#1

Для ініціалізації свого проекту створіть допоміжну функцію do\_setup(args\_dict), яка буде викликати функцію setup з параметрами зі словника args\_dict.

Структура словника для параметра args\_dicts має бути наступною

{

"name": "useful",

"version": "1",

"description": "Very useful code",

"url": "http://github.com/dummy\_user/useful",

"author": "Flying Circus",

"author\_email": "flyingcircus@example.com",

"license": "MIT",

"packages": ["useful"],

}

Вариант, который прошел

from setuptools import setup

def do\_setup(args\_dict):

setup(\*\*args\_dict)

Вариант системы

from setuptools import setup

def do\_setup(args\_dict, requires):

setup(name=args\_dict['name'],

version=args\_dict['version'],

description=args\_dict['description'],

url=args\_dict['url'],

author=args\_dict['author'],

author\_email=args\_dict['author\_email'],

license=args\_dict['license'],

packages=args\_dict['packages'],

№2

Модифікуємо приклад попередньої задачі. Для функції do\_setup необхідно передбачити другий параметр, який буде списком залежностей.

Функція do\_setup(args\_dict, requiers) повинна викликати функцію setup з параметрами зі словника args\_dict та параметром install\_requires, який набуває значення requiers.

Структура словника для параметра args\_dicts має бути наступною

{

"name": "useful",

"version": "1",

"description": "Very useful code",

"url": "http://github.com/dummy\_user/useful",

"author": "Flying Circus",

"author\_email": "flyingcircus@example.com",

"license": "MIT",

"packages": ["useful"],

}

from setuptools import setup

def do\_setup(args\_dict, requires):

setup(name=args\_dict['name'],

version=args\_dict['version'],

description=args\_dict['description'],

url=args\_dict['url'],

author=args\_dict['author'],

author\_email=args\_dict['author\_email'],

license=args\_dict['license'],

packages=args\_dict['packages'],

install\_requires=requires)

№3

родовжуємо модифікувати приклад. Для функції do\_setup необхідно передбачити третій параметр, який буде словником, де ми можемо вказати список "точок входу" для ключа console\_scripts.

Функція do\_setup(args\_dict, requiers, entry\_points) повинна викликати функцію setup з параметрами словника args\_dict та параметром install\_requires, який набуває значення requiers. Третій параметр entry\_points приймає словник, де ми можемо вказати список "точок входу" для ключа console\_scripts.

from setuptools import setup

def do\_setup(args\_dict, requires, entry\_points):

setup(name=args\_dict['name'],

version=args\_dict['version'],

description=args\_dict['description'],

url=args\_dict['url'],

author=args\_dict['author'],

author\_email=args\_dict['author\_email'],

license=args\_dict['license'],

packages=args\_dict['packages'],

install\_requires=requires,

entry\_points=entry\_points

)

№4

Далі підуть завдання на повторення та закріплення матеріалу. Можна використовувати будь-які техніки, з якими ви зіткнулися у процесі навчання. І почнемо ми з функцій.

У Python існує рядкова функція isdigit(). Ця функція повертає True, якщо всі символи в рядку є цифрами, і є принаймні один символ, інакше — False. Напишіть функцію з ім'ям is\_integer, яка розширюватиме функціональність isdigit(). При перевірці рядка необхідно ігнорувати початкові та кінцеві прогалини в рядку. Після виключення зайвих прогалин рядок вважається таким, що представляє ціле число, якщо:

* її довжина більша або дорівнює одному символу
* вона повністю складається з цифр
* передбачити виняток, що, можливо, є початковий знак "+" або "-", після якого мають йти цифри

def is\_integer(s):

if len(s) == 0:

return False

elif len(s.strip()) == 0:

return False

elif s.strip().isdigit():

return True

elif s.strip()[0] == "+" or s.strip()[0] == "-":

return True

else:

return False

№5

Дуже часто люди у своїх повідомленнях не ставлять великі літери, особливо це стало масовим явищем в еру мобільних. пристроїв. Розробіть функцію capital\_text, яка прийматиме на вхід рядок з текстом і повертатиме рядок з відновленими великими літерами.

Функція повинна:

* зробити великою першу літеру в рядку, попри прогалини
* зробити великою першу літеру після точки, попри прогалини
* зробити великою першу літеру після знака оклику, попри прогалини
* зробити великою першу літеру після знака питання, попри прогалини
* import re
* s = 'hello world! awesome? yes.  no'
* *def* capital\_text(*s*):
* new\_s = ""
* new\_list = []
* *s* = *s*.strip()
* sptitters\_list = re.findall(*r*'[.]|[!]|[?]', *s*)
* list = re.split(*r*'[.!?]', *s*)
* if len(sptitters\_list) != len(list):
* if len(sptitters\_list) > len(list):
* num = len(sptitters\_list) - len(list)
* for i in range(num):
* list.append("")
* elif len(sptitters\_list) < len(list):
* num = len(list) - len(sptitters\_list)
* for i in range(num):
* sptitters\_list.append("")
* for sentense in list:
* new\_list.append(sentense.strip().capitalize())
* for item, conc in zip(new\_list, sptitters\_list):
* new\_s = new\_s + item + conc + " "
* print(new\_s)
* return new\_s[:-1]
* capital\_text(s)

#6

Всі ви, можливо, стикалися з ребусами "Знайди слово". Вони існують як текстові варіанти, так і як програми для мобільних додатків. Нагадаємо коротко суть ребуса. У великому квадраті з набором букв необхідно знайти слово по горизонталі та інколи по вертикалі.



Реалізуйте функцію solve\_riddle(riddle, word\_length, start\_letter, reverse=False) для знаходження слова, що шукається в рядку ребуса.

Параметри функції:

* riddle - рядок із зашифрованим словом.
* word\_length – довжина зашифрованого слова.
* start\_letter - літера, з якої починається слово (мається на увазі, що до початку слова літера не зустрічається в рядку).
* reverse - вказує, у якому порядку записане слово. За замовчуванням — в прямому. Для значення True слово зашифроване у зворотньому порядку, наприклад, у рядку 'mi1rewopret' зашифроване слово 'power'.

Функція повинна повертати перше знайдене слово. Якщо слово не знайдене, повернути пустий рядок.

riddle = 'mi1powerpret'

start\_letter = "p"

word\_length = 5

*def* solve\_riddle(*riddle*, *word\_length*, *start\_letter*, *reverse*=False):

    if *reverse* == False:

        index = *riddle*.find(*start\_letter*)

        word = *riddle*[index: index+*word\_length*]

        print(word)

        return word

    else:

*riddle* = *riddle*[::-1]

        index = *riddle*.find(*start\_letter*)

        word = *riddle*[index: index+*word\_length*]

        print(word)

        return word

solve\_riddle(riddle, word\_length, start\_letter, *reverse*=False)

#7

У четвертому модулі розв'язували завдання нормалізації даних. Розширмо її.

При аналізі даних часто виникає необхідність позбавитися екстремальних значень, перш ніж почати працювати з даними далі. Напишіть функцію data\_preparation, яка приймає набір даних, список списків (Приклад: [[1,2,3],[3,4], [5,6]]).

Функція повинна видаляти з переданих списків найбільше і найменше значення, але якщо розмір списку понад два елементи. Після видалення даних з кожного списку необхідно злити їх разом в один список, відсортувати його за зменшенням та повернути отриманий список як результат (Для прикладу вище результат буде наступним: [6, 5, 4, 3, 2]).

list\_data = [[1, 2, 3], [3, 4], [5, 6]]

*def* data\_preparation(*list\_data*):

    prepared\_list = []

    sub\_lists = len(*list\_data*)

    for element in range(sub\_lists):

        num\_of\_item = len(*list\_data*[element])

        if num\_of\_item > 2:

*list\_data*[element].remove(max(*list\_data*[element]))

*list\_data*[element].remove(min(*list\_data*[element]))

            prepared\_list.extend(*list\_data*[element])

        else:

            prepared\_list.extend(*list\_data*[element])

    prepared\_list.sort(*key*=None, *reverse*=True)

    print(prepared\_list)

data\_preparation(list\_data)

#8

Підсумкове завдання модуля два було на обчислення арифметичного виразу. У задачі на повторення ми підемо трохи іншим шляхом і виконаємо схоже завдання, одночасно закріпивши знання роботи зі рядками та списками. Розбиття рядка на лексеми є процес перетворення вихідного рядка в список з підрядків, званих лексемами (token).

В арифметичному виразі лексемами є: оператори, числа та дужки. Операторами у нас будуть такі символи: \*, /, - та +. Оператори та дужки легко виділити у рядку — вони складаються з одного символу і ніколи не є частиною інших лексем. Числа виділити складніше, оскільки ці лексеми можуть складатися з кількох символів. Тому будь-яка безперервна послідовність цифр — це одна числова лексема.

Напишіть функцію, яка приймає параметр рядок, що містить математичний вираз, і перетворює його в список лексем. Кожна лексема має бути або оператором, або числом, або дужкою.

Приклад:

"2+ 34-5 \* 3" => ['2', '+', '34', '-', '5', '\*', '3']

З метою спрощення вважаємо, що числа можуть бути тільки цілими, і вхідний рядок завжди міститиме математичний вираз, що складається з дужок, чисел та операторів.

Зверніть увагу, що лексеми можуть відокремлюватися один від одного різною кількістю прогалин, а можуть і не відокремлюватися зовсім. Прогалини не є лексемами та до підсумкового списку потрапити не повинні.

s = "2+ 34-5 \* 3"  # = > ['2', '+', '34', '-', '5', '\*', '3']

*def* token\_parser(*s*):

    l = len(*s*)

    integ = []

    i = 0

    while i < l:

        s\_int = ''

        a = *s*[i]

        while '0' <= a <= '9':

            s\_int += a

            i += 1

            if i < l:

                a = *s*[i]

            else:

                break

        i += 1

        if s\_int != '':

            integ.append(s\_int)

        if a in "()+-/\*":

            integ.append(a)

    print(integ)

token\_parser(s)

#9

**ЗАВДАННЯ: РОБОТА З ВКЛАДЕНИМИ СПИСКАМИ**

**ЗАВДАННЯ**

Підсписком (sublist) називають список, що є складовою більшого списку. Підсписок може містити один елемент, множину елементів або бути порожнім.

Наприклад, [1], [2], [3] та [4] є підписками списку [1, 2, 3, 4]. Список [2, 3] також входить до складу [1, 2, 3, 4], але при цьому список [2, 4] не є підсписком [1, 2, 3, 4], оскільки у вихідному списку числа 2 і 4 не є сусідами.

Порожній список є підсписком будь-якого списку.

Напишіть функцію all\_sub\_lists, що повертає список, який містить всі можливі підписки заданого.

Наприклад, якщо функції передано аргумент список [1, 2, 3], то функція має повернути наступний список: [[], [1], [2], [3], [1, 2], [2, 3], [1, 2, 3]].

Функція all\_sub\_lists повинна повертати щонайменше один список з порожнім підписом [[]].

data = [4, 6, 1, 3]

*def* all\_sub\_lists(*data*):

    lists = [[]]

    for i in range(len(*data*) + 1):

        for j in range(i):

            lists.append(*data*[j: i])

    print(lists)

    lists.sort(*key*=(len))

    print(lists)

all\_sub\_lists(data)

#10

При роботі веб-сервісів спілкування, за протоколом HTTP, найчастіше відбувається в форматі [JSON](https://ua.wikipedia.org/wiki/JSON). І надсилання даних на сервер при запиті POST - це необхідність використовувати словник, оскільки структура формату JSON ідентична словнику Python.

Реалізуйте допоміжну функцію, яка формуватиме запит на сервер у вигляді словника. Дана функція make\_request(keys, values) приймає два параметри у вигляді списків. Функція повинна створити словник із ключами з списку keys та значеннями зі списку values.

* Порядок відповідності збігається з індексами списків keys та values.
* Якщо довжина keys та values не збігаються, поверніть порожній словник.

keys = ['Ivan', 'Petro', 'Oksana', 'Iryna', 'Stas']

values = [20, 35, 16, 18, 34, 44]

# new\_dict = {user: age for (user, age) in zip(users, ages)}

# print(new\_dict)

*def* make\_request(*keys*, *values*):

    if len(*keys*) == len(*values*):

        dict = {key: value for (key, value) in zip(*keys*, *values*)}

    else:

        dict = {}

    print(dict)

make\_request(keys, values)

#11

Як ви знаєте, раніше телефони були з кнопками, та й зараз вони є подекуди у вжитку. Тоді текстові повідомлення набиралися за допомогою цифрові кнопки. Як інженери телефонів створили набір тексту? Рішення було в тому, що одна кнопка була асоційована одночасно з декількома літерами, а вибір залежав від кількості натискань на кнопку. Одноразове натискання призводило до появи першої літери у відповідному цій кнопці списку, наступні натискання змінювали її на наступну.

Символи, що відповідають кнопкам на телефонах

| **Кнопка** | **Символи** |
| --- | --- |
| 1 | . , ? ! : |
| 2 | A B C |
| 3 | D E F |
| 4 | G H I |
| 5 | J K L |
| 6 | M N O |
| 7 | P Q R S |
| 8 | T U V |
| 9 | W X Y Z |
| 0 | Пробіл |

Напишіть функцію sequence\_buttons, що показує послідовність кнопок, яку необхідно натиснути, щоб на екрані телефону з'явився текст, введений користувачем.

Створіть словник, який відповідає символам з кнопками, які потрібно натиснути.

Приклад: якщо функції sequence\_buttons передати рядок "Hello, World!", функція повинна повернути "4433555555666110966677755531111".

Вимоги:

* функція коректно обробляє малі та великі літери.
* функція ігнорує символи, що не входять до зазначеного списку

*def* sequence\_buttons(*string*):

    characters = ".,?!:abcdefghijklmnopqrstuvwxyz "

    characters = tuple(characters)

    buttons = ("1", "11", "111", "1111", "11111", "2", "22", "222", "3", "33", "333",

               "4", "44", "444", "5", "55", "555", "6", "66", "666", "7", "77", "777", "7777", "8", "88", "888", "9", "99", "999", "9999", "0")

    TRANS = {}

    for char, but in zip(characters, buttons):

        TRANS[ord(char)] = but

        TRANS[ord(char.upper())] = but.upper()

    translated\_name = ""

    for ch in *string*:

        ch = ch.translate(TRANS)

        translated\_name = translated\_name + ch

    print(translated\_name)

sequence\_buttons("Hello, World!")

#12

Реалізуйте функцію file\_operations(path, additional\_info, start\_pos, count\_chars), яка додає додаткову інформацію в файл на шляху path з параметра additional\_info, і після цього повертає рядок з позиції start\_pos довжиною count\_chars.

Вимоги:

* функція повинна відкривати файл за допомогою with за шляхом path в режимі додавання інформації
* записувати в кінець файлу рядок additional\_info
* після запису функція має відкрити той самий файл для читання
* прочитати та повернути рядок з позиції start\_pos завдовжки count\_chars за допомогою функції seek.

Важливо: для проходження завдання необхідно використовувати менеджер контексту with, методи seek, write та read.

def file\_operations(path, additional\_info, start\_pos, count\_chars):

with open(path, 'a') as fh:

fh.write(additional\_info)

with open(path, 'r') as fh:

fh.seek(start\_pos)

res = fh.read(count\_chars)

return res

#13

Є файл зі списком працівників компанії. У кожному рядку файлу записано інформацію лише одного співробітника. Формат запису, в межах навчання приймемо спрощений, такий як: ім'я співробітника, символ пропуску та його посада, тобто, ким він працює.

John courier

Pipe cook

Створіть функцію get\_employees\_by\_profession(path, profession). Функція повинна у файлі (параметр path) знайти всіх співробітників зазначеної професії (параметр profession)

Вимоги:

* відкрийте файл за допомогою with для читання
* отримайте рядки з файлу за допомогою методу readlines()
* за допомогою методу find знайдіть усі рядки у файлі, де є вказана profession, та помістіть записи до списку
* об'єднайте всі ці рядки в списку в один рядок за допомогою методу join (пам'ятайте про переклад рядків '\n' та зайві прогалини, які треба прибрати)
* приберіть значення змінної 'profession' (замініть на порожній рядок "" методом replace)
* поверніть отриманий рядок із файлу

def get\_employees\_by\_profession(path, profession):

list = []

with open(path, 'r') as fh:

while True:

line = fh.readline()

if profession in line:

clear\_line = line.replace(profession, "").replace("\n", "").strip()

list.append(clear\_line)

if not line:

break

string = " ".join(list)

print(string)

return string

#14

## ЗМІНА ФАЙЛІВ. ПОВТОРЕННЯ

### ЗАВДАННЯ

Напишіть функцію to\_indexed(start\_file, end\_file), яка зчитуватиме вміст файлу, додаватиме до прочитаних рядків порядковий номер і зберігати їх у такому вигляді у новому файлі.

Кожний рядок у створеному файлі повинен починатися з його номера, двокрапки та пробілу, після чого має йти текст рядка з вхідного файлу.

Нумерація рядків іде від 0.

Не прошел, хотя и выглядит правильно:

# def to\_indexed(source\_file, output\_file):

#     lines = []

#     with open(source\_file, 'r') as fh:

#         while True:

#             line = fh.readline().strip()

#             print(line + "!!!")

#             lines.append(line)

#             if not line:

#                 break

#     with open(output\_file, 'w') as fh:

#         for num, line in enumerate(lines[0:-1]):

#             fh.writelines('{}: {}'.format(num, line))

#             print('{}: {}'.format(num, line))

Прошел, ответ от ментора:  
*def* to\_indexed(*source\_file*, *output\_file*):

    with open(*source\_file*, 'r') as f:

        lines = f.readlines()

    new\_lines = []

    for i in range(0, len(lines)):

        new\_lines.append(str(i) + ': ' + lines[i])

    with open(*output\_file*, 'w') as f1:

        for i in new\_lines:

            f1.write(i)

№15

Рекурсія добре підходить до задачі flatentening. Це процес вирівнювання списків, який полягає в позбавленні вкладеної структури. Наприклад список вигляду [1, 2, [3, 4, [5, 6]], 7] має бути перетворений на плоский (flat) список [1, 2, 3, 4, 5, 6, 7]

Напишіть функцію flatten, яка приймає на вхід список, рекурсивно вирівнюватиме цей список і повертатиме пласку версію списку.

Для виконання завдання можна дотримуватися наступного алгоритму:

Якщо вхідний список порожній, то:

Повертаємо порожній список

Якщо перший елемент списку є списком, то:

Отримуємо перший список, рекурсивно викликавши функцію з першим елементом списку

Отримуємо другий список, рекурсивно викликавши функцію з рештою списку без першого елемента

Повертаємо конкатенацію двох списків

Якщо перший елемент списку не є списком, то:

Отримуємо перший список із першого елемента списку

Отримуємо другий список, рекурсивно викликавши функцію з рештою списку без першого елемента

Повертаємо конкатенацію двох списків

def flatten(data):

if data == []:

return data

if isinstance(data[0], list):

return flatten(data[0]) + flatten(data[1:])

return data[:1] + flatten(data[1:])

#16

Розберемо просту техніку кодування та декодування на основі серій. Наприклад, у нас є список із повторюваними спостереженнями якоїсь величини, вона може приймати значення X, Y або Z. Значення з'являються у довільному порядку та зберігаємо ми їх у списку ["X", "X", "X", "Z", "Z", "X", "X", "Y", "Y", "Y", "Z", "Z"] або рядку "XXXZZXXYYYZZ".

Внаслідок чого ми можемо зменшити обсяг інформації, що зберігається? Стиснення можна досягти заміною групи повторюваних значень на одноразове значення та лічильник повторів: ["X", 3, "Z", 2, "X", 2, "Y", 3, "Z", 2]

Напишіть рекурсивну функцію decode для декодування списку, закодованого цим способом. Як аргумент функція приймає закодований список. Функція має повернути розшифрований список елементів.

*def* decode(*data*):

    output = ""

*def* \_decode(*data*, *output*):

        while *data*:

*output* = *output* + *data*[0] \* *data*[1]

            del *data*[0]

            del *data*[0]

*output* = \_decode(*data*, *output*)

        if len(*data*) == 0:

            return *output*

        return *output*

    output\_recurs = \_decode(*data*, output)

    print(list(output\_recurs))

    return list(output\_recurs)

data = ["X", 3, "Z", 2, "X", 2, "Y", 3, "Z", 4, "W", 4]

decode(data)

№17

Повернемося до попереднього завдання та виконаємо зворотне. Напишіть рекурсивну функцію encode для кодування списку або рядка. Як аргумент функція приймає список ( наприклад ["X", "X", "X", "Z", "Z", "X", "X", "Y", "Y", "Y", "Z", "Z" ]) або рядок (наприклад, "XXXZZXXYYYZZ"). Функція повинна повернути закодований список елементів (наприклад ["X", 3, "Z", 2, "X", 2, "Y", 3, "Z", 2]).

*def* encode(*data*):

*def* \_encode(*data*):

*data* = "".join(*data*)

        if not *data*:

            return ""

        first = *data*[0]

        counter = 0

        for letter in *data*:

            if letter != first:

                break

            counter += 1

        result = *f*"{first}{counter}" + \_encode(*data*[counter:])

        # result = list(result)

        return result

    res = list(\_encode(*data*))

    output = []

    for item in res:

        if item.isdigit():

            output.append(int(item))

        else:

            output.append(item)

    return output