(c.upper())] = l.upper()

def translate(name):

translated = ""

for ch in name:

ch=ch.translate(TRANS)

translated = translated + ch

return translated

#8

 минулому модулі ми працювали із системою оцінок ECTS. Напишіть функцію formatted\_grades, яка приймає на вхід словник оцінювання студентів за предмет наступного вигляду:

students = {"Nick": "A", "Olga": "B", "Mike": "FX", "Anna": "C"}

І повертає список відформатованих рядків, щоб під час виведення наступного коду:

for el in formatted\_grades(students):

print(el)

Виходила наступна таблиця:

1|Nick | A | 5

2|Olga | B | 5

3|Mike | FX | 2

4|Anna | C | 4

* перший стовпець — ширина 4 символи, вирівнювання по правому краю
* другий стовпець — ширина 10 символів, вирівнювання по лівому краю
* третій та четвертий стовпець — ширина 5 символів, вирівнювання по центру
* вертикальний символ | не входить у ширину стовпця

grades = {"A": 5, "B": 5, "C": 4, "D": 3, "E": 3, "FX": 2, "F": 1}

*def* formatted\_grades(*students*):

    completed = []

    for item, (key, value) in enumerate(*students*.items(), 1):

        completed.append("{*:>4*}|{*:<10*}|{*:^5*}|{*:^5*}".format(

            item, key, value, grades[value]))

    return completed

for el in formatted\_grades(*students*={"Nick": "A", "Olga": "B", "Mike": "FX", "Anna": "C"}):

    print(el)

#9

Попрацюємо трохи зі специфікацією у форматуванні рядків. Напишіть функцію formatted\_numbers, яка повертає список відформатованих рядків, щоб під час виведення наступного коду:

for el in formatted\_numbers():

print(el)

Виходила наступна таблиця:

| decimal | hex | binary |

|0 | 0 | 0|

|1 | 1 | 1|

|2 | 2 | 10|

|3 | 3 | 11|

|4 | 4 | 100|

|5 | 5 | 101|

|6 | 6 | 110|

|7 | 7 | 111|

|8 | 8 | 1000|

|9 | 9 | 1001|

|10 | a | 1010|

|11 | b | 1011|

|12 | c | 1100|

|13 | d | 1101|

|14 | e | 1110|

|15 | f | 1111|

* всі стовпці мають ширину 10 символів
* у заголовків таблиці вирівнювання по центру
* перший стовпець десяткових чисел — вирівнювання по лівому краю
* другий стовпець шістнадцяткових чисел — вирівнювання по центру
* третій стовпець двійкових чисел — вирівнювання з правого краю
* вертикальний символ | не входить у ширину стовпця

Як ви вже зрозуміли, функція formatted\_numbers виводить таблицю чисел від 0 до 15 у десятковому, шістнадцятковому та бінарному форматі.

*def* formatted\_numbers():

    header = "|{*:^10*}|{*:^10*}|{*:^10*}|".format("decimal", "hex", "binary")

    body = []

    body.append(header)

    for num in range(16):

        line = ('|{*:<10d*}|{*:^10x*}|{*:>10b*}|'.format(

            num, num, num))

        body.append(line)

    return body

for el in formatted\_numbers():

    print(el)

#10

Напишіть функцію find\_word, яка приймає два параметри: перший text та другий word. Функція виконує пошук зазначеного слова word у тексті text за допомогою функції search та повертає словник.

При виклику функції:

print(find\_word(

"Guido van Rossum began working on Python in the late 1980s, as a successor to the ABC programming language, and first released it in 1991 as Python 0.9.0.",

"Python"))

Результат має бути наступного виду при збігу:

{

'result': True,

'first\_index': 34,

'last\_index': 40,

'search\_string': 'Python',

'string': 'Guido van Rossum began working on Python in the late 1980s, as a successor to the ABC programming language, and first released it in 1991 as Python 0.9.0.'

}

де

* result — результат пошуку True або False
* first\_index — початкова позиція збігу
* last\_index — кінцева позиція збігу
* search\_string — частина рядка, в якому був збіг
* string — рядок, переданий у функцію

Якщо збігів не виявлено:

print(find\_word(

"Guido van Rossum began working on Python in the late 1980s, as a successor to the ABC programming language, and first released it in 1991 as Python 0.9.0.",

"Python"))

Результат:

{

'result': False,

'first\_index': None,

'last\_index': None,

'search\_string': 'python',

'string': 'Guido van Rossum began working on Python in the late 1980s, as a successor to the ABC programming language, and first released it in 1991 as Python 0.9.0.'

}

Прошло проверку

import re

def find\_word(text, word):

result = re.search(word, text)

return {'result': bool(result),

'first\_index': result.span()[0] if result else None,

'last\_index': result.span()[1] if result else None,

'search\_string': word,

'string': text}

Мой собственный результат

import re

*def* find\_word(*text*, *word*):

    dict = {"result": False, "first\_index": None, "last\_index": None}

    search = re.search(*word*, *text*)

    if search != None:

        dict["result"] = True

    nums = search.span()

    print(nums, type(nums))

    dict["first\_index"] = nums[0]

    dict["last\_index"] = nums[1]

    dict["search\_string"] = *word*

    dict["string"] = *text*

    return dict

print(find\_word(

    "Guido van Rossum began working on Python in the late 1980s, as a successor to the ABC programming language, and first released it in 1991 as Python 0.9.0.",

    "Python"))

№11

Для закріплення матеріалу напишіть функцію find\_all\_words, яка шукає збіг слова в тексті. Функція повертає список всіх знаходжень слова в параметрі word в тексті у вигляді будь-якого написання, тобто, наприклад, можливі варіанти написання слова "Python" як pYthoN, pythOn, PYTHOn і т.і. головне, щоб зберігався порядок слів, регістр не має значення.

Підказка: функції модуля re приймають ще ключовий параметр flags і ми можемо визначити нечутливість до регістру, надавши йому значення re.IGNORECASE

import re

*def* find\_all\_words(*text*, *word*):

    search = re.findall(*word*, *text*, re.IGNORECASE)

    return search

#12

У шостій задачі ми писали функцію is\_spam\_words, яка визначала, чи є чи ні стоп-слова у тексті повідомлення. Підемо далі та застосуємо функцію sub модуля re для заміни вказаних у списку стоп-слів на деякий шаблон. Наприклад, всі "погані" слова замінюватимемо зірочками. Напишіть функцію replace\_spam\_words, яка приймає рядок (параметр text), перевіряє його на вміст заборонених слів зі списку (параметр spam\_words), та повертає результат рядок, але замість заборонених слів, підставлений шаблон з \*, причому довжина шаблону дорівнює довжині забороненого слова. Визначити нечутливість до регістру стоп-слів.

import re

*def* replace\_spam\_words(*text*, *spam\_words*):

    new\_text = *text*

    for spam in *spam\_words*:

        a = "\*"\*len(spam)

        new\_text = re.sub(spam, a, new\_text, *flags*=re.IGNORECASE)

    return new\_text

print(replace\_spam\_words(

    'Guido van Rossum began working on Python in the late 1980s, as a successor to the ABC programming PYTHOn language, and first released pYthoN it in 1991 as Python 0.9.0. pythOn ', ['began', 'Python']))

#13

**ЗАВДАННЯ: ПОШУК EMAIL**

import re

*def* find\_all\_emails(*text*):

    result = re.findall(*r*"[a-zA-Z]{1}[\w\.]+@+\w+\.+\w{2,3}", *text*)

    return result

#14

## ЗАВДАННЯ: ПОШУК НОМЕРА ТЕЛЕФОНУ

шаблон якого наступний: +380(67)777-7-777 або +380(67)777-77-77

import re

*def* find\_all\_phones(*text*):

    result = re.findall("([+]\d{3}\S{7}-\d{1}-\d{3})", *text*)

    result.extend(re.findall("([+]\d{3}\S{7}-\d{2}-\d{2})", *text*))

    return result

#15

І останнє завдання на регулярні висловлювання — це пошук у тексті гіперпосилань.

Напишіть регулярний вираз для функції find\_all\_links, яка буде в тексті (параметр text) знаходити всі лінки та повертати список отриманих з тексту збігів.

З метою спрощення приймемо, що:

* Початок гіперпосилання може бути http:// або https://
* доменне ім'я — це набір латинських букв, цифр, символів підкреслення \_ та точок .
* символи точок . не можуть зустрічатися поруч

import re

*def* find\_all\_links(*text*):

    result = []

    iterator = re.finditer(

*r*"((http|https)\:\/\/)(\w+[.]\w+|\w+)\.([a-zA-Z]){2,6}", *text*)

    for match in iterator:

        result.append(match.group())

    return result