Відповідно до навчальної програми з інформатики для учнів загальноосвітніх навчальних закладів

Козолуп Є.В.

Програмування в школі. Мова Python

Навчальний посібник

9 клас

Суми 2017

УДК 004.45
ББК 32.973+74.262
К 59

Відповідно до навчальної програми з інформатики для учнів загальноосвітніх навчальних закладів

Козолуп Є.В.

Програмування в школі. Мова Python : Навчальний посібник. 9 клас. / Є.В. Козолуп. - Суми, 2017. - 56 с.

У навчальному посібнику розглядаються теми з розділу "Алгоритми опрацювання табличних величин" за навчальною програмою для учнів 9 класу на прикладі мови програмування Python.

Матеріал посібника розділений на 6 лекцій, кожна з яких містить теоретичний матеріал, проілюстрований великою кількістю наочних прикладів (кодів програм) з детальним поясненням, контрольні запитання та практичні завдання для самостійного опрацювання.

Призначений для учнів загальноосвітніх шкіл, вчителів. Рекомендується як посібник і для продовження самостійного вивчення мови програмування Python.

ЗМІСТ

Майбутньому програмісту!

Дев'ятикласнику! У минулому році ти мав чудову можливість почати знайомитися з основними аспектами програмування на прикладі мови програмування Python. Ти вже знаєш про її середовище розробки, основні функції та методи, і вже вмієш створювати прості та цікаві програми даною мовою.

В цьому навчальному році ми пропонуємо тобі вдосконалити та поглибити свої знання з програмування в цілому та конкретно теми "Алгоритми опрацювання табличних величин".

У даному навчальному посібнику міститься весь потрібний теоретичний матеріал для вивчення даної теми. Звертай велику увагу на все, що виділено **жирним** шрифтом та на пункт ***Зверніть увагу!*** Ну і як же можна навчитися створювати програми без практичної роботи за комп'ютером?! У пункті ***Практичні завдання*** є перелік задач для кращого засвоєння матеріалу.

Успіхів тобі!

Лекція 1. Поняття списку, табличних величин. Введення та виведення табличних величин. Багаторядкове текстове поле

1.1 Поняття списку

Для впорядкування значень одного або різних типів даних використовують більш складні типи даних, в порівнянні з тими, які ми вчили у минулому році.

Так для того, щоб не створювати 5 окремих змінних, можна створити одну, яка буде містити значення цих 5-ти змінних. Потім ці дані можна отримати та використати у ході роботи програми.

Тобто створити такий собі ***список (list),*** *послідовність із декількох значень одного або різних типів даних*.

У Python створення списку дуже схоже на створення змінної, але дещо відрізняється:

назва_списку=[*значення1*, *значення2*, *значення3*, …]

Наприклад, створимо список *a* зі значеннями *1, "два"* та *3.0*. Маємо:

```
>>> a=[1, "два", 3.0]
>>> a
[1, "два", 3.0]
```

Як ви могли помітити, всі елементи цього списку відносяться до різних типів даних, а саме *int*, *str* та *float* відповідно.

Отож, тепер ми вміємо створювати списки, але тут постає питання: як звернутися до окремого елемента списку? А звертатися до елементів ми будемо за їх *індексами*, тобто за порядковим номером. Кожен елемент списку має свій унікальний порядковий номер, який визначається автоматично, відповідно до розташування елементів у списку. **Зверніть увагу!** У Python індекс першого елемента не 1 а 0, другого 1 і т. д. Це особливість даної мови, і про це не можна забувати. Індекс потрібно вказувати у квадратних дужках біля назви списку:

назва_списку[*індекс*]

Отже вираз a[1] буде дорівнювати значенню "два", тобто другому елементу нашого списку *a*:

```
>>> a=[1, "два", 3.0]
>>> a[1]
```

```
"два"
>>> a[1]=="два"
True
```

Якщо ж ми введемо a[3], то Python виведе помилку, оскільки, елемента з індексом 3 не існує:

```
>>> a[3]
Traceback (most recent call last):
  File "<pyshell#1>", line 1, in <module>
    a[3]
IndexError: list index out of range
```

1.2 Основні дії зі списками

Працюючи зі списками, ми можемо виконувати три основні дії з ними: додавати списки, множити їх на число та отримувати зрізи.

Звичайно, це лише основні дії, які виконуються без використання додаткових функцій, а от більш детально про алгоритми опрацювання списків ми ознайомимося в ***лекції 2***.

1) Додавання списків

Додавання списків — це дія при якій до одного списку додаються елементи іншого. Наприклад:

```
>>> a=[1, 2.0, 3]
>>> b=[4.0, 5, 6.0]
>>> a+b
[1, 2.0, 3, 4.0, 5, 6.0]
```

Зверніть увагу! У прикладі зі списками від перестановки доданків результат буде змінюватися. Нові елементи до списку додаються **в кінець**:

```
>>> a+b
[1, 2.0, 3, 4.0, 5, 6.0]
>>> b+a
[4.0, 5, 6.0, 1, 2.0, 3]
```

Таким чином, ми можемо додавати і окремі елементи. Наприклад, ми маємо список *a* та змінну *b* і якщо ми хочемо додати значення змінної в кінець списку, то ми повинні її додати до списку, у іншому випадку до неї додати список. **Зверніть увагу!** Змінна при додаванні до списку повинна бути представлена у вигляді списку (взята в квадратні дужки):

```
>>> a=[1, 2, 3]
>>> b=4
>>> a+[b]
[1, 2, 3, 4]
>>> [b]+a
[4, 1, 2, 3]
```

2) Множення списку на число (дублювання)

При множенні списку *a* на ціле число 3 утвориться список, який буде в собі містити значення трьох списків *a:*

```
a=[1, 2, 3]
>>> b=3
>>> a*b
[1, 2, 3, 1, 2, 3, 1, 2, 3]
```

Зверніть увагу! Множити можна виключно на ціле число!

3) Отримання зрізів

Зріз *- це певна частина списку.* Для його отримання потрібно вказати початковий і кінцевий елементи (індекси) зрізу:

*назва_списку***[***початковий_елемент***:***кінцевий_елемент*]

Наприклад:

```
>>> a=[0, 1, 2, 3, 4, 5, 6, 7]
>>> a[2:5]
[2, 3, 4]
>>> a[0:3]
[0, 1, 2]
```

Зверніть увагу! Початковий елемент зрізу включається, а кінцевий - ні.

Також, якщо вам потрібно встановити початковим елементом зрізу перший елемент списку, або кінцевим елементом зрізу останній елемент списку, їх можна не вказувати:

```
>>> a=[1, 2, 3, 4, 5, 6, 7, 8]
>>> a[:5]
[1, 2, 3, 4, 5]
>>> a[3:]
[4, 5, 6, 7, 8]
```

Окрім цього, можна вказати крок зрізу. ***Крок зрізу*** *— це число, на яке буде змінюється індекс кожного наступного елемента при побудові зрізу.* Крок зрізу повинен обов'язково бути цілим числом.

Крок вказується одразу після кінцевого елемента:

назва_списку[*початковий_елемент*:*кінцевий_елемент*:*крок_зрізу*]

Наприклад, для того, щоб отримати зріз із пропущеним кожним другим числом потрібно встановити 2 в якості кроку зрізу:

```
>>> a=[0, 1, 2, 3, 4, 5, 6, 7, 8]
>>> a[::2]
[0, 2, 4, 6, 8]
```

Примітка. Також, якщо встановити -1 як крок зрізу, можна отримати список обернений до даного:

```
>>> a=[0, 1, 2, 3, 4, 5, 6, 7, 8]
>>> a[::-1]
[8, 7, 6, 5, 4, 3, 2, 1, 0]
```

1.3 Табличні величини

Тепер переходимо до поняття табличних величин.

Таблична величина *— це впорядкований* ***список****, який складається з елементів одного типу.* Так як працювати із елементами одного типу набагато простіше, у шкільній програмі ми будемо працювати саме з табличними величинами. Але, звичайно, справжній програміст повинен вміти працювати з будь-якими даними.

Чому ж саме «таблична» величина? А це тому, що будь-який однотипний список можна представити у вигляді таблиці (табл. 1):

Індекс	0	1	2	3	4	5	6
Елемент	"один"	"два"	"три"	"чотири"	"п'ять"	"шість"	"сім"

Табл. 1

І почнемо ми з введення та виведення табличних величин. Розглядати методи введення та виведення ми будемо як у командному, так і в графічному інтерфейсі.

1.4 Введення та виведення табличних величин. Командний інтерфейс

З командним інтерфейсом все просто, бо для запиту списку будемо використовувати все ту ж функцію *input()*, але, оскільки список – це

декілька значень, в якості допомоги ми застосуємо метод ***split()***. Цей метод допоможе запитати список з декількох елементів в один рядок, розділених певним розділовим знаком. Застосовується він наступним чином:

назва_списку=input()**.split(**"*розділовий_знак*"**)**

Розділовим знаком може бути будь-який фрагмент текстового рядка. А якщо його не вказувати, то автоматично розділовим знаком буде вважатися пробіл:

```
>>> a=input().split()
4 7 8 3 3 0 5 4 3 0
```

Спробуємо застосувати цю конструкцію для створення простої задачі. Спочатку запитаємо список значень у користувача, а потім зобразимо його в зворотному порядку:

```
>>> a=input().split()
4 7 8 3 3 0 5 4 3 0
>>> a[::-1]
['0', '3', '4', '5', '0', '3', '3', '8', '7', '4']
```

Все дуже просто! Але, як ви могли помітити, всі елементи нашого списку — текстові фрагменти. Це тому, що ми не вказали тип наших елементів, а вони за замовчуванням представляють з себе текст. Тут вже трішки складніше, бо ми повинні застосувати функцію переведення для кожного елементу списку при його введенні. У цьому нам допоможе наступна конструкція (всі елементи списку — цілі числа, тому застосуємо функцію *int()*, у випадку із дійсними числами використовувати функцію *float()*):

a=[int(i) for i in input().split()]

i — елемент списку input().split().

Тобто, для кожного елемента *i* із нашого введеного списку *input().split()* потрібно застосувати функцію *int()*. Так, форма запису циклу відрізняється від тої, яку ми використовували в 8-му класі, але й так можна представляти цикл *for*: в один рядок.

```
>>> a=[int(i) for i in input().split()]
4 7 8 3 3 0 5 4 3 0
>>> a
[4, 7, 8, 3, 3, 0, 5, 4, 3, 0]
```

Ну ось, нарешті, все як потрібно. Тепер можна переходити до виведення нашого списку.

Виведення відбувається за допомогою вже відомої нам функції *print()*:

```
>>> a=[int(i) for i in input().split()]
4 7 8 3 3 0 5 4 3 0
>>> print(a)
[4, 7, 8, 3, 3, 0, 5, 4, 3, 0]
```

Але форма зображення списку бажає кращого. Наприклад, у нас є бажання, аби список зображався як рядок із елементів, розділених пробілом, або щось в цьому стилі. Для цього потрібно застосувати той самий цикл *for* для кожного елемента із нашого списку *a*. Тілом циклу буде функція виведення нашого елементу *i*:

```
>>> a=[int(i) for i in input().split()]
4 7 8 3 3 0 5 4 3 0
>>> for i in a:
      print(i)
4
7
8
3
3
0
5
4
3
0
```

Ми майже досягли потрібного результату, але елементи виводяться в стовпчик, тобто кожен новий елемент з нового рядка.

Функція *print()* має чудовий атрибут, який допоможе нам у вирішенні нашої проблеми. Це атрибут ***end***, який вказує що повинно стояти після виведення всіх елементів функції *print()*. За замовчуванням значенням атрибута *end* є рядок '\n', тобто перехід на наступний рядок. Ми ж встановимо пробіл. От що вийшло:

```
>>> a=[int(i) for i in input().split()]
```

```
4 7 8 3 3 0 5 4 3 0
>>> for i in a:
        print(i, end=" ")

4 7 8 3 3 0 5 4 3 0
```

Як ви бачите, ми досягли потрібного результату, та в залежності від умови завдання, можна встановити будь-який інший знак.

А тепер переходимо до графічного інтерфейсу.

1.5 Введення та виведення табличних величин. Графічний інтерфейс

Одразу ж створимо нове вікно із розмірами 300x300 пікселів із назвою "Табличні величини" та створимо напис "Введіть список значень:":

```
from tkinter import *
Window=Tk()
Window.geometry("300x300")
Window.title("Табличні величини")
Label(Window, text="Введіть список значень:").place(x=70, y=25)
Window.mainloop()
```

Ми вже знайомі з текстовим полем *Enrty,* і можемо використати його для введення списків, але *Enrty* - це однорядкове текстове поле, тому для введення великих списків його використовувати буде не доречно. В даному випадку краще використати багаторядкове текстове поле ***Text***. Створюється воно за допомогою функції ***Text()*** наступним чином:

назва_поля=**Text**(*вікно*, *атрибут1*, *атрибут2* ...)

Даний об'єкт має такі атрибути:

- **bg=**"колір" — колір поля;
- **fg=**"колір" — колір тексту поля;
- **font=**"шрифт та розмір шрифту" - шрифт тексту;
- **width=***число* — ширина, зазначається у кількості знаків;
- **height=***число* — висота, зазначається у кількості знаків;
- **bd=***число* — ширина контуру поля (у пікселях);

Для роботи з багаторядковим текстовим полем існують наступні методи:

- **insert()** - метод додавання даних;

- **delete()** — метод видалення даних;
- **get()** - метод отримання даних.

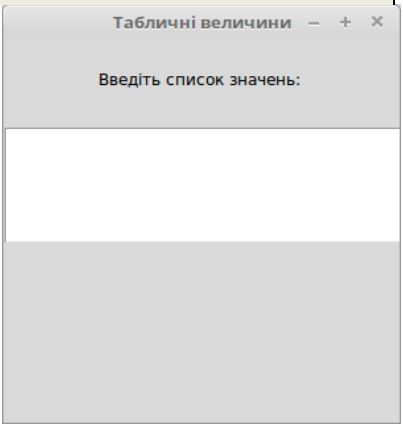
Методи *delete* та *get* мають такі параметри: початкова та кінцева позиції. Позиція позначається у вигляді "x.y". x-номер рядка, y-номер стовпчика. А метод *insert* має початкову позицію та текст, який буде додаватися. Давайте розглянемо приклади:

```
...
text1.insert('1.0', 'Текст')
text1.delete('1.0', '3.2')
text1.get('1.0', END)       # Отримати всі дані
...
```

Переходимо до створення багаторядкового текстового поля з шириною 42 знаки та з висотою 5 знаків білого кольору. Розмістимо його в 70-ти пікселях від верхнього краю:

```
...
text=Text(Window, width=42, height=5, bg="white", font='Arial 11')
text.place(x=0, y=70)
...
```

Після запуску програми ми вже можемо вводити текст в наше багаторядкове текстове поле (Мал. 1).



Мал. 1

Тепер переходимо до виведення нашого списку. Найефективніший спосіб виведення — побудова таблиці. Та в Python немає такого елемента як таблиця, але є метод розміщення об'єктів у вигляді таблиці. Цей метод називається ***grid***, а розміщувати ми будемо написи із елементами списку.

Та перед цим створимо рамку для нашої таблиці. Саме в рамці і будуть розміщуватися елементи списку. Рамка створюється за допомогою функції ***Frame()***, вона має наступні атрибути:

- **bg=**"колір" — колір поля;**width=***число* — ширина, зазначається у кількості знаків;

- **width**=*число* — ширина, зазначається у кількості знаків;
- **height**=*число* — висота, зазначається у кількості знаків;
- **bd**=*число* — ширина контуру поля (у пікселях);

Рамку, як і інші об'єкти вікна *tkinter,* потрібно розміщувати за допомогою метода *place()*, що ми і зробимо:

```
...
frame=Frame(Window, bd=0.5, relief="solid")
frame.place(x=5, y=200)
...
```

relief — це ще один атрибут, який встановлює тип границі. *Solid* — звичайна лінія.

На самостійне опрацювання! Знайдіть у Всесвітній мережі якомога більше типів границі.

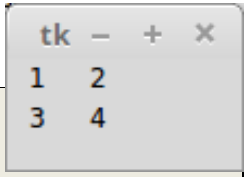
А зараз переходимо до створення самої таблиці за допомогою метода *grid.* Застосовується він так:

об'єкт.**grid**(row=*число*, column=*число*)

row — номер рядка;

column — номер стовпчика.

Ось простий приклад (мал. 2):



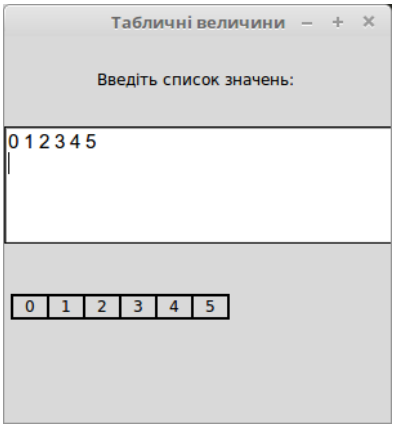
Мал. 2

```
from tkinter import *
Window=Tk()
frame=Frame(Window)
frame.place(x=0, y=0)
label1=Label(frame, text="1", width=3).grid(row=0, column=0)
# створення та розміщення напису в рамці frame
label2=Label(frame, text="2", width=3).grid(row=0, column=1)
# створення та розміщення напису в рамці frame
label3=Label(frame, text="3", width=3).grid(row=1, column=0)
# створення та розміщення напису в рамці frame
label4=Label(frame, text="4", width=3).grid(row=1, column=1)
# створення та розміщення напису в рамці frame
Window.mainloop()
```

Також створимо подію *Return* для нашого вікна, і сам алгоритм виведення списку (створення написів та їх розміщення):

```
...
def fun1(event):
    a=[int(i) for i in text.get("1.0", END).split()]
    # створення списка із елементами, які ввів користувач
    c=0
    # початкове значення стовпчика
    for i in a: # цикл for для всих елементів списку a
        label=Label(frame,    text=i,    bd=0.5,    relief="solid",
width=3)
        # створення напису
        label.grid(row=0, column=c)
        # розміщення напису
        c=c+1
        # зміна номера стовпчика
...
Window.bind("<Return>", fun1)
#створення події
...
```

Пояснення. Спочатку ми повинні всі отримані дані перетворити у однотипний список, елементи якого – цілі числа. Потім встановимо початкові значення стовпця від яких ми будемо відштовхуватися. Для рядка нову змінну ми не створюємо, оскільки виводитися весь список буде в один рядок. Його значення постійне і дорівнює 0. Запускаємо цикл для кожного елемента зі списку a. Кожен елемент буде виводитися у вигляді тексту напису у стовпчики, а номер стовпчика при кожному повторенні буде змінюватися на 1.



Мал. 3

Давайте протестуємо нашу програму (мал. 3).

Цікава інформація

Чи пам'ятаєте ви функцію ***range()***, яку ми використовували для створення циклів з лічильником? Так от, головне призначення даної

функції – створення однотипних списків, елементи яких – цілі числа, які утворюють алгебраїчну прогресію. Ця функція має наступні параметри: початковий елемент, кінцевий елемент та крок:

range(*початковий_елемент*, *кінцевий_елемент*, *крок*)

Зверніть увагу! Кінцевий елемент не включається.

Наприклад, створимо список із чисел починаючи з 0 і до 101 зі кроком 3, ось що маємо:

```
>>> a=range(1,101,3)
>>> for i in a:
        print(i, end=" ")

0 3 6 9 12 15 18 21 24 27 30 33 36 39 42 45 48 51 54 57 60 63 66
69 72 75 78 81 84 87 90 93 96 99
```

Контрольні запитання

1. Сформулюйте визначення списку.
2. Які основні дії можна виконувати зі списками?
3. Як отримати зріз списку?
4. Як запитується список значень у: a) командному; b) графічному інтерфейсі?
5. Які методи використовуються для роботи із багаторядковим текстовим полем?
6. Для чого служить метод grid?
7. Назвіть всі нові об'єкти вікна tkinter, з якими ви познайомилися в цій лекції?

Практичні завдання

Командний інтерфейс

1. Створіть просту програму мовою Python, яка буде додавати між собою два списки (значення користувач вводить сам), а потім виводить результат.

2. Створіть програму мовою Python. На вході маємо два списки і ціле число. Якщо введене число більше 0, то до першого списку додається другий, інакше до другого перший. На виході маємо новий утворений список.

3. Створіть програму мовою Python, яка запитує цілочисельний список у користувача та виводить повідомлення "Так" у випадку, якщо список читається однаково як зліва на право, так і навпаки. У випадку. якщо умова не виконується, виводитиметься повідомлення "Ні".

Графічний інтерфейс

1. Створіть нове вікно *tkinter* та багаторядкове текстове поле. Розміри вікна та атрибути текстового поля вказати самостійно. В текстове поле користувач буде вводити список однотипних значень, а виводитися буде той самий список, тільки в зворотному порядку (у вигляді таблиці).

2. Створіть нове вікно *tkinter* та багаторядкове текстове поле. Розміри вікна та атрибути текстового поля вказати самостійно. В текстове поле користувач буде вводити список температурних показників за рік (починаючи з січня). На вході ми матимемо 4 однорядкові таблиці з показниками температур за певний сезон. Для більш зручного орієнтування додати написи з сезонами перед таблицями.

Лекція 2. Алгоритми опрацювання списків

Ми продовжуємо знайомитися з табличними величинами та з алгоритмами їх опрацювання, але на відміну від попередньої лекції, у цій ми будемо більше працювати не з самими списками, а з їх елементами.

2.1 Знаходження довжини списку

Інколи нам доводиться створювати програми, при цьому не знаючи що саме введе користувач, тому просто порахувати кількість значень списку на пальцях у нас ніяк не вийде. Для цього існує спеціальна функція ***len()***:

len(*назва_списку*)

```
>>> a=[0, 1, 2, 3, 4, 5, 6, 7, 8]
>>> len(a)
9
```

За допомогою цієї функції ми можемо звернутися до останнього елемента списку, чого раніше ми не могли зробити. Для цього від довжини списку потрібно відняти 1, щоб отримати індекс останнього елемента, адже нумерація в списку починається не з 1, а з 0!

```
>>> a[len(a)-1]
8
```

2.2 Заміна та видалення значень

У цій частині хочеться наголосити на тому, що список — це такий тип даних, що легко піддається змінам. Елементи списку можна змінювати, видаляти, сортувати, на відміну від його "родича" - кортежа, про якого ви можете більш детально дізнатися в рубриці **Цікава інформація**.

Почнемо зі зміни значень. Щоб замінити значення елемента з індексом *i* потрібно використати наступну конструкцію:

назва_списку[i]=*нове_значення*

```
>>> a=[0, 1, 2, 3, 4, 5, 6, 7, 8]
>>> a[4]="чотири"
>>> a
[0, 1, 2, 3, 'чотири', 5, 6, 7, 8]
```

Аналогічно і видалення елемента відбувається за його індексом з допомогою оператора ***del***:

del *назва_списку*[*індекс*]

```
>>> del a[3]
>>> a
[0, 1, 2, 'чотири', 5, 6, 7, 8]
>>> del a[len(a)-1]
>>> a
[0, 1, 2, 'чотири', 5, 6, 7]
```

Також видаляти елемент можна і за його значенням. Для цього використовується метод ***remove()***:

назва_списку.**remove**(*значення*)

```
>>> a=[1, 2, 3, 4]
>>> a.remove(2)
>>> a
[1, 3, 4]
```

Зверніть увагу! Якщо значення не вказано, буде видалений останній елемент.

2.3 Знаходження кількості заданих елементів

Окрім того, що ми можемо знайти загальну кількість всіх елементів, ми можемо знаходити кількість заданих елементів. Таку дію можна виконати за допомогою метода ***count()***:

назва_списку.**count**(*елемент*)

Тобто, якщо нам потрібно порахувати скільки раз у списку зустрічається число 7, ми зробимо це так:

```
>>> a=[5, 8, 7, 2, 9, 8, 1, 7, 3, 10]
>>> a.count(7)
2
```

2.4 Знаходження максимального та мінімального значення списку

Все, що нам потрібно, це згадати вивчені в 8-му класі функції *max()* та *min()*:

```
>>> a=[3, 5, 8, 3, 7, 2, 9, 0, 8, 1, 7, 3, 10, 4]
>>> min(a)
0
>>> max(a)
10
```

2.5 Знаходження суми та середнього значення елементів списку

І тут знову на допомогу нам приходить вбудована функція для сумування всіх елементів списку і називається вона ***sum()***:

```
>>> a=[1, 2, 3, 4, 5]
>>> sum(a)
15
```

Для знаходження середнього значення елементів списку потрібно просто суму поділити на кількість елементів:

```
>>> a=[1, 2, 3, 4, 5]
>>> sum(a)/len(a)
3.0
```

2.6 Застосування циклу for для опрацювання табличних величин

З конструкцією for і in range(n) ми знайомі з 8-го класу, і використовували ми її для створення циклу з лічильником, але тоді ми ще не знали, що цикл *for* призначений саме для обробки списків. Тепер ми вже знаємо, що *range(n)* — список *n* чисел від *0* до *n-1*. А якщо ви ще не знайомі з даною функцією — рекомендую ознайомитися з рубрикою **Цікава інформація** з першої лекції. А надалі бажано не оминати цю рубрику.

Основна ж конструкція цього циклу виглядає так:

for *i* **in** *назва_список*:

дії

i – змінна елемента списку.

Тобто виконати певні дії для кожного елемента списку. Вже зрозуміло, що кількість повторень залежить від кількості елементів. В тілі циклу змінна *i* набуватиме значення елемента списку над яким відбувається робота.

Ми вже працювали з подібним циклом, коли виводили елементи списку:

```
>>> a=[9, 8, 7, 6, 5, 4, 3, 2, 1]
>>> for i in a:
        print(i, end=" ")

9 8 7 6 5 4 3 2 1
```

Як ми бачимо, при кожному повторі циклу значення змінної *i* змінюється на значення наступного елемента списку. Цей елемент виводиться, а після того, як будуть опрацьовані всі елементи списку, цикл завершиться.

За допомогою циклу *for* ми можемо представити практично всі функції для опрацювання списків, про які ми нещодавно дізналися. Наприклад, функція *sum()*. Все, що нам потрібно – створити певну змінну. При кожному повторенні циклу значення цієї змінної буде збільшуватися на значення елемента *i*. На початку значення змінної дорівнює 0:

```
>>> a=[1, 2, 3, 4, 5]
>>> s=0
>>> for i in a:
        s=s+i

>>> print(s)
15
```

Тепер розглянемо приклад із функцією *count()*. Цей алгоритм ми представимо у вигляді повноцінної програми, а отже на початку у користувача буде запитуватися список цілих чисел. Потім ще одне ціле число. Кількість таких чисел у списку потрібно підрахувати.

Аналогічно, як і в першому випадку, ми створюємо нову змінну. При кожному повторенні циклу буде перевірятися, чи елемент списку дорівнює введеному числу; і в позитивному випадку змінна кількості елементів буде збільшуватися на 1. Ну і в кінці виводитиметься кількість потрібних елементів. Ось код програми:

```
a=[int(i) for i in input().split()]
b=int(input())
c=0
for i in a:
   if i==b:
       c=c+i
print(c)
```

Також окрім стандартної конструкції циклу *for*, його можна представити одним рядком так, як представляємо його, коли запитуємо список у користувача. Звичайно така конструкція має більше недоліків ніж

переваг, але якщо ви хочете для кожного елемента застосувати будь-яку одну функцію, то такий запис буде дуже зручним. Отже, у квадратних дужках ми повинні вказати функцію з атрибутом *i* та конструкцію циклу *for*:

назва_списку=[*функція(i)* for i in *назва_списку*]

Зверніть увагу! Жодних розділових знаків встановлювати не потрібно.

Ну наприклад, у нас є список із дійсних чисел, а нам потрібно всі числа округлити до цілого. Саме тут нам і допоможе дана конструкція циклу *for*:

```
>>> a=[1.2, 2.3, 3.4, 4.5, 5.6]
>>> a=[round(i) for i in a]
>>> a
[1, 2, 3, 4, 6]
```

2.7 Приклади програм

Створимо дві програми: одну з командним інтерфейсом та одну з графічним.

На вході ми маємо список з дійсних чисел. Це будуть показники температури за рік, тобто 12 чисел по одному на кожен місяць. Задача програми вивести аналіз цих температур за наступними пунктами: кількість місяців, в яких температура була нижча за 0; максимальну та мінімальну температури та середнє значення температури за весь рік (округлене до десятків). Для створення цієї програми ми використаємо вбудовані функції: *sum()*, *min()* та *max(),* і розробимо власний алгоритм для підрахунку кількості місяців, в яких температура була нижче нуля.

Спочатку будемо запитувати список із дійсних чисел у користувача та обов'язково встановимо повідомлення, про те, що потрібно ввести саме 12 чисел. Отже, що ми маємо:

```
a=[float(i) for i in input(
      "Введіть показники температур (12 чисел): ").split()]
c=0
for i in a:
    if i<0:
        c=c+1
print("Місяців із показником температури менше 0:", c)
```

```
print("Максимальна температура:", max(a))
print("Мінімальна температура:", min(a))
print("Середнє значення температур:", round(sum(a)/12, 1))
```

Протестуємо нашу програму та поспостерігаємо за виведенням результату (мал. 4).

```
Введіть показники температур (12 чисел): -10 -9 9.4 16 25 23.6 24 27.4
18 15 8.8 -6
Місяців із показником температури менше 0: 3
Максимальна температура: 27.4
Мінімальна температура: -10.0
Середнє значення температур: 11.9
>>>
```

Мал. 4

Переходимо до наступної задачі, яку ми створимо за допомогою модуля *tkinter*. Тепер нам потрібно написати програму, яка буде аналізувати оцінки учня за такими пунктами: найбільший та найменший бал, середній бал та кількість оцінок початкового, середнього, достатнього та високого рівня (у вигляді таблиці). І для початку створимо нове вікно та текстове поле, а також додамо напис із текстом "Введіть список оцінок:":

```
from tkinter import *
Window=Tk()
Window.geometry("300x300")
Window.resizable(False, False)
Label(Window, text="Введіть список оцінок:").place(x=80, y=25)
text=Text(Window, width=37, height=5, bg="white")
text.place(x=0, y=70)
Window.mainloop()
```

Зверніть увагу на напис! Ми не давали йому назву, бо більше не будемо звертатися до нього, а отже і назва йому не потрібна.

Тепер створимо написи, в яких будуть на виході міститися вихідні дані, а також кнопку для запуску алгоритму підрахунку результатів.

```
...
but=Button(Window, text="Проаналізувати")
but.place(x=100, y=120)
maximum=Label(Window, text="Максимальний бал:")
maximum.place(x=20, y=160)
minimum=Label(Window, text="Мінімальний бал:")
```

```
minimum.place(x=20, y=180)
serednii=Label(Window, text="Середній бал:")
serednii.place(x=20, y=200)
...
```

Переходимо до створення таблиці. Для цього застосуємо метод *grid()*:

```
...
frame=Frame(Window)
frame.place(x=70, y=230)
Label(frame, text="Н", width=5).grid(row=0, column=0)
Label(frame, text="С", width=5).grid(row=0, column=1)
Label(frame, text="Д", width=5).grid(row=0, column=2)
Label(frame, text="В", width=5).grid(row=0, column=3)
n_label=Label(frame, text="", width=5)
n_label.grid(row=1, column=0)
s_label=Label(frame, text="", width=5)
s_label.grid(row=1, column=1)
d_label=Label(frame, text="", width=5)
d_label.grid(row=1, column=2)
v_label=Label(frame, text="", width=5)
v_label.grid(row=1, column=3)
...
```

Перший рядок таблиці — це клітинки з літерами, які відповідають рівню оцінок: Н (низький), С (середній), Д (достатній), В (високий). Другий рядок — клітинки з порожнім написом, який після обробки заповниться вихідними даними.

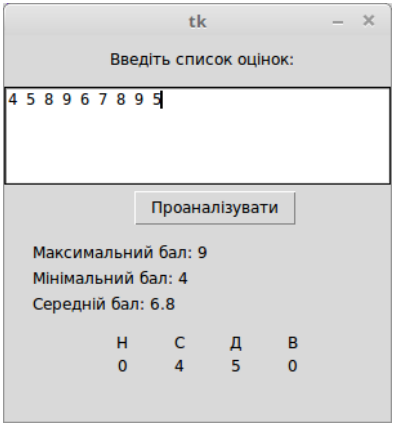
Переходимо до алгоритму аналізу оцінок. Створюємо подію та функцію з алгоритмом:

```
...
def fun(event):
    a=[int(i) for i in text.get('1.0', END).split()]
    maximum["text"]="Максимальний бал: "+str(max(a))
    minimum["text"]="Мінімальний бал: "+str(min(a))
    serednii["text"]="Середній бал: "+str(round(sum(a)/len(a), 1))
    n=0
    s=0
```

```
    d=0
    v=0
    for i in a:
        if i<=3:
            n=n+1
        elif i>3 and i<=6:
            s=s+1
        elif i>6 and i<=9:
            d=d+1
        else:
            v=v+1
    n_label["text"]=n
    s_label["text"]=s
    d_label["text"]=d
    v_label["text"]=v
...
but.bind("<Button-1>", fun)
...
```

Як ви бачити, тут були використані функції *min()* та *max()*, алгоритм підрахунку середнього балу та власний алгоритм підрахунку кількості оцінок за їх рівнем. **Пояснення.** Цикл *for* проходить по кожному елементу списку і перевіряє, до якого із рівнів буде відноситися дана оцінка. Після цього кількість оцінок відповідного рівня збільшується на 1.

Давайте переглянемо, як працює дана програма (мал. 5)



Мал. 5

Цікава інформація

Табличних величин багато і списки – це далеко не єдиний тип даних, який відноситься до послідовностей. Їх існує дуже багато, та нам потрібно знати тільки про декілька із них. Наприклад, про **кортежі**.

Кортеж *— це тип даних, який відноситься до незмінюваних послідовностей.* Тобто, на відміну від списку, кортежі можна тільки створити, та доповнювати, а от видаляти чи замінювати елементи – неможливо. Кортежі потрібні, аби тільки зберігати дані для їх подальшого використання, отже на кортежі буде витрачатися менше пам'яті та часу, а отже це може дуже допомогти в ході роботи програми.

Як ми вже знаємо елементи списків вводяться в квадратні дужки, що стосується кортежів, елементи вводяться в *круглі* дужки:

назва_кортежа=(*елемент1*, *елемент2*...)

А от що стосується алгоритмів опрацювання, вони залишаються такими ж: можна отримувати зрізи, звертатися до окремих елементів, підраховувати суму, кількість значень та інше.

Контрольні запитання

1. Яке призначення функції *len()*?
2. Які основні дії можна виконувати із списками?
3. Як отримати зріз списку?
4. Як запитується список значень у: a) командному; b) графічному інтерфейсі?
5. Які методи використовуються для роботи із багаторядковим текстовим полем?
6. Для чого служить метод grid?
7. Назвіть всі нові об'єкти вікна tkinter з якими ви познайомилися в цій лекції?

Практичні завдання

Командний інтерфейс

Створіть програму мовою Python за наступною задачею:

a) На вході маємо цілочисельний список. Програма повинна піднести кожен елемент до квадрату та вивести новий список;
b) на вході маємо список з дійсних чисел. На виході маємо округлену суму всіх елементів даного списку;
c) на вході маємо цілочисельний список. Програма повинна вивести кількість елементів списку менших за 0;

d) маємо цілочисельний список, який вводе користувач. Програма повинна замінити всі від'ємні числа на протилежні до них;
e) користувач вводе список з необмеженої кількості літер українського алфавіту. Програма повинна порахувати скільки голосних літер в даному списку. Використовувати цикл *for* заборонено!

Графічний інтерфейс

1. Створіть нове вікно *tkinter*, та багаторядкове текстове поле. Розміри вікна та атрибути текстового поля вказати самостійно. В текстове поле користувач буде вводити список з дійсних чисел, а виводитися буде той самий список (у вигляді таблиці), тільки замість всіх значень, що більші за 0 і менші за 10 буде виводитися 1.

2. Створіть нове вікно *tkinter*, та багаторядкове текстове поле. Розміри вікна та атрибути текстового поля вказати самостійно. В текстове поле користувач буде вводити список з показниками цін на певний товар в різних магазинах. На виході маємо три написи, один з мінімальною ціною, один з максимальною, а один з середньою ціною по всіх магазинах.

Практична робота №1

Тема: Складання і виконання алгоритмів знаходження сум і кількостей значень елементів табличних величин за заданими умовами.

Для виконання даної практичної роботи необхідно знати:

- принципи введення та виведення табличної величини в командному та графічному інтерфейсі;
- основні функції та методи для опрацювання табличних величин ;
- як застосовується цикл *for* до табличної величини.

Завдання

Відкрийте середовище розробки програмного забезпечення IDLE та створіть новий файл Python. На вході маємо цілочисельний список значень. На виході повний аналіз цього списку за кожним із цих пунктів:

a) кількість значень списку;
b) середнє значення списку;
c) сума всіх елементів списку;
d) кількість нульових елементів;
e) кількість чисел *a* (*a* запитується в користувача);
f) максимальний та мінімальний елементи списку (2 числа в один рядок);
g) кількість додатних елементів;
h) кількість від'ємних елементів;
i) кількість елементів більших за число *b* (*b* запитується у користувача).

Лекція 3. Рядок тексту

У цій лекції, на відміну від більшості попередніх ми не будемо працювати із числами: дійсними або цілими. Ця лекція присвячена виключно роботі з текстом, але от питання: як відноситься рядок тексту до табличних величин?

Ми вже знаємо, що існує не один тип даних, який відноситься до табличних величин. Це списки та кортежі, про які ми дізналися в попередній лекції, а також ще багато інших. Тому існують функції переведення складних типів даних між собою (зі списків в кортежі та навпаки і т.п.). Але нас цікавить тільки одна, функція ***list()***, яка призначена для перетворення будь-якої табличної величини в список. Розглянемо на прикладі:

```
>>> a=(1, 2, 3)
>>> a=list(a)
>>> a
[1, 2, 3]
```

Зверніть увагу! Застосовувати подібні функції до окремих значень не можна, лише до табличних величин, або простих різнотипних списків:

```
>>> a=1
>>> list(a)
Traceback (most recent call last):
  File "<pyshell#12>", line 1, in <module>
    list(a)
TypeError: 'int' object is not iterable
```

Давайте подивимося, що відбудеться з текстом, якщо застосувати цю функцію до рядка тексту:

```
>>> a="text"
>>> a=list(a)
>>> a
['t', 'e', 'x', 't']
```

Саме так, текст, хоч і відноситься до простих типів даних, але він все одно є найпростішою табличною величиною, бо текстовий фрагмент складається з менших однотипних частинок — символів.

3.1 Робота з рядком тексту

Оскільки рядок тексту — це таблична величина, то для нього будуть справедливі більшість функцій та алгоритмів, які ми вивчали раніше. Наприклад, ми можемо отримувати зріз або звернутися до окремого символу рядка тексту. Це робиться так, як і зі списком:

```
>>> a="Hello, World!"
>>> a[:5]
'Hello'
>>> a[7:12]
'World'
>>> a[5]
','
```

Також із вже відомих нам функцій є функція *len()*, яка повертає кількість елементів у тексті:

```
>>> a="Hello, World!"
>>> len(a)
13
```

Чи пам'ятаєте ви метод *split()*, яким ми користувалися для введення списку значень? А за допомогою даного методу ми можемо розбити рядок тексту за певним розділовим знаком. Наприклад:

```
>>> a="слово1 слово2 слово3"
>>> a=a.split()
>>> a
['слово1', 'слово2', 'слово3']
>>> a="слово1, слово2, слово3"
>>> a=a.split(", ")
>>> a
['слово1', 'слово2', 'слово3']
```

Коротко про інші методи для обробки тексту дивіться в таблиці 2.

Метод	Призначення
рядок_тексту.**isdigit()**	Для перевірки: чи складається рядок тексту тільки з цифр. Відповідь у вигляді *True/False*.
рядок_тексту.**isalpha()**	Для перевірки: чи складається рядок тексту тільки з символів. Відповідь у вигляді *True/False*.

рядок_тексту.**isalnum()**	Для перевірки: чи складається рядок тексту з цифр та символів. Відповідь у вигляді *True/False*.
рядок_тексту.**islower()**	Для перевірки: чи складається рядок тексту з символів у нижньому регістрі. Відповідь у вигляді *True/False*.
рядок_тексту.**isupper()**	Для перевірки: чи складається рядок тексту з символів у верхньому регістрі. Відповідь у вигляді *True/False*.
рядок_тексту.**istitle()**	Для перевірки: чи починається кожне слово рядка з великої літери. Відповідь у вигляді *True/False*.
рядок_тексту.**upper()**	Переведення рядка тексту до верхнього регістру.
рядок_тексту.**lower()**	Переведення рядка тексту до нижнього регістру.
розділовий_знак.**join**(*список*)	Для збірки тексту зі списку з розділовим знаком. Метод обернений до *split()*. **Зверніть увагу!** Всі елементи списку повинні бути текстовими фрагментами.
рядок_тексту.**capitalize()**	Переведення першого символу до верхнього регістру, а всі інші до нижнього.
рядок_тексту.**lstrip()**	Видалення пробілів на початку рядка.
рядок_тексту.**rstrip()**	Видалення пробілів в кінці рядка.
рядок_тексту.**strip()**	Видалення пробілів на початку та в кінці рядка.
рядок_тексту.**swapcase()**	Для заміни регістру.
рядок_тексту.**title()**	Переведення першої літери кожного слова до верхнього регістру.

Табл. 2

Рядок тексту відноситься до незмінних послідовностей, як і кортежі, тому ми можемо тільки створювати та доповнювати його, і отримувати дані:

```
>>> a="Hello, World!"
>>> a[3]="L"
Traceback (most recent call last):
  File "<pyshell#3>", line 1, in <module>
```

```
    a[3]="L"
TypeError: 'str' object does not support item assignment
>>> del a[5]
Traceback (most recent call last):
  File "<pyshell#4>", line 1, in <module>
    del a[5]
TypeError: 'str' object doesn't support item deletion
```

Давайте розглянемо приклад простої задачі. На вході маємо слово, в будь-якому регістрі та будь-якою мовою. Програма аналізує слово та перевіряє, чи може це слово однаково читатися як зліва на право, так і справа на ліво. Відповідь ми отримаємо у вигляді *True/False*.

Здавалося, що це задача в одну дію, але це не так. Спочатку ми повинні перевести всі символи до одного регістру та видалити непотрібні пробіли на початку та в кінці, які користувач може випадково ввести, а потім вивести результат перевірки, тобто:

```
a=str(input("Введіть слово: "))
# видалення зайвих пробілів
a=a.strip()
# переведення символів до нижнього регістру
a=a.lower()
print(a==a[::-1])
```

Запустимо нашу програму та для приклад: введемо слова "дід" та "баба" (мал. 6).

```
======================
Введіть слово: дід
True
>>>
======================
Введіть слово: баба
False
>>>
======================
Введіть слово:  ДІд
True
>>>
```

Мал. 6

Для кращого засвоєння матеріалу створимо більш складну програму. На вході маємо те саме слово, а на виході – два слова, утворені з нього, які складаються: перше — лише з літер верхнього регістру, а друге — лише нижнього (мал. 7).

Для розв'язання даної задачі ми створимо два нових порожніх рядки тексту та запустимо цикл *for* для кожного символу введеного слова, далі – перевірка регістру даного символу і, в залежності від результату, цей символ додається до порожнього рядка:

```
a=str(input("Введіть слово: "))
```

```
# видалення зайвих пробілів
a=a.strip()
b=""
c=""
for i in a:
    if i.isupper():
        b=b+i
    else:
        c=c+i
print(b)
print(c)
```

```
========================
Введіть слово: СлоВо
СВ
лоо
>>> 
```

Мал. 7

Контрольні запитання

1. Яке призначення функції *list()*?
2. Чи відноситься рядок тексту до табличних величин?
3. Текст – це змінна або незмінна величина?
4. Які основні операції можна виконувати з рядком тексу як табличною величиною?
5. Назвіть основні методи для обробки текстових величин.

Практичні завдання

Командний інтерфейс

1. Створіть нову програму мовою Python. На вході маємо ім'я, прізвище та по батькові користувача саме в такому порядку і в один рядок. На виході маємо текстовий рядок з даними користувача, але в іншому порядку: прізвище, ім'я та по-батькові.

2. Створіть нову програму мовою Python. На вході маємо два слова, записаних в один рядок. Якщо перше слово є оберненим до другого, то на виході матимемо *True*, а якщо ні – *False*. Регістр літер в словах не має значення.

3. Створіть нову програму мовою Python. Користувач вводить будь-які слова в один рядок через крапку. Задача програми вивести цей самий рядок, слова якого розділені крапкою, але кожне слово в зворотному порядку.

4. Створіть нову програму мовою Python. На вході маємо два слова. Програма повинна вивести спільні літери в цих словах.

Лекція 4. Поняття матриці. Алгоритми опрацювання двовимірних списків

Вже три лекції поспіль ми говоримо про табличні величини, але, окрім уявної таблиці, яку представляє з себе список, інших таблиць ми не бачили. А от в даній лекції ми попрацюємо зі справжніми двовимірними таблицями, із однотипних елементів.

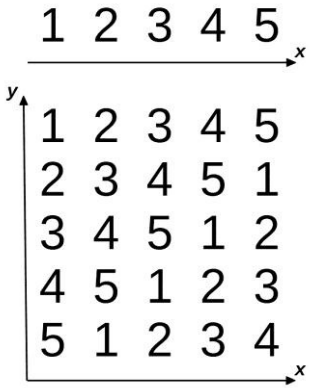
4.1 Поняття матриці

У мові Python таблицю звикли представляти у вигляді *двовимірного списку*. **Двовимірним** називають *список, який складається із одновимірних списків*[1].

Таку таблицю із даними одного типу називають ***матрицею*** (мал. 8).

1	2	3	4	5
2	3	4	5	1
3	4	5	1	2
4	5	1	2	3

Мал. 8



Мал.9

Давайте, до кінця зрозуміємо, що таке двовимірний, а що таке одновимірний список, і чим вони відрізняються. В назвах вже вказана головна відмінність — кількість вимірів або *координатних осей*. Одновимірний список відображається за допомогою однієї осі (вісь *x*), а матриці за допомогою двох (вісь *x* та *y*), як на мал. 9.

Ще в **лекції 1** був оговорений факт, що елементом списку може бути будь-що, навіть інший список, то можна створити простий двовимірний список із п'яти простих. Ось так:

```
>>>
a=[[1,2,3,4,5],[2,3,4,5,1],[3,4,5,1,2],[4,5,1,2,3],[5,1,2,3,4]]
>>> a
[[1, 2, 3, 4, 5], [2, 3, 4, 5, 1], [3, 4, 5, 1, 2], [4, 5, 1, 2,
3], [5, 1, 2, 3, 4]]
```

Аналогічно, як і з одновимірними списками, ми можемо звертатися до певного елемента за його індексом:

```
>>> a[1]
```

1 Дана тавтологія є повністю виправдана

```
[2, 3, 4, 5, 1]
```

Та для того, щоб звернутися до конкретного елемента в середині отриманого списку, індекс вказується двічі: спочатку для загального списку, а потім для внутрішнього:

назва_списку[*індекс*][*індекс*]

```
>>> a[1]
[2, 3, 4, 5, 1]
>>> a[1][2]
4
```

І таким способом ми можемо отримувати зрізи загального та внутрішнього списків:

```
>>> a[:2]
[[1, 2, 3, 4, 5], [2, 3, 4, 5, 1]]
>>> a[1][:4]
[2, 3, 4, 5]
```

4.2 Введення елементів матриць

Матрицю прийнято вводити рядками, тобто використовувати просту конструкцію запиту списку відповідну кількість разів і після кожного повторного запиту додавати новий список до основного. Наприклад ось так:

```
>>> a=[]
>>> for i in range(3):
        b=[int(i) for i in input().split()]
        a=a+[b]

3 4 5 6
4 5 6 7
5 6 7 8
>>> a
[[3, 4, 5, 6], [4, 5, 6, 7], [5, 6, 7, 8]]
```

В даному випадку ми запитували матрицю із трьох рядків, про що свідчить кількість повторень.

Тепер давайте розглянемо випадок, коли ми не знаємо, зі скількох рядків буде складатися наша матриця. Просто на початку потрібно запитати у користувача кількість рядків:

```
>>> n=int(input("N="))
N=5
```

І потім запустити цикл із *n-ю* кількістю повторень.

4.3 Виведення матриць

Тепер перейдемо до виведення матриці. Коли ми виводили одновимірний список ми користувалися наступною конструкцією:

for i in *список*:

print(i, end="")

Проте в даному випадку список не один, а декілька, і при кожному повторенні зміна *i* також буде списком вже із конкретних значень, тому ми повинні запустити ще один цикл, для кожного елемента нашого отриманого списку *i* (використаємо якусь іншу змінну, за звичай *j*), і тепер ми будемо виводити не *i*, а *j*:

```
>>> a=[[1, 2, 3],[2, 3, 1],[3, 1, 2]]
>>> for i in a:
        for j in i:
                print(j, end=" ")



1 2 3 2 3 1 3 1 2
```

Як бачите, вся матриця вивелася в один рядок. Ну звичайно, адже після обробки кожного внутрішнього списку потрібно переміститися на новий рядок:

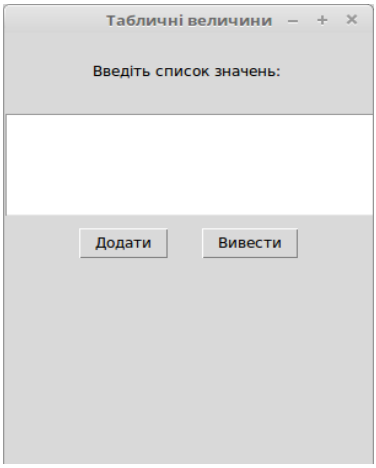
```
>>> a=[[1, 2, 3],[2, 3, 1],[3, 1, 2]]
>>> for i in a:
        for j in i:
                print(j, end=" ")
        print()



1 2 3
2 3 1
3 1 2
```

4.4 Введення та виведення. Графічний інтерфейс

З графічним інтерфейсом все просто: ми доповнимо вже існуючий алгоритм з першої лекції, так як зробили це раніше.

Тож маємо вікно, напис та текстове поле, розміри вікна встановимо 300x400 пікселів (мал. 10). Тепер додамо дві кнопки: "Додати" та "Вивести":

```
from tkinter import *
Window=Tk()
Window.geometry("300x400")
Window.title("Табличні величини")
Label(Window, text="Введіть список значень:").place(x=70, y=25)
text=Text(Window, width=42, height=5, bg="white", font='Arial 11')
text.place(x=0, y=70)
but1=Button(Window, text="Додати")
but1.place(x=60, y=170)
but2=Button(Window, text="Вивести")
but2.place(x=160, y=170)
Window.mainloop()
```



Мал. 10

Вводити матрицю ми будемо рядками. Додавати рядок за рядком, після кожного натиску кнопки "Додати". Рядок додається до матриці і видаляється із текстового поля, для введення нового. Після всіх змін можна натиснути кнопку "Вивести", при чому нижче з'явиться введена матриця.

Отже, спочатку алгоритм введення:

```
...
a=[]
def dodatu(event):
    b=text.get("1.0", END).split()
    global a
    a=a+[b]
    text.delete("1.0", END)
...
but1.bind("<Button-1>", dodatu)
...
```

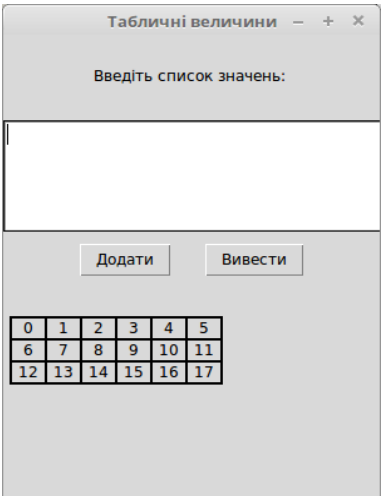
Пояснення. Змінна *a* буде містити матрицю, тому на початку це просто порожній список. Функція *dodatu()* запускається одразу після натискання на кнопку, і перша дія — це зчитування списку з текстового поля. **Зверніть увагу!** Ми не вказували тип даних елементів списку, адже задачею програми є лише виведення цих елементів; якщо ж від вас буде вимагатися подальша обробка і аналіз даних, тип потрібно вказувати. Далі просто додавання нових елементів у матрицю, як і з командним інтерфейсом та видалення даних з текстового поля. **Зверніть увагу**, що змінна *a* повинна зберігати своє значення й поза функцією, тому ми назначили її глобальною.

Обов'язково перед тим, як виводити таблицю, створимо рамку:

```
...
frame=Frame(Window, bd=0.5, relief="solid")
frame.place(x=5, y=230)
...
```

Ну і тепер алгоритм виведення:

```
...
def vuvestu(event):
    r=0
    # початкове значення рядка
    for i in a:
        c=0
        # початкове значення стовпчика
        for j in i:
            label=Label(frame,  text=j,
bd=0.5, relief="solid",
                width=3)
            # створення напису
            label.grid(row=r, column=c)
            c=c+1
            # зміна номера стовпчика
        r=r+1
...
but2.bind("<Button-1>", vuvestu)
...
```



Мал. 11

Алгоритм аналогічний до того, що був у **лекції 1**, але тепер циклів у нас два: один для рядків, а інший для елементів у цих рядках. Тепер змінюється не тільки номер стовпчика, а й номер рядка.

Запустимо програму та поглянемо, як вона працює (мал. 11)

4.5 Алгоритми опрацювання матриць

Стосовно матриць: зберігаються всі принципи та методи з одновимірних списків, із нового — лише подвійні цикли, аби пройти всі елементи матриці.

Тож для прикладу створимо простий алгоритм підрахунку суми всіх елементів цілочисельної матиці. На вході запитується кількість рядків матриці, та безпосередньо її елементи. На виході – сума матриці. Та, на жаль застосувати нам вже відому функцію *sum()* до всієї матриці у нас не вийде, адже підрахувати суму можна тільки числа, а не списки. Та можна додати суми всіх внутрішніх списків:

```
n=int(input("n="))
a=[]
for i in range(n):
    b=[int(i) for i in input().split()]
    a=a+[b]
s=0
for i in a:
    s=s+sum(i)
print(s)
```

Розглянемо ще один нескладний приклад. На вході маємо матрицю із n рядків, на виході – кількість нульових значень. Спочатку створимо нову змінну, потім запускаємо цикл для всіх елементів матриці, і, якщо елемент дорівнює 0, змінна кількості збільшується на 1:

```
n=int(input("n="))
a=[]
for i in range(n):
    b=[int(i) for i in input().split()]
    a=a+[b]
c=0
for i in a:
    for j in i:
```

```
        if j==0:
            c=c+1
print(c)
```

Тепер давайте вирішимо наступну задачу. На вході матриця із n рядків, на виході нова матриця, рядки якої складаються із стовпців попередньої. Тобто, якщо на вході a=[[1, 2], [3, 4]], то на виході c=[[1,3], [2, 4]].

Для розв'язання даної задачі нам допоможе нова функція zip(), застосовується вона так:

zip(матриця*)**

Зірочка — обов'язковий елемент, який означає, що змінна введена після неї буде саме двовимірним списком. Та для того, щоб в результаті отримати список, потрібно ще застосувати функцію *list()* перед нею:

```
n=int(input("n="))
a=[]
for i in range(n):
    b=[int(i) for i in input().split()]
    a=a+[b]
c=list(zip(*a))
for i in c:
    for j in i:
        print(j, end=" ")
    print()
```

```
 RESTART: /media/yevhenii/
n=3
4 5
6 7
8 9
4 6 8
5 7 9
>>>
 RESTART: /media/yevhenii/
n=5
12 13 14 15
10 9 8 7
50 51 52 53
-7 -6 -5 -4
0 0 0 0
12 10 50 -7 0
13 9 51 -6 0
14 8 52 -5 0
15 7 53 -4 0
>>>
```

Мал. 12

Розглянемо, як працює ця програма на практиці (мал. 12).

Контрольні запитання

1. Дайте визначення двовимірного списку.
2. Чим відрізняється одновимірний та двовимірний список?
3. Як звертатися до окремого елемента в матриці? Як отримати зріз?
4. Як застосовується цикл *for* для двовимірного списку?

Практичні завдання

Командний інтерфейс

1. Створіть нову програму мовою Python. На вході маємо цілочисельну матрицю, яка складається з *n* рядків. Задача програми:

1) піднести кожен елемент до квадрата та вивести нову матрицю;
2) додати до кожного елемента матриці число *a* (*a* запитується у користувача) та вивести нову матрицю;
3) отримати квадратний корінь з кожного елемента матриці та вивести її;
4) округлити кожен елемент залишивши *y* чисел після коми та вивести матрицю (*y* запитується у користувача).

Примітка. Всі описані дії повинні виконуватися послідовно як одна програма.

Графічний інтерфейс

1. Створіть нову програму мовою Python, нове вікно tkinter та три нових поля: два однорядкових і одне багаторядкове. Також створіть дві кнопки: "Додати" та "Виконати". Розмістіть їх таким чином, щоб багаторядкове текстове поле знаходилося вище від кнопки "Додати", а однорядкові текстові поля і кнопка "Виконати" нижче. За допомогою багаторядкового текстового поля і кнопки "Додати" програма буде запитувати цілочисельну матрицю в користувача. А в однорядкові текстові поля користувач повинен ввести числа *a* і *b*. Після натискання на кнопку "Виконати", програма ділить кожен елемент матриці на *a* і округлює з точністю *b* чисел після коми, та виводить нову матрицю у вигляді таблиці.

Лекція 5. Алгоритми упорядкування даних

Чи пам'ятаєте ви уроки з теми Microsoft Excel? На цих уроках ви вже знайомилися із такими прийомами, як сортування, фільтрація та пошук, коли працювали з електронними таблицями. А тепер ми навчимося застосовувати ці прийоми при роботі із одновимірними та двовимірними списками мовою Python.

5.1 Сортування

Ми знаємо, що при виконанні цього процесу впорядкування елементів списку всі елементи залишаються, проте змінюють своє положення в залежності від умови, яку ви зазначили.

Наприклад, у нас є одновимірний список, із цілих чисел:

```
>>> a=[1, 5, 2, 4, 3]
```

Наша задача полягає у тому, щоб відсортувати дані від найменшого до найбільшого. Алгоритм сортування даного списку за даною умовою виглядає так:

1. a=[1, 5, 2, 4, 3]
2. a=[1, 2, 5, 4, 3]
3. a=[1, 2, 3, 5, 4]
4. a=[1, 2, 3, 4, 5]

Щоб представити цей алгоритм у вигляді програмного коду мовою Python, ми можемо розробити програму, яка буде переставляти елементи безпосередньо в початковому списку за індексами в залежності від зазначеної умови, і т.п. Та буде набагато зручніше створити новий список і з його допомогою відсортувати введений:

```
>>> a=[5, 2, 1, 4, 3]
>>> b=[]
>>> for i in range(len(a)):
        b=b+[min(a)]
        a.remove(min(a))

>>> b
[1, 2, 3, 4, 5]
>>> a
[]
```

Пояснення. Як і було сказано, був створений новий список *b*, потім запущено цикл *for* 5 разів (або len(a)). У тілі циклу дві операції: пошук мінімального елемента, додавання цього елемента в новий список та видалення цього елемента з початкового списку.

Таким чином при кожному повторенні елементи "перестрибують" з одного списку в інший, але тепер в тій послідовності, якій нам хочеться:

1. a=[5, 2, 4, 3]; b=[1]
2. a=[5, 4, 3]; b=[1, 2]
3. a=[5, 4]; b=[1, 2, 3]
4. a=[5]; b=[1, 2, 3, 4]
5. a=[]; b=[1, 2, 3, 4, 5]

Давайте, для прикладу, ускладнимо дану задачу. На вході ми матимемо цілочисельну матрицю з *n* рядків. Задача така ж: в кожному рядку відсортувати значення від найменшого до найбільшого. Принцип залишається той же, але тепер нам потрібно створювати два нових списки, та використовувати подвійний цикл для обробки:

```
n=int(input("n="))
a=[]
for i in range(n):
       A=[int(i) for i in input().split()]
       a=a+[A]
       # запит матриці
c=[] # нова матриця, до якої додаватимуться впорядковані рядки
for i in a:
      b=[]
      # новий рядок, до якого додаватимуться впорядковані елементи
      for j in range(len(i)):
            b=b+[min(i)]
            i.remove(min(i))
      c=c+[b] # додавання впорядкованого рядка до нової матриці
for i in c:
      for j in i:
            print(j, end=" ")
      print()
# виведення нової матриці
```

Задача аналогічна попередній, тому пояснювати її не потрібно, лише зроблю деякі примітки. Список *b* при кожному запуску верхнього циклу буде набувати значення порожнього списку, а початковим списком в даному випадку слугує список *i*. Тому саме від нього рахуємо мінімальний елемент, кількість елементів, та саме з нього видаляємо пройдений елемент.

```
======================
n=4
1 6 9 4
-4 -2 -6 -8
2 3 8 4
9 5 3 2
1 4 6 9
-8 -6 -4 -2
2 3 4 8
2 3 5 9
>>>
```

Мал. 14

Після запуску бачимо, що програма працює повністю правильно (мал. 14).

5.2 Фільтрація

Фільтрацією ми звикли називати процес відсіювання непотрібної інформації. Видаляти ми нічого не будемо, а просто створимо новий список і будемо додавати до нього тільки ті елементи, які нам потрібні. Аналогічно як і у випадку з сортуванням, але непотрібних елементів в новому списку не буде.

Одразу приклад задачі. На вході маємо одновимірний цілочисельний список. Задача програми – відфільтрувати список за наступною умовою: елементи списку повинні бути більші або дорівнювати *0*.

Створимо новий список і запустимо цикл для обробки всіх елементів. Якщо елемент задовольняє умову, то додамо його до нового порожнього списку і виведемо його:

```
a=[int(i) for i in input().split()]
b=[]
for i in a:
    if i>=0:
        b=b+[i]
for i in b:
    print(i, end=" ")
```

```
======================
-1 2 -3 4 -5 6 -7 -8
2 4 6
>>>
```

Мал. 13

Спостерігаємо за виконанням програми (мал. 13).

Ну і приклад з матрицею. На вході маємо цілочисельну матрицю з *n*-ю кількістю рядків, на виході – лише додатні елементи рядків.

Отже, відбуватиметься перевірка, і якщо елемент рядка менше від 0, він не буде додаватися до нової матриці, яка після перевірки виводитиметься:

```
n=int(input("n="))
a=[]
for i in range(n):
    b=[int(i) for i in input().split()]
    a=a+[b]
c=[] # нова матриця
for i in a:
    d=[] # новий рядок
    for j in i:
        if j>=0:
            d=d+[j]
    c=c+[d]
for i in c:
    for j in i:
        print(j, end=" ")
    print()
# виведення нової матриці
```

5.3 Пошук

Розглянемо на прикладі, як можна застосовувати, та які бувають алгоритми пошуку при роботі зі списками. Нехай у нас є два одновимірних списки: один з назвами місяців, а другий – із середнім показником температури в цьому місяці:

```
m=["січень","лютий","березень","квітень","травень","червень",
"липень","серпень","вересень","жовтень","листопад","грудень"]
t=[-13.4, -15, 10, 17, 25.6, 28, 27.5, 29.3, 15, 13.4, 9, -10.4]
```

При цьому кожен елемент першого списку із певним індексом буде відповідати елементу з таким же індексом із іншого списку. Тобто, в *лютому* температура була *-15* градусів.

Задача програми: на вході маємо місяць, а на виході – температуру в цьому місяці. Доповнимо код, і от що в нас повинно вийти:

```
m=["січень","лютий","березень","квітень","травень","червень",
"липень","серпень","вересень","жовтень","листопад","грудень"]
t=[-13.4, -15, 10, 17, 25.6, 28, 27.5, 29.3, 15, 13.4, 9, -10.4]
```

```
month=str(input("Введіть місяць: ")).lower()
# переведення до нижнього регістру введеного слова
for i in m:
     if month==i:
          print(t[m.index(i)])
```

Пояснення. Спочатку у користувача запитується місяць, потім запускається цикл для кожного елемента списку *m*, а в ньому перевірка, чи збігається цей елемент із введеним текстом. Якщо так, то на екран буде виводитися елемент зі списку *t* з таким же індексом, що і елемент *i* зі списку *m*. Нагадую, що метод *index()* призначений для визначення індексу елемента із певного списку.

З одновимірними списками все просто – ми пов'язуємо їх між собою за допомогою індексів і здійснюємо пошук.

Тепер переходимо до матриць.Знову проста задача. На вході маємо цілочисельну матрицю із *n* рядків та *n* стовпців. Користувач вводить ціле число, а задачею програми є знайти це число в матриці і вивести ті числа, що знаходяться вище та нижче від даного, розділивши пробілом. Якщо елемента вище або нижче немає, замість нього виводиться *0*.

Спочатку стандартний алгоритм запиту матриці та алгоритм пошуку елементів схожий на алгоритм в попередній задачі:

```
n=int(input("n="))
a=[] # нова матриця
a1=[] # в подальшому перший і останній рядок з n нулів
for i in range(n):
     a1=a1+[0]
a=a+[a1]
# створення та додавання першого рядка з n нулів
for i in range(n):
     b=[int(i) for i in input().split()]
     a=a+[b]
# введення та формування матриці
a=a+[a1]
# додавання останнього рядка з n нулів
c=int(input("Введіть число: "))
for i in a:
     for j in i:
```

```
            if j==c:
                  i1=a.index(i)-1 # індекс рядка вище
                  i2=a.index(i)+1 # індекс рядка нижче
                  i3=i.index(j) # індекс елемента в рядку
                  print(a[i1][i3], a[i2][i3])
                  # виведення пари чисел
```

Зверніть увагу на те, як ми досягли виведення нулів. Було створено два рядки: один на початку, а один в кінці з *n*-ю кількістю нулів. Після того, як умова виконується, обчислюються індекси рядків, які знаходяться вище та нижче від даного, та індекс елемента, і за даними індексами отримується елемент, який знаходиться вище та нижче від даного.

```
n=5
1 2 3 4 5
2 3 4 5 6
3 4 5 6 7
4 5 6 7 8
5 6 7 8 9
Введіть число: 7
6 8
6 8
6 0
>>>
```

Мал. 15

Давайте запустимо нашу програму (мал. 15).

Контрольні запитання

1. При роботі з якою прикладною програмою ви зустрічалися з поняттям сортування, фільтрації та пошуку.
2. Поясніть яким чином відбувається сортування табличних величин мовою програмування Python?
3. Дайте визначення поняттю фільтрація.
4. Як відбувається пошук в одновимірних та двовимірних списках (матрицях)?

Практичні завдання

Командний інтерфейс

1. Створіть нову програму мовою Python. На вході маємо одновимірний список із цілих чисел. Задачею програми є:
   a) отримати квадратний корінь кожного елемента списку та відсортувати список від найменшого до найбільшого елемента;
   b) відсортувати список від найбільшого до найменшого елемента;

c) отримати зріз до середини списку та відсортувати його від найменшого елемента до найбільшого.

2. Створіть нову програму мовою Python. На вході маємо одновимірний список із дійсних чисел. Задачею програми є відфільтрувати цей список за наступними умовами:

1) вивести список із елементів, які не менше 0;
2) вивести список із елементів, які не мають дробової частини;
3) вивести список із елементів, які входять у проміжок $\left[\frac{1}{2}; 100\right]$;
4) вивести список із елементів, які є розв'язком нерівності $2x + 6 > 0$;
5) вивести список із елементів, які менші за суму елементів зліва та справа від нього.

3. Створіть нову програму мовою Python. На вході маємо два списки: назва товару та ціна товару. **Примітка.** Кількість елементів цих списків повинна бути однакова. Після цього користувач вводить назву та кількість товару, який хоче замовити. Програма повинна вивести повну вартість замовленого товару.

Практична робота №2

Тема: Складання і виконання алгоритму пошуку значень у таблиці.

Для виконання даної практичної роботи необхідно знати основні алгоритми впорядкування значень табличних величин та вміти їх застосовувати до одновимірного та двовимірного списків.

Завдання 1

Відкрийте середовище розробки програмного забезпечення IDLE та створіть новий файл Python. На вході маємо список з цілих чисел. Програма повинна створити новий з квадратів чисел введеного списку. Після цього користувач у новому рядку вводе елемент першого списку, а на виході маємо елемент другого, який йому відповідає.

Завдання 2

Відкрийте середовище розробки програмного забезпечення IDLE та створіть новий файл Python. На вході маємо цілочисельну матрицю та ціле число. Задача програми знайти це число в матриці та вивести число, яке стоїть справа від даного. **Примітка.** Якщо числа праворуч не має, вивести 0.

Лекція 6. Візуалізація табличних величин за допомогою графічних примітивів. Побудова діаграм

Згадаймо про такий спосіб зображення табличної величини, як *діаграма*. Взагалі, діаграми бувають різними: векторна діаграма, гістограма, графік та інші. Ми ж спробуємо навчитися зображати табличні величини у вигляді *гістограми*.

6.1 Побудова гістограми

Гістограма *або* ***стовпчикова діаграма*** *— графічний спосіб зображення табличних даних у вигляді прямокутних стовпчиків, розміщених один біля одного, розміри яких залежать від значень табличної величини* (мал).

Спробуємо створити подібну діаграму засобами мови програмування Python. І для прикладу поєднаємо два інтерфейси: командний та графічний.

Спочатку на вході в командному інтерфейсі у користувача запитується список із оцінками з семи предметів: математики (мат.), інформатики (інф.), української мови (укр. мова), української літератури (укр. літ.), фізики (фіз.), біології (біол.) та географії (геогр.):

```
a=[int(i) for i in input().split()]
```

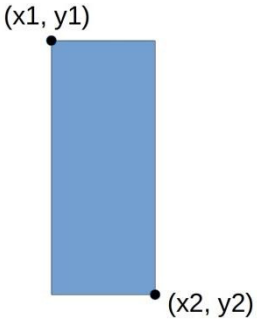
Тепер підключимо модуль tkinter та підготуємо полотно для діаграми. Так як максимальна оцінка 12 балів, нехай кожен бал в стовпчиковій діаграмі буде займати 20 пікселів, отже максимальна висота полотна — 240, а ширина — 260 пікселів (7 стовпчиків та 6 пропусків по 20 пікселів):

```
from tkinter import *
Window=Tk()
Window.title("Діаграма")
Window.geometry("260x240")
canv=Canvas(Window, width=260, height=240)
canv.place(x=0, y=0)
Window.mainloop()
```

Ну, а зараз переходимо до виведення самої діаграми. Ми запустимо цикл для кожного елемента *i* у списку *a* із функцією створення прямокутника за чотирма координатами: *x1, y1, x2, y2* (мал. 16). Одразу

потрібно зазначити, які з цих координат мають постійне значення, а які будуть змінюватися.

В даному випадку лише змінна *y2* буде константою, тобто незмінною і буде мати значення 240. Початкове значення *x1* — 0, *x2* — 20 пікселів (ширина стовпця), а значення змінної *y1* залежатиме безпосередньо від оцінки (змінної *i*). Як вже раніше було сказано 1 бал стовпчика займає 20 пікселів, а отже довжину стовпчика можна визначити за формулою *i**20. Тепер ми маємо максимальну довжину 240 і довжину стовпчика *i**20, а значить залишок це і будить наш *y1*, маємо *y1*=240-*i**20.

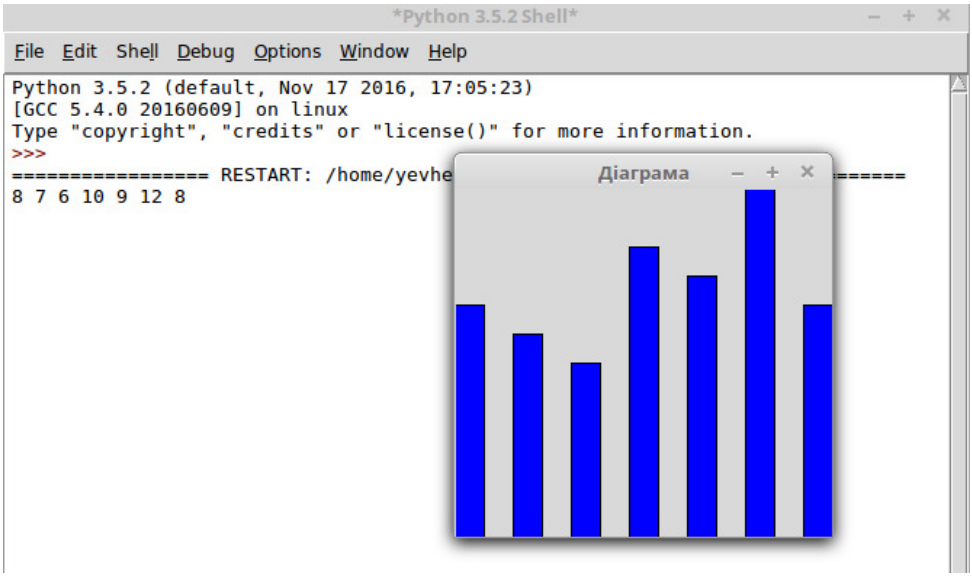


Мал. 16

Стосовно змінних *x1* та *x2*, то після кожного повторення їх значення будуть збільшуватися на 40 пікселів (20 пікселів — ширина стовпця та 20 пікселів — пропуск між ними). Ось так виглядатиме загальний алгоритм:

```
a=[int(i) for i in input().split()]
from tkinter import *
Window=Tk()
Window.title("Діаграма")
Window.geometry("260x240")
canv=Canvas(Window, width=260, height=240)
canv.place(x=0, y=0)
x1=0
x2=20
y2=240
for i in a:
     y1=240-i*20
     canv.create_rectangle(x1, y1, x2, y2, fill="blue")
     x1=x1+40
     x2=x2+40
Window.mainloop()
```

Запустимо програму та введемо такі значення: 8, 7, 6, 10, 9, 12, та 8 (мал. 17).



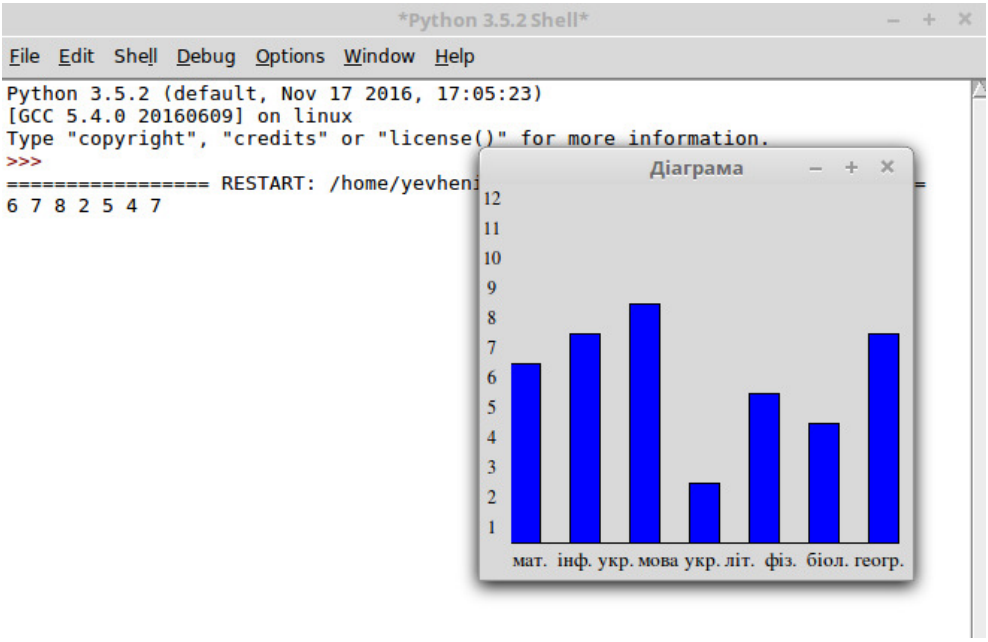
Мал. 17

Ми отримали в результаті те, що і хотіли отримати. Проаналізуємо наш результат. Перший і останній рядок є повністю однаковими, а передостанній стовпчик займає весь наданий простір. Для того, щоб не заплутатися в даній діаграмі, було б доречно додати координатні осі. Їх значення ми виведемо за принципом, за яким ми виводили таблиці в графічному інтерфейсі, але в цьому випадку границі виводитися не будуть. Внесемо наступні зміни:

```
...
Window.geometry("290x265")
canv=Canvas(Window, width=260, height=240)
canv.place(x=20, y=0)
...
canv.create_line(0,240,240,240)
# побудова горизонтальної лінії
frame1=Frame(Window)
frame1.place(x=0, y=0)
r=0
for i in range(12,0,-1):
     Label(frame1, text=i, font="Times 9", width=2).grid(row=r,
column=0)
```

```
      r=r+1
# виведення вертикальної таблиці
frame2=Frame(Window)
frame2.place(x=20, y=241)
c=0
b=["мат.", " інф.", "укр. мова", "укр. літ.", " фіз.", " біол.",
"геогр."]
for i in b:
      Label(frame2, text=i, font="Times 10").grid(row=0, column=c)
      c=c+1
# виведення горизонтальної таблиці
...
```

Та як це буде виглядати при роботі? Результат роботи на мал. 18.



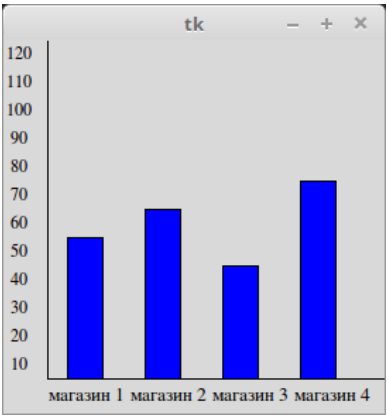
Мал. 18

Контрольні запитання

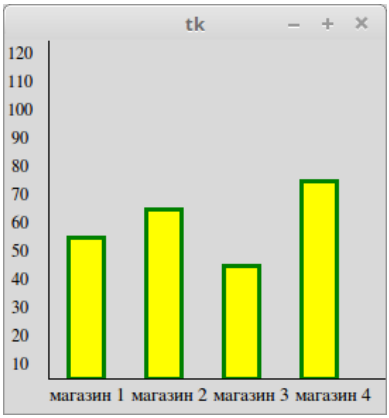
1. Дайте визначення поняттю діаграма.
2. Чим відрізняється гістограма від інших типів діаграм?
3. Як створити стовпчикову діаграму засобами мови програмування Python?

Практичні завдання

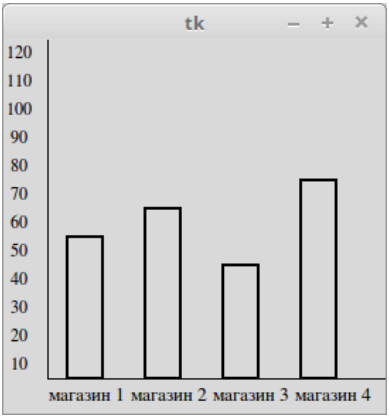
1. Створіть нову програму мовою Python, для порівняння цін на товар в різних магазинах. Максимальна вартість товару – 120 грн.. На вході в командному інтерфейсі у користувача записується список із 4 дійсних чисел від 0 до 120. На виході у графічному інтерфейсі маємо гістограму з показниками цін на даний товар за зразком (мал. 19).



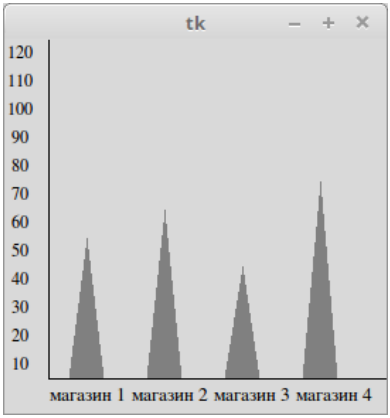
1)



2)



3)



4)

Мал. 19

ЛІТЕРАТУРА

1. Курс по библиотеке Tkinter языка Python [Електронний ресурс]. – Режим доступу: https://ru.wikiversity.org/wiki/Курс_по_библиотеке_Tkinter_языка_Python

2. Строки. Функции и методы строк [Електронний ресурс]. – Режим доступу: https://pythonworld.ru/tipy-dannyx-v-python/stroki-funkcii-i-metody-strok.html

3. Списки (list). Функции и методы списков [Електронний ресурс]. – Режим доступу: https://pythonworld.ru/tipy-dannyx-v-python/spiski-list-funkcii-i-metody-spiskov.html

4. Python 3.5.4 documentation [Електронний ресурс]. – Режим доступу: https://docs.python.org/3.5/

КОЗОЛУП Євгеній Вікторович

Програмування в школі. Мова Python

Навчальний посібник

9 клас

Рецензент:

Бабенко І.В. – вчитель інформатики Сумської спеціалізованої школи I-III ступенів №17, м. Суми, Сумської області, спеціаліст I категорії.

ДЛЯ ЗАМІТОК