МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

С.Ю. Борю

ПРОГРАМУВАННЯ: PYTHON 3.X – ПРАКТИЧНИЙ Q&A

Збірник задач і вправ для здобувачів ступеня вищої освіти бакалавра спеціальності «Комп'ютерні науки» освітньо-професійної програми «Комп'ютерні науки»

Затверджено
вченою радою ЗНУ
Протокол № від

Запоріжжя
2025

УДК 004 (076)
Б68

Борю С.Ю. Програмування: Python 3.x – практичний Q&A : збірник задач і вправ для здобувачів ступеня вищої освіти бакалавр спеціальності «Комп'ютерні науки» освітньо-професійної програми «Комп'ютерні науки» Запоріжжя : Запорізький національний університет, 2025, 43 с.

Збірник задач і вправ розроблений для здобувачів ступеня вищої освіти бакалавра спеціальності «Комп'ютерні науки» та є допоміжним матеріалом у вивченні дисципліни «Програмування». Збірник має формат «питання-відповідь» (Q&A), що дозволяє студентам ефективно закріпити теоретичні знання та одразу перейти до їх практичного застосування.

Основна увага приділена розгляду типових проблемних ситуацій та поширених питань про особливості функціонування мови Python – від управління змінними та типами даних до роботи з функціями вищого порядку та концепціями об'єктно-орієнтованого програмування. Видання допоможе студентам не лише писати працездатний код, але й розуміти механізми його роботи, уникати поширених помилок.

Збірник задач і вправ «Програмування: Python 3.x – практичний Q&A» призначений для студентів спеціальності «Комп'ютерні науки» освітньо-професійної програми «Комп'ютерні науки», але представлене видання може бути корисним студентам різних спеціальностей.

Рецензент
Н. В. Матвійшина, кандидат технічних наук, доцент кафедри комп'ютерних наук.

Відповідальний за випуск
О. С. Пшенична, кандидат педагогічних наук, в. о. завідувача кафедри комп'ютерних наук.

ВСТУП

Дисципліна «Програмування» належить до обов’язкових дисциплін циклу професійної підготовки.

У межах дисципліни «Програмування» вивчаються поняття процедурних мов програмування, інформаційних об'єктів програм, операторів, процедур, стандартних бібліотек підпрограм а також основні технологічні методи практичного застосування мовних засобів програмування для розробки програмного продукту, призначеного для практичного розв’язання задач інформатики та математики.

Зокрема, розглядається одна із сучасних і дуже популярних нині – система програмування Python. Вивчення мови програмування та поведінки інтерпретатора системи програмування досить часто викликає ряд питань у програміста.

У збірнику задач і вправ розглядаються питання, що часто задаються з програмування мовою Python і наводяться ґрунтовні відповіді на них.

Згідно з вимогами освітньої-професійної програми здобувачі освіти мають досягти таких компетентностей та програмних результатів навчання:

– здатність до абстрактного мислення, аналізу та синтезу;

– здатність застосовувати знання у практичних ситуаціях;

– знання та розуміння предметної області та розуміння професійної діяльності;

– здатність вчитися і оволодівати сучасними знаннями;

– здатність до логічного мислення, побудови логічних висновків, використання формальних мов і моделей алгоритмічних обчислень, проектування, розроблення й аналізу алгоритмів, оцінювання їх ефективності та складності, розв’язності та нерозв’язності алгоритмічних проблем для адекватного моделювання предметних областей і створення програмних та інформаційних систем;

– здатність проектувати та розробляти програмне забезпечення із застосуванням різних парадигм програмування: узагальненого, об’єктно-орієнтованого, функціонального, логічного, з відповідними моделями, методами й алгоритмами обчислень, структурами даних і механізмами управління;

– застосовувати знання основних форм і законів абстрактно-логічного мислення, основ методології наукового пізнання, форм і методів вилучення, аналізу, обробки та синтезу інформації в предметній області комп'ютерних наук;

– використовувати сучасний математичний апарат неперервного та дискретного аналізу, лінійної алгебри, аналітичної геометрії, в професійній діяльності для розв’язання задач теоретичного та прикладного характеру в процесі проектування та реалізації об’єктів інформатизації;

– проектувати, розробляти та аналізувати алгоритми розв’язання обчислювальних та логічних задач, оцінювати ефективність та складність

алгоритмів на основі застосування формальних моделей алгоритмів та обчислюваних функцій;

– розробляти програмні моделі предметних середовищ, вибирати парадигму програмування з позицій зручності та якості застосування для реалізації методів та алгоритмів розв’язання задач в галузі комп’ютерних наук.

Збірник задач і вправ «Програмування: Python 3.x – практичний Q&A» призначений для студентів спеціальності «Комп’ютерні науки» освітньо-професійної програми «Комп’ютерні науки», але представлене видання може бути корисним студентам різних спеціальностей.

ПРАКТИЧНА РЕАЛІЗАЦІЯ ЧАСТО ЗАСТОСОВУВАНИХ АЛГОРИТМІВ

У процесі вивчення нової системи програмування, особливо такої як інтерпретатор мови Python, часто виникає низка стандартних питань практичної реалізації часто застосовуваних алгоритмів.

Система програмування Python дуже специфічна та суттєво відрізняється від класичних процедурних мов. У зв'язку з цим при початковому вивченні цієї системи програмування часто виникає низка питань – як вирішити і реалізувати деякі завдання, що часто повторюються.

Нижче розглядаються питання, які часто виникають при практичному написанні коду мовою Python. Наведено методику вирішення цих питань та приклади програм.

1. Як помістити в numpy масив матрицю, значення якої знаходяться у файлі

```
import sys

import numpy as np
filename="mat_dat.txt"
try:
    f1=open(filename, "r") # , encoding='UTF-8')
except IOError:
  print ("No file ", filename)
  sys.exit(4)
try:
  txt=f1.readline()[:-1]
except IOError:
  print    ("помилка    структури    файлу    {0}:    "        \
        "1-а запис не знайдено...".format(filename) )
  sys.exit(8)
try:
  w= f1.readline().split()
except IOError:
  print    ("помилка    структури    файлу    {0}:    "        \
        "2-а запис не знайдено...".format(filename) )
  sys.exit(8)
if (len(w) < 2 ):
   print    ("помилка    структури    файлу    {0}:    "    \
"2-а    запис    не    містить    значень    розмірності    "    \
 " матриці...".format(filename) )
   sys.exit(8)
try:
  n= int( w[0])
  m= int( w[1])
```

```
    except ValueError:
       print     ("помилка     структури     файлу     {0}:     "     \
    "2-а  запис  не  містить  коректних  значень  розмірності  "\
      "матриці...".format(filename) )
       sys.exit(8)

    mat=np.zeros((n,m), float)

    print("З          фалу          ",          filename,          \
         " зчитуються матрица розміром [",n,",",m,"]")

    for i in range(0,n):
        nstr=i+3
        try:
            w= f1.readline().split()
            #print(w)
        except IOError:
            print     ("помилка     структури     файлу     {0}:     \"
          "{1}-а       запис       не       знайдено......"       \
              .format(filename, nstr) )
            sys.exit(8)
        try:
            for j in range(m):
                mat[i,j]=float(w[j])
        except ValueError:
            print    ("помилка    структури    файлу    {0}:    "    \
          "{1}-а   запис   в   {2}   позиції   помилка   запису   "   \
          "      числового      значення      матиці..."             \
              .format(filename, nstr, j+1) )
            sys.exit(8)
        except IndexError:
            print     ("помилка     структури     файлу     {0}:"     \
          "   {1}-а   запис   в   {2}   позиції   не   містить   "   \
"                           числового    значення    матиці..."    \
         .format(filename, nstr, j+1) )
    f1.close()

    print(mat)
```

Файл із значеннями елементів матриці має формат:

```
Матриця для вирішення задач .....
5 5       розмірність матриці
0.1 0.2 0.3 0.4 0.5
0.5 0.6  0.7  0.8 0.9
1 2 3 4 5
10 20 30 40  50
100 200 300  400 500
```

Завдання для самостійного вирішення.

1. Напишіть функцію створення матриці та читання значень її елементів із заданого файлу. Структура файлу описана вище – у прикладі.
2. Напишіть функцію запису значень елементів матриці в заданий файл. Структура файлу описана вище – у прикладі.
3. Напишіть програму, яка використовує розроблені функції, яка виконує множення двох заданих матриць. Значення елементів матриць розміщені у файлах. Значення елементів результуючої матриці також записати у файл.

2. Як «перехопити» всі помилки (RuntimeWarning і Exception) під час виконання функції `eval`

Приклад 1. Розробимо функцію, яка при першому виклику запитує з консолі функцію користувача, потім розраховує значення функції від аргументу x. При наступних запитах – розраховує значення функції від аргументу x. У разі помилки – видає повідомлення про помилку та запитує іншу функцію користувача. Використовуються глобальні змінні `_flag_fuser, _fu_fuser`

```
import sys

import warnings

_flag_fuser=True
_fu_fuser=''
def fuser(x):
    global _flag_fuser, _fu_fuser
    if (_flag_fuser):
        print("введіть f(x)=", end='')
        _fu_fuser=input()
        _flag_fuser=False
    _z=0.0
    with warnings.catch_warnings():
        warnings.filterwarnings('error')
        try:
                _z=eval(_fu_fuser)
                print('     "{0:s}"({1:.3f})={2:.3f}     '     \
                            .format(_fu_fuser, x, _z))
        except Exception as e:
            #print('e=',e)
            print('помилка',  e,  '\ прозрахунку  значень  `  \
                      ` введенной функции="',  \
                    _fu_fuser,'"(',x,')', sep='')
                _flag_fuser=True
                fuser(x)
    #print('_z=',_z)
```

```
    return _z

xx=0.0

while (xx != 666):
    xx=float(input('x='))
    zzz=fuser(xx)
    print('zzz=',zzz)

sys.exit(0)
```

Приклад 2. Розробити функцію `user(xxx)`, яка приймає на вхід одновимірний масив значень (`numpy array`) `xxx`. Ця функція повинна запитувати з консолі функцію користувача `f(x)` у вигляді рядка. Після успішного введення функція повинна розрахувати значення `f(x)` для кожного елемента вхідного масиву `xxx`, використовуючи змінну `x` як поточний аргумент. У разі виникнення будь-якої помилки під час обчислення (синтаксична помилка у виразі, математична помилка, або недійсне значення `numpy`), функція має видати детальне повідомлення про помилку та повторно запитати іншу функцію користувача з консолі.

```
import sys
import numpy as np

def user(xxx):
    '''
повертає numpy масив значень функції користувача від аргументів, що передаються як numpy вхідний масив xxx функція користувача запитується з консолі у разі помилки - видає повідомлення про помилку та запитує іншу функцію користувача
    '''

    #print("введите f(x)=", end='')
    Flag=True
    while True:
        print("введіть f(x)=", end='')
        fu=input() #"введите f(x)=")
        z=np.empty(xxx.size,dtype=float)
        for i in range(0,len(xxx)):
            x=xxx[i]
            with warnings.catch_warnings():
                warnings.filterwarnings('error')
                try:
                    z[i]=eval(fu)
                    #print(i,z[i])
                except Exception as e:
                    #print('e=',e)
                    print("помилка", e,  \
                    "\nрозрахунку       значень       "       \
                " функции f(x)='",fu,"'")
```

```
                Flag=False
                break

        if (Flag):
            break
        Flag=True
    #print(z)
    return z
```

Варіант використання для побудови графіка, введених користувачем функцій:

```
import matplotlib as mpl
import numpy as np
x = np.linspace(-2, 2, 1000)
plt.plot(x, user(x), label='користувача 1')
plt.plot(x, user(x), label=' користувача 2')

plt.grid(True) # Сітка
plt.xlabel('x label')
plt.ylabel('y label')
plt.title("Simple Plot")
plt.legend()
plt.savefig('pic_5.pdf', fmt='pdg')

plt.show()
```

Завдання для самостійного вирішення.

Розробити функцію `user1(xxx, fun_str)`, яка приймає на вхід одновимірний масив значень (`numpy array) xxx` та текстовий рядок `fun_str` - запис формули, за якою виконується розрахунок значення функції. Для запису аргументу використовувати ім'я x. Приклад x**2+12*x-25.

Функція повинна розрахувати значення `f(x)` для кожного елемента вхідного масиву `xxx`, використовуючи змінну `x` як поточний аргумент. У разі виникнення будь-якої помилки під час обчислення (синтаксична помилка у виразі, математична помилка, або недійсне значення `numpy`), функція має видати детальне повідомлення про помилку та завершити роботу.

3. Яку версію Python встановлено

Використовуємо пакет `sys`:

```
import sys
```

```
print( 'Версія Python {}.{}.{} на платформі {}'. \
        format(            sys.version_info.major,            \
    sys.version_info.minor, \
        sys.version_info.micro, sys.platform ) )
```

Завдання для самостійного вирішення.

Розробите функцію, яка повертає список текстових рядків, що містять інформацію про версію інтерпретатора, що використовується, а саме:

Основний (мажорний) номер версії

Другорядний (мінорний) номер версії

Мікро (патч) номер версії

Ідентифікатор платформи, на якій виконується інтерпретатор Python.

Виконайте тестування цієї функції на різних версіях та платформах.

4. Приклад отримання списку слів, поділених заданими роздільниками

```
# з вхідного рядка str отримуємо list слів,
# розділених символами роздільниками з рядка raz
def mysplit(str, raz=' \n\r\t.;,?!\t'):
    import re
    __r="["
    for __b in raz:
        __r=__r + __b + "|"
        #print(__b)
    __r=__r[:-1]+"]"
    __split_regex = re.compile(__r) ###r'[.|!|?|…]')
    __sentences =                          \
            filter(lambda t: t,            \
            [t.strip()                     \
            for t in                       \
            __split_regex.split(str)])
    __w1=[]
    for __s in __sentences:
        __w1.append(__s)
    return __w1

# текст для разбора на слова
txt='123,,,345;;123\n\n\r\r;\txxx   \t ' \
    ' апрол   iЇ   abc!!  .ccc..апрол     abc sds  ёЁ. ' \
    ' привет! куку? привет....ааа  ббб  ввв\n!!!ггг ддд '

print(txt)
# список w слів роздільники ' \n\r\t.;,?!\t’
```

```
    w=mysplit(txt,' \n\r\t.;,?!\t')

    # друк списку
    print(w)
```

Ще один варіант:

```
def toster_word_split(s, SEPARATORS=' \n\r\t.;,?!\t'):
    result=[]
    current_word=''
    for char in s:
        if char in SEPARATORS:
            if current_word:
                result.append(current_word)
                current_word=''
        else:
            current_word+=char

    if current_word:
        result.append(current_word)
    return result

txt='  111 222 3333... ;;;43444   555  2222....@'
w1=toster_word_split(txt)
print(w1)

вывод:
['111', '222', '3333', '43444', '555', '2222', '@']
```

Завдання для самостійного вирішення.

Напишіть програму, яка за заданим ім'ям текстового файлу виводить на консоль кількість слів, що складаються із заданої кількості літер.

5. Програма підрахунку слів у файлі

```
#! Програма підрахунку слів у файлі
import os
import sys

def get_words(filename):

    with open(filename, encoding="utf8") as file:
        text = file.read()
    text = text.replace("\n", " ")
    text = text.replace(",", " "). \
               replace(".", " "). \
               replace("?", " "). \
```

```
                replace("!", " "). \
                replace("'"," "). \
                replace('"'," "). \
                replace('(',' '). \
                replace(")", " "). \
                replace("%", " "). \
                replace(":", " "). \
                replace("\\n", " ")
# і так далі……
    text = text.lower()
    words = text.split()
    words.sort()
    return words


def get_words_dict(words):
    words_dict = dict()

    for word in words:
        if word in words_dict:
            words_dict[word] = words_dict[word] + 1
        else:
            words_dict[word] = 1
    return words_dict


#######################
dirName=input("Введіть шлях до файлу(Enter-тек. дір): ")
if len(dirName) == 0:
    dirName=os.getcwd()

if not os.path.isdir(dirName):
    print("Помилка-директорія ", dirName,  \
                                  "ненайдена.....")
    sys.exit(8)

names = os.listdir(dirName)
print("поточна директорія: ",dirName, \
                                 "\n список файлов:")
for name in names:
    print(name)
while True:
    filename = input("Введіть ім'я файлу (Enter-stop): ")
    if len(filename) == 0:
       sys.exit(0)
    if not os.path.exists(filename):
       print("Помилка - файл ", filename, \
                                   " не знайден.....")
       continue
    else:
        break

# отримуємо повне ім'я
```

```
fullname = os.path.join(dirName, filename)
print("Обробка файлу ", fullname)
words = get_words(filename)
words_dict = get_words_dict(words)
print("Кількість слів: %d" % len(words))
print("Унікальні слова: %d" % len(words_dict))
print("Всі слова:")
for word in words_dict:
    print(word.ljust(20), words_dict[word])
```

Завдання для самостійного вирішення.

Напишіть програму, яка за заданим ім'ям текстового файлу виводить на консоль кількість унікальних слів у файлі, що містять більше трьох літер. Програма має також виводити ці слова.

6. Приклад підрахунку кількості слів у тексті та виведення впорядкованої таблиці частоти слів

```
# у словник dic поміщаємо
# <key> : <+1 до старого значення>

def toD(dic, key):
    if key in dic:
        dic[key]+=1
    else:
        dic[key]=1


# текст для розбору на слова, розділені пробілом.

txt=''' 123
345
123
апрол
іЇ
abc
ссс
апрол
abc
sds
єЄ
привет
куку
привет
ааа
ббб
ввв
```

```
    ггг
    ддд
    '''

    print (txt.split())     #  друк усіх слів тексту

    D={}                    # словник частоти слів

    for t in txt.split():  # формуємо словник

        toD(D,t)

    print(D) # друк словника

    # список D1 зі словника та його
    # упорядкування за ключом словника
    # Якщо хочемо за значенням, пишіть key=lambda x:x[1]
    D1=sorted(D.items(),key=lambda x:x[0]) # по ключам
    print(D1)   # друк списку

    # відсортований список у словник D2 та його друк
    D2={D1[i][0] : D1[i][1] for i in range(len(D1)) }
    print(D2)
#друк таблицею D2
for kk, vv in D2.items():
    print(kk,'\t',vv)
```

Завдання для самостійного вирішення.

Напишіть програму, яка за заданим ім'ям текстового файлу виводить на консоль кількість слів у файлі, а також список слів файлу та їх кількість. Програма повинна вивести слово, яке найчастіше використовується і слово, яке використовується найрідше.

7. Приклад підрахунку частоти літер у тексті та виведення впорядкованої таблиці частоти літер

```
# в txt знаходиться текст

def toD(dic, key):
'''
у словник dic поміщаємо
<key> : <+1 до старого значення>
'''
if key in dic:
  dic[key]+=1
else:
```

```
  dic[key]=1
D={} # словник частоти літер
kword=0
for t in txt: # формуємо словник
  toD(D,t)
  kword+=1
print(D) # друк словника
# список D1 зі словника та його упорядкування
# за ключом словника
# Якщо хочемо за значенням, пишіть key=lambda x:x[1]
D1=sorted(D.items(),key=lambda x:x[0]) # за ключами
print(D1) # друк списку
# відсортований список у словник D2 та його друк
D2={D1[i][0] : D1[i][1] for i in range(len(D1)) }
print(D2)
#друк таблицею
for kk, vv in D2.items():
    if (kk == '\n'):
        kk=r'\n'
    elif (kk == '\r'):
        kk=r'\r'
    elif (kk == '\t'):
        kk=r'\t'
    elif (kk == ' '):
        kk=r'bl'
    else:
        kk=' '+kk
    print('{0:2s}\t==> {1:5d}\t' \
        '{2:6.2f}%'.format(kk,vv,float(vv/kword)*100) )
```

Завдання для самостійного вирішення.

Напишіть програму, яка за заданим ім'ям текстового файлу виводить на консоль кількість не пробільних букв у файлі (розділові знаки та інші роздільники слів так само не враховувати). Програма повинна вивести літеру, яка використовується найчастіше і літеру, яка використовується найрідше.

8. Перевірка наявності файлу чи каталогу за вказаним шляхом

Потрібно перевірити коректність введеної користувачем адреси файлу чи каталогу. Зробити це можна за допомогою функції `os.path.exists`, яка повертає `true`, якщо об'єкт файлової системи існує і `false` – якщо ні.

Функція `os.path.isfile` перевіряє, чи об'єкт є файлом, а `os.path.isdir` – чи є каталогом.

У наведеному нижче скрипті перевіряється наявність об'єкта за вказаною користувачем адресою, після чого перевіряється файл або каталог. Залежно від виду об'єкта виводиться та чи інша інформація.

```
# Скрипт перевіряє наявність шляху.
# Якщо файл, то виводить його розмір, дати створення,
# відкриття та модифікації.

# Якщо каталог, виводить список вкладених файлів
# і каталогів.

import os
import datetime

testpath = input(' Введіть адресу: ')

if os.path.exists(testpath):
    if os.path.isfile(testpath):
        print('ФАЙЛ')
        print('Розмір:',
               os.path.getsize(testpath)//1024,'Кб')
        print('Дата створення:',\
              datetime.datetime.  \
                   fromtimestamp(  \
                    int(os.path.getctime(testpath))))
        print('Дата останнього відкриття:',\
              datetime.datetime.fromtimestamp( \
                   int(os.path.getatime(testpath))))
        print(Дата останньої зміни:',\
              datetime.datetime.fromtimestamp( \
                    int(os.path.getmtime(testpath))))
    elif os.path.isdir(testpath):
        print('КАТАЛОГ')
        print(' Список об'єктів у ньому:', \
                        os.listdir(testpath))
else:
    print ("Об'єкт не знайдено")
```

У скрипті використовуються функції:

`os.path.getsize` – повертає розмір файлу,

`os.path.getctime` – час створення,

`os.path.getatime` – час останнього відкриття,

`os.path.getmtime` – дата останньої зміни.

Метод `datetime.datetime.fromtimestamp` дозволяє виводити час у місцевому форматі.

Завдання для самостійного вирішення.

Напишіть програму, яка по імені папки виводить основну статистику її вмісту: повний шлях до папки, сумарний розмір файлів у папці та кількість під папок та їх сумарний розмір. Ім'я папки необхідно отримати як параметр командою рядка дзвінка скрипта.

9. Чому отримано виняток UnboundLocalError, хоча змінна має значення?

Може здатися несподіваним отримати `UnboundLocalError` в раніше працюючому коді, який додали операцію присвоєння десь усередині функції.

Цей код:

```
>>>
>>> x = 10
>>> def bar():
...     print(x)
>>> bar()
10
```

працює, але наступний код:

```
>>>
>>> x = 10
>>> def foo():
...     print(x)
...     x += 1
```

призводить до **`UnboundLocalError`**:

```
>>>
>>> foo()
Traceback (most recent call last):
  ...
UnboundLocalError: local variable 'x' referenced before assignment
```

Це відбувається тому, що, коли відбувається присвоєння змінній області видимості, вона стає локальною в цій області і приховує інші змінні з таким же ім'ям у зовнішніх областях.

Коли остання інструкція у `foo` надає нове значення змінної x, компілятор вирішує, що це локальна змінна. Отже, коли ранній `print` намагається надрукувати неініціалізовану змінну, виникає помилка.

У прикладі вище можна отримати доступ до змінної, оголосивши її **глобальною**:

```
>>>
>>> x = 10
>>> def foobar():
...     global x
...     print(x)
...     x += 1
>>> foobar()
10
```

Це явне оголошення потрібно для того, щоб нагадати, що (на відміну від зовні аналогічної ситуації зі змінними класу та екземпляра), насправді змінюється значення змінної у зовнішній області видимості:

```
>>>
>>> print(x)
11
```

Можна зробити подібну річ у вкладеній області видимості, використовуючи ключове слово **`nonlocal`**:

```
>>>
>>> def foo():
...     x = 10
...     def bar():
...         nonlocal x
...         print(x)
...         x += 1
...     bar()
...     print(x)
>>> foo()
10
11
```

Завдання для самостійного вирішення.

Наведіть приклади неправильного використання локальних та глобальних змінних у функцію користувача. Розробіть функцію, яка «викликає» іншу функцію користувача, ім'я та параметри якої передаються параметрами під час виклику функції. Функція повинна за виклику заданої функції «перехоплювати» виняток UnboundLocalError.

10.Які правила для глобальних і локальних змінних у Python?

У Python, змінні, на які тільки посилаються всередині функції, вважаються **глобальними**. Якщо змінній присвоюється нове значення десь у тілі функції, вважається, що вона **локальна**, і, якщо це необхідно, потрібно явно вказувати її як глобальну.

Хоча це трохи дивно на перший погляд, це легко зрозуміло. З одного боку, вимога `global` для змінних, що присвоюються, запобігає ненавмисним побічним ефектам. З іншого боку, якщо `global` був обов'язковим для всіх глобальних посилань, то `global` використовувалось весь час. Було б необхідним оголосити як глобальне кожне посилання на вбудовану функцію або компонент модуля, що імпортується.

Завдання для самостійного вирішення.

Напишіть програму, що ілюструє правила поведінки глобальних та локальних змінних у програмах мовою Python.

11.Чому анонімні функції (`lambda`), визначені в циклі з різними значеннями, повертають той самий результат?

Наприклад, розроблено наступний код:

```
>>>
>>> squares = []
>>> for x in range(5):
... squares.append(lambda: x**2)
```

Це дає список із 5 функцій, які розраховують x**2. Очікується, що, будучи викликаними, вони повернуть, відповідно, 0, 1, 4, 9, і 16. Однак, всі вони повертають 16:

```
>>>
>>> squares[2]()
16
>>> squares[4]()
16
```

Це трапляється, оскільки x не є локальною для `lambda`, а визначена у зовнішній області видимості, і використовується тоді, коли вона викликається – а не коли визначається.

В кінці циклу, x=4, тому всі функції повертають 4**2, тобто 16. Це також можна перевірити, змінивши значення x і подивившись на результат:

```
>>>
>>> x = 8
>>> squares[2]()
64
```

Щоб уникнути подібного, необхідно зберігати значення змінних локально:

```
>>>
>>> squares = []
>>> for x in range(5):
... squares.append(lambda n=x: n**2)
```

Тут n=x створює локальну для функції змінну n і обчислюється в момент визначення функції:

```
>>>
>>> squares[2]()
4
>>> squares[4]()
16
```

Такі висновки стосуються не тільки анонімних, а й звичайних функцій.

Завдання для самостійного вирішення.

Напишіть програму, що ілюструє помилку використання не анонімної функції (lambda), а звичайної функції користувача.

12.Як організувати спільний доступ до глобальних змінних для декількох модулів?

Канонічний спосіб організувати подібний доступ – створити окремий модуль (часто званий `config` або `cfg`). Необхідно просто додати `import config` у кожний модуль програми. При цьому модуль стає доступним через глобальне ім'я. Оскільки існує лише один екземпляр модуля, будь-які зміни, зроблені в модулі, відображаються скрізь.

Наприклад:

```
config.py:
x = 0

mod.py:
import config
config.x = 1

main.py:
import config
import mod
print(config.x)
```

З тих же міркувань, модулі можна використовувати як основу імплементації синглтона.

Завдання для самостійного вирішення.

Напишіть програму, що ілюструє використання глобальної змінної спільно з трьох модулів. Продемонструйте протокол її роботи

13.Як правильно використовувати імпортування?

У загальних випадках не слід використовувати конструкцію `from modulename import *`. Це засмічує простір імен того, хто імпортує. Деякі розробники уникають цієї ідіоми навіть для невеликої кількості модулів, які були спроектовані для такого імпортування, як-от `Tkinter` та `threading`.

Модулі слід імпортувати на початку файлу. Це допомагає швидко зрозуміти, які модулі вимагає код, і чи знаходиться ім'я модуля в області

видимості. Запис одного імпорту на рядок спрощує додавання та видалення операторів імпорту, хоча множинний імпорт в одному рядку займатиме менше місця на екрані.

Хорошою практикою є імпортування модулів у такому порядку:

- стандартні бібліотечні модулі (наприклад, `sys`, `os`, `getopt`, `re`)
- модулі сторонніх розробників (все, що встановлено у директорії `site-packages`), наприклад, PIL, NumPy тощо.
- локально створені модулі

Іноді буває необхідно помістити імпорт у функцію або клас, щоб уникнути проблем із циклічним імпортом.

Циклічний імпорт відмінно працює, якщо обидва модулі використовують форму `import <module>`. Але вони зазнають невдачі, коли другий модуль хоче отримати ім'я з першого (from module import name) і імпорт знаходиться на зовнішньому рівні. Це тому, що імена першого модуля ще недоступні, оскільки перший модуль зайнятий імпортом другого.

У цьому випадку, якщо другий модуль використовується тільки в одній функції, імпорт можна легко помістити в цю функцію. На той час, як його буде викликано, перший модуль вже закінчить ініціалізацію і другий модуль здійснить свій імпорт.

Може виявитися необхідним перемістити імпорт з початку файлу, якщо один є платформно-залежним модулем. У цьому випадку імпорт усіх модулів на початку файлу виявиться неможливим. У цій ситуації хорошим рішенням буде імпорт потрібних модулів у відповідному платформно-залежному коді.

Імпорти слід переміщувати у вкладені області видимості (наприклад, всередину визначень функцій) лише у випадку виникнення певної проблеми, як-от циклічний імпорт, або коли є необхідність скоротити час ініціалізації модуля.

Ця техніка є корисною, якщо значна частина імпортованих модулів не є необхідною відразу, а їх потреба залежить від того, як саме виконуватиметься програма. Імпорт також можна розмістити всередині функції, якщо конкретні модулі використовуються лише у цій функції.

Слід звернути увагу, що перше завантаження модуля може виявитися дорогим через затримку на його ініціалізацію. Проте, повторні завантаження є «безкоштовними», оскільки вони потребують лише кількох пошуків у словниках (що швидше, ніж повна ініціалізація). Навіть якщо ім'я модуля зникло з області видимості, модуль, швидше за все, досі знаходиться в `sys.modules`.

Завдання для самостійного вирішення.

В одній папці є такі файли

#**mcom.py**

com=0

```
def pp():
   print(com, " ", end="")
   return
```

#m1.py

```
import mcom
def pm1():
   mcom.com+=1
   return
```

#m2.py

```
import mcom
def pm2():
   mcom.com+=5
   return
```

#**main_m1_m2.py**

```
import mcom
import m1
import m2
mcom.pp()
m1.pm1()
mcom.pp()
m2.pm2()
mcom.pp()
print(" ").
```

Що буде виведено на консоль після виконання main_m1_m2.py:

1) 0 1 1
2) 0 1 6
3) 1 2 7
4) 0 0 0

14.Чому значення за замовчанням поділяються між об'єктами?

Цей тип помилки часто зустрічається серед початківців. Розроблено функцію:

```
def foo(mydict={}): # Небезпека: посилання між викликами
... compute something ...
mydict[key] = value
return mydict
```

При першому виклику функції, словник `mydict` містить одне значення. При другому виклику `mydict` містить вже два елементи, оскільки, коли функція `foo()` починає виконуватися, `mydict` вже містить попередній елемент.

Часто очікується, що виклик функції створює нові об'єкти для значень за замовчуванням. Однак, це не відповідає дійсності. Значення за замовчуванням створюються лише один раз, у момент, коли функція визначається. Якщо цей об'єкт змінюється, як словник у наведеному прикладі, наступні виклики функції використовуватимуть саме змінений об'єкт.

За визначенням, незмінні об'єкти (числа, рядки, кортежі та `None`) безпечні при зміні. Зміна змінних об'єктів, таких як словники, списки та екземпляри класів користувача можуть призвести до несподіваних наслідків.

Тому, гарною практикою є не використовувати об'єкти, що змінюються, як значення за замовчуванням. Замість цього рекомендується використовувати `None` і всередині функції перевіряти аргумент на `None`, а потім створювати новий список або словник.

Наприклад, не слід писати:

```
def foo(mydict={}):
    ...
```

Але можна написати так:

```
def foo(mydict=None):
    if mydict is None:
      mydict = {} # create a new dict for local namespace
```

Однак, ця особливість може бути корисною. Коли у є функція, яка довго виконується, техніка, що часто застосовується – кешування параметрів і результатом кожного виклику функції є:

```
def expensive(arg1, arg2, _cache={}):
  if (arg1, arg2) in _cache:
     return _cache [(arg1, arg2)]
  # Розрахунок значення
  result = ... expensive computation ...
  _cache[(arg1, arg2)] = result # Кладемо результат у кеш
  return result
```

Завдання для самостійного вирішення.

Програміст написав програму:

```
a=10
def fun(x, y=a):
  print(f'fun:: x={x} y={y}')

def fun1(x,y=None):
   if y==None:
```

```
    y=a;
  print(f'fun::x={x} y={y}')


a=1
print(f'a={a}')
fun(1)
a=666
print(f'a={a}')
fun(2)
```

припускаючи, що другий параметр функції fun y буде приймати значення змінної a. Але під час виконання програми було отримано:

```
a=1
fun:: x=1 y=10
a=666
fun:: x=2 y=10
```

поясніть причину такої поведінки програми і перепишіть fun так, щоб значенням другого вхідного параметра y завжди було актуальне значення змінної a.

15.Як передати опціональні чи іменовані параметри з однієї функції до іншої?

Отримати такі параметри можна за допомогою специфікаторів * та ** у списку аргументів функції; вони повертають кортеж позиційних аргументів та словник іменованих параметрів. Після цього можна передати їх в іншу функцію, використовуючи її виклик * і **:

```
def f(x, *args, **kwargs):
    ...
    kwargs['width'] = '14.3c'
    ...
    g(x, *args, **kwargs)
```

Завдання для самостійного вирішення.

Напишіть функцію mprint, яка мала всі властивості системної функції print, але додатково виводила інформацію не тільки на консоль, а й у заданий текстовий файл. Примітка: Використовуйте перший параметр цієї функції для відкриття та закриття текстового файлу.

16.Чому зміна списку 'y' також змінює список 'x'?

Якщо написано код:

```
>>>
>>> x = []
>>> y = x
>>> y.append(10)
>>> y
[10]
>>> x
[10]
```

Може виникнути здивування тому, що додавання елемента до списку, на який посилається змінна `y`, також змінює список, на який посилається змінна `x`.

Два факти призводять до такого результату:

- Змінні – це просто посилання на об'єкти. `y = x` не створює копію списку – це створює змінну `y`, яка посилається той самий об'єкт, як і `x`.
- Списки змінюються.

Після виклику `append`, вміст об'єкта було змінено з [] на [10]. Оскільки `x` і `y` посилаються на той самий об'єкт, використання будь-якого з них дає [10].

Якщо використовуються незмінні об'єкти:

```
>>>
>>> x = 5 # числа незмінні
>>> y = x
>>> x = x + 1 # 5 не можна змінити. Ми створюємо НОВИЙ об'єкт
>>> x
6
>>> y
5
```

можна бачити, що `x` і `y` більше не дорівнюють, оскільки числа незмінні, і `x = x + 1` не змінює число 5 шляхом збільшення. Натомість, створюється новий об'єкт 6 і присвоюється змінної `x` (тобто, змінюється те, який об'єкт посилається `x`). Після цього є 2 об'єкти (6 та 5) та 2 змінні, які на них посилаються.

Деякі операції (наприклад, `y.append(10)` і `y.sort()`) змінюють об'єкт, тоді як зовні схожі операції (наприклад, `y = y + [10]` і `sorted(y)`) створюють новий об'єкт. Взагалі в Python (і у всіх випадках у стандартній бібліотеці) метод, який змінює об'єкт, повертає `None`, щоб допомогти уникнути помилок. Тому якщо написано `y = y.sort()`, то це не дає відсортовану копію `y`, замість цього буде `None`, що швидше за все призведе до помилки, що легко діагностується.

Однак, існує один клас операцій, де та сама операція поводиться по-різному з різними типами: розширені оператори присвоювання. Наприклад, += змінює списки, але не кортежі або числа (`a_list += [1, 2, 3]` еквівалентно `a_list.extend([1, 2, 3])`) і змінює список, в той час як `some_tuple += ( 1, 2, 3)` та `some_int += 1` створюють новий об'єкт.

Якщо необхідно знати, чи посилаються 2 змінні на один об'єкт чи ні, можна використовувати оператор `is` або вбудовану функцію `id`.

Завдання для самостійного вирішення.

Виправте наведений на початку параграфа код
x = []
y = x
y.append(10)
print(x)
Висновок прогами:
[10]
так, що б програма виводила:
[]

17.Як створювати функції вищого порядку?

Є два шляхи: використовувати вкладені функції або об'єкти, що викликаються. Наприклад, з використанням вкладених функцій:

```
Def linear(a, b):
    def result(x):
       return a * x + b
return result
```

Використання об'єкта, що викликається:

```
class linear:

def __init__(self, a, b):
   self.a, self.b = a, b

def __call__(self, x):
   return self.a * x + self.b
```

В обох випадках,

```
taxes = linear(0.3, 2)
```

дає функцію, що (наприклад) `taxes(10e6) == 0.3 * 10e6 + 2.`

Використання об'єкта, що викликається – трохи повільніше, і в результаті виходить більший обсяг коду. Однак, слід зауважити, що кілька функцій можуть розділяти свою сигнатуру за допомогою успадкування:

```
class exponential(linear):
  #__init__ успадковується
  def __call__(self, x):
     return self.a* (x** self.b)
```

Об'єкт може зберігати свій стан для кількох викликів:

```
class counter:
```

```
value = 0

def set(self, x):
  self.value = x

def up(self):
  self.value = self.value + 1

def down(self):
  self.value = self.value - 1

count = counter()
inc, dec, reset = count.up, count.down, count.set
```

Тут `inc`, `dec`, `reset` виступають у ролі функцій, які спільно використовують ту саму змінну.

Завдання для самостійного вирішення.

Напишіть функцію, яка зберігала значення локального параметра між викликами. Напишіть програму, яка тестує цю функцію.

18.Як скопіювати об'єкт у Python?

У загальному випадку це можна зробити за допомогою модуля `copy`.

Деякі об'єкти можна скопіювати простіше. Словники мають метод `copy`:

```
newdict = olddict.copy()
```

Послідовності можуть бути скопійовані шляхом зрізів:

```
new_l = l[:]
```

Завдання для самостійного вирішення.

Напишіть функцію користувача, яка копіює об'єкт - словник не застосовуючи системний метод copy. Напишіть програму, яка тестує цю функцію.

19.Як дізнатися доступні методи та атрибути об'єкта?

`dir(x)` повертає список методів та атрибутів.

Функція `dir`() Python — це потужний інструмент для інтроспекції, який допомагає досліджувати об'єкти, показуючи список усіх атрибутів і методів.

Нижче наведено приклади використання `dir`() у різних сценаріях.

1. Використання `dir()` без аргументів

Якщо викликати `dir()` без аргументів, вона поверне список імен, визначених у поточній локальній області видимості (локальний простір імен).

```
x = 10
y = "Hello"

print(dir())
```

Приклад виведення:

```
['__annotations__', '__builtins__', '__doc__', '__loader__', '__name__', '__package__', '__spec__', 'x', 'y']
```

Як видно, у списку з'явилися створені змінні'x'і'y'.

2. Використання `dir()` із вбудованими типами даних

`dir()` дозволяє дізнатися, які методи та атрибути доступні для роботи зі стандартними типами, такими як рядки, списки, словники тощо.

Приклад зі списком:

```
my_list = [1, 2, 3]
print(dir(my_list))
```

Приклад виведення (частина списку):

```
['__add__', '__class__', ..., 'append', 'clear', 'copy', 'count', 'extend', 'index', 'insert', 'pop', 'remove', 'reverse', 'sort']
```

Це показує, що для списку доступні такі методи, як `append()`, `sort()` та інші.

Приклад з рядком:

```
my_string = "Python"
print(dir(my_string))
```

Приклад виведення (частина списку):

```
['__add__', '__class__', ..., 'capitalize', 'center', 'count', 'encode', 'endswith', 'find', 'format', ...]
```

3. Використання `dir()` з модулями

Можна передати в `dir()` імпортований модуль, щоб побачити всі функції, класи та змінні, які в ньому містяться.

```
import math
print(dir(math))
```

Приклад виведення (частина списку):

```
['__doc__', '__file__', '__loader__', ..., 'acos', 'acosh', 'asin', 'asinh', 'atan', 'atan2', 'atanh', 'ceil', 'comb', 'copysign', 'cos', 'es 'e', 'erf', 'erfc', 'exp', 'expm1', 'fabs', 'factorial', 'floor', 'fmod', 'frexp', 'fsum', 'gamma', 'gcd', 'hypot', 'inf', 'is' 'isnan', 'isqrt', 'lgamma', 'log', 'log1p', 'log10', 'log2', 'modf', 'nan', 'perm', 'pi', 'pow', 'prod', 'sin,', 'sin,', 'tan', 'tanh', 'tau', 'trunc']
```

Це допомагає швидко зрозуміти, як взаємодіяти з модулем та які функції він надає (наприклад, `sqrt`, `pii` т.д.).

4. Використання `dir()` з користувальницькими класами та об'єктами

Ви також можете використовувати `dir()` для дослідження власних класів та їх екземплярів.

```
class MyClass:
    def __init__(self):
        self.name = "Python"
        self._salary = 1000 # Атрибут із підкресленням (не публічний)

    def greet(self):
        return "Hello," + self.name

obj = MyClass()
print(dir(obj))
```

Приклад виведення (частина списку):

```
['__class__', '__delattr__', ..., 'greet', 'name', '_salary']
```

У списку є як стандартні вбудовані атрибути, так і визначені користувачем `greet`, `name` і `_salary`.

Завдання для самостійного вирішення.

Що буде виведено на консоль командою:
dir(dir)?
type(dir)
type(dir())

20.Як дізнатися ім'я об'єкта?

Взагалі, ніяк, оскільки об'єкти насправді не мають імен. Важливо: надання завжди пов'язує ім'я з об'єктом. Це вірно і для інструкцій `def` та `class`.

```
>>>

>>> class A:
... pass
...
>>> B = A
>>>
>>> a = B()
>>> b = a
>>> print(b)
<__main__.A object at 0x7fbcc3ee5160>
>>> print(a)
<__main__.A object at 0x7fbcc3ee5160>
```

Можливо, клас має ім'я: однак, хоча він пов'язаний з двома іменами і запитується через ім'я B, створений екземпляр все ще вважається екземпляром класу A. Однак, неможливо сказати, ім'я екземпляра a або b, оскільки обидва вони пов'язані з одним і тим же значенням.

Завдання для самостійного вирішення.

Наведіть приклад успадкування класів та копіювання об'єктів класу у різні змінні.

21.Який пріоритет у оператора «кома»?

Кома не є оператором в Python.

```
>>>
>>> "a" in "b", "a"
(False, 'a')
```

Оскільки кома – не оператор, але роздільник між виразами, приклад вище виконується як би було введено:

```
("a" in "b"), "a"
```

А не

```
"a" in ("b", "a")
```

Те ж саме і для операторів присвоєння (=, += та інші). Вони не є операторами як такими, лише синтаксичними роздільниками в операціях присвоювання.

Завдання для самостійного вирішення.

Наведіть приклады використання роздільника кома.

22.Чи є Python еквівалент тернарного оператора "?:" в C?

Так. Синтаксис:

```
[on_true] if [expression] else [on_false]
```

```
x, y = 50, 25
small = x if x < y else y
```

Завдання для самостійного вирішення.

Напишіть оператор присвоєння змінної z значення x якщо x лежить у числовому діапазоні від -10 до +5 і x в діапазоні від +10 до +100. В іншому випадку, надайте значення z значення нулю.

23.Чи можна писати обфусцовані однорядники?

Можна:

```
from functools import reduce

# Простые числа < 1000
print(list(filter(None,map(lambda y:y*reduce(lambda x,y:x*y!=0,
map(lambda x,y=y:y%x,range(2,int(pow(y,0.5)+1))),1),range(2,1000)))))

# Первые 10 чисел Фибоначчи
print(list(map(lambda x,f=lambda x,f:(f(x-1,f)+f(x-2,f)) if x>1 else 1:
f(x,f), range(10))))

# Множество Мандельброта
print((lambda Ru,Ro,Iu,Io,IM,Sx,Sy:reduce(lambda x,y:x+y,map(lambda y,
Iu=Iu,Io=Io,Ru=Ru,Ro=Ro,Sy=Sy,L=lambda yc,Iu=Iu,Io=Io,Ru=Ru,Ro=Ro,i=IM,
Sx=Sx,Sy=Sy:reduce(lambda x,y:x+y,map(lambda x,xc=Ru,yc=yc,Ru=Ru,Ro=Ro,
i=i,Sx=Sx,F=lambda xc,yc,x,y,k,f=lambda xc,yc,x,y,k,f:(k<=0)or (x*x+y*y
>=4.0) or 1+f(xc,yc,x*x-y*y+xc,2.0*x*y+yc,k-1,f):f(xc,yc,x,y,k,f):chr(
64+F(Ru+x*(Ro-Ru)/Sx,yc,0,0,i)),range(Sx))):L(Iu+y*(Io-Iu)/Sy),range(Sy
))))(-2.1, 0.7, -1.2, 1.2, 30, 80, 24))
#     \___ ___/  \___ ___/  |   |   |__ lines on screen
#         V          V      |   |______ columns on screen
#         |          |      |__________ maximum of "iterations"
#         |          |_________________ range on y axis
#         |____________________________ range on x axis
```

Не намагайтеся це робити вдома!

Завдання для самостійного вирішення.

Напишіть програму, яка читає файл з вихідним текстом пітон програми та «перетворює» текст на програму записану одним рядком.

24.Чому -22 // 10 равно -3?

Оскільки i % j має той самий знак, що j.

А ще

```
i == (i // j) * j + (i % j)
```

Завдання для самостійного вирішення.

Напишіть програму, яка наочно демонструє роботу оператора поділу % та //

25.Як змінити рядок?

Ніяк, оскільки рядки незмінні. У більшості ситуацій потрібно просто зробити новий рядок з різних частин. Однак, якщо це потрібно, можна використовувати `io.StringIO` або модуль `array`:

```
>>>
>>> import io
>>> s = "Hello, world"
>>> sio = io.StringIO(s)
>>> sio.getvalue()
'Hello, world'
>>> sio.seek(7)
7
>>> sio.write("there!")
6
>>> sio.getvalue()
'Hello, there!'

>>> import array
>>> a = array.array('u', s)
>>> print(a)
array('u', 'Hello, world')
>>> a[0] = 'y'
>>> print(a)
array('u', 'yello, world')
>>> a.tounicode()
'yello, world'
```

Завдання для самостійного вирішення.

Напишіть програму, яка наочно демонструє не змінність рядка.

26.Як використовувати рядки для функцій/методів?

Існує кілька прийомів.

1) Кращий – використання словника, що ставить відповідність рядку функції. Його головна перевага – рядки нічого не повинні збігатися з назвами функцій.

```
def a():
    pass

def b():
    pass

dispatch = {'go': a, 'stop': b}  # Note lack of
                                 # parens for funcs
```

```
dispatch[get_input()]()
```

2) Використання вбудованої функції **getattr**:

```
import foo
getattr(foo, 'bar')()
```

3) Використання **locals** или **eval:**

```
def myFunc():
    print("hello")

fname = "myFunc"

f = locals()[fname]
f()

f = eval(fname)
f()
«»»»»»»»»»»»»»»»»»»»
```

Завдання для самостійного вирішення.

Напишіть програму, яка наочно демонструє використання системної функції eval.

27.Як видалити всі символи нового рядка в кінці рядка?

Можна використовувати `S.rstrip("\r\n")` для видалення символів нового рядка, без видалення кінцевих пробілів:

```
>>>
>>> lines = ("line 1 \r\n"
...          "\r\n"
...          "\r\n")
>>> lines.rstrip("\r\n")
'line 1 '
```

Завдання для самостійного вирішення.

Напишіть програму, яка видаляє із заданого текстового файлу всі символи «повернення каретки» - '\r'

28.Як видалити повторювані елементи у списку?

У Python є кілька способів видалити повторювані елементи зі списку. Найпопулярнішими є:

Використання множини (`set`). Простий і швидкий спосіб, але порядок елементів не зберігається (якщо використовуєш звичайний `set`):

```
my_list = [1, 2, 2, 3, 4, 4, 5]
unique_list = list(set(my_list))
print(unique_list)
```

Зі збереженням порядку (через цикл або `dict.fromkeys`)

Варіант 1: Через цикл:

```
my_list = [1, 2, 2, 3, 4, 4, 5]
unique_list = []
for item in my_list:
    if item not in unique_list:
        unique_list.append(item)
print(unique_list)
```

Варіант 2: Через `dict.fromkeys()` (працює з Python 3.7+, де `dict` зберігає порядок)

```
my_list = [1, 2, 2, 3, 4, 4, 5]
unique_list = list(dict.fromkeys(my_list))
print(unique_list)
```

Використання `collections.OrderedDict` (для старіших версій Python)

```
from collections import OrderedDict

my_list = [1, 2, 2, 3, 4, 4, 5]
unique_list = list(OrderedDict.fromkeys(my_list))
print(unique_list)
```

Який спосіб зазвичай обирають? Якщо порядок важливий, то `list(set(my_list)`. Якщо потрібно зберегти порядок, то `dict.fromkeys(my_list)` чи цикл.

Завдання для самостійного вирішення.

Напишіть програму, яка видаляє повторювані елементи з великого списку (100000 чисел), використовуючи розглянуті способи. З'ясуйте, який спосіб потребує менше часу роботи програми.

29.Як створити багатовимірний список?

Наступний варіант є невдалим:

```
>>>
>>> A = [[None] * 2] * 3
```

Це виглядає правильно, якщо надрукувати:

```
>>>
>>> A
[[None, None], [None, None], [None, None]]
```

Але якщо надавати значення, то воно з'явиться в декількох місцях:

```
>>>
>>> A[0][0] = 5
>>> A
[[5, None], [5, None], [5, None]]
```

Причина в тому, що оператор * не створює копію, а лише посилання на наявний об'єкт. *3 створює список із 3 посилань на той самий список. Зміна в одному рядку змінює інші, які, ймовірно, є неправильними.

Можливі шляхи вирішення:

```
A = [None] * 3
for i in range(3):
A[i] = [None] * 2
w, h = 2, 3
A = [[None] * w for i in range(h)]
```

Або можна використовувати спеціальні модулі, що надають матриці. Найбільш відомим є `NumPy`.

Завдання для самостійного вирішення.

Напишіть програму, яка створить тривимірний масив – куб чисел розміру NxNxN (N<=10). Вважайте, що лівий верхній елемент має координати (0, 0, 0). Нумерація елементів куба виконується відповідно до декартової системи координат. Позиція елемента визначається кортежем (i, j, k). Використовуйте таку формулу для визначення значення числа, яке знаходиться в позиції (i, j, k) z=100*i+ 10*j+k

30.Чому a_tuple[i] += ['item'] не працює, а додавання працює?

Це пов'язано з тим, що розширений оператор присвоювання (+=) є одночасно оператором присвоювання і залежить від фундаментальної різниці між змінними та незмінними об'єктами в Python. Хоча сам кортеж (tuple) є незмінним об'єктом, його елементи можуть вказувати на змінні об'єкти (наприклад, списки).

Це обговорення стосується загалом випадків, коли розширені оператори присвоєння застосовуються до елементів кортежу, які посилаються на об'єкти, що змінюються. Для ілюстрації цієї проблеми зазвичай використовують список і оператор += як зразок.

Якщо написати:

```
>>>
>>> a_tuple = (1, 2)
>>> a_tuple[0] += 1
Traceback (most recent call last):
...
TypeError: 'tuple' object does not support item assignment
```

Причина виключення має бути зрозумілою: 1 додається до об'єкта a_tuple[0], але коли ми намагаємося присвоїти результат 2, до першого елемента в кортежі, ми отримуємо помилку, оскільки ми не можемо змінити елемент кортежу.

Тобто цей вислів робить наступне:

```
>>>
>>> result = a_tuple[0] + 1
>>> a_tuple[0] = result
Traceback (most recent call last):
...
TypeError: 'tuple' object does not support item assignment
Коли ми пишемо щось на кшталт:
>>>
>>> a_tuple = (['foo'], 'bar')
>>> a_tuple[0] += ['item']
Traceback (most recent call last):
...
TypeError: 'tuple' object does not support item assignment
```

Виняток трохи більш несподіваний, але дивовижний той факт, що, незважаючи на помилку, елемент додався!

```
>>> a_tuple[0]
['foo', 'item']
```

Щоб зрозуміти, що сталося, потрібно знати, що:

- якщо об'єкт визначає метод `__iadd__`, він викликається, коли виконується += і повернуте значення використовується для присвоювання;
- для списків `__iadd__` еквівалентний виклику `extend` для списку.

Таким чином,

```
>>> a_list = []
>>> a_list += [1]
>>> a_list
```

еквівалентний:

```
>>> result = a_list.__iadd__([1])
>>> a_list = result
```

Отже, приклад з кортежем еквівалентний:

```
>>> result = a_tuple[0].__iadd__(['item'])
```

```
>>> a_tuple[0] = result
Traceback (most recent call last):
...
TypeError: 'tuple' object does not support item assignment
```

`__iadd__`завершився успішно і список збільшився, але присвоювання закінчилося помилкою.

Завдання для самостійного вирішення.

Напишіть програму, яка демонструє роботу розширеного оператора присвоєння.

31.Генератори списків та кортежів - у чому різниця між ними?

Основна відмінність між генераторами списків (`list comprehensions`) і тим, що зазвичай називають "генераторами кортежів" (`generator expressions`) у Python, полягає в тому, що перші створюють повний список одразу і зберігають його в пам'яті, а другі є генераторами і виробляють елементи "на льоту" при необхідності, заощаджуючи пам'ять.

Генератор списків використовує квадратні дужки [] і негайно створює та заповнює весь список у пам'яті.

Синтаксис:

```
[вираз for елемент in ітерируемый_об’єкт]
```

Результат: Об'єкт типу `List`.

Використання пам'яті: Використовує більше пам'яті, оскільки зберігає всю колекцію.

Приклад: `my_list = [i * 2 for i in range(1000)]` - створює список із 1000 елементів відразу.

Генератори (`вирази`-генератори, `generator expressions`)

Так званий "генератор кортежів" насправді є виразом-генератором (`generator expression`) та використовує круглі дужки ().

Він не створює кортеж безпосередньо (у Python немає спеціального "генератора кортежів" як окремого синтаксису, подібного до генератора списків). Натомість він повертає об'єкт-генератор, який є ітератором.

Синтаксис:

```
(вираз for елемент in ітерируемый_об’єкт)
```

Результат: Об'єкт типу `generator`.

Використання пам'яті: Ефективний з погляду пам'яті, тому що генерує значення по одному за раз, тільки коли вони запитуються (ледачі обчислення).

Приклад: `my_generator = (i * 2 for i in range(1000))` - створює генератор, який почне робити значення лише за ітерації.

Зауважимо, що вираз-генератор можна перетворити на кортеж, обернувши їх у функцію `tuple()`: `my_tuple = tuple(i * 2 for i in range(10))`.

Завдання для самостійного вирішення.

Напишіть програму яка створює список та кортеж використовуючи "генератор списків" (list comprehensions) та "генератор кортежів" (generator expressions).

Наприклад:

```
my_list = [i * 2 for i in range(1000)]
```

- створює список із 1000 елементів відразу і

```
my_generator = (i * 2 for i in range(1000))
```

- створює генератор, який почне робити значення лише за ітерації.

Знайдіть розмір об'єктів `list_comp` та `my_generator` в байтах. Порівняйте ці розміри. Поясніть результати порівняння.

32.Як закодувати (і розкодувати) зображення у текстовий рядок?

Для кодування зображення в текстовий рядок Python зазвичай використовується формат Base64. Цей метод дозволяє перетворити бінарні дані зображення в послідовність символів ASCII, які можна легко передавати по каналах, призначених тільки для тексту (наприклад, в JSON-об'єктах, HTTP-запитах або електронних листах).

У Python для цього використовується вбудований модуль`base64`.

Приклад кодування зображення у рядок Base64

Для виконання цього завдання вам знадобиться файл зображення (наприклад,`my_image.jpg`), розташований у тій же директорії, що і ваш Python-скрипт.

```
import base64

def encode_image_to_base64(image_path):
    """
    Кодує файл зображення у текстовий рядок у форматі Base64.
    """
    try:
```

```
        # Відкриваємо зображення в бінарному режимі ('rb' - read
binary)
        with open(image_path, "rb") as image_file:
            # Читаємо бінарні дані файлу
            image_bytes = image_file.read()

            # Кодуємо байти в Base64
            encoded_bytes = base64.b64encode(image_bytes)

            # Декодуємо байти Base64 у рядок у кодуванні UTF-8
            base64_string = encoded_bytes.decode('utf-8')

            return base64_string

    except FileNotFoundError:
        print(f"Помилка: Файл не знайдений на шляху {image_path}")
        return None
    except Exception as e:
        print(f"Відбулася помилка: {e}")
        return None

# Приклад використання:
image_file_name = "my_image.jpg" # Замініть на ім'я файлу
encoded_string = encode_image_to_base64(image_file_name)

if encoded_string:
    print("Зображення успішно закодовано в рядок Base64.")
    # Щоб не засмічувати консоль довгим виведенням, надрукуємо
лише частину рядка
    print(f"Початок рядка: {encoded_string[:100]}...")

    # Також ви можете зберегти рядок у текстовому файлі
    with open("encoded_image.txt", "w") as text_file:
        text_file.write(encoded_string)
    print("Закодований рядок збережено у файл encoded_image.txt")
```

Як розкодувати рядок назад у зображення? Можна використовувати наступний код:

```
import base64

def decode_base64_to_image(base64_string, output_path):
    """
    Декодує рядок Base64 у файл зображення.
    """
    try:
        # Кодуємо рядок назад у байти Base64
        encoded_bytes = base64_string.encode('utf-8')

        # Декодуємо байти Base64 у вихідні бінарні дані зображення
        decoded_image_bytes = base64.b64decode(encoded_bytes)

        # Записуємо бінарні дані у новий файл у бінарному режимі
('wb' - write binary)
```

```
        with open(output_path, 'wb') as image_file:
            image_file.write(decoded_image_bytes)

        print(f"Зображення успішно розкодоване та збережене як
{output_path}")

    except Exception as e:
        print(f"Відбулася помилка при декодуванні: {e}")

# Приклад використання (продовження попереднього прикладу):
if encoded_string:
    decode_base64_to_image(encoded_string, "recovered_image.jpg")
```

Завдання для самостійного вирішення.

Розробите програму, яка тестує розглянуті вище функції кодування та декодування зображення.

Питання для самоконтролю

1. Яке основне правило щодо використання конструкції `from modulename import *` і чому її зазвичай слід уникати?
2. Назвіть рекомендований порядок групування операторів імпорту у файлі Python (три основні групи).
3. За яких двох виняткових умов (пов'язаних із проблемами або продуктивністю) допускається переміщувати оператори імпорту всередину визначень функцій?
4. Чому зазвичай рекомендується записувати один оператор імпорту на рядок, а не використовувати множинний імпорт в одному рядку?
5. Поясніть механізм, через який зміна вмісту змінної `y` (наприклад, додавання елемента до списку) також призводить до зміни вмісту змінної `x`, якщо `x = []` і `y = x`?
6. Чому оператор `a_tuple[i] += ['item']` призводить до помилки, хоча елемент `a_tuple[i]` є змінним об'єктом (списком)?
7. У чому полягає фундаментальна відмінність між змінними (`mutable`) та незмінними (`immutable`) об'єктами, що впливає на роботу з кортежами?
8. У наданому коді для обчислення функції користувача використовується конструкція `warnings.filterwarnings('error')`. Яка мета цього прийому при роботі з бібліотекою `numpy` та динамічним обчисленням (`eval`)?
9. У чому полягає проблема спільного стану у функціях `inc`, `dec`, `reset`, які використовують спільну змінну?
10. У контексті обчислення функції, введеної користувачем, яку роль відіграє вбудована функція `eval()` і чому її використання вимагає ретельної обробки помилок?

ВИКОРИСТАНА ЛІТЕРАТУРА

1. Unseen Perspectives. Python in Practice: From Fundamentals to Functional Code: Your essential first step into coding, Leanpub, 2025, 120 p.
2. Stepic, R. Python GUI with Tkinter – From Beginner to Pro. Leanpub/STREM Academy, 2025. 601 p.
3. Oliveira, W. Python for Beginners: Mastering the Basics of Python. Independently published, 2025. 500 p.
4. Hellmann, D. Python 3 Standard Library by Example. 2nd ed., 2022. 1456 p.
5. Горобець О. Ю., Горобець С. В., Хахно К. Ю. Мова Python для інженерних та наукових задач : підручник. Київ : КПІ ім. Ігоря Сікорського, 2024. 277 с.
6. Копей В. Б. Мова програмування Python для інженерів і науковців : навч. посіб. Івано-Франківськ : ІФНТУНГ, 2019. 272 с.

РЕКОМЕНДОВАНА ЛІТЕРАТУРА

1. Горобець О. Ю., Горобець С. В., Хахно К. Ю. Мова Python для інженерних та наукових задач : підручник. Київ : КПІ ім. Ігоря Сікорського, 2024. 277 с.
2. Копей В. Б. Мова програмування Python для інженерів і науковців : навч. посіб. Івано-Франківськ : ІФНТУНГ, 2019. 272 с.
3. Sach L., O'Hanlon M. Create Graphical User Interfaces with Python: How to build windows, buttons, and widgets for your Python projects. Cambridge : Raspberry Pi Trading, 2023. 156 p

Інформаційні ресурси

1. NUMPY. URL : https://numpy.org/doc/
2. PYTHON. URL : https://www.python.org/doc/
3. MATPLOTLIB. URL : https://matplotlib.org/
4. SCIPY. URL : https://docs.scipy.org/doc/
5. SYMPY. URL : http://www.sympy.org/en/index.html

ЗМІСТ

ВСТУП.....3
1. Як помістити в numpy масив матрицю, значення якої знаходяться у файлі..5
2. Як «перехопити» всі помилки (RuntimeWarning і Exception) під час виконання функції eval.....7
3. Яку версію Python встановлено.....9
4. Приклад отримання списку слів, поділених заданими роздільниками.....10
5. Програма підрахунку слів у файлі.....11
6. Приклад підрахунку кількості слів у тексті та виведення впорядкованої таблиці частоти слів.....13
7. Приклад підрахунку частоти літер у тексті та виведення впорядкованої таблиці частоти літер.....14
8. Перевірка наявності файлу чи каталогу за вказаним шляхом.....15
9. Чому отримано виняток UnboundLocalError, хоча змінна має значення? .17
10. Які правила для глобальних і локальних змінних у Python? 18
11. Чому анонімні функції (lambda), визначені в циклі з різними значеннями, повертають той самий результат?.....19
12. Як організувати спільний доступ до глобальних змінних для декількох модулів? 20
13. Як правильно використовувати імпортування? 20
14. Чому значення за замовчанням поділяються між об'єктами?.....22
15. Як передати опціональні чи іменовані параметри з однієї функції до іншої? 24
16. Чому зміна списку 'y' також змінює список 'x'? 25
17. Як створювати функції вищого порядку?.....26
18. Як скопіювати об'єкт у Python?.....27
19. Як дізнатися доступні методи та атрибути об'єкта? 27
20. Як дізнатися ім'я об'єкта? 29
21. Який пріоритет у оператора «кома»? 30
22. Чи є Python еквівалент тернарного оператора "?:" в C? 30
23. Чи можна писати обфусцовані однорядники?.....31
24. Чому -22 // 10 равно -3?.....31
25. Як змінити рядок?.....32
26. Як використовувати рядки для функцій/методів? 32
27. Як видалити всі символи нового рядка в кінці рядка? 33
28. Як видалити повторювані елементи у списку? 33
29. Як створити багатовимірний список? 35
30. Чому a_tuple[i] += ['item'] не працює, а додавання працює?.....35
31. Генератори списків та кортежів - у чому різниця між ними?.....37
32. Як закодувати (і розкодувати) зображення у текстовий рядок?.....38
Питання для самоконтролю.....40
ВИКОРИСТАНА ЛІТЕРАТУРА.....41
РЕКОМЕНДОВАНА ЛІТЕРАТУРА.....41

Навчальне видання
(українською мовою)

Борю Сергій Юрійович

ПРОГРАМУВАННЯ: PYTHON 3.X – ПРАКТИЧНИЙ Q&A

Збірник задач і вправ для здобувачів ступеня вищої освіти бакалавра спеціальності «Комп’ютерні науки» освітньо-професійної програми «Комп’ютерні науки»

Рецензент *Н. В Матвіїшина*
Відповідальний за випуск *О. С. Пшенична*
Коректор *С. Ю. Борю*