

Навчальний посібник по Python
Івановського Олексія Анатолійовича
Доцента кафедри «Конструювання машин» НТУУ(КПІ) Київського
політехнічного інституту (Україна).

ВСТУП

Навчальна дисципліна "Інформатика" належить до циклу професійної підготовки навчального плану підготовки бакалавра циклу навчальних дисциплін професійної та практичної підготовки (за вибором студентів).

Предмет навчальної дисципліни – процес навчання і дозволить використовувати знання основ програмування при розробці інформаційних технологій, щозараз охоплюють майже всі сфери життя і діяльності та відіграють важливу.

Метою навчальної дисципліни є формування у студентів **здатностей:**

- застосовувати професійні знання й уміння на практиці;
- адаптуватися до різних професійних ситуацій, проявляти творчий підхід, ініціативу;
- аналізувати проблеми, ставити постановку цілей і завдань, виконувати вибір та вирішення проблем;
- вирішувати проблеми в професійній;

Розвиток сучасних технологій неможливий без використання комп'ютерної техніки та програмного забезпечення. Підготовка фахівців вимагає великих знань та навичок володіння комп'ютерною технікою, а також знання основ алгоритмізації та програмування мовами високого рівня.

Python – це універсальна мова, що широко використовується в усьому світі для самих різних цілей – бази даних і оброблення текстів, вбудовування інтерпретатора в ігри, програмування GUI, швидке створення прототипів (RAD) програмування Internet і Web додатків – серверних (CGI), клієнтських (роботи), Web- серверів і серверів додатків.

Серед переваг мови Python можна виділити переносимість написаних програм, на комп'ютери різної архітектури та з різними операційними системами, лаконічність запису алгоритмів, можливість отримати ефективний код програм за швидкістю виконання. Зручність мови Python ґрунтується на тому, що вона є мовою високого рівня, має набір конструкцій структурного програмування та підтримує модульність. Гнучкість та універсальність мови Python забезпечує її широке розповсюдження.

Навчальний посібник орієнтований на програмістів початківців, що мають початкові поняття про основи алгоритмізації. Кожен розділ посібника містить відповідні приклади, що представляють собою програми, що написані

мовою Python. Приклади допоможуть ефективно засвоїти основи програмування мовою Python.

У першому розділі підручника розглянуто архітектуру та структуру комп'ютерів, основні відомості про апаратне та програмне забезпечення, властивості алгоритмів, приділено увагу графічному способу подання алгоритмів, наведено умовні графічні позначення у схемах алгоритмів, правила графічного запису алгоритмів та базові структури алгоритмів. Крім цього, наведено математичні основи ЕОМ, форми та формати подання даних, основні системи числення, правила та приклади переведення як цілих так і дробових чисел з однієї системи числення в іншу.

У другому розділі розглядаються основні засоби програмування в мові Python. Особливу увагу приділено простим та складним типам даних.

У третьому розділі розглянуто основні алгоритмічні конструкції (умови, цикл).

Четвертий розділ присвячено елементам функціонального та модульного програмування. Визначено поняття рекурсії та наведено детальні приклади роботи рекурсивних функцій.

У п'ятому розділі описано роботу з файлами і каталогами.



Зміст

Розділ 1: Підготовка до виконання програми

Огляд

Програмування — це створення інструкцій, які виконує комп'ютер. Інструкції можуть показати комп'ютеру вивести рядок , внести дані в інтернеті або розрахувати вміст файлу та зберегти його в базі даних них. Ці інструкції називаються кодом. Програмісти пишуть код на різних мовах програмування. Раніше програмування було на багато складніше, оскільки програми були змушені використовувати вкрай складні низькорівніві мови програмування. Коли мова програмування є низькорівнева, це означає, що вона ближче до двоїчної системи записів (у нулях і одиницях), чим високорівнева мова програмування (мова, яка більше нагадує англійську), і тому її складно розуміти. Python — це мова програмування з відкритим вихідним кодом, створена голландським програмістом Гвідо ван Россумом та названий на честь британської трупи коміків «Монті Пайтон» (Monty Python). Одним з ключових ідей ван Россума було те, що програмісти витрачають більше часу на читання коду, ніж його написання, тому вирішив створити мову яка легко читається. Python є однією з найпопулярніших і найпростіших в освоєнні мов програмування у світі. Він працює на всіх основних операційних системах та комп'ютерах і застосовується скрізь, де тільки можна – від створення веб-серверів до настільних програм. Завдяки популярності цієї мови, на програмістів Python сьогодні є великий попит.

Налаштування середовища програмування

Python — мова високого рівня. Це значить що можна уникнути прямої взаємодії з регістрами, адресами пам'яті, стеками викликів тощо. Комп'ютер розуміє інструкції - компілятору та інтерпретатору. Інтерпретатор перекладає одну інструкцію (або оператор) за раз у машинний код. Але компілятор бере всю програму, а потім перекладає її на машинну мову за один раз. Для інтерпретації цих програм потрібен інтерпретатор Python. Отже, вам потрібно отримати інтерпретатор і встановити його, перш ніж писати свої програми. Вам потрібно вибрати правильний інтерпретатор на основі вашої операційної системи.

Використання командного рядка

Ви можете використовувати текстовий редактор, щоб написати програму на Python. Наприклад, можна використовувати блокнот. Але перш ніж це зробити,

давайте почнемо з простого командного рядка, щоб перевірити, чи інстальовано Python у вашій системі. Отже, відкрийте командний рядок і введіть python (див. малюнок):

```
Командний рядок - python
Microsoft Windows [Version 10.0.19044.2006]
(c) Корпорація Майкрософт. Усі права захищені.

C:\Users\Admin>python
Python 3.9.0 (tags/v3.9.0:9cf6752, Oct  5 2020, 15:34:40) [MSC v.1927 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

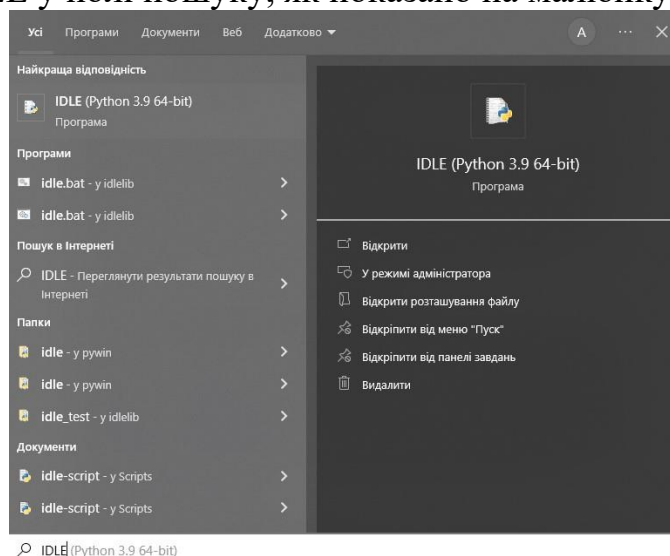
Якщо на вашому комп'ютері інстальовано Python, ви помітите детальну інформацію, як показано на малюнку . Тепер ви можете використовувати кілька простих команд для подальшої перевірки. Кожного разу, коли ви вводите команду та натискаєте клавішу Enter (або Return), оператор перевірятиметься.:
C:\Users\python

```
>>> 7+12
19
>>> 7<12
True
>>>
```

Якщо ввести exit() (або або Ctrl-Z плюс Return), то можна вийти з оболонки

Використання IDLE

Також можливо запустити IDLE, щоб отримати оболонку python, де можливо виконувати команди python. Наприклад, щоб отримати IDLE у Windows, ви можете ввести IDLE у полі пошуку, як показано на малюнку :



Що таке оболонка? Простіше кажучи, це середовище, яке використовується для запуску інших програм. Ми можемо використовувати оболонку як для

інтерфейсу командного рядка, так і для графічного інтерфейсу користувача (GUI). Але зазвичай ми використовуємо його для посилання на інтерфейс командного рядка операційної системи (ОС). Розробники часто називають терміни «оболонка» та «термінали» в одному контекст

Ця оболонка очікує отримання певної команди від користувача, потім виконує команду, а потім відображає результат. Після завершення цього циклу він очікує на отримання наступної команди від користувача. Введіть `print("Hello World!")` після `>>>` в оболонці та натисніть клавішу Enter. Ви можете негайно побачити результат Hello world у наступному рядку.

Щоб виконати деякі основні команди або перевірити, чи готові ви до програмування на Python, цей процес підходить. Але проблема в тому, що коли виходите з оболонки, то втрачаємо всі ці команди. Тому повинні використовувати текстовий файл, щоб написати програму на Python і зберегти файл із розширенням `.py`. Файл із розширенням `.py` називається сценарієм python.

Python постачається з програмою IDLE (скорочення від `interactive development environment` – інтерактивне середовище розробки). Це інструмент, який використовується для розробки програм простим, швидким і надійним способом. Різні типи IDE надають різні функції. Такі як:

- інструменти інтеграції забезпечують інструменти мови програмування та скриптів ,
- інтелектуальний редактор коду надає помічників кодингу, таких як попередження помилок, які допоможуть автоматично завершити код, забезпечує зручну навігацію та відстеження коду у великих проектах з великою кількістю файлів та папок з підпроектами,
- компілятор забезпечує компіляцію там, де потрібна мова програмування, налагоджувач забезпечує налагодження та перевіряє скомпіловані двійкові файли, підтримка різних мов програмування та скриптів зробить середовище IDE повнозадачним, підтримуючи кілька пов'язаних мов програмування та мов скриптів, які можуть бути в одному проекті ,
- інтерактивна консоль надає інтерактивну оболонку або консоль для видачі команд, пов'язаних з проектом, та отримання результатів у термінальному режимі ,

- контроль версій забезпечує версійність коду та полегшує відстеження змін ,плагіни надають безліч корисних функцій як доповнення до існуючої IDE.

Також IDE включає інструменти, спеціально призначені для розробки програмного забезпечення. Такі інструменти містять:

- Редактор, призначений для обробки коду (наприклад, з підсвічуванням синтаксису та автозавершення).
- Засоби складання, виконання та налагодження.
- Систему контролю версій.

Також IDLE ми будемо називати інтерпретатором мови Python. Інтерпретатор мови Python – це програма, яка може читати інструкції програми на Python та їх виконувати. Інтерпретатор може використовуватись у двох режимах: інтерактивному та сценарному. В інтерактивному режим інтерпретатор чекає від вас, що ви наберете інструкцію Python на клавіатурі. Після цього інтерпретатор її виконує та чекає від вас наступного інструкції. У сценарному режимі інтерпретатор читає вміст файлу, що містить інструкції Python. Такий файл називається програмою Python або сценарієм Python. Інтерпретатор виконує кожен інструкцію в програмі Python під час читання файлу.

Коли інтерпретатор Python розпочинає роботу в інтерактивному режимі, ми бачимо, що в консольному вікні буде виведено щось на зразок:

```
Python 3.9.0 (tags/v3.9.0:9cf6752, Oct 5 2020, 15:34:40)
[MSC v.1927 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more
information.
>>>
```

Такий символ означає >>> що інтерпретатор чекає набір інструкцій Python. Наприклад можна ввести таку інструкцію:

```
>>> print('Доброго ранку!')
```

Цю інструкцію можна представити як команду, яку надсилаємо до інтерпретатора.

Python. Якщо набрати цю інструкцію точно, як показано, то на екрані буде надруковано

повідомлення :

```
>>> print('Доброго ранку!')
Доброго ранку!
>>>
```

Це підказка про те, що інтерпретатор очікує від вас набір інструкцій Python. Після виведення повідомлення знову з'являється підказка >>>, яка говорить про

те, що інтерпретатор чекає наступної інструкції. Якщо в інтерактивному режимі набрати інструкцію неправильно, інтерпретатор виведе на екрані повідомлення про помилку. Це робить інтерактивний режим дуже корисним під час вивчення мови Python. У міру вивчення нових компонентів мови їх можна випробовувати в інтерактивному режимі та отримувати безпосередній зворотний зв'язок від інтерпретатора.

```
>>> print('Доброго ранку!'}}
SyntaxError: closing parenthesis '}' does not match opening
parenthesis '('
>>>
```

Інтерактивний режим є корисним для тестування програмного коду. Водночас інструкції,

які вводимо в інтерактивному режимі, не зберігаються як програма. Вони просто виконуються і їх результати відображаються на екрані. Якщо нам потрібно зберегти список інструкцій Python як програми, то ці інструкції слід зберегти у файлі. Для того, щоб виконати цю програму, інтерпретатор Python слід запустити у сценарному режимі. Наприклад, припустимо, нам потрібно написати програму Python, яка виводить на екран наведені далі три рядки тексту:

```
бігати
стояти
спати
```

Для написання програми слід створити файл у простому текстовому редакторі, такому як

Блокнот (який встановлений на всіх комп'ютерах з Windows), що містить наступні

інструкції:

```
print ('бігати')
print ('стояти')
print ('спати.')
```

При збереженні програми Python їй слід дати ім'я з розширенням `.py`, яке ідентифікує

її як програму Python. Наприклад, наведену вище програму можна зберегти під назвою `test.py`. Щоб виконати програму, слід перейти в каталог, в якому збережен файл, та в командній оболонці операційної системи набрати команду:

```
python test.py
```

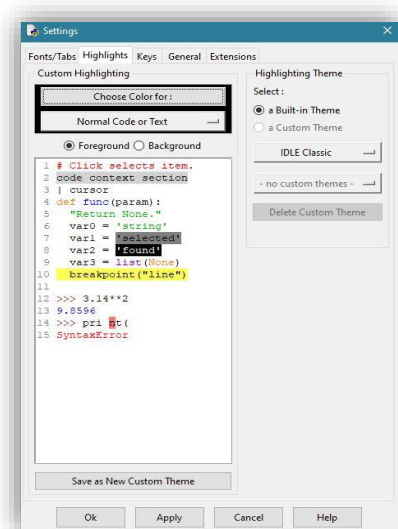
Ця команда запустить інтерпретатор Python у сценарному режимі, внаслідок чого він виконає інструкції у файлі `test.py`. Коли програма закінчиться, інтерпретатор Python припинить свою роботу.

Середовище IDLE також має вбудований текстовий редактор з функціональними можливостями, спеціально призначені для того, щоб допомагати вам писати свої програми

на Python. Наприклад, редактор IDLE "розцвічує" код таким чином, що ключові слова та інші частини програми розмічуються на екрані спеціальним кольором. Це спрощує читання програм. Програмний код, що вводиться у вікні редактора, а також у вікні Python оболонки, виділяється кольором наступним чином:

- ключові слова Python відображаються в оранжевому кольорі;
- коментарі – у червоному кольорі;
- рядкові літерали – у зеленому кольорі;
- визначені у програмі імена, такі як імена функцій та класів, - у синьому кольорі;
- вбудовані функції – у фіолетовому кольорі.

Редактор IDLE має функціональні можливості, які допомагають підтримувати у програмах Python однакове виділення відступами. Найкориснішою із цих функціональних можливостей є автоматичне виділення відступом. Коли написан рядок, який закінчується двокрапкою, зокрема вираз `if`, перший рядок циклу або заголовок функції, а потім натискається клавіша `<Enter>`, редактор автоматично виділяє відступом рядки, які набираються згодом. За замовчанням у середовищі IDLE для кожного рівня виділення відступом як відступ використовуються чотири пробіли. Кількість пробілів можна змінити, вибравши в меню Options команду Configure IDLE. Перейдіть на вкладку Fonts/Tabs (Шрифти/Відступи) діалогового вікна, і побачим панель повзунка, який дозволяє змінити кількість пропусків, що використовуються як ширина відступу. Однак в Python чотири пробіли є стандартною шириною відступу, рекомендується залишити поточне налаштування.



У вікні редактора можна зберегти поточну програму шляхом вибору відповідної команди з меню File:

- Save (Зберегти);
- Save As (Зберегти як);
- Save Copy As (Зберегти копію як).

Команди Save та Save As працюють так само, як і в будь-якій програмі Windows.

Команда

Save Copy As працює як Save As, але залишає вихідну програму у вікні редактора. Після того, як програма набрана в редакторі, її можна виконати, натиснувши клавішу <F5> або вибравши в меню Run (Виконати) команду Run Module (Виконати модуль). Якщо після останнього внесення зміни до вихідного коду програма не була збережена то з'явиться діалогове вікно . Треба натиснути кнопку ОК, щоб зберегти програму.



Розділ 2: Коментарі, змінні та оператори

Використання коментарів

Використання коментарів у програмі є стандартною практикою. Ці коментарі можуть допомогти іншим краще зрозуміти ваш код.

У Python ви бачите такі параметри для коментарів:

Випадок : однорядкові коментарі з використанням тегів `#`. Ось приклад: `# Це однорядковий коментар`.

Коментарі - це прості нотатки або деякі тексти. Ви використовуєте їх для читачів, але не для інтерпретатора Python. Інтерпретатор Python ігнорує текст всередині блоку коментарів. У індустрії програмного забезпечення багато технічних рецензентів переглядають ваш код. Коментарі допомагають зрозуміти логіку програми. Розробник може забути логіку через кілька місяців. Ці коментарі можуть допомогти йому пригадати свою логіку. Вважається що за краще використовувати багато однорядкових коментарів із використанням тегів `#`.

Приклад

Коли інтерпретатор Python бачить коментар, він ігнорує його.

```
# Перевірка того, чи 9 більше 8
print(9>8) '''
# Тепер я показую кілька однорядкових коментарів #
Ділення 9 на 3
# І друк результату
Print(9/3)
Вивід(Output)
True
6
```

Вступ до змінних

У математиці часто пишуть щось на зразок $x=10$. Якщо x є змінною, яка представляє число 10. Python працює так само, за винятком того, що тут можливо представляти як числові, так і нечислові значення за допомогою змінних.

Рядки проти чисел

Рядки - це тексти. Вони дуже поширені в програмуванні на Python. Ви вже бачили використання рядків у попередніх демонстраціях. Наприклад, щоб надрукувати Hello World! Друкується наступне:

```
print("Hello World!")
```

Також можливо використовувати рядкову змінну на зразок наступного: `my_text = "Hello World!"` а потім надрукуйте результат, використовуючи такий рядок коду: `print(my_text)`

Зауважте, що в обох випадках друкується Hello World! всередині подвійних лапок. Це загальний формат для оголошення рядкової змінної.

```
>>> print("Доброго ранку.")
```

Ви можете використовувати одинарні або подвійні лапки для друку вихідних повідомлень. Обидва в порядку. Python не розрізняє одинарні та подвійні лапки. Вам потрібно пам'ятати простий факт: ви берете рядковий літерал у відповідні одинарні або подвійні лапки. Одинарні лапки для друку простих повідомлень і подвійні лапки для форматованих рядків.

Для того щоб надрукувати число без змінної, ви можете просто ввести число у функції `print()` наступним чином.

```
print(1) #Друкує 1
print(9.2) # Виводить 9.2
print(-3,531) # Виводить -3,531
Ви також можете використовувати числові змінні, наприклад:
# Використання змінних
int=125
print(my_int) # Виводить 125
float=25.763
print(my_float) # Виводить 25.763
# Різниця між числами та рядками
print("1"+"2") #Виводить 12
print(1+2) # 3
```

int float complex

У Python підтримуються чотири різновиди чисел:

- цілі числа – `int(2,3,26,89,100)`
- числа з плаваючою крапкою або дійсне – `float(2.5, 34.65, 345,19)`
- комплексні числа - `complex(6+8a, 4l-23d)`
- та логічні значення – `True and False`

Цілі числа мають найпростіший тип, далі йдуть числа з плаваючою крапкою або дійсне та найскладніший тип - комплексні числа. Таким чином, якщо в операції беруть участь ціле число і дійсне, то ціле число буде автоматично перетворено в дійсне число, а потім проведена операція над дійсними числами. Результатом цієї операції буде дійсне число.

Для перевірки типу будь-якого значення та змінною можна використовувати функцію `type()`:

```
>>> a=8
>>> b =19.2
>>> c =6+4s
>>> type(a); type(b); type(c)
<class 'int'> <class 'float'> <class 'complex'>
```

Також можна перетворювати значення будь-якого типу за допомогою відповідних функцій `int()`, `float()` або `complex()`:

```
>>> a=8
>>> int(a)
8
>>> float(a)
8.0
>>> complex(a)
(8+0j)
```

Комплексне число не можна перетворити за допомогою функцій `int()` і `float()` до цілого чи дійсного. Функція `int()` відкидає дробову частину числа, а не округляє його:

```
>>> a =8.7
>>> int(a)
8
>>> b = -7.2
>>> int(b)
-7
```

Арифметичні операції з числами та вбудовані функції з одним та двома аргументами

$x + y$	Додавання
$x - y$	Віднімання
x / y	множення
$x // y$	ділення
$x \% y$	Остаток від ділення:
$x ** y$	Зведення x до ступеня y

abs(x)	Модуль числа x
round(x)	Округлення
round(x, n)	Округлює число x до n знаків після коми:
pow(x, y)	Праця з степенями
divmod(x, y)	видає два числа: приватне та залишок
=	присвоєння

Також при праці з числами ми використовуємо функції з послідовностями або довільним числом аргументів:

max(a, b,...) - Максимальне число з двох чи більше

min(a, b,...) - Мінімальна кількість із двох або більше

sum(seq) - Відсортований список

Правила присвоєння

Використовуйте оператор присвоєння (наприклад, =), щоб присвоїти значення змінній. Далі ви використовуєте описові назви для ваших змінних. Намагайтеся уникати односимвольних імен змінних, за винятком деяких циклів або функцій. Ви будете знати про цикли та функції та використовуватимете їх у наступних розділах. У деяких фрагментах коду ви можете побачити щось на зразок x=7 . Але в реальній програмі виберіть краще ім'я. Наприклад, щоб призначити ідентифікаційний номер (скажімо, 7) працівнику організації, ви можете написати щось на кшталт man_id=7

Назва вашої змінної не повинна суперечити ключовим словам Python. Ключове слово є зарезервованим словом, ви не можете використовувати його як звичайні ідентифікатори.

Зарезервовані слова:

False , await, else, import, pass, None, break, except, in, raise, True, class, finally, is, return, and, continue, for, lambda, try, as, def, from, non, local, while, assert, del, global , not, with, async, elif, if, or, yield.

У Python, на відміну багатьох інших комп'ютерних мов, не потрібно ні оголошувати тип змінної, ні включати обмежувач кінця рядка. Рядок програми завершується там, де вона завершується. Змінні створюються автоматично під час першого присвоєння. Змінним Python можуть надаватися будь-які об'єкти. Нижче наведено приклад:

```
>>> x = "Hello"
```

```
>>> print(x)
Hello
>>> x = 5
>>> print(x)
5
```

Змінна `x` спочатку посилається на рядковий об'єкт "Hello", а потім на об'єкт цілого числа 5. Нове привласнення перевизначає усі попередні. Команда `del` видаляє змінну. При спробі вивести вміст змінної після її видалення відбувається помилка, якби змінна ніколи не створювалася:

```
>>> x = 5
>>> print(x)
5
>>> del x
>>> print(x)
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
NameError: name 'x' is not defined
>>>
```

В іменах змінних враховується регістр символів; вони можуть містити будь-які алфавітно-цифрові символи, а також символи підкреслення, але повинні починатися з літери або символу підкреслення.

Оператори

Оператори — це спеціальні символи, які використовуються для виконання певних завдань або обчислень. Ці *оператори* працюють з деякими значеннями, які називаються *операндами*. Наприклад, у виразі: $2+3$, $+$ є оператором, а 2,3 є операндами.

Наприклад присвоїти значення змінній ми робимо за допомогою оператора присвоювання (`=`). Python підтримує багато інших операторів, які можна класифікувати наступним чином:

1. Арифметичні оператори
2. Оператори присвоєння
3. Оператори порівняння
4. Логічні оператори
5. Оператори ідентифікації
6. Оператори належності

Арифметичні оператори:

- $**$ — піднесення до степеня;
- $-x$ — унарний мінус;

- `/, //` – звичайне ділення, ділення з округленням вниз;
- `%` – остача від ділення;
- `*` – множення;
- `--` – віднімання;
- `+` – додавання.

```
a = 3
b = 2
c = a**b # c = 9
print("c = ", c)
```

Оператори порівняння

<code>==</code>	дорівнювати
<code>!=</code>	не дорівнює
<code>></code>	більше
<code><</code>	менше
<code>>=</code>	більше або дорівнює
<code><=</code>	менше або дорівнює

```
5 < 6
a = 8
b = 3
c = a == b # c = False
c = a < b # c = False
c = a != b # c = True
```

Оператори присвоєння

“`=`”

ім’я = значення

- ім’я (змінна), яке потрібно зв’язати зі значенням (об’єктом) ;
- значення, яке присвоюється імені , яке може бути виразом, одиночним значенням чи списком.

```
a = 5
b = 7
c = a < b < 8 # c = True
c = a != b != 8 # c = True
c = a == b == 8 # c = False
```


Логічні оператори

and	Обидві умови мають бути дотримані
or	Будь-яка з умов має бути виконана

```
>>> num1 = 34
>>> num2 = 8.5
>>> num1 > 12 and num2 != 12
True
>>> num1 == 34 and num2 >= 8
True
>>> num1 != 34 and num2 != 12
False
>>> num1 <= 12 and num1 == 0
False
>>> num1 != 34 or num2 != 12
True
>>> num1 < 1 or num2 > 9.6
False
>>> str1 = "a"
>>> str2 = "b"
>>> str1 < "c" and str2 != "a"
True
```

Оператори ідентифікації

Оператор ідентифікації python порівнює розташування об'єктів у пам'яті. Python надає два вбудованих оператора ідентифікації.

Оператор ідентифікації	Принцип
is	Повертає True, якщо два об'єкти вказують на одне й те саме місце пам'яті
is not	Повертає True, якщо два об'єкти не вказують на те саме місце пам'яті

```
num = 5.0 if (type(num) is int): print("True") else:
print("False")
False
num = 5.0 if (type(num) float): print("True") else:
print("False")
True
```

Оператори належності

У Python існує два оператори приналежності: `in` та `not in`. Ці оператори перевіряють, чи є значення частиною якоїсь послідовності: рядки, списку, кортежу, словника.

`in` - Повертає `True`, якщо значення є у послідовності, інакше повертає `False`.

```
a = "b" in "abc"
```

```
print(a)
```

```
true
```

`not in` - Повертає `True`, якщо значення немає у послідовності. Якщо значення є у послідовності, то повертає `False`.

```
a = 5 not in (2, 3, 4)
```

```
print(a)
```

```
true
```

Арифметические операторы

У Python підтримуються чотири різновиди чисел:

- цілі числа – `int(2,3,26,89,100)`
- числа з плаваючою крапкою або дійсне – `float(2.5, 34.65, 345,19)`
- комплексні числа - `complex(6+8a, 4l-23d)`
- та логічні значення – `True` and `False`

Цілі числа мають найпростіший тип, далі йдуть числа з плаваючою крапкою або дійсне та найскладніший тип - комплексні числа. Таким чином, якщо в операції беруть участь ціле число і дійсне, то ціле число буде автоматично перетворено в дійсне число, а потім проведена операція над дійсними числами. Результатом цієї операції буде дійсне число.

Для перевірки типу будь-якого значення та змінною можна використовувати функцію `type()`:

```
>>> a=8
```

```
>>> b =19.2
```

```
>>> c =6+4s
```

```
>>> type(a); type(b); type(c)
```

```
<class 'int'> <class 'float'> <class 'complex'>
```

Також можна перетворювати значення будь-якого типу за допомогою відповідних функцій `int()`, `float()` або `complex()`:

```
>>> a=8
>>> int(a)
8
>>> float(a)
8.0
>>> complex(a)
(8+0j)
```

Комплексне число не можна перетворити за допомогою функцій `int()` і `float()` до цілого чи дійсного. Функція `int()` відкидає дробову частину числа, а не округляє його:

```
>>> a = 8.7
>>> int(a)
8
>>> b = -7.2
>>> int(b)
-7
```

Арифметичні операції з числами та вбудовані функції з одним та двома аргументами

$x + y$	Додавання
$x - y$	Віднімання
x / y	множення
$x // y$	ділення
$x \% y$	Остаток від ділення:
$x ** y$	Зведення x до ступеня y
<code>abs(x)</code>	Модуль числа x
<code>round(x)</code>	Округлення
<code>round(x, n)</code>	Округлює число x до n знаків після коми:
<code>pow(x, y)</code>	Праця з степенями
<code>divmod(x, y)</code>	видає два числа: приватне та залишок
<code>=</code>	присвоєння

Також при праці з числами ми використовуємо функції з послідовностями або довільним числом аргументів:

`max(a, b,...)` - Максимальне число з двох чи більше

`min(a, b,...)` - Мінімальна кількість із двох або більше

`sum(seq)` - Відсортований список

Розділ 3: Загальні типи даних

У цьому розділі показано використання деяких поширених типів даних і вбудованих функцій у Python. Функція — це блок коду, який допомагає вам виконати заплановану роботу.

У Python є багато типів даних (рядками, числами та логічними значеннями). Це найпоширеніші типи даних.

Демонстрація коду з рядками

Наприклад, `\n` представляє escape-символ. Якщо ви використовуєте його в рядку, символ «п» відображається, а текстовий курсор переміщується вниз до наступного рядка. Ось вам приклад:

```
>>>
print("Hello\nWorld!")
Hello
World!
```

Так само, якщо ви хочете надрукувати табуляцію між літерами, ви можете використовувати `\t`.

Приклад

```
>>>
print("Hello\tWorld!")
Hello World!
```

`\n` і `\t` дуже поширені. Крім цього, можете побачити використання `\b` для зворотного простору. Але поведінка `\b` і `\a` може відрізнитися. Для ілюстрації, кожен із них добре працює в командній оболонці Python, але вони не працюють в IDLE.

```
>>>
print("Abc\bcd")
Abd
```

Тепер виконайте той самий рядок коду в IDLE. У цьому випадку такий результат:

```
>>>
print("Abc\bcd")
Abcd
```

Можно надрукувати рядок із зворотною косою рисою. У цьому випадку можна використовувати необроблені рядки, додавши `r` (або `R`) перед подвійними лапками наступним чином:

```
>>>
print(r"Hello\World
") Hello\World
```

Тут є можливість вставити символ (який є подвійним в цьому випадку) після зворотної риски. Тепер ви можете отримати наступний результат:

"Hello World!"

Крім того, можна використовувати подвійні лапки всередині одинарних лапок таким чином:

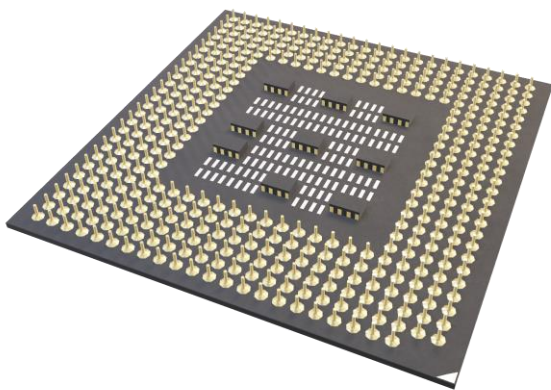
```
>>> print(' "Hello
World!" ') "Hello
World!"
```

Демонстрація коду з числами

Подібно до рядків, щоб надрукувати число без змінної, можна ввести число всередині функції print().

```
print(1)
print(5.7)
print(-6.789)
```

Можно виконувати як основні, так і складні арифметичні операції в операторі друку.



#Виконання деяких простих і складних арифметичних операцій

```
print(1+2)
print(10 - 3)
print( 25* 3)
print(12.88/4)
print(1+2*3)
print((1+2)*3)
```

Використання змінних для чисел:

Використання змінних

```
my_int=125
print(my_int)
print(my_float)
```

Для рядкових змінних операторів плюс (+) об'єднує рядки, а для чисел він їх додає:

#Різниця між числом і рядками

```
my_string1="10"
my_string2 =
"22"
my_int1=10
my_int2=22
print(my_string1+my_string2)
print(my_int1+my_int2)
```

Якщо використовувати оператор +, потрібно конвертувати їх у той самий тип даних. Інакше ви побачите помилки.

Є поняття f-рядків. Ви можете поставити букву «f» перед рядком, а потім додати в нього змінну. Мені подобається такий підхід, тому що його легко читати та розуміти. Щоб вставити змінну в рядок, потрібно загорнути її у фігурні дужки. Давайте розглянемо наступний приклад:

```
user_name="Alexei"
print(f"Привіт,
{user_name}!")
    Привіт, Alexei!
# Знаходження
максимуму
    print(max(1,2,3,4,5))
#Знаходження мінімуму
```

```
print(min(1, 2, 3, 4, 5))
```

Вивід:

5

1

Функція `max()` використовується для пошуку найбільшого серед чисел, а функція `min()` використовується для пошуку найменшого серед чисел.

Прийняття даних користувача

Для цього використовуємо функцію `input()`. Використовуючи цю функцію, можемо передавати підказку. Підказка може допомогти користувачеві дізнатися про інформацію, яку йому потрібно надати. Наприклад функція `input()` призупиняє програму та чекає на введення користувача. Після введення даних можна продовжувати працювати. Напишемо програму, де користувач може вказати своє ім'я та вік. Зберігаємо ці вхідні дані в деяких змінних. Потім друкуємо інформацію з додатковими повідомленнями.

```
# Введіть назву
user_name = input("Введіть
ім'я:") #Введіть ім'я
user_age = input("Ваш
вік:")
print("Привіт, ", user_name, "! твій вік",
user_age)) print(f"Привіт, {user_name}!
твій вік {user_age}")
print("Привіт, " + user_name + "! твій вік " +
str(user_age))
```

Ім'я:Олексій

Вік:33

Привіт, Олексій ! твій вік 33

Привіт, Олексій ! твій вік 33

Привіт, Олексій ! твій вік 33

Розділ : Оптимальна праця з кодом та використання різноманітних операторів.

Оптимальна праця з кодом та використання різноманітних операторів є невід'ємною частиною програмування. Часто потрібно перевірити певні умови в програмі. На основі цих умов керуємо процесом виконання програми.

Часто виконуємо умовні тести, використовуючи різні оператори if. Наприклад, у програмі можемо використовувати оператор if, ланцюжок if-else або if-elif-else. Вибір залежить від того, скільки умов потрібно перевірити в конкретній точці.

Використання оператора if

Якщо її значення більше 10, програма може сказати, що поточне значення змінної більше 10. Створимо новий файл і введемо в нього наступне:

```
a = 11
if a > 10:
    print("Значення a більше ніж 10.")
```

Використання операторів if-else

Але якщо *a* менше або дорівнює 10, програма НЕ друкуватиме вихідні дані. (Щоб перевірити це, можемо змінити значення *a* на значення менше 10 і виконати програму знову.)

Якщо додати ще кілька рядків до попередньої демонстрації та використання оператора else..

```
else:
    print("Значення a менше або дорівнює 10.")
    Щоб перевірити це, створимо a=5 і запустимо таку програму:
a = 5
if a > 10:
    print("Значення a більше ніж10.")
else:
    print("Значення a менше або дорівнює 10.")
```

Використовуємо один знак рівності (=), щоб присвоїти значення змінній. Але ми використовуємо (==) для оператора рівності. Він повертає True, якщо значення обох сторін оператора збігаються; інакше повертає False.

Використання elif з інструкцією if-else

Дуже часто програма перевіряє дві умови. У такому випадку використання

операторів if-else недостатньо. Програма має вивести, чи число більше за 10, чи менше за 10, чи дорівнює 10. Використовуємо if-elif-else для цього сценарію.

```
user_input = input("Enter a valid  
number only:") # Skipping the  
validation of the user's input  
a =  
float(user_inp  
ut) if a > 10:  
    print("The number is greater  
than 10.") elif a == 10:  
    print("The number is  
equal to 10.") else:  
    print("The number is less than 10.")
```

Python виконує лише один блок із ланцюжка if-else або if-elif-else. Керування виконується в послідовному порядку, поки одна умова не стане True . Коли умова стає True, усі рядки з False.

Ітерація з використанням циклів

У програмуванні ітерація використовується для повторення. Коли пишете програму або реалізуєте алгоритм, може знадобитися повторювати певну частину коду, доки не буде виконано певну умову. Інструкції циклу допомагають перебирати ці частини коду

Мета ітерації

Повторення та виконання одного і того ж блоку коду знову і знову називається ітерацією. Бесконечная ітерація, при якій блок коду виконується до цієї порції, поки не буде виконана яка-небудь умова. У Python невизначена ітерація виконується за допомогою циклу while. Ітератор (від англ. iterator - перелічувач) - інтерфейс, що надає доступ до елементів колекції (масиву або контейнера) та навігацію по них. У різних системах ітератори можуть мати різні загальноприйняті назви. У термінах систем управління базами даних ітератори називають курсорами convenience. For now, let's start with while loops.

Цикл while

Цикл while продовжує виконуватися, доки виконується певна умова. Він має такий вигляд:

while “умова”:

інструкція

інструкція

інструкція

Обов’язкові умови:

- Цикл починається із зарезервованого ключового слова while
- Після зазначеної умови стоїть двокрапка (:). У середині циклу while може бути багато операторів.
- Робити відступ у рядках.
- Перевірка умову перед введенням цього блоку операторів. Якщо умова виконується, елемент керування може увійти в блок і виконати ці оператори.
- Для виходу з циклу while, умова має бути хибною. В іншому випадку блок операторів усередині циклу продовжить виконання.
- Коли умова стає хибною, блок операторів усередині циклу while більше не виконуватиметься.
- Якщо входите в цикл, але не задовольняєте критеріям виходу, то потрапляєте в пастку нескінченного циклу.

Цикл for

У Python існує ще один цикл, який називається for. У певних ситуаціях цей цикл забезпечує зручний спосіб вираження кроків циклу while. Для написання циклу з лічильником повторень застосовується інструкція for. У Python інструкція for призначена до роботи з послідовністю значень даних. Коли ця інструкція виконується, вона повторно виконується кожного значення послідовності.

```
for змінна in (значення, значення)
```

```
    інструкція
```

```
    інструкція
```

```
    інструкція
```

Будемо позначати for як вираз. У виразі for змінна це ім'я змінної. У середині дужок знаходиться послідовність розділених комами значень. У другого рядка, розташовується блок інструкцій, який виконується під час кожної ітерації циклу. Інструкція for виконується в такий спосіб: змінній надається перше значення у списку, а потім виконуються інструкції, які розташовані в блоці. Потім змінною

присвоюється таке значення у списку, і інструкції у блоці виконуються знову. Цей процес триває доти, доки змінної не буде присвоєно останнє значення у списку.

```
print(от 1 до 5 )  
for num in [ 1, 2, 3, 4, 5):  
    print (num)
```

Під час першої ітерації циклу для змінної `num` присвоюється значення 1, а потім виконується

інструкція у рядку 3 (друкує значення 1). Під час наступної ітерації циклу змінної `num` присвоюється значення 2, і виконується інструкція у рядку 3 (друкує значення 2). Цей процес триває до того часу, поки змінної `num` не присвоюється останнє значення у списку. Оскільки у списку лише п'ять значень, цикл зробить п'ять ітерацій.

Цикли `for` можна використовувати для переміщення даних між змінною ними та об'єктами, що ітеруються. Наприклад, можна використовувати два цикли `for`, щоб взяти всі рядки з двох різних списків, зробити великими всі символи в цих рядках і помістити змінені рядки до нового списку.

Використання оператора `break`

Використовуємо інструкцію `break` — інструкцію з ключовим словом `break`, — чтобы прекратить цикл. Наступний цикл викониться сто раз.

```
i = 1  
while True:  
    if i > 100: break  
    print(i)  
    i += 1
```

Якщо ви додасте інструкцію `break`, цикл викониться лише один раз. Показано значення `True`. В цьому випадку вирази всередині циклу стануть виконуватися бескінченно. Однак використання оператора `break` прериває виконання циклу, як тільки він буде виконано 100 раз.

Використання оператора `continue`

У Python є ще один оператор який називається `continue`. Можимо використовувати інструкцію щоб продовжити, або перервати поточну ітерацію циклу чи продовжити з наступними ітераціями. Наприклад потрібно вивести всі числа від 1 до 5, крім числа 3. Це можна створити, використовуючи цикл `for` та інструкцію `continue`.

```
for i in range(1, 6):
```

```
if i == 3:
    continue
print(i)
```

В цьому циклі, коли змінна **i** приймає значення 3, виконується інструкція `continue` — тоді замість того, щоб повністю завершитися, як в випадку з ключевим словом `break`, цикл продовжує працювати. Він переходить в наступні ітерації, пропускаючи код, який повинен бути виконаний. Коли змінна **i** приймає значення 3, Python виконує інструкцію продовжити, а не виводить число 3.

Вкладений цикл

Є можливість різними способами комбінувати цикли. Наприклад, можна помістити один цикл в інший, або створити цикл всередині. Немає ніяких обмежень по кількості циклів, які можна помістити всередині других циклов, хоча ці обмеження важливі. Коли цикл знаходиться всередині другого циклу, другий цикл є вкладеним у перший. В цьому випадку цикл, містить внутрішньо іншого циклу, називається зовнішнім, а вкладений цикл — внутрішнім. Коли є вкладений цикл, внутрішній цикл виконується перебор свого ітерованого об'єкта один раз для ітерації зовнішнього циклу.

```
for i in range(1, 3):
    print(i)
    for letter in ["a", "b", "c"]:
        print(letter)
```

Вкладений цикл `for` буде перебирати список `["a", "b", "c"]` настільки раз, скільки разів виконується зовнішній цикл. Якщо зробили так, щоб зовнішній цикл виконувався три рази, то і внутрішній цикл також перебирав свій список тричі.

```
list1 = [1, 2, 3, 4]
list2 = [5, 6, 7, 8]
added = []
for i in list1:
    for j in list2:
        added.append(i + j)
print(added)
```

Перший цикл виконує ітерування кожного цілого числа у списку `list1`. Для

кожного елемента в цьому списку другий цикл перебирає кожне число у власному об'єкті, що ітерується, потім додає його до числа з list1 і додає результат до списку added.

Терміни:

Нескінченний цикл: цикл, який ніколи не завершується.

Зовнішній цикл: цикл, який містить вкладений цикл.

Внутрішній цикл: цикл, вкладений в інший цикл.

Інструкція break: інструкція з ключовим словом break, що використовується для припинення циклу.

Інструкція continue: інструкція з ключовим словом continue, використовується, щоб перервати поточну ітерацію циклу і продовжити з наступної ітерації.

Ітерування (перебір): використання циклу для отримання доступу до кожному елементу об'єкта, що ітерується.

Змінна індексу: змінна, що зберігає ціле число, яке представляє індекс в об'єкті, що ітерується.

Цикл for: цикл, що перебирає об'єкт, що ітерується — наприклад, рядок, список, кортеж чи словник.

Цикл while: цикл, що виконує код доти, доки вираз приймає значення True.

Цикл: фрагмент коду, що безперервно виконує інструкції, поки задовольняє визначена в коді умова.

Додаткові типи даних

Тут показано використання деяких розширених типів даних, які дуже поширені в програмуванні Python.

Робота зі списками

Списки використовуються для зберігання послідовності елементів. У списку можливо мати як кілька елементів так і багато елементів. Також можна побачити наявність змішаних типів даних. Формально можна сказати, що список може містити послідовність об'єктів, які можуть походити від різних типів даних.

Визначається список приблизно так: list_name=[value1,value2,value3,..]

Розділяється значення квадратними дужками []. Таке розташування допомагає зберегти кілька значень в одній змінній.

Списки в Python є послідовністю, що видозмінюється , тобто їх елементи можуть змінюватись. Також – це впорядковані за розташуванням колекції об'єктів довільних типів, розмір яких обмежений. Отже, вираз у формі список [індекс] може з'являтися зліва оператора привласнення. Наприклад:

```
1 numbers = [1, 2, 3, 4, 5)
2 print(numbers)
3 numbers[0] = 99
4 print(numbers)
```

Інструкція у рядку 2 покаже:

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

На відміну від рядків, списки складаються не з символів, а з різних об'єктів (значень, даних), і заключаються в квадратні дужки []. Об'єкти відокремлюються один від друга за допомогою коми. Кожна, що зберігається у списку порція даних називається елементом. Значення, укладені у дужки та відокремлені комами, є елементами списку. Списки можуть складатися з різних об'єктів: чисел, рядків та навіть інших списків. В останньому випадку списки називають вкладеними. Ось деякі приклади списків:

```
[154 , 159 , 148 , 123 , 178]                # список
цілих чисел
[18.3 , 18.1 , 13.0 , 149. , 18.4]            # список
дійсних чисел
['Оксана', 'Олена', 'Максим']                 # список
рядків
['Львів', 'Дніпро', 164 , 17]                 # змішаний
список
[[2 , 0, 1], [0, 1, 0], [0, 0, 1]]            # список
списків
```

Як і над рядками, над списками можна виконувати операції з'єднання та повторення:

```
>>> [8, 'березня', 2022]+[19 , 'лютого', 2022]
[8, 'березня', 2022 , 19, 'лютого', 2022]
>>> [2, 3, 4]*2
[2, 3, 4, 2, 3, 4]
```

Створити список можна такими способами:

- за допомогою функції list ([<послідовність>]). Функція дозволяє конвертувати будь-яку послідовність до списку. Якщо параметр не вказано, створюється порожній список:

```
list()                #Створюємо порожній список
```

```

[]
list ("String")           #Перетворимо рядок на
список
['S', 't', 'r', 'i', 'n', 'g']
list((1, 2, 3, 4, 5))     #Перетворимо кортеж на
список
[1, 2, 3, 4, 5]

```

На відміну від рядків, списки змінюються послідовності. Якщо представити рядок як об'єкт у пам'яті, то коли над ним виконуються операції конкатенації та повторення, то цей рядок не змінюється, а в результаті операції створюється інший рядок в іншому місці пам'яті. У рядок не можна додати інший символ або видалити існуючий, не створивши нового рядка. З списком справа інакше. При виконанні операцій нові списки можуть не створюватиметься, а змінюватиметься безпосередньо оригінал. Зі списків можна видаляти елементи, додавати нові. При цьому слід пам'ятати, багато що залежить від того, як ви розпоряджаєтеся змінними. Символ у рядку не можна змінити, елемент списку можна.

Робота з кортежами

Кортежі є ще одним важливим типом даних, схожим на списки. Кортеж — це контейнер, який зберігає об'єкти у порядку. У відміну від списків, кортежі незмінні, тобто їх вміст не можна змінити. Як тільки створюється кортеж, значення будь-якого його елемента вже не можна змінити та не можна додавати та видаляти елементи. Кортежі визначаються за допомогою круглих дужок. Елементи в кортежі повинні бути розділені комами. Для створення кортежів використовують один із двох варіантів синтаксису. 1:

```

my_tuple = tuple()
my_tuple

```

2:

```

my_tuple = ()
my_tuple

```

Щоб додати до кортежу нові об'єкти, створіть його другим способом, вказавши через кому кожен бажаний елемент.

```

tuples = ("Ukraine", 2022, True)
tuples

```

Якщо кортеж містить лише один елемент, після цього елемента потрібно поставити кому. Таким чином Python відрізняє кортеж від числа у дужках, що

визначають порядок виконання операцій. Після створення кортежу до нього не можна додавати нові елементи або змінити існуючі. При спробі змінити елемент у кортежі після його створення Python згенерує виняток.

Перевірити, чи міститься елемент у кортежі, можна за допомогою ключового слова `in`.

```
tuples = ("Ukraine", 2022, True)
tuples
"2022" in tuples
>>True
```

Помістіть перед `in` ключове слово `not` для перевірки відсутності елемента у кортежі.

```
tuples = ("Ukraine", 2022, True)
tuples
"Days" not in tuples
>>True
```

Кортежі зручно використовувати, коли маєте справу зі значеннями, які ніколи не зміняться, і ви бажаєте бути впевненими, що їх не змінять інші частини вашої програми.

Словники

Словники – ще один вбудований контейнер для зберігання об'єктів. Вони використовуються для зв'язування одного об'єкта, званого ключем, з іншим. Таке зв'язування називається відображенням. Результатом буде пара ключ-значення. Пари ключ-значення додаються до словника. Потім можна знайти у словнику ключ і отримати відповідне йому значення. Однак не можна використовувати значення для знаходження ключа. Словники є змінними, так що в них можна додавати нові пари ключ-значення. На відміну від списків та кортежів, словники не зберігають об'єкти в певному порядку. Їх корисність полягає у зв'язках між ключами та значеннями - існує безліч ситуацій, в яких вам потрібно буде зберігати дані попарно. Наприклад, у словнику можна зберегти інформацію про будь-що.

```
dict = dict()
dict
або
dict = {}
```


dict

При створенні словників до них можна додавати пари ключ-значення. Обидва варіанти синтаксису припускають відокремлення ключа від значення двокрапкою. Пари ключ-значення відокремлюються комами. На відміну від кортежів, якщо є тільки одна пара ключ-значення, кома після неї не потрібна.

```
>>> animal = {'cat' : 'кіт', 'dog' : 'пес', 'bird' : 'птаха', 'mouse' : 'миша'}
```

Словники, як і списки, є типом даних, що змінюється: можна змінювати, додавати та видаляти елементи пари 'ключ:значення'. Спочатку словник можна створити порожнім, наприклад, `dic = {}` і лише потім заповнити його елементами. Додавання та зміна має однаковий синтаксис: словник[ключ] = значення. Ключ може бути, як вже існуючим (тоді відбувається зміна значення), і новим (відбувається додавання елемента словника). Вилучення елемента словника здійснюється за допомогою функції `del (dic [key])` або так `pop(key)`.

```
>>> dic = {'cat' : 'кішка', 'dog' : 'пес', 'bird' : 'птаха', 'mouse' : 'миша'}
>>> dic ['cat'] = 'кіт'
>>> dic = {'cat' : 'кіт', 'dog' : 'пес', 'bird' : 'птаха', 'mouse' : 'миша'}
>>> dic = {'cat' : 'кішка', 'dog' : 'пес', 'bird' : 'птаха', 'mouse' : 'миша'}
>>> del (dic ['mouse'])
>>> dic = {'cat' : 'кішка', 'dog' : 'пес', 'bird' : 'птаха'}
```

Значенням у словнику може бути будь-який об'єкт. На відміну від значення словника, ключ словника має бути незмінним. Ключем словника може бути рядок чи кортеж, але не список чи словник. Для визначення наявності ключа у словнику використовуйте ключове слово `in`. Слово `in` не можна використовувати для перевірки наявності у словнику значення.

Функции и модули

Обзор функций

Функція - це група інструкцій, що існує всередині програми з метою виконання певного завдання.

Функція створюється (або, як кажуть програмісти, визначається) за допомогою ключового слова `def` у наступному форматі. Визначення функції починається з коду, що виконується, `def`. Потім ім'я функції з параметрами (якщо вони є). Нарешті, оператор `def` закінчується двокрапкою.

```
def
    hello(name
    ):

    print('Hel
    lo')
```

Базові правила які треба використовувати при створенні функції:

- в якості імені функції не можна використовувати одне з ключових слів Python
- ім'я функції не може містити пробелі
- перший символ повинен бути одним із букви від a до z, від A до Z або символом підкреслення
- після першого символу можна використовувати букви від a до z або від A до Z, цифри від 0 до 9 либо символи подчеркивания;
- символи в верхньому і нижньому реєстрах відрізняються

У Python кожен рядок у блоці має бути виділений відступом. Останній рядок виділений

відступом після заголовку функції є останнім рядком у блоці функції.

```
def fun():
    print('Hello')
    print('End')
#остання строка в цьому блоці
print('Morning')#інструкція не в блоці
```

Коли рядки виділяються відступом, слід переконатися, що кожен рядок починається

з однакової кількості пропусків. Інакше станеться помилка. Наприклад, наведене нижче визначення функції викликає помилку, тому що всі рядки виділені різною кількістю пропусків.

```
def fun():
    print('Hello')
    print('End')
print('Morning')
```

Функція використовується для обробки даних які отримує з основної гілки програми. Данні передаються функції при її виклику у дужках і називаються аргументами.

```
def summa (a,b):
```

```

    c = a + b
    return c
num1 = int(input('Перше число: '))
num2 = int(input('Друге число: '))
summa (num1 , num2 )

```

Кількість параметрів в об'явленні функцій відповідає кількості аргументів, які передають функції при виклику.

```

def add(x, y):
    return x + y

```

Проста функція

Функція може приймати довільну кількість аргументів або не приймати їх все. Також формуються функції з довільним числом аргументів.

```

>>> def func(a, b, c=2): # c,a,b - аргумент
...     return a + b + c

```

Є концепції позиційних (позиційних) і іменованих (ключових) аргументів.

**args* (аргументи, не пов'язані з ключовими словами)

***kwargs* (аргументи ключових слів)

Ми використовуємо символ підстановки або «*», як це — **args* або ***kwargs* — як аргумент нашої функції, коли у нас є сумніви щодо кількості аргументів, які ми повинні передати у функції».

Спеціальний синтаксис **args* у визначеннях функцій у Python використовується для передачі змінної кількості аргументів у функцію. Він використовується для передачі списку аргументів змінної довжини без ключів.

Синтаксис передбачає використання символу *** для прийому змінної кількості аргументів; за домовленістю, воно часто використовується зі словом *args*.

**args* дозволяє вам приймати більше аргументів, ніж кількість формальних аргументів, які ви визначили раніше. За допомогою **args* до ваших поточних формальних параметрів можна додати будь-яку кількість додаткових аргументів (включаючи нуль додаткових аргументів).

Наприклад, ми хочемо створити функцію множення, яка приймає будь-яку кількість аргументів і здатна перемножувати їх усі разом. Це можна зробити за допомогою **args*.

Використовуючи ***, змінна, яку ми пов'язуємо з ***, стає ітерованою, тобто ви можете робити такі дії, як ітерації по ній, запускати деякі функції вищого

порядку.

Чим вони відрізняються. Ми просто складаємо позиції аргументів і параметрів. Аргумент №1 відповідає параметру №1, аргумент №2 — параметру №2 і так далі.

Для виклику функції необхідні все три аргумента. Якщо пропустити хоча б один із них — буде видано повідомлення про помилку.

```
def printThese (a,b,c) :  
    print(a, "is stored in a")  
    print(b, "is stored in b")  
    print(c, "is stored in c")  
printThese(1,2,3)  
"""  
1 is stored in a  
2 is stored in b  
3 is stored in c  
"""
```

```
def printThese (a,b,c) :  
    print(a, "is stored in a")  
    print(b, "is stored in b")  
    print(c, "is stored in c")  
printThese(1,2)  
"""  
TypeError: printThese() missing 1 required positional  
argument: 'c'  
"""
```

Якщо при об'явленні функції налаштувати значення за замовчуванням — вказати відповідний аргумент при виклику функції вже необов'язково.

Параметр стає додатковим.

```
def printThese (a,b,c=None) :  
    print(a, "is stored in a")  
    print(b, "is stored in b")  
    print(c, "is stored in c")  
printThese(1,2)  
"""
```

```
1 is stored in a
2 is stored in b
None is stored in c
"""
```

Опціональні параметри, крім того, можна задавати при виклику функцій, використовуючи їх імена.

У наступному прикладі встановлюємо три параметри за значенням за замовчуванням None і дивимося на те, як їх можна позначити, використовуючи їх імена та не звертаючи уваги на порядок слідування аргументів, застосовуваних при виклику функцій.

```
def printThese (a=None, b=None, c=None) :
    print(a, "is stored in a")
    print(b, "is stored in b")
    print(c, "is stored in c")
printThese(c=3, a=1)
"""
```

```
1 is stored in a
None is stored in b
3 is stored in c
"""
```

Оператор * частіше всього асоціюється з людьми з операцією множення, але в Python він має інший сенс.

Цей оператор дозволяє «розпаковувати» об'єкти, всередині яких зберігаються деякі елементи. Ось приклад:

```
a = [1, 2, 3]
b = [*a, 4, 5, 6]
print(b) # [1, 2, 3, 4, 5, 6]
```

Беремо список a, розпаковуємо, та поміщаємо в список b.

Ми знаємо про те, що оператор «зірочка» в Python здатний «витаскувати» з об'єктів елементи. Знаємо ми і про те, що існують два параметри виду функцій. А саме, *args — це скорочення від «аргументів» (аргументи), а **kwargs — скорочення від «аргументів ключового слова» (іменовані аргументи).

Деякі з цих конструкцій використовуються для розпакування аргументів відповідного типу, що дозволяє викликати функції зі списком аргументів

змінної довжини. Наприклад — створювана функція, яка має виводити результати.

```
def printScores(student, *args):
    print(f"Student Name: {student}")
    for arg in args:
        print(arg)
printArgs("Alex", 100, 95, 88, 92, 99)
"""
Student Name: Alex
100
95
88
92
99
"""
```

«args» — це всього лише набір символів, яким прийнято обозначати аргументи. Саме головне тут — оператор *. Все що йде що именно идет після нього, ролі не грає. Завдяки використанню * ми створили список позиційних аргументів на основі того, що було передано функціями при виклику. Завдяки їм створюється словник, в якому містяться іменовані аргументи, передані функції при його виклику.

```
def printPetNames(owner, **pets):
    print(f"Owner Name: {owner}")
    for pet, name in pets.items():
        print(f"{pet}: {name}")
printPetNames("Jonathan", dog="Brock", fish=["Larry",
"Curly", "Moe"], turtle="Shelldon")
"""
Owner Name: Jonathan
dog: Brock
fish: ['Larry', 'Curly', 'Moe']
turtle: Shelldon
"""
```

Використовуйте загальноприйняті конструкції *args і **kwargs для захоплення позиційних та іменованих аргументів.

Конструкцію **kwargs не можна розполагати до *args. Якщо це зробити — буде видано повідомлення про помилку.

Використовуйте конфлікти між цими іменованими параметрами та **kwargs, у

випадках, коли значення планується передавати як ****kwarg**-аргумент, але значення ключа поєднується з іменем іменованого параметра.

Оператор *****можна використовувати не тільки в оголошеннях функцій, але і при їх виклику.

Програма Python для ілюстрації ***args** для змінної кількості аргументів

```
def myFun(*argv):  
    for arg in argv:  
        print(arg)
```

```
myFun('Привіт', 'Доброго', 'КПІ')
```

Програма Python для ілюстрації ***args** з першим додатковим аргументом

```
def myFun(arg1, *argv):  
    print("Перший аргумент :", arg1)  
    for arg in argv:  
        print("Наступний аргумент через *argv :", arg)
```

```
myFun('Привіт', 'Доброго ранку', 'КПІ')
```

Що таке Python ****kwargs**

Спеціальний синтаксис ****kwargs** у визначеннях функцій у Python використовується для передачі списку аргументів змінної довжини з ключовими словами. Ми використовуємо назву **kwargs** із подвійною зіркою. Причина в тому, що подвійна зірочка дозволяє нам пропускати ключові аргументи (і будь-яку їх кількість).

Аргумент ключового слова – це місце, де ви надаєте ім'я змінній, коли ви передаєте її у функцію.

Можна думати про **kwargs** як про словник, який відображає кожне ключове слово на значення, яке ми передаємо разом із ним. Ось чому, коли ми повторюємо **kwargs**, здається, що немає жодного порядку, у якому вони були надруковані.

приклад 1:

Програма Python для ілюстрації `*kwargs` для змінної кількості аргументів ключового слова. Тут `**kwargs` приймає аргумент змінної довжини за ключовим словом, переданий викликом функції. *перша*='Доброго' *перша* є ключовим, а 'Доброго' є значенням. простими словами, те, що ми призначаємо, є цінним, а кому ми призначаємо, це ключ.

```
def myFun(**kwargs):  
    for key, value in kwargs.items():  
        print("%s == %s" % (key, value))
```

```
myFun(перша='Доброго', друга='Ранку', остання='Україна')
```

Програма для ілюстрації `**kwargs` для змінної кількості аргументів ключового слова з одним додатковим аргументом. Все теж саме, але одна зміна полягає в тому, що ми передаємо аргумент, не пов'язаний із ключовим словом, який прийнятний позиційним аргументом (`arg1` у `myFun`). і ключові аргументи, які ми передаємо, прийнятні для `**kwargs`.

```
def myFun(arg1, **kwargs):  
    for key, value in kwargs.items():  
        print("%s == %s" % (key, value))
```

```
myFun("Привіт", перша='Доброго', друга='Ранку',  
остання='Україна')
```

Використання `*args` і `**kwargs` для виклику функції

Тут ми передаємо `*args` і `**kwargs` як аргумент у функції `myFun`. Передача `*args` до `myFun` просто означає, що ми передаємо позиційні аргументи та аргументи змінної довжини, які містяться в `args`. тому «Доброго» переходять до `arg1`, «ранку» переходять до `arg2`, а «Україна» переходять до `arg3`. Коли ми передаємо `**kwargs` як аргумент `myFun`, це означає, що він приймає аргументи ключових слів. Тут «`arg1` є ключовим, а значенням є «Доброго», яке передається в `arg1`, і так само «ранку» і «Україна» передаються в `arg2` і `arg3` відповідно. Після передачі всіх даних ми друкуємо всі дані в рядках.


```
def myFun(arg1, arg2, arg3):  
    print("arg1:", arg1)  
    print("arg2:", arg2)  
    print("arg3:", arg3)
```

```
# Now we can use *args or **kwargs to  
# pass arguments to this function :  
args = ("Доброго", "ранку", "Україна")  
myFun(*args)
```

```
kwargs = {"arg1": "Доброго", "arg2": "ранку", "arg3":  
"Україна"}  
myFun(**kwargs)
```

Приклад 2 Передаємо *args і **kwargs як аргумент у функції myFun. де «Доброго», «ранку», «Україна» передається як *args, а first=»Доброго», mid=»ранку», last=»Україна» передається як **kwargs і друкується в тому самому рядку.

```
def myFun(*args, **kwargs):  
    print("args: ", args)  
    print("kwargs: ", kwargs)
```

```
# Тепер ми можемо використовувати як *args ,**kwargs  
# щоб передати аргументи цій функції:  
myFun('Доброго', 'ранку', 'Україна', перша="Доброго",  
друга="ранку", остання="Україна")
```

Використання *args і **kwargs для встановлення значень об'єкта

*args отримує аргументи як кортеж.

**kwargs отримує аргументи як словник.

```
class car(): #визначення класу автомобіля  
    def __init__(self,*args): #args отримує необмежену  
кількість. аргументів у вигляді масиву  
        self.speed = args[0]
```

```
        self.color=args[1]

#створення об'єктів класу автомобіля

audi=car(200,'red')
bmw=car(250,'black')
mb=car(190,'white')

print(audi.color)
print(bmw.speed)
```

```
з **kwargs
class car():
    def __init__(self,**kwargs):
        self.speed = kwargs['s']
        self.color = kwargs['c']
```

```
audi=car(s=200,c='red')
bmw=car(s=250,c='black')
mb=car(s=190,c='white')

print(audi.color)
print(bmw.speed)
```

Модулі.

Модуль колекції в Python надає різні типи контейнерів. Контейнер — це об'єкт, який використовується для зберігання різних об'єктів і забезпечує спосіб доступу до об'єктів, що містяться, і повторення їх. Деякі з вбудованих контейнерів — кортеж, список, словник тощо. У цій статті ми обговоримо різні контейнери, які надає модуль колекцій.

Counters
OrderedDict
DefaultDict
ChainMap
NamedTuple
Deque
UserDict
UserList
UserString

Лічильник є підкласом словника. Він використовується для збереження підрахунку елементів у ітерації у формі неупорядкованого словника, де ключ представляє елемент у ітерації, а значення — кількість цього елемента в ітерації.

Примітка. Це еквівалент пакету або набору інших мов.

`class collections.Counter([iterable-or-mapping])` итерируемое-или-отображение

Ініціалізація об'єктів лічильника

Об'єкт `counter` можна ініціалізувати за допомогою функції `counter()`, і цю функцію можна викликати одним із таких способів:

1. З послідовністю предметів
2. Зі словником, що містить ключі та підрахунки
3. З аргументами ключових слів, що відображають імена рядків у підрахунках
4. Ініціалізація об'єктів лічильника

Об'єкт `counter` можна ініціалізувати за допомогою функції `counter()`, і цю функцію можна викликати одним із таких способів:

- З послідовністю предметів
 - Зі словником, що містить ключі та підрахунки
 - З аргументами ключових слів, що відображають імена рядків у підрахунках
- ```
from collections import Counter
```

```
З послідовністю елементів
print(Counter(['B', 'B', 'A', 'B', 'C', 'A', 'B',
 'B', 'A', 'C']))
```

```
зі словником
print(Counter({'A':3, 'B':5, 'C':2}))
```

```
з ключовими аргументами
```

```
print(Counter(A=3, B=5, C=2))
```

## **OrderedDict**

**OrderedDict** — це підклас словника, який запам'ятовує порядок введення ключів. Єдина різниця між `dict()` і `OrderedDict()` полягає в тому, що:

**OrderedDict** зберігає порядок вставки ключів. Звичайний `dict` не відстежує порядок вставки, і його ітерація дає значення в довільному порядку. І навпаки, порядок вставлення елементів запам'ятовується **OrderedDict**.

**OrderedDict** також є підкласом словника, але на відміну від словника, він запам'ятовує порядок, у якому було вставлено ключі.

```
class collections.OrderDict()
A Python program to demonstrate working
of OrderedDict
```

```
from collections import OrderedDict
```

```
print("This is a Dict:\n")
```

```
d = {}
d['a'] = 1
d['b'] = 2
d['c'] = 3
d['d'] = 4
```

```
for key, value in d.items():
 print(key, value)
```

```
print("\nThis is an Ordered Dict:\n")
```

```
od = OrderedDict()
od['a'] = 1
od['b'] = 2
od['c'] = 3
od['d'] = 4
```

```
for key, value in od.items():
 print(key, value)
```

Під час видалення та повторного вставлення того самого ключа ключ буде натиснутий до останнього, щоб зберегти порядок вставлення ключа.

## DefaultDict

DefaultDict також є підкласом словника. Він використовується для надання деяких значень за замовчуванням для ключа, який не існує та ніколи не викликає KeyError.

### Syntax:

```
class collections.defaultdict(default_factory)
```

default\_factory — це функція, яка надає значення за замовчуванням для створеного словника. Якщо цей параметр відсутній, виникає KeyError.

### Ініціалізація об'єктів DefaultDict

Об'єкти DefaultDict можна ініціалізувати за допомогою методу DefaultDict(), передаючи тип даних як аргумент.

```
from collections import defaultdict
```

```
Визначення слова
d = defaultdict(int)

L = [1, 2, 3, 4, 2, 4, 1, 2]

Ітерація по списку
за ведення рахунку
for i in L:
 # Значення за замовчуванням 0
 # тому немає потреби
 # спочатку ввести ключ

 d[i] += 1

print(d)
```

## ChainMap

ChainMap інкапсулює багато словників в один блок і повертає список словників.

```
class collections.ChainMap(dict1, dict2)
```

### Доступ до ключів і значень із ChainMap

Значення з ChainMap можна отримати за допомогою імені ключа. До них також можна отримати доступ за допомогою методів keys() і values().

## Adding new dictionary

Новий словник можна додати за допомогою методу `new_child()`. Щойно доданий словник додається на початку `ChainMap`.

```
Python code to demonstrate ChainMap and
new_child()
```

```
import collections
```

```
initializing dictionaries
dic1 = { 'a' : 1, 'b' : 2 }
dic2 = { 'b' : 3, 'c' : 4 }
dic3 = { 'f' : 5 }
```

```
initializing ChainMap
chain = collections.ChainMap(dic1, dic2)
```

```
printing chainMap
print ("All the ChainMap contents are : ")
print (chain)
```

```
using new_child() to add new dictionary
chain1 = chain.new_child(dic3)
```

```
printing chainMap
print ("Displaying new ChainMap : ")
print (chain1)
```

## NamedTuple

`NamedTuple` повертає об'єкт кортежу з іменами для кожної позиції, які відсутні у звичайних кортежах. Наприклад, розглянемо кортеж імен `student`, де перший елемент представляє `fname`, другий представляє `lname`, а третій елемент представляє `DOB`. Припустімо, що для виклику `fname` замість запам'ятовування позиції індексу ви можете викликати елемент за допомогою аргументу `fname`, тоді отримати доступ до елемента кортежів буде дуже легко. Ця функція надається `NamedTuple`.

1. `_make()`: Ця функція використовується для повернення `namedtuple()` із ітерованого елемента, переданого як аргумент.

2. `_asdict()`: Ця функція повертає `OrderedDict()` у вигляді, створеному з відображених значень `namedtuple()`.

## Deque

***Deque (Doubly Ended Queue)*** — це оптимізований список для швидшого додавання та висування операцій з обох боків контейнера.

Він забезпечує часову складність для операцій додавання та висування порівняно зі списком із часовою складністю.

```
class collections.deque(list)
```

Ця функція приймає список як аргумент.

### Вставлення елементів

Елементи в deque можна вставляти з обох кінців. Для вставки елементів з правого боку використовується метод `append()`, а для вставлення елементів з лівого боку використовується метод `appendleft()`.

### **Removing Elements**

Елементи також можна видалити з двох сторін. Для видалення елементів справа використовуйте метод `pop()`, а для видалення елементів зліва використовуйте метод `popleft()`.

### **UserDict**

**UserDict** — це контейнер, схожий на словник, який діє як обгортка навколо об'єктів словника. Цей контейнер використовується, коли хтось хоче створити власний словник із зміненими або новими функціями.

```
class collections.UserDict([initialdata])
```

### **UserList**

**UserList** — це список, схожий на контейнер, який діє як обгортка навколо об'єктів списку.

Це корисно, коли хтось хоче створити власний список із зміненими чи додатковими функціями.

```
class collections.UserString(seq)
```

## **Класи**

Клас — це визначений користувачем план або прототип, з якого створюються об'єкти. Класи забезпечують засоби групування даних і функціональних можливостей. Створення нового класу створює новий тип об'єкта, що дозволяє створювати нові екземпляри цього типу.

Кожен екземпляр класу може мати атрибути, приєднані до нього для підтримки його стану. Екземпляри класу також можуть мати методи (визначені їхнім

класом) для зміни свого стану.

Щоб зрозуміти необхідність створення класу в Python, давайте розглянемо приклад. Скажімо, ви хочете відстежити кількість собак, які можуть мати різні атрибути, такі як порода та вік. Якщо використовується список, першим елементом може бути порода собаки, а другим — її вік.

### **Синтаксис: Визначення класу**

```
клас ClassName:
 # Заява
```

### **Синтаксис: визначення об'єкта**

```
obj = ClassName()
print(obj.attr)
```

Клас створює визначену користувачем структуру даних, яка містить власні члени даних і функції-члени, доступ до яких можна отримати та використовувати, створивши екземпляр цього класу. Клас схожий на план для об'єкта.

Деякі моменти щодо класу Python:

Класи створюються за ключовим словом `class`.

Атрибути - це змінні, які належать до класу.

Атрибути завжди загальнодоступні, і до них можна отримати доступ за допомогою оператора та крапки (`.`). Напр.: `Myclass.Myattribute`

Визначення класу

```
продемонструвати визначення
клас
class Dog:
 pass
```

У наведеному вище прикладі ключове слово `class` вказує на те, що ви створюєте клас, за яким слідує назва класу (у цьому випадку `Dog`).

### **Об'єкти класу**

Об'єкт — це екземпляр класу. Клас схожий на проект, тоді як екземпляр є копією класу з фактичними значеннями.

Об'єкт складається з:

1. Стан: представлено атрибутами об'єкта. Він також відображає властивості об'єкта.
2. Поведінка: представлена методами об'єкта. Він також відображає



реакцію об'єкта на інші об'єкти.

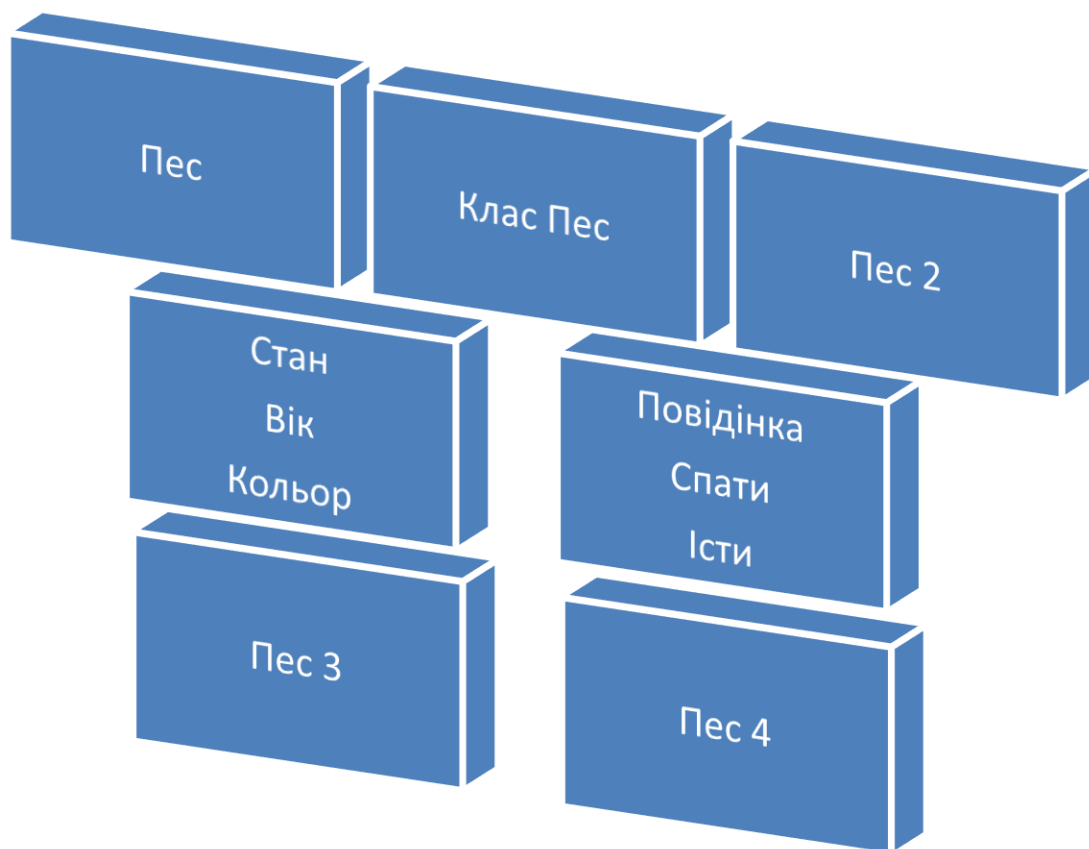
3. Ідентифікація: вона дає унікальне ім'я об'єкту та дозволяє одному об'єкту взаємодіяти з іншими об'єктами



Оголошення об'єктів (також називається створенням екземпляра класу)

Коли об'єкт класу створюється, клас називається екземпляром.

Усі екземпляри мають спільні атрибути та поведінку класу. Але значення цих атрибутів, тобто стану, є унікальними для кожного об'єкта. Один клас може мати будь-яку кількість екземплярів.



Створення об'єкта

# продемонструвати створення екземпляра

```

клас

class Dog:

 # Простий клас
 # атрибут
 attr1 = "тварина"
 attr2 = "dog"

 # Метод зразка
 def fun(self):
 print("Я", self.attr1)
 print("Я", self.attr2)

Код драйвера
Створення екземпляра об'єкта
Сірко = Dog()
Доступ до атрибутів класу
і метод через об'єкти

print(Сірко.attr1)
Сірко.fun()

```

У наведеному вище прикладі створюється об'єкт, який по суті є собакою на ім'я Сірко. Цей клас має лише два атрибути класу, які говорять нам, що Сірко — це собака та тварина.

## The self

Методи класу повинні мати додатковий перший параметр у визначенні методу. Ми не надаємо значення для цього параметра, коли викликаємо метод, Python надає його.

Якщо у нас є метод, який не приймає аргументів, ми все одно маємо один аргумент.

Коли ми викликаємо метод цього об'єкта як `myobject.method(arg1, arg2)`, це автоматично перетворюється Python на `MyClass.method(myobject, arg1, arg2)` — це все, про що йдеться у спеціальному `self`.

`self` — це стандартне ім'я першого аргументу для методів об'єкта. У ході виконання екземпляра методу об'єкта в першому аргументі автоматично

виводиться клас (передати його спеціально не потрібно).

```
class MyClass:
```

```
 def say(self):
 print(self)
```

```
my_1 = MyClass() # Создаём объект класса.
```

### метод `__init__`

Метод `__init__` схожий на конструктори. Конструктори використовуються для ініціалізації стану об'єкта. Як і методи, конструктор також містить набір інструкцій (тобто інструкцій), які виконуються під час створення об'єкта. Він запускається, як тільки створюється екземпляр об'єкта класу. Метод корисний для виконання будь-якої ініціалізації, яку ви хочете виконати з вашим об'єктом.

# Приклад класу з методом `__init__`

```
class Person:
```

```
 # метод ініціалізації або конструктор
```

```
 def __init__(self, name):
 self.name = name
```

```
 # Метод зразка
```

```
 def say_hi(self):
 print('Привіт , мене звати', self.name)
```

```
p = Person('Оксана')
```

```
p.say_hi()
```

### **Змінні класу та екземпляра**

Змінні екземпляра призначені для даних, унікальних для кожного екземпляра, а змінні класу призначені для атрибутів і методів, спільних для всіх екземплярів класу. Змінні екземпляра — це змінні, значення яких присвоюється всередині конструктора або методу з `self`, тоді як змінні класу — це змінні, значення яких присвоюється в класі.

Визначення змінних екземплярів за допомогою конструктора.

```

class Dog:

 # клас змінної
 animal = 'dog'

 # ініціалізуємо конструктор
 def __init__(self, poroda, color):

 # створюємо змінну
 self.poroda = poroda
 self.color = color

Rodger = Dog("Pudel", "brown")
Buzo = Dog("Bulldog", "black")

print('Sirko :')
print('Sirko є', Sirko.animal)
print('Poroda: ', Sirko.breed)
print('Color: ', Sirko.color)

print('Ralf :')
print('Ralf is a', Ralf.animal)
print('Poroda: ', Ralf.breed)
print('Color: ', Ralf.color)

print("Доступ до змінної класу за допомогою імені класу")
print(Dog.animal)
Sirko :
Sirko є dog
Poroda: Pudel
Color: brown
Ralf :
Ralf is a dog
Poroda: Bulldog
Color: black
Доступ до змінної класу за допомогою імені класу
dog

```

**Визначення змінних екземплярів за допомогою звичайного методу.**

```

class Dog:

 # змінна
 animal = 'dog'

```

```

створюємо конструктор
def __init__(self, poroda):

 # Змінна екземпляра
 self.poroda = poroda

Додає змінну екземпляра
def setColor(self, color):
 self.color = color
Отримує змінну екземпляра

def getColor(self):
 return self.color

Sirko = Dog("bulldog")
Sirko.setColor("red")
print(Sirko.getColor())
red

```

## Конструктори в Python

Конструктори зазвичай використовуються для створення екземпляра об'єкта. Завдання конструкторів полягає в тому, щоб ініціалізувати (призначити значення) членам даних класу під час створення об'єкта класу. У Python метод `__init__()` називається конструктором і завжди викликається під час створення об'єкта.

Синтаксис створення конструктора:

```

def __init__(self):
 # тіло конструктора

```

Типи конструкторів:

**1.Конструктор за замовчуванням:** Конструктор за замовчуванням — це простий конструктор, який не приймає жодних аргументів.

Його визначення має лише один аргумент, який є посиланням на екземпляр, що створюється.

**2.параметризований конструктор:** конструктор з параметрами відомий як параметризований конструктор.

Параметризований конструктор приймає свій перший аргумент як посилання на екземпляр, що створюється, відомий як `self`, а решта аргументів надає той хто створює програму.

Приклад конструктора за замовчуванням:

```
class KPI:

 # створення конструктора
 def __init__(self):
 self.kpi = "KPI"

 def print_KPI(self):
 print(self.kpi)

створення об'єкта
obj = KPI()

obj.print_KPI()
```

### **KPI**

Приклад параметризованого конструктора

```
class Addition(Додавання):
 first = 0
 second = 0
 answer = 0

 # параметризований конструктор
 def __init__(self, f, s):
 self.first = f
 self.second = s

 def display(self):
 print("Перший = " + str(self.first))
 print("Другий = " + str(self.second))
 print("Додавання двох чисел = " +
str(self.answer))

 def calculate(self):
 self.answer = self.first + self.second

створення об'єкта класу
це викличе параметризований конструктор
```

```

obj1 = Addition(2000, 4000)

створення другого об'єкта того самого класу
obj2 = Addition(16, 44)

виконати Додавання на obj1
obj1.calculate()

obj2.calculate()

obj1.display()

obj2.display()
Перший = 2000
Другий = 4000
Додавання двох чисел = 6000
Перший = 16
Другий = 44
Додавання двох чисел = 60

```

## Деструктори в Python

Деструктори викликаються, коли об'єкт знищується.

Метод `__del__()` відомий як метод деструктора в Python. Він викликається, коли всі посилання на об'єкт видалено, тобто коли об'єкт збирається як сміття.

Синтаксис оголошення деструктора:

```

def __del__(self):
 # тіло деструктора

```

Примітка: Посилання на об'єкти також видаляється, коли об'єкт виходить за межі посилання або коли програма завершується.

Приклад 1: Ось простий приклад деструктора. Використовуючи ключове слово `del`, ми видалили всі посилання на об'єкт «obj», тому деструктор викликався автоматично.

```

class Burger:

 def __init__(self):
 print('Створено бургер.')

```

```

Видалення (виклик деструктора)
def __del__(self):
 print('Викликано деструктор, видалено бургер.')

obj = Burger()
del obj

```

**Створено бургер.**

**Викликано деструктор, видалено бургер.**

Примітка: деструктор було викликано після завершення програми або коли всі посилання на об'єкт видалено, тобто коли кількість посилань стає нульовою, а не коли об'єкт вийшов за межі області видимості.

Приклад 2

```

class A:
 def __init__(self, bb):
 self.b = bb

class B:
 def __init__(self):
 self.a = A(self)
 def __del__(self):
 print("KPI")

def fun():
 b = B()

```

fun()

У цьому прикладі, коли викликається функція fun(), вона створює екземпляр класу B, який передає себе в клас A, який потім встановлює посилання на клас B, що призводить до циклічного посилання.

Як правило, збирач сміття Python, який використовується для виявлення цих типів циклічних посилань, видалив би його, але в цьому прикладі використання спеціального деструктора позначає цей елемент як «незбірний».

Просто він не знає порядку, в якому знищувати об'єкти, тому залишає їх. Таким чином, якщо ваші екземпляри задіяні в циклічних посиланнях, вони залишатимуться в пам'яті стільки, скільки працює програма.

### **Спадкування в Python**

Одним із основних понять у мовах об'єктно-орієнтованого програмування (ООП) є успадкування. Це механізм, який дозволяє створювати ієрархію класів, які мають спільний набір властивостей і методів, шляхом отримання класу від іншого класу. Спадкування — це здатність одного класу отримувати або



успадковувати властивості іншого класу.

Перевагами спадкування є:

1. Він добре відображає відносини в реальному світі.
2. Він забезпечує повторне використання коду.
3. Нам не потрібно писати один і той же код знову і знову. Крім того, це дозволяє нам додавати більше функцій до класу, не змінюючи його.
4. Він має транзитивний характер, що означає, що якщо клас В успадковує інший клас А, то всі підкласи В автоматично успадковують клас А.
5. Спадкування пропонує просту, зрозумілу структуру моделі.
6. Менші витрати на розвиток і обслуговування є результатом спадщини.

Створення спадкування

```
Class BaseClass:
```

```
 {Body}
```

```
Class DopClass(BaseClass):
```

```
 {Body}
```

Створення батьківського класу

Створення класу Person з методами Display.

```
class Person(object):
```

```
def __init__(self, name, kpi):
```

```
 self.name = name
```

```
 self.kpi = kpi
```

```
#перевірка є ця людина викладачем КПІ
```

```
def Display(self):
```

```
 print(self.name, self.kpi)
```

```
emp = Person("Олена", КПІ)
```

```
emp.Display()
```

**Олена КПІ**

Створення дочірнього класу

Тут Emp — ще один клас, який успадковує властивості класу Person (базового класу).

```
class Emp(Person):
```

```
 def Print(self):
```

```
 print("Викликано клас Emp")
```

```
Emp_details = Emp("Олена", KPI)

виклик функції батьківського класу
Emp_details.Display()
Виклик функції дочірнього класу
Emp_details.Print()
```

**Олена КПІ**

**Викликано клас Emp**

Приклад успадкування в Python

```
class UNI(object):

 # Конструктор
 def __init__(self, name):
 self.name = name

 # Щоб отримати ім'я
 def getName(self):
 return self.name

 def isEmployee(self):
 return False

Успадкований або підклас

class Employee(UNI):
 # Тут ми повертаємо true
 def isEmployee(self):
 return True

emp = UNI("KPI")
print(emp.getName(), emp.isEmployee())

emp = Employee("UNI")
print(emp.getName(), emp.isEmployee())
KPI true
KPI false
```

**Типи успадкування Python**

Наслідування та композиція є двома основними поняттями в об'єктно-

орієнтованому програмуванні, які моделюють відносини між двома класами. Вони визначають дизайн програми та визначають, як додаток має розвиватися у міру додавання нових функцій чи зміни вимог.

Обидва реалізують повторне використання коду, але роблять це по-різному. Моделі успадкування – це стосунки. Це означає, що коли у вас є клас похідне від чогось, який успадковується від базового класу, ви створили відношення, де похідне є спеціалізованою версією основного успадкування.

Класи, що успадковуються від іншого, називаються похідними класами, підкласами чи підтипами.

Класи, у тому числі отримані інші класи, називаються базовими класами чи суперкласами.

Вважається, що похідний клас породжує, успадковує чи розширює базовий клас. Є базовий клас `Animal`, і ви створюєте його для створення класу `Horse`. У спадкових відносинах говориться, що `Horse` — це `Animal`. Це означає, що `Horse` успадковує інтерфейс і реалізацію `Animal`, а об'єкти `Horse` можуть використовуватись для заміни об'єктів `Animal` у додатку.

Композиція — це концепція, яка моделює відносини. Вона дозволяє створювати складні типи, комбінуючи об'єкти інших типів. Це означає, що клас `Composite` може містити об'єкт другого класу `Component`.

Класи, що містять об'єкти інших класів, зазвичай називаються композитами, а класи, які використовуються для створення більш складних типів, називаються компонентами.

Наприклад, наш клас `Кінь` може бути складений іншим об'єктом типу `Хвіст`. Композиція дозволяє виразити ці відносини, сказав, що у `Horse` є хвіст.

Композиція дозволяє повторно використовувати код, додаючи об'єкти до інших об'єктів, у відмінності від спадкування інтерфейсу та реалізації інших класів. Класи `Horse` і `Dog` можуть використовувати функціональність `Tail` через композицію, не виводячи один клас з другого.

Усе в Python є об'єктом. Модулі — це об'єкти, визначення класів і функції — це об'єкти, і, звичайно, об'єкти, створені з класів, теж є об'єктами. Наслідування є обов'язковою функцією кожного об'єктно-орієнтованого мови програмування.

Найпростіший спосіб побачити спадкування в Python — це перейти в інтерактивну оболонку Python і написати трохи коду. Ми почнемо з описами самого простого із можливих класів:

```
class MyClass:
```

```
... pass
```

```
...
```

Тепер, коли у нас об'явлений клас, ми можемо використовувати функцію `dir()` для отримання списку його членів:

```
>>> c = MyClass()
```

```
>>> dir(c)
```

```
['__class__', '__delattr__', '__dict__', '__dir__',
'__doc__', '__eq__',
'__format__', '__ge__', '__getattribute__', '__gt__',
'__hash__', '__init__',
'__init_subclass__', '__le__', '__lt__', '__module__',
'__ne__', '__new__',
'__reduce__', '__reduce_ex__', '__repr__', '__setattr__',
'__sizeof__',
'__str__', '__subclasshook__', '__weakref__']
```

`dir()` повертає список усіх членів у вказаному об'єкті.

Кожен клас, який ви створюєте на Python, буде явно успадкований від об'єкта. Виключення з цього правила є класом, який використовується для позначення помилок шляхом виклику винятку.

```
Програма Python для демонстрації
```

```
єдиного успадкування
```

```
основний клас
```

```
class Parent:
```

```
 def func1(self):
```

```
 print("Ця функція знаходиться в батьківському
класі.")
```

```
додатковий
```

```
class Child(Parent):
```

```
 def func2(self):
```

```
 print("Ця в додатковому.")
```

```
object = Child()
```

```
object.func1()
```

```
object.func2()
```

Ця функція знаходиться в батьківському класі.

Ця функція знаходиться в додатковому .

### Множинне успадкування:

Якщо клас може бути похідним від кількох базових класів, такий тип успадкування називається множинним успадкуванням. У множинному успадкуванні всі функції базових класів успадковуються в похідному класі.

# множинне успадкування(mixed)

```
class Mother:
 mothername = ""

 def mother(self):
 print(self.mothername)
```

```
class Father:
 fathername = ""

 def father(self):
 print(self.fathername)
```

# Похідний клас

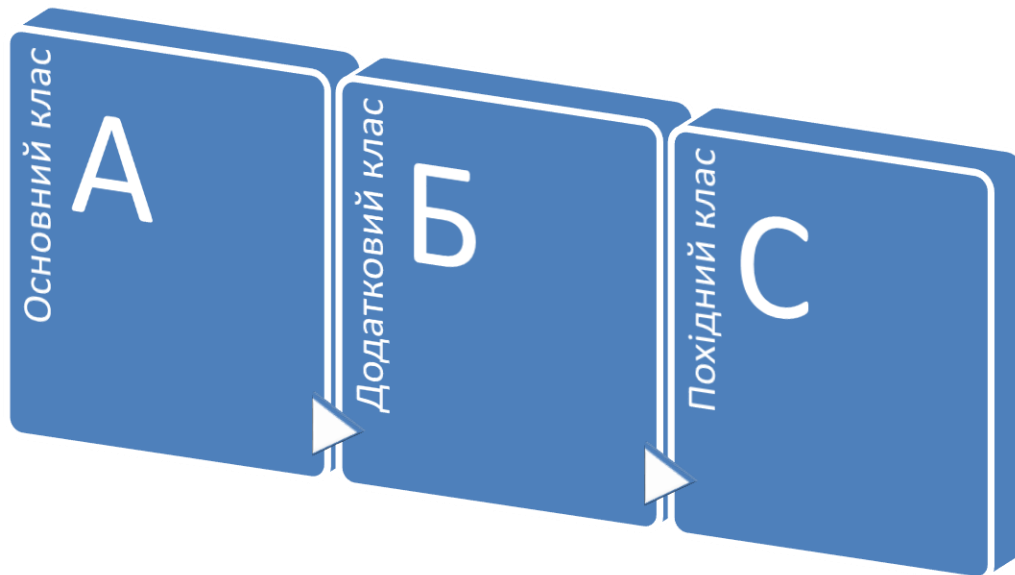
```
class Son(Mother, Father):
 def parents(self):
 print("Father :", self.fathername)
 print("Mother :", self.mothername)
```

```
Driver's codes
s1 = Son()
s1.fathername = "Maxim"
s1.mothername = "Olena"
s1.parents()
Father : Maxim
Mother : Olena
```

### Багаторівневе успадкування:

У багаторівневому успадкуванні ознаки базового класу та похідного класу далі успадковуються в новий похідний клас. Це схоже на стосунки між дитиною та дідусем.

## Багаторівневе успадкування



```
багаторівневе успадкування
```

```
основний клас
```

```
class Grandfather:
```

```
 def __init__(self, grandfathername):
 self.grandfathername = grandfathername
```

```
додатковий
```

```
class Father(Grandfather):
```

```
 def __init__(self, fathername, grandfathername):
 self.fathername = fathername
```

```
 # виклик конструктора класу Grandfather
 Grandfather.__init__(self, grandfathername)
```

```
Похідний клас
```

```
class Son(Father):
```

```
 def __init__(self, sonname, fathername,
grandfathername):
 self.sonname = sonname
```

```

 # виклик конструктора класу Father
 Father.__init__(self, fathername,
grandfathername)

def print_name(self):
 print('Grandfather name :', self.grandfathername)
 print("Father name :", self.fathername)
 print("Son name :", self.sonname)

s1 = Son('Prince', 'Rampal', 'Lal mani')
print(s1.grandfathername)
s1.print_name()

```

## Модулі

### Tkinter

**Tkinter** — це бібліотека, яка найчастіше використовується для розробки GUI (графічного інтерфейсу користувача) на Python. Це стандартний інтерфейс Python для набору інструментів Tk GUI, який постачається разом із Python. Оскільки Tk і Tkinter доступні на більшості платформ Unix, а також у системі Windows, розробка GUI програм за допомогою Tkinter стає найшвидшою та найпростішою.

Віджети в Tkinter — це елементи програми GUI, яка надає користувачам різні елементи керування (такі як мітки, кнопки, поля зі списком, прапорці, панелі меню, радіокнопки та багато іншого) для взаємодії з програмою.є об'єктно-орієнтований інтерфейс інструментарію Tk GUI.

Деякі інші бібліотеки Python доступні для створення наших власних програм із графічним інтерфейсом

Kivy

Python Qt

## wxPython

| Віджети     | Опис                                                                                                                  |
|-------------|-----------------------------------------------------------------------------------------------------------------------|
| Label       | Він використовується для відображення тексту або зображення на екрані                                                 |
| Button      | Він використовується для додавання кнопок до програми                                                                 |
| Canvas      | Він використовується для малювання малюнків та інших макетів, таких як тексти, графіка тощо.                          |
| ComboBox    | Він містить стрілку вниз для вибору зі списку доступних параметрів                                                    |
| CheckBox    | Він відображає кілька параметрів для користувача як перемикач                                                         |
| RadioButton | Він використовується для введення однорядкового тексту від користувача                                                |
| Entry       | Він використовується для введення однорядкового тексту від користувача                                                |
| Frame       | Він використовується як контейнер для зберігання та організації віджетів                                              |
| Message     | Він працює так само, як і Label і стосується багаторядкового тексту, який не редагується                              |
| Scale       | Він використовується для надання графічного зображення повзунок, який дозволяє вибрати будь-яке значення з цієї шкали |
| Scrollbar   | Використовується для прокручування вмісту вниз.<br>Він забезпечує контролер слайдів.                                  |
| SpinBox     | Це дозволяє користувачеві вибирати з заданого набору значень                                                          |
| Text        | Це дозволяє користувачеві редагувати багаторядковий текст і відформатувати так, як це має відображатися               |
| Menu        | Він використовується для створення всіх видів меню, які використовуються програмою                                    |



## Приклад 1

```
from tkinter import *
from tkinter.ttk import *
потрібно написати код
створити головне вікно
створення програми
об'єкт головного вікна під назвою root
root = Tk()
надання заголовка головному вікну
root.title("Перша_програма")
Мітка (Label) - це те, що буде виведено
показати на вікні
label = Label(root, text="Привіт !").pack()
виклик методу основного циклу, який використовується
коли ваша програма буде готова до запуску
і вказує, щоб код продовжував відображатися
root.mainloop()
```

tkinter надає нам різноманітні загальні елементи графічного інтерфейсу, які ми можемо використовувати для створення інтерфейсу, наприклад кнопки, меню та різноманітні поля введення та області відображення. Ми називаємо ці елементи віджетами.

## Віджети

Загалом віджет — це елемент графічного інтерфейсу користувача (GUI), який відображає інформацію або дає можливість користувачеві взаємодіяти з ОС. У Tkinter віджети є об'єктами; екземпляри класів, які представляють кнопки, рамки тощо.

Кожен окремий віджет є об'єктом Python. Під час створення віджета ви передаєте його батьківський елемент як параметр функції створення віджета. Єдиним винятком є «кореневе» вікно, яке є вікном верхнього рівня, яке міститиме все інше, і воно не має батьківського елемента.

## Приклад2

```
from tkinter import *
створити вікно
root = Tk()
```

```
рамка у вікні
frame = Frame(root)
метод упакування
frame.pack()

кнопка всередині рамки
button = Button(frame, text = 'КПІ')
button.pack()
Цикл подій Tkinter
root.mainloop()
```

### Класи віджетів

Tkinter підтримує наступні основні віджети –

*Label, Button, Canvas, ComboBox, CheckButton, Entry, Frame, Message, Scale, Scrollbar, SpinBox, Text Menu*

### Управління розташуванням

Створення нового віджета не означає, що він з'явиться на екрані. Щоб відобразити його, нам потрібно викликати спеціальний метод: `grid`, `pack` (приклад вище) або `place`.

| Метод          | Опис                                                                                                                                                                               |
|----------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>pack()</b>  | Менеджер розташування сітки поміщає віджети у двовимірну таблицю. Головний віджет розбивається на кілька рядків і стовпців, і кожна частина отриманої таблиці може містити віджет. |
| <b>grid()</b>  | Головний віджет розбивається на кілька рядків і стовпців, і кожна частина отриманої таблиці може містити віджет.                                                                   |
| <b>place()</b> | Менеджер розташування Pack пакує віджети в рядки або стовпці.                                                                                                                      |

### Створення кнопки в tkinter

Кроки:

- 1.Імпортуйте модуль tkinter
- 2.Створити головне вікно (`root = Tk()`)
- 3.Додайте скільки завгодно віджетів.

Імпорт модуля tkinter подібний до імпорту будь-якого іншого модуля.

```
from tkinter import *
```

Також є модуль `tkinter.ttk` який забезпечує доступ до набору віджетів на тему Tk, представленого в Tkinter. Основна ідея `tkinter.ttk` полягає в тому, щоб

відокремити, наскільки це можливо, код який реалізує поведінку віджета, відносно коду, що реалізує його зовнішній вигляд. `tkinter.ttk` використовується для створення сучасних GUI (графічний інтерфейс користувача) програм, які не можуть бути створені самим `tkinter`.

### Приклад 3

```
from tkinter import *
root = Tk()
Відкрите вікно розміром 100x100
root.geometry('100x100')
Створити кнопку
btn = Button(root, text = 'Натисни мене !', bd = '5',
 command = root.destroy)
Встановити положення кнопки у верхній частині вікна.
btn.pack(side = 'top')
root.mainloop()
```

### Створення кнопки без використання тематичного віджета `tk`.

Створення кнопки за допомогою тематичного віджета `tk` (`tkinter.ttk`).  
Імпорт модуля `tkinter` подібний до імпорту будь-якого іншого модуля.

### Приклад4

```
from tkinter import *

імпортовано модуль tkinter.ttk і
автоматично змінив всі віджети
які присутні в модулі tkinter.
from tkinter.ttk import *

Створити об'єкт
root = Tk()

Ініціалізація вікна tkinter розміром 100x100

root.geometry('100x100')

btn = Button(root, text = 'Натисни мене !',
 command = root.destroy)
```

```
btn.pack(side = 'top')
root.mainloop()
```

## Label

Tkinter Label — це віджет, який використовується для реалізації вікон відображення, де ви можете розмістити текст або зображення. Текст, який відображається цим віджетом, може бути змінений у будь-який час. Він також використовується для виконання таких завдань, як підкреслення частини тексту та переносу тексту на кілька рядків. Важливо зауважити, що мітка може використовувати лише один шрифт одночасно для відображення тексту. Щоб використовувати мітку, просто потрібно вказати, що в ній відображати (це може бути текст, малюнок або зображення)

```
w = Label (text="KPI" , font="Arial 32")
```

```
w = Label (основне вікно, параметри, ...)
```

параметри: нижче наведено список найбільш часто використовуваних параметрів для цього віджета. Ці параметри можна використовувати як пари ключ-значення, розділені комами:

**anchor:** ці параметри використовуються для керування розташуванням тексту, якщо у віджеті більше місця, ніж потрібно для тексту. Типовим є `anchor=CENTER`, що центрує текст у вільному просторі.

**bg:** Цей параметр використовується для встановлення нормального фонового зображення, що відображається за міткою та індикатором.

**height:** цей параметр використовується для встановлення вертикальних розмірів нової рамки.

**width:** ширина мітки . Якщо цей параметр не встановлено, розмір мітки буде відповідати її вмісту.

**bd:** Цей параметр використовується для встановлення розміру рамки навколо індикатора. Стандартне значення `bd` встановлено на 2 пікселі.

**font:** Якщо ви відображаєте текст у мітці (за допомогою параметра `text` або `textvariable`), параметр шрифту використовується, щоб визначити, яким шрифтом відображатиметься цей текст у мітці.

**cursor:** Використовується для вказівки, який курсор показувати, коли миша

переміщується над міткою. За замовчуванням використовується стандартний курсор.

**bitmap:** як випливає з назви, вона пов'язана зі змінною Tkinter із міткою. Якщо змінна змінена, текст мітки оновлюється.

**fg:** мітка, яка використовується для текстових і растрових міток. Значення за замовчуванням залежить від системи. Якщо ви відображаєте растрове зображення, це кольор, який з'явиться в позиції 1-бітів растрового зображення.

**image:** цей параметр використовується для відображення статичного зображення у віджеті мітки.

**padx:** Цей параметр використовується для додавання додаткових пробілів між лівим і правим краєм тексту в мітці. Значенням за замовчуванням для цього параметра є 1.

**pady:** цей параметр використовується для додавання додаткових пробілів між верхньою та нижньою частинами тексту в межах мітки. Стандартне значення для цього параметра – 1.

**justify:** цей параметр використовується для визначення способу вирівнювання кількох рядків тексту. Використовуйте LEFT, RIGHT або CENTER як його значення. Зауважте, що для розміщення тексту всередині віджета використовуйте опцію прив'язки. Значенням за умовчанням для вирівнювання є CENTER.

**relief:** цей параметр використовується для визначення зовнішнього вигляду декоративної рамки навколо етикетки.

**wraplength:** Замість того, щоб мати лише один рядок як текст мітки, його можна розбити на кількість рядків, де кожен рядок містить кількість символів, визначену для цього параметра.

Приклад5:

```
from tkinter import *

top = Tk()

top.geometry("450x300")

мітка для імені користувача
user_name = Label(top, text = "Ім'я").place(x = 40,
```

```

60)

мітка для user_password
user_password = Label(top,
 text = "Пароль").place(x = 40,
 y = 100)

submit_button = Button(top,
 text = "Надіслати").place(x =
40,
 y = 130)

user_name_input_area = Entry(top,
width = 30).place(x = 110, y = 60)
user_password_entry_area = Entry(top,
width = 30).place(x = 110, y = 100)
top.mainloop()

```

## RadioButton

Radiobutton — це стандартний віджет Tkinter, який використовується для реалізації вибору одного з багатьох. Перемикачі можуть містити текст або зображення, і можливо пов'язани функцію або методом Python кожною кнопкою. Коли кнопка натиснута, Tkinter автоматично викликає цю функцію або метод.

button = Radiobutton(основа, text="Імя кнопки", змінні = "спільна змінна", значення = "значення кожної кнопки", параметри= значення, ...)

Приклад 6.

Перемикачі, у формі кнопкового поля. Щоб відобразити вікно кнопок, параметр індикатора має бути встановлено на 0.

```

from tkinter import *
Створення робочого вікна

```

```

m = Tk()
m.geometry("175x175")
Рядкова змінна Tkinter
можливість зберігати будь-яке значення рядка
v = StringVar(m, "1")
Dictionary to create multiple buttons
values = {"RadioButton 1" : "1",
 "RadioButton 2" : "2",
 "RadioButton 3" : "3",
 "RadioButton 4" : "4",
 "RadioButton 5" : "5"}
Цикл використовується для створення кількох радіокнопок
замість створення кожної кнопки окремо
for (text, value) in values.items():
 Radiobutton(m, text = text, variable = v,
 value = value, indicator = 0,
 background = "red").pack(fill = X, ipady
= 5)
Нескінченний цикл можна завершити
переривання клавіатури або миші
або будь-якою попередньо визначеною функцією
(destroy()) чи cntrl-c
mainloop()

```

Фон цих кнопкових коробок червоний.

#### Приклад7

Змінемо блоки кнопок на стандартні перемикачі. Для цього видаляємо індикатор.

```

from tkinter import *
from tkinter.ttk import *
m = Tk()
m.geometry("175x175")
v = StringVar(m, "1")
Словник для створення кількох кнопок
values = {"RadioButton 1" : "1",
 "RadioButton 2" : "2",
 "RadioButton 3" : "3",
 "RadioButton 4" : "4",

```

```
"RadioButton 5" : "5"]}
```

```
for (text, value) in values.items():
 Radiobutton(m, text = text, variable = v,
 value = value).pack(side = TOP, ipady = 5)
```

```
mainloop()
```

### Приклад 8

Додаємо різноманітні стилі

```
from tkinter import *
from tkinter.ttk import *
m = Tk()
m.geometry('175x175')
v = StringVar(m, "1")
Клас стилю для додавання стилю до Radiobutton
Його можна використовувати для стилізації будь-якого
віджета ttk
style = Style(m)
style.configure("TRadiobutton", background = "green",
 foreground = "red", font = ("arial", 10,
 "bold"))
Словник для створення кількох кнопок
values = {"RadioButton 1" : "1",
 "RadioButton 2" : "2",
 "RadioButton 3" : "3",
 "RadioButton 4" : "4",
 "RadioButton 5" : "5"}
Словник для створення кількох кнопок
for (text, value) in values.items():
 Radiobutton(m, text = text, variable = v,
 value = value).pack(side = TOP, ipady =
5)

mainloop()
```



## Checkbutton

Віджет Checkbutton – це стандартний віджет Tkinter, який використовується для реалізації вибору включення/вимкнення. Кнопки з прапорцями можуть містити текст або зображення. Коли натиснути кнопку, Tkinter викликає цю функцію або метод.

`w = Checkbutton (1, 2)`

1: цей параметр використовується для подання батьківського вікна.

2: доступно безліч опцій, і їх можна використовувати як пари ключ-значення, розділені комами.

**activebackground:** цей параметр використовується для представлення кольору фону, коли кнопка-прапорець знаходиться під курсором.

**Activeforeground:** цей параметр використовується для представлення кольору переднього плану, коли кнопка-прапорець знаходиться під курсором.

**bg:** цей параметр використовується для представлення звичайного кольору фону, що відображається за міткою та індикатором.

**bd:** цей параметр використовується для представлення розміру кордону навколо індикатора, значення за замовчуванням – 2 пікселі.

**comand:** Ця опція пов'язана з функцією, яка буде викликатись при зміні стану прапорця.

**cursor:** при використанні цієї опції курсор миші зміниться на цей зразок, коли він опиниться над кнопкою-прапорцем.

**disabledforeground:** колір переднього плану, який використовується для відображення тексту вимкненої контрольної кнопки. За промовчаням використовується точкова версія кольору переднього плану за замовчуванням.

**font:** цей параметр використовується для представлення шрифту для тексту.

**fg:** цей параметр використовується для представлення кольору для візуалізації тексту.

**hight:** цей параметр використовується для представлення кількості рядків тексту на прапорці, за замовчуванням - 1.

**highlightcolor:** цей параметр використовується для представлення кольору виділення фокуса, коли у фокусі знаходиться кнопка-прапорець.

**justify:** цей параметр використовується для керування вирівнюванням тексту: ЦЕНТР, ВЛІВО або ВПРАВО.

**offvalue:** відповідна керуюча змінна за умовчанням встановлена в 0, якщо кнопка не відзначена. Ми можемо змінити стан неперевіреної змінної на іншу.

**onvalue:** відповідна керуюча змінна за умовчанням встановлена в 1, якщо кнопка позначена. Ми можемо змінити стан зазначеної змінної інше.

**padx:** цей параметр використовується для вказівки, скільки місця слід залишити

зліва та праворуч від прапорця та тексту. Значення за замовчуванням – 1 піксель.

**pady:** цей параметр використовується для вказівки, скільки місця слід залишити над та під прапорцем та текстом. Значення за замовчуванням – 1 піксель.

**relief:** тип кордону кнопки. За замовчуванням встановлено значення FLAT.

**selectcolor:** цей параметр використовується для представлення кольору прапорця, коли він встановлений. За промовчанням selectcolor = "червоний".

**selectimage:** зображення відображається на кнопці-прапорці, якщо вона встановлена.

**state:** представляє стан контрольної кнопки. За умовчанням він встановлений у нормальний стан. Ми можемо змінити його на DISABLED, щоб не відповідала кнопка. Стан кнопки АКТИВНИЙ, коли вона знаходиться у фокусі.

**text:** ця опція використовувала символи нового рядка («\n») для відображення кількох рядків тексту.

**underline:** цей параметр використовується для відображення індексу символу в тексті, який потрібно підкреслити. Індксація починається з нуля у тексті.

**variable:** цей параметр використовується для представлення пов'язаної змінної, яка відстежує стан прапорця.

**width:** цей параметр використовується для представлення ширини прапорця. а також представлені у кількості знаків, що представлені у вигляді текстів.

**wraplength:** ця опція розбиватиме текст на кількість частин.

Методи:

У цих віджетах використовуються такі методи:

deselect (): цей метод викликається для вимкнення прапорця.

flash (): кнопка-прапорець блимає між активним та нормальним кольорами.

invoke (): цей метод викликає метод, пов'язаний із контрольною кнопкою.

select (): цей метод викликається для увімкнення прапорця.

toggle(): цей метод використовується для перемикання між різними кнопками.

```
from tkinter import *
```

```
root = Tk()
```

```
root.geometry("300x200")
```

```
w = Label(root, text ='КПІ', font = "50")
```

```
w.pack()
```

```
Checkbutton1 = IntVar()
```

```
Checkbutton2 = IntVar()
```

```
Checkbutton3 = IntVar()

Button1 = Checkbutton(root, text = "Заява",
 variable = Checkbutton1,
 onvalue = 1,
 offvalue = 0,
 height = 2,
 width = 10)

Button2 = Checkbutton(root, text = "Студент",
 variable = Checkbutton2,
 onvalue = 1,
 offvalue = 0,
 height = 2,
 width = 10)

Button3 = Checkbutton(root, text = "Курс",
 variable = Checkbutton3,
 onvalue = 1,
 offvalue = 0,
 height = 2,
 width = 10)

Button1.pack()
Button2.pack()
Button3.pack()

mainloop()
```

## **Canvas**

Віджет Canvas дозволяє відображати різноманітну графіку в програмі. Його можна використовувати для малювання простих форм до складних графіків. Ми також можемо відображати різні види спеціальних віджетів відповідно до наших потреб.

Сінтаксис

```
C = Canvas(root, height, width, bd, bg, ..)
```

**Необов'язкові параметри**

- root = кореневе вікно.
- висота = висота віджета полотна.
- ширина = ширина віджета полотна.
- bg = фоновий колір для полотна.
- bd = межа вікна полотна.
- кортеж scrollregion (w, n, e, s), визначений як область для прокручування ліворуч, зверху, знизу та праворуч.
- колір підсвічування, що відображається у виділенні фокуса.
- курсор Його можна визначити як курсор для полотна, яке може бути колом, кнопкою, стрілкою тощо.
- confine визначає, чи можна отримати доступ до полотна за межами області прокручування.

Деякі поширені методи малювання:

- **Creating an Oval**

```
oval = C.create_oval(x0, y0, x1, y1, options)
```

- **Creating an дуга**

```
arc = C.create_arc(20, 50, 190, 240, start=0,
extent=110, fill="red")
```

- **Creating a Line**

```
line = C.create_line(x0, y0, x1, y1, ..., xn, yn,
options)
```

- **Creating a polygon**

```
oval = C.create_polygon(x0, y0, x1, y1, ...xn, yn,
options)
```

```
from tkinter import *
```

```
root = Tk()
```

```
C = Canvas(root, bg="yellow",
 height=250, width=300)
```

```
line = C.create_line(108, 120,
 280, 60,
 fill="red")

arc = C.create_arc(180, 150, 80,
 210, start=0,
 extent=220,
 fill="blue")

oval = C.create_oval(80, 30, 140,
 180,
 fill="black")

C.pack()
mainloop()
```

## Combobox

Комбінований список — це комбінація списку та поля введення. Це один із віджетів Tkinter, який містить стрілку вниз для вибору зі списку параметрів. Це допомагає користувачам вибирати відповідно до списку відображених параметрів. Коли користувач клацає стрілку розкривного меню в полі введення, внизу поля введення відображається спливаюче вікно прокрученого списку. Вибраний параметр відображатиметься в полі введення, лише якщо вибрано параметр зі списку.

```
combobox = ttk.Combobox(master, option=value, ...)
```

```
import tkinter as tk
from tkinter import ttk
```

```
створення вікна
```

```
window = tk.Tk()
```

```
window.title('Combobox')
```

```
window.geometry('500x250')
```

```
створення мітки
```

```
ttk.Label(window, text = "КПІ Combobox Widget",
 background = 'green', foreground = "blue",
 font = ("Times New Roman", 15)).grid(row = 0, column = 1)
```

```
мітка
```

```
ttk.Label(window, text = "Вибрати місяць :",
 font = ("Times New Roman", 10)).grid(column = 0,
 row = 5, padx = 10, pady = 25)
```

```
Combobox створення
```

```
n = tk.StringVar()
```

```
monthchoosen = ttk.Combobox(window, width = 27, textvariable = n)
```

```
Додавання списку що випадає зі списком
```

```
monthchoosen['values'] = (' January',
 ' February',
 ' March',
 ' April',
 ' May',
 ' June',
```

```
' July',
' August',
' September',
' October',
' November',
' December')
```

```
monthchoosen.grid(column = 1, row = 5)
monthchoosen.current()
window.mainloop()
```

Приклад 2: поле зі списком із початковими значеннями за замовчуванням.

Ми також можемо встановити початкові значення за замовчуванням у віджеті Combobox, як показано у прикладі коду нижче.

```
import tkinter as tk
from tkinter import ttk
```

```
window = tk.Tk()
window.geometry('350x250')
```

```
ttk.Label(window, text = "Вибрати місяць :",
 font = ("Times New Roman", 10)).grid(column = 0,
 row = 15, padx = 10, pady = 25)
```

```
n = tk.StringVar()
monthchoosen = ttk.Combobox(window, width = 27,
 textvariable = n)
```

```
monthchoosen['values'] = (' January',
 ' February',
 ' March',
 ' April',
 ' May',
```

```
' June',
' July',
' August',
' September',
' October',
' November',
' December')
```

```
monthchoosen.grid(column = 1, row = 15)
```

```
Показує лютий як значення за замовчуванням
```

```
monthchoosen.current(1)
```

```
window.mainloop()
```

## **Entry**

Віджет Entry — це віджет Tkinter, який використовується для введення або відображення одного рядка тексту.

### **Syntax :**

```
entry = tk.Entry(1, 2)
```

1)Родительское вікно або фрейм, в якому відображається виджет.

2) Віджет введення надає наступні параметри:

- bg: нормальний колір фону, що відображається за меткою та індикатором.
- bd: розмір границі навколо індикатора. За умовчанням 2 пікселя.
- font: шрифт, використовуваний для тексту.
- fg: колір, використовуваний для візуалізації тексту.
- justify: якщо текст складається з кількох рядків, цей параметр визначає, як текст вирівнюється за шириною: ЦЕНТР, ВЛЕВО або ВПРАВО.
- relief: зі значенням за замовчуванням рельєф = ПЛОСКИЙ. Ви можете встановити цю опцію для будь-якого з інших стилів, наприклад: SUNKEN, RIGID, RAISED, ROOVE.
- show: Звичайні символи, які вводять користувачі, з'являються в записі. Щоб зробити .password. запис, який відображає кожен символ у вигляді зірочки, встановіть show = "\*".



- `textvariable`: щоб мати можливість витягнути поточний текст із віджету введення, ви повинні встановити цю опцію для екземпляра класу `StringVar`.

Методи:

`get()`: повертає поточний текстовий запис у вигляді рядка.

`delete()`: видаляє символи з віджета

`insert(index, 'name')`: вставляє строку 'name' перед символом по цьому індексу.

```
import tkinter as tk
```

```
root=tk.Tk()
```

```
root.geometry("600x400")
```

```
name_var=tk.StringVar()
```

```
passw_var=tk.StringVar()
```

```
def submit():
```

```
 name=name_var.get()
```

```
 password=passw_var.get()
```

```
 print("Імя: " + name)
```

```
 print("Пароль : " + password)
```

```
 name_var.set("")
```

```
 passw_var.set("")
```

```
name_label = tk.Label(root, text = 'Username',
```

```
font=('calibre',10, 'bold'))

name_entry = tk.Entry(root,textvariable = name_var,
font=('calibre',10,'normal'))

passw_label = tk.Label(root, text = 'Password', font =
('calibre',10,'bold'))

passw_entry=tk.Entry(root, textvariable = passw_var, font
= ('calibre',10,'normal'), show = '*')

sub_btn=tk.Button(root,text = 'Розташування', command =
submit)

name_label.grid(row=0,column=0)
name_entry.grid(row=0,column=1)
passw_label.grid(row=1,column=0)
passw_entry.grid(row=1,column=1)
sub_btn.grid(row=2,column=1)

root.mainloop()
```

## **Text**

Використовується там, де користувач хоче вставити багатострочні текстові поля. Цей виджет можна використовувати для різних програм, де потрібен багатострочний текст, наприклад, для обміну повідомленнями, виправлення чи відображення інформації та багатьох інших завдань. Ми можемо залишати мультимедійні файли, такі як зображення та посилання, а також у текстовому віджеті.

```
text = tk.Text(1, 2)
```

2- параметри

1- робоче вікно.

- bg - колір фону
  - fg - колір переднього плану
  - bd - границя виджета.
  - hight - висота виджета.
  - width - ширина виджета.
  - font - Тип шрифту тексту.
  - cursor - Тип використовуваного курсору.
  - insetofftime - час у мілісекундах, протягом якого курсор не мигає.
  - inserttontime - час у мілісекундах, протягом якого горить індикатор курсора.
  - padx - горизонтальний відступ.
  - pady - вертикальне заповнення.
  - state - визначає, чи буде виджет реагувати на рух миші або клавіатури.
  - Highligthickness - визначає товщину виділення у фокусі.
  - insertionwidth - визначає ширину вставленого символу.
  - relief - тип бордюри, який може бути.
  - yscrollcommand - зробити виджет вертикально прокручиваемым.
  - xscrollcommand - зробити виджет горизонтально прокручиваемым.
- ext(root, bg, fg, bd, height, width, font, ..)

Параметри:

- root- корневое окно.
- bg - колір фону
- fg - колір переднього плану
- bd - границя виджета.
- hight - висота виджета.
- width - ширина виджета.
- font - Тип шрифту тексту.
- cursor - Тип використовуваного курсору.

- insetofftime - час у мілісекундах, протягом якого курсор не мигає.
- inserttontime - час у мілісекундах, протягом якого горить індикатор курсора.
- padx - горизонтальний відступ.
- pady - вертикальне заповнення.
- state - визначає, чи буде виджет реагувати на рух миші або клавіатури.
- Highligthickness - визначає товщину виділення у фокусі.
- insertionwidth - визначає ширину вставленого символу.
- relief - тип бордюри, який може бути ЗАТУШЕННИМ, ПОВИШЕННИМ, КАНАВНИМ і КОНЬКОМ.
- yscrollcommand - зробити виджет вертикально прокручуваним.
- xscrollcommand - зробити виджет горизонтально прокручуваним.

#### *Деякі загальні методи*

- index (індекс) - отримати індекс.
- insert (index) - вставити строку по індексу.
- see (index) - перевірка, чи бачити строку по індексу.
- get (startindex, endindex) – отримати символи в діапазоні.
- delete (startindex, endindex) - видаляє символи

#### Методи обробки тегів

- tag\_delete (ім'я тега) - для видалення даного тега.
- tag\_add (тег, початковий індекс, кінцевий індекс) - помітити рядок у вказаному діапазоні
- tag\_remove (тег, початковий індекс, кінцевий індекс) - видалити тег із зазначеного діапазону

#### Методи обробки меток

- mark\_names () - отримати всі відмітки в заданому діапазоні.
- index (mark) - получити индекс отметки.
- mark\_gravity () - щоб отримати гравітацію даної відмітки.

## Приклад

```
from tkinter import *

root = Tk()
root.geometry("300x300")
root.title(" Q&A ")

def Take_input():
 INPUT = inputtxt.get("1.0", "end-1c")
 print(INPUT)
 if(INPUT == "120"):
 Output.insert(END, 'Вірно')
 else:
 Output.insert(END, "Невірно")

l = Label(text = "Скільки 24 * 5 ? ")
inputtxt = Text(root, height = 10,
 width = 25,
 bg = "light yellow")

Output = Text(root, height = 5,
 width = 25,
 bg = "light cyan")

Display = Button(root, height = 2,
 width = 20,
 text = "Показати",
 command = lambda:Take_input())

l.pack()
inputtxt.pack()
Display.pack()
Output.pack()

mainloop()
```

## Message

Віджет повідомлень, який використовується для відображення повідомлень користувача про наведені додатки python.

Текст повідомлення містить більше однієї строки.

```
w = Message(1, 2)
```

1: цей параметр використовується для зображення батьківського вікна.

2: доступно безліч опцій, і їх можна використовувати як пару ключів-значень, розділених зап'ятыми.

Параметри:

З цим віджетом можна використовувати наступні часто використовувані опції: -

- **bg**: цей параметр використовується для представлення звичайного кольору фону.
- **bd**: цей параметр використовується для представлення розміру границі, значення за умовчанням - 2 пікселя.
- **cursor**: при використанні цієї опції курсор миші зміниться на цей шаблон, коли він буде обраний.
- **fonts**: цей параметр використовується для представлення шрифту, використовується для тексту.
- **fg**: цей параметр використовується для представлення кольору, використовується для візуалізації тексту.
- **high**: цей параметр використовується для представлення кількості рядків тексту в повідомленні.
- **image**: цей параметр використовується для відображення графічного зображення на відео.
- **justify**: цей параметр використовується для керування вирівнюванням тексту: ЦЕНТР, ВЛЕВО або ВПРАВО.
- **padx**: цей параметр використовується для вказівки, скільки місць залишити слів і справа від віджета та тексту. Значення за умовчанням - 1 піксель.

- `padu`: цей параметр використовується для вказівки, скільки місць потрібно поставити над і під виджетом. Значення за умовчанням - 1 піксель.
- `relief`: тип границі виджета. За замовчуванням встановлено значення `FLAT`.
- `text`: ця опція використовувала символи нової строки («\n») для відображення кількох рядків тексту.

### Приклад

```
from tkinter import *

root = Tk()
root.geometry("300x200")

w = Label(root, text = 'КПІ', font = "50")
w.pack()

msg = Message(root, text = "Знати хочуть багато,
добувати знання - не всі.")

msg.pack()

root.mainloop()
```

### Menu

Меню є важливою частиною будь-якого графічного інтерфейсу користувача. Зазвичай меню використовують для забезпечення зручного доступу до різноманітних операцій, таких як збереження або відкриття файлу, вихід з програми або маніпулювання даними.

Меню верхнього рівня відображаються безпосередньо під рядком заголовка кореневого або будь-якого іншого вікна верхнього рівня.

```
menu = Menu(1, 2)
```

1: цей параметр використовується для зображення батьківського вікна.

2: доступно безліч опцій, і їх можна використовувати як пару ключів-значень, розділених зап'ятыми.

```
імпорт
```

```
from tkinter import *
from tkinter.ttk import *
from time import strftime

створення вікна tkinter
root = Tk()
root.title('Menu ')

Створення панелі меню
menubar = Menu(root)

Додавання меню «Файл» і команд
file = Menu(menubar, tearoff = 0)
menubar.add_cascade(label = 'File', menu = file)
file.add_command(label = 'New File', command = None)
file.add_command(label = 'Open...', command = None)
file.add_command(label = 'Save', command = None)
file.add_separator()
file.add_command(label = 'Exit', command = root.destroy)

Додавання меню редагування та команд
edit = Menu(menubar, tearoff = 0)
menubar.add_cascade(label = 'Edit', menu = edit)
edit.add_command(label = 'Cut', command = None)
edit.add_command(label = 'Copy', command = None)
edit.add_command(label = 'Paste', command = None)
edit.add_command(label = 'Select All', command = None)
```



```

edit.add_separator()
edit.add_command(label = 'Find...', command = None)
edit.add_command(label = 'Find again', command = None)

Додавання меню довідки
help_ = Menu(menubar, tearoff = 0)
menubar.add_cascade(label = 'Help', menu = help_)
help_.add_command(label = ' Help', command = None)
help_.add_command(label = 'Demo', command = None)
help_.add_separator()
help_.add_command(label = 'information', command = None)

дисплей меню
root.config(menu = menubar)
mainloop()

```

## Spinbox

Віджет Spinbox використовується для вибору з фіксованого числа значень. Це альтернативний виджет Entry, який надає користувачеві діапазон значень.

w = повідомлення (1, 2)

1: цей параметр використовується для зображення батьківського вікна.

2: доступно безліч опцій, і їх можна використовувати як пару ключів-значень, розділених зап'ятыми.

Параметри:

З цим виджетом можна використовувати наступні часто використовувані опції:

-

- **activebackground**: цей параметр використовується для зображення кольору фону, коли ползунок і стрілки знаходяться під курсором.
- **bg**: цей параметр використовується для представлення звичайного кольору фону, що відображається за меткою та індикатором.

- **bd:** цей параметр використовується для представлення розміру межі навколо індикатора, значення за умовчанням - 2 пікселя.
- **cursor:** при використанні цієї опції курсор миші змінюється на цей узор, коли він відкривається над типом.
- **disabledforeground:** цей параметр використовується для відображення кольору переднього плану віджета, коли він вимкнений.
- **disabledbackground:** цей параметр використовується для зображення кольору фону віджета, коли він відключений.
- **шрифт:** цей параметр використовується для представлення шрифту, використовується для тексту.
- **fg:** цей параметр використовується для представлення кольору, використовується для візуалізації тексту.
- **format:** цей параметр використовується для форматування рядків і не має значення за умовчанням.
- **from\_:** цей параметр використовується для представлення мінімального значення.
- **justify:** цей параметр використовується для керування вирівнюванням тексту: ЦЕНТР, ВЛЕВО або ВПРАВО.
- **relief:** цей параметр використовується для позначення типу границі, а його значення за умовчанням встановлено на SUNKEN.
- **repeatdelay:** цей параметр використовується для керування автоматичним повторенням кнопки, його значення за умовчанням - у мілісекундах.
- **Repeatinterval:** ця опція схожа на repeatdelay.
- **state:** цей параметр використовується для представлення стану віджета, а його значення за умовчанням - НОРМАЛЬНЕ.
- **textvariable:** цей параметр використовується для керування поведінкою тексту віджета.
- **to:** Указує максимальний предел значення віджета. Другой определяется параметром from\_.
- **validate:** цей параметр використовується для керування перевіркою значень віджета.
- **validatecommand:** цей параметр пов'язаний із зворотним викликом функції, яка використовується для перевірки вмісту віджета.
- **значення:** цей параметр використовується для представлення кортежу, що містить значення для цього віджета.
- **vcmd:** ця опція аналогічна команді перевірки.

- `wrap`: цей параметр завершує кнопки вгору та вниз на `Spinbox`.
- `xscrollcommand`: ця опція встановлена для методу `set()` смуги прокрутки, щоб зробити цей виджет горизонтально прокручуваним.

Методи:

У цих виджетах використовуються такі методи:

- `delete(startindex, endindex)`: цей метод використовується для видалення символів, присутніх у вказаному діапазоні.
- `get(startindex, endindex)`: цей метод використовується для отримання символів, присутніх у вказаному діапазоні.
- `Identify(x, y)`: цей метод використовується для ідентифікації елемента віджета у вказаному діапазоні.
- `index(індекс)`: цей метод використовується для отримання абсолютного значення даного індексу.
- `invoke(елемент)`: цей метод використовується для виклику зворотного виклику, пов'язаного з віджетом.

```
from tkinter import *

root = Tk()
root.geometry("300x200")

w = Label(root, text = 'KPI', font = "50")
w.pack()

sp = Spinbox(root, from_ = 0, to = 20)
sp.pack()

root.mainloop()
```

## Progressbar widget

Мета цього віджету – переконати користувача в тому, що щось відбувається. Він може працювати в одному з двох режимів.

У детермінованому режимі віджет показує індикатор, який переміщається з початку до кінця під керуванням програми.

У невизначеному режимі віджет анімується, тому користувач думатиме, що

щось відбувається. У цьому режимі індикатор підстрибує між кінцями віджету.

```
widget_object = Progressbar(parent, **options)

from tkinter import *
from tkinter.ttk import *

створення вікна tkinter
root = Tk()

Віджет панелі прогресу
progress = Progressbar(root, orient = HORIZONTAL,
 length = 100, mode = 'determinate')

Функція, відповідальна за оновлення
значення індикатора виконання
def bar():
 import time
 progress['value'] = 20
 root.update_idletasks()
 time.sleep(1)

 progress['value'] = 40
 root.update_idletasks()
 time.sleep(1)

 progress['value'] = 50
 root.update_idletasks()
```

```
time.sleep(1)

progress['value'] = 60
root.update_idletasks()
time.sleep(1)

progress['value'] = 80
root.update_idletasks()
time.sleep(1)
progress['value'] = 100

progress.pack(pady = 10)

Ця кнопка буде ініціалізована
індикатор прогресу
Button(root, text = 'Start', command = bar).pack(pady =
10)

нескінченний цикл
mainloop()
```

## Приклад 2

```
from tkinter import *
from tkinter.ttk import *

root = Tk()
```

```
progress = Progressbar(root, orient = HORIZONTAL,
 length = 100, mode = 'indeterminate')
```

```
def bar():
```

```
 import time
```

```
 progress['value'] = 20
```

```
 root.update_idletasks()
```

```
 time.sleep(0.5)
```

```
 progress['value'] = 40
```

```
 root.update_idletasks()
```

```
 time.sleep(0.5)
```

```
 progress['value'] = 50
```

```
 root.update_idletasks()
```

```
 time.sleep(0.5)
```

```
 progress['value'] = 60
```

```
 root.update_idletasks()
```

```
 time.sleep(0.5)
```

```
 progress['value'] = 80
```

```
 root.update_idletasks()
```

```
 time.sleep(0.5)
```

```
 progress['value'] = 100
```

```
root.update_idletasks()
time.sleep(0.5)
```

```
progress['value'] = 80
root.update_idletasks()
time.sleep(0.5)
```

```
progress['value'] = 60
root.update_idletasks()
time.sleep(0.5)
```

```
progress['value'] = 50
root.update_idletasks()
time.sleep(0.5)
```

```
progress['value'] = 40
root.update_idletasks()
time.sleep(0.5)
```

```
progress['value'] = 20
root.update_idletasks()
time.sleep(0.5)
progress['value'] = 0
```

```
progress.pack(pady = 10)
```

```
Button(root, text = 'Start', command = bar).pack(pady = 10)
```

```
mainloop()
```

## Scrollbar

Віджет смуги прокручування використовується для прокручування вмісту вниз. Ми також можемо створити горизонтальні смуги прокручування для віджету Entry.

```
w = Message(1, 2)
```

1: цей параметр використовується для подання батьківського вікна.

2: доступно безліч опцій, і їх можна використовувати як пари ключ-значення, розділені комами.

- **activebackground:** цей параметр використовується для представлення кольору фону віджету, коли він знаходиться у фокусі.
- **bg:** цей параметр використовується для представлення кольору фону віджету.
- **bd:** цей параметр використовується для представлення ширини межі віджету.
- **command:** ця опція може бути встановлена для процедури, пов'язаної зі списком, яка може викликатися щоразу при переміщенні смуги прокручування.
- **cursor:** в цьому варіанті курсор миші змінюється на тип курсору, встановлений для цього параметра, який може бути стрілкою, точкою тощо.
- **elementborderwidth:** цей параметр використовується для представлення ширини кордону навколо стрілок і повзунка. Стандартне значення -1.
- **highlightbackground:** цей параметр використовується для виділення виділення кольором, коли віджет не має фокусу.
- **highlightcolor:** цей параметр використовується для виділення кольору у фокусі, коли у фокусі знаходиться віджет.
- **Highlightthickness:** цей параметр використовується для представлення товщини виділення у фокусі.



- `jump`: цей параметр використовується для керування поведінкою переходу під час прокручування. Якщо він встановлений в 1, зворотний виклик викликається, коли користувач відпускає кнопку миші.
- `orient`: цей параметр може бути встановлений на ГОРИЗОНТАЛЬНО або ВЕРТИКАЛЬНО залежно від орієнтації смуги прокручування.
- `repeatdelay`: цей параметр вказує на тривалість, до якої повинна бути натиснута кнопка, перш ніж повзунок почне багаторазово переміщатися в цьому напрямку. За замовчуванням 300 мс.
- `repeatinterval`: значення інтервалу повтору за замовчуванням - 100.
- `takefocus`: ви можете вкладати фокус через віджет смуги прокручування
- `troughcolor`: цей параметр використовується для представлення кольору
- `width`: цей параметр використовується для представлення ширини смуги прокручування.

У цих віджетах використовуються такі методи:

- `get ()`: цей метод використовується для повернення двох чисел `a` та `b`, які представляють поточну позицію смуги прокручування.
- `set (first, last)`: цей метод використовується для підключення смуги прокручування до іншого виджету.

```
from tkinter import *
root = Tk()
root.geometry("150x200")
```

```
w = Label(root, text = 'КПІ',
 font = "50")
```

```
w.pack()
```

```
scroll_bar = Scrollbar(root)
```

```
scroll_bar.pack(side = RIGHT,
 fill = Y)
```

```
mylist = Listbox(root,
 yscrollcommand = scroll_bar.set)
```

```

for line in range(1, 26):
 mylist.insert(END, "КПІ " + str(line))

mylist.pack(side = LEFT, fill = BOTH)

scroll_bar.config(command = mylist.yview)

root.mainloop()

```

## ScrolledText Widget

```

import tkinter as tk
from tkinter import ttk
from tkinter import scrolledtext
win = tk.Tk()
win.title("ScrolledText ")
Мітка
ttk.Label(win,
 text = "ScrolledText Widget приклад",
 font = ("Times New Roman", 15),
 background = 'green',
 foreground = "white").grid(column = 0,
 row = 0)

Створення тексту
віджет області
text_area = scrolledtext.ScrolledText(win,
 wrap = tk.WORD,
 width = 40,
 height = 10,
 font = ("Times New
Roman",
 15))

text_area.grid(column = 0, pady = 10, padx = 10)

Розташування курсора в текстовій області

```

```
text_area.focus()
win.mainloop()
```



### **Приклад - Виджет ScrolledText, який створює текст tkinter для читання**

```
import tkinter as tk
import tkinter.scrolledtext as st
```

```
win = tk.Tk()
win.title("ScrolledText Widget")
```

```
tk.Label(win,
 text = "ScrolledText",
 font = ("Times New Roman", 15),
 background = 'green',
 foreground = "white").grid(column = 0,
 row = 0)
```

```
Створюємо текстову область
віджет з атрибутом " для читання"
```

```
text_area = st.ScrolledText(win,
```

```

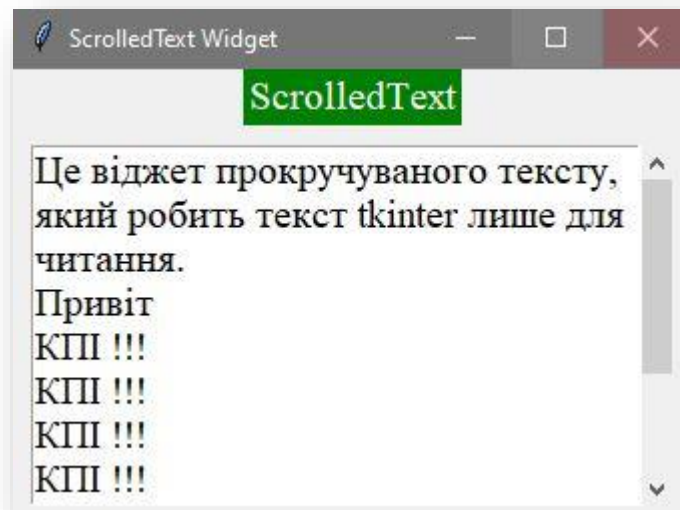
width = 30,
height = 8,
font = ("Times New Roman",
 15))

text_area.grid(column = 0, pady = 10, padx = 10)

Вставка тексту, для читання
text_area.insert(tk.INSERT,
"""\
Цей віджет тексту, який робить текст tkinter лише для
читання.
Привіт
КПІ !!!
КПІ !!!
КПІ !!!
КПІ !!!
КПІ !!!
КПІ !!!
КПІ !!!
""")

текст тільки для читання
text_area.configure(state = 'disabled')
win.mainloop()

```



## Listbox Widget

Віджет Listbox використовується для відображення різних типів елементів. Ці елементи повинні мати один і той самий тип і колір шрифту. Елементи також мають бути текстового типу. Користувач може вибрати один або кілька елементів із поданого списку відповідно до вимог.

```
listbox = Listbox(root, bg, fg, bd, height, width, font, ..)
```

### Optional parameters

- **root** - вікно.
- **bg** - колір фону
- **fg** - колір переднього плану
- **bd** - межа GUI
- **height** - висота віджету
- **width** - ширина
- **font** - шрифт.
- **highlightcolor** - Колір елементів списку під час фокусування.
- **yscrollcommand** - для вертикального прокручування.
- **xscrollcommand** - для прокручування по горизонталі.
- **cursor** - Курсор на віджеті, який може бути стрілкою, точкою тощо.

### Common methods

- **yview** - дозволяє вертикально прокручувати віджет.
- **xview** - дозволяє горизонтально прокручувати віджет.
- **get()** - щоб отримати елементи списку в заданому діапазоні.
- **activate(index)** - щоб вибрати рядки з вказаним індексом.
- **size()** - повертає кількість наявних рядків.
- **delete(start, last)** - видалити рядки у вказаному діапазоні.

- **nearest(y)** - повертає індекс найближчого рядка.
- **curseselection()** - повертає кортеж для всіх номерів рядків, які вибираються.

```
from tkinter import *

створити кореневе вікно.
top = Tk()

створити об'єкт списку
listbox = Listbox(top, height = 10,
 width = 15,
 bg = "grey",
 activestyle = 'dotbox',
 font = "Helvetica",
 fg = "yellow")

Визначте розмір вікна.
top.geometry("300x250")

Визначте мітку для списку.
label = Label(top, text = " Щось смачне")

вставити елементи за їх
індексами та іменами.
listbox.insert(1, "Вареники")
listbox.insert(2, "Борщ")
listbox.insert(3, "Котлета по київськи")
listbox.insert(4, "Піцца")
listbox.insert(5, "Шоколад")

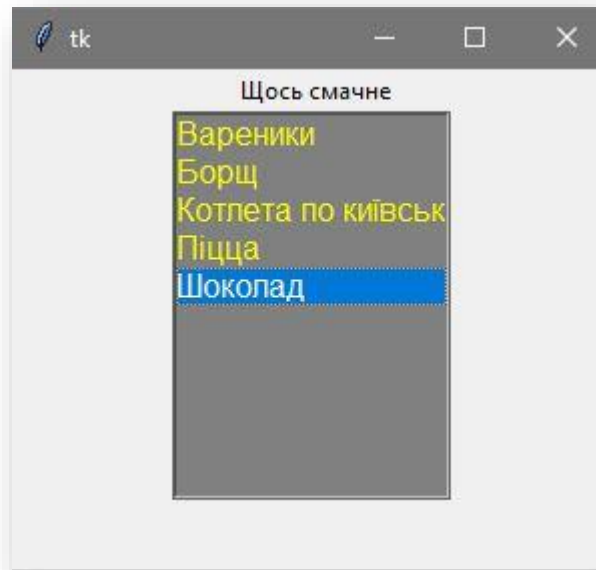
запакуйте віджети
label.pack()
listbox.pack()

Відображати до користувача
```

```
top.mainloop()
```

Давайте видалимо елементи зі створеного вище списку

```
Видалити елементи зі списку
вказавши індекс.
listbox.delete(2)
```



## Frame

Рамка(frame) — це прямокутна область на екрані.

Фрейм також можна використовувати як базовий клас для реалізації складних віджетів. Він використовується для організації групи віджетів.

Синтаксис:

```
f = frame(1, 2)
```

1: цей параметр використовується для створення батьківського вікна.

2: доступно безліч опцій, і їх можна використовувати як пари ключ-значення, розділені комами.

Опції:

З цим віджетом можна використовувати наступні варіанти: -

bg: цей параметр використовується для представлення звичайного кольору фону, що відображається за міткою та індикатором.

bd: цей параметр використовується для представлення розміру рамки навколо індикатора, а значення за замовчуванням становить 2 пікселі.

cursor: допомогою цього параметра курсор миші змінюватиметься на цей

візерунок, коли він перебуватиме над кадром.

height: вертикальний розмір нової рамки.

highlightcolor: цей параметр використовується для представлення кольору виділення фокусу, коли рамка має фокус.

highlightthickness: цей параметр використовується для представлення кольору виділення фокусу, коли кадр не має фокусу.

highlightbackground: цей параметр використовується для представлення товщини виділення фокусу.

relief: Тип рамки. За замовчуванням встановлено значення FLAT.

width: цей параметр використовується для визначення ширини кадру.

```
from tkinter import *
root = Tk()
root.geometry("300x150")

w = Label(root, text = 'КПІ', font = "50")
w.pack()

frame = Frame(root)
frame.pack()

bottomframe = Frame(root)
bottomframe.pack(side = BOTTOM)

b1_button = Button(frame, text = "КПІ1", fg = "red")
b1_button.pack(side = LEFT)

b2_button = Button(frame, text = "КПІ2", fg = "brown")
b2_button.pack(side = LEFT)

b3_button = Button(frame, text = "КПІ3", fg = "blue")
b3_button.pack(side = LEFT)

b4_button = Button(bottomframe, text = "КПІ4", fg
= "green")
b4_button.pack(side = BOTTOM)

b5_button = Button(bottomframe, text = "КПІ5", fg
```



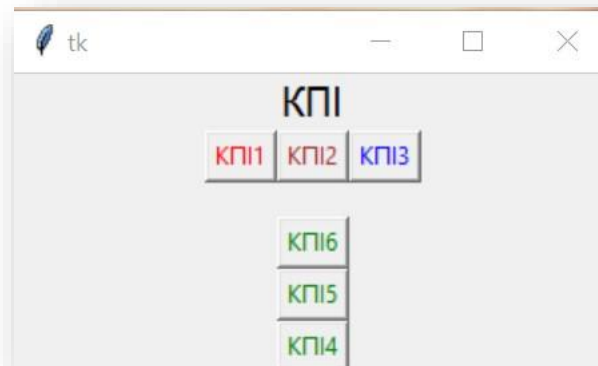
```

="green")
b5_button.pack(side = BOTTOM)

b6_button = Button(bottomframe, text ="КПІ6", fg
="green")
b6_button.pack(side = BOTTOM)

root.mainloop()

```



## Scale

Віджет «Scale» використовується кожен раз, коли ми хочемо вибрати конкретне значення з діапазону значень. Він надає ползунок, через який ми можемо вибирати значення, переміщаючись слева направо або зверху вниз в залежності від орієнтації ползунка.

```
S = Scale(root, bg, fg, bd, command, orient, from_, to, ..)
```

## Параметри

- **root** – вікно.
- **bg** – колір фону
- **fg** – колір переднього плану
- **bd** – межа GUI
- **orient** – орієнтація (вертикальна або горизонтальна)
- **from\_** – початкове значення
- **to** – кінцеве значення
- **troughcolor** – встановити колір
- **state** – вирішує, чи буде віджет реагувати чи не реагувати.

- `sliderlength` – визначає довжину повзунка.
- `label` – щоб відобразити мітку у віджеті.
- `highlightbackground` – щоб відобразити мітку у віджеті
- `cursor` – курсор на віджеті, який може бути стрілкою, колом, крапкою тощо.

### Методи

`set(value)` – встановити значення для масштабу.

`get()` – отримати значення масштабу.

### **Горизонтальний скейл**

```
from tkinter import *
root = Tk()
root.geometry("400x300")
v1 = DoubleVar()
def show1():

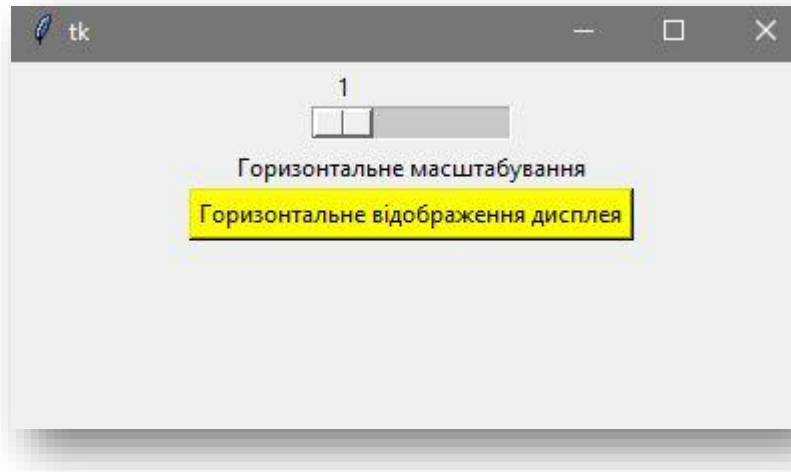
 sel = "Horizontal Scale Value = " + str(v1.get())
 l1.config(text = sel, font = ("Courier", 14))
s1 = Scale(root, variable = v1,
 from_ = 1, to = 100,
 orient = HORIZONTAL)
l3 = Label(root, text = "Горизонтальне масштабування")

b1 = Button(root, text = "Горизонтальне відображення
дисплея",
 command = show1,
 bg = "yellow")

l1 = Label(root)

s1.pack(anchor = CENTER)
l3.pack()
b1.pack(anchor = CENTER)
l1.pack()
```

```
root.mainloop()
```



### Вертикальний скейл

```
from tkinter import *
root = Tk()
root.geometry("400x300")
v2 = DoubleVar()

def show2():
 sel = "Vertical Scale Value = " + str(v2.get())
 l2.config(text = sel, font = ("Courier", 14))

s2 = Scale(root, variable = v2,
 from_ = 50, to = 1,
 orient = VERTICAL)

l4 = Label(root, text = "Вертикальний масштаб")

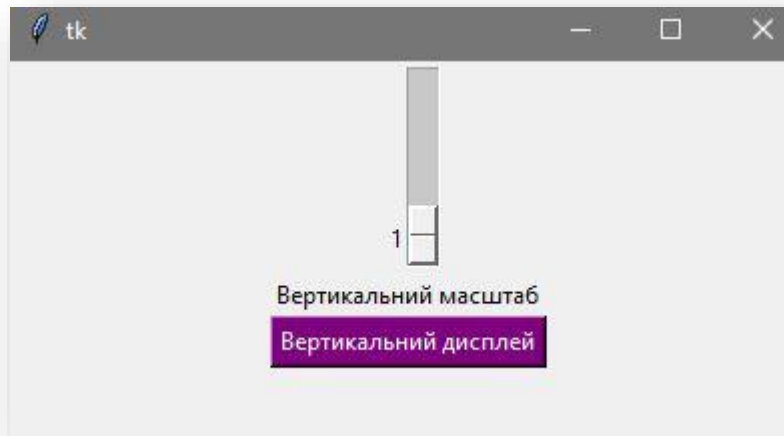
b2 = Button(root, text = "Вертикальний дисплей",
 command = show2,
 bg = "purple",
 fg = "white")

l2 = Label(root)

s2.pack(anchor = CENTER)
```

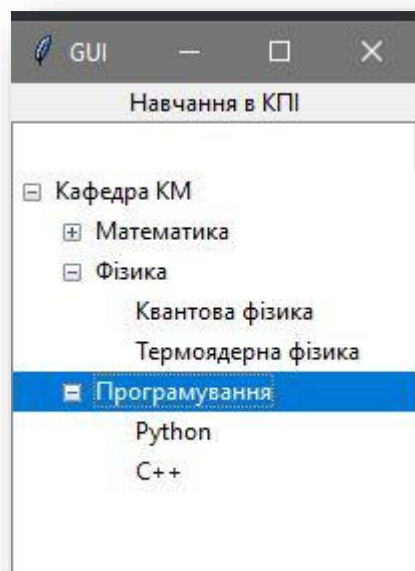
```
l4.pack()
b2.pack()
l2.pack()

root.mainloop()
```



## Treeview

Цей віджет корисний для візуалізації та дозволу навігації по ієрархії елементів. Він може відображати більше однієї функції кожного елемента в ієрархії. Він може побудувати дерево як інтерфейс користувача, як у провіднику Windows. Тому тут ми будемо використовувати Tkinter, щоб побудувати ієрархічне деревоподібне представлення в програмі Python GUI.



```
from tkinter import *
from tkinter import ttk
Створення вікна програми
app = Tk()
Визначення назви програми
app.title("GUI")
Визначення мітки програми та виклик геометрії
метод управління, тобто упакувати, щоб організувати
віджети у формі блоків перед їх пошуком
у батьківському віджеті
ttk.Label(app, text = "Навчання в КПІ").pack()
Створення вікна перегляду дерева
treeview = ttk.Treeview(app)
treeview.pack()
Вставлення елементів до дерева
Вставлення батьківського елемента
treeview.insert('', '0', 'item1',
 text = 'Кафедра КМ')
Вставка базової гілки у дереві
treeview.insert('', '1', 'item2',
 text = 'Математика')
treeview.insert('', '2', 'item3',
 text = 'Фізика')
treeview.insert('', 'end', 'item4',
 text = 'Програмування')

Вставлення більше одного атрибута елемента
treeview.insert('item2', 'end', 'Алгебра',
 text = 'Алгебра')
treeview.insert('item2', 'end', 'Геометрія',
 text = 'Геометрія')
treeview.insert('item3', 'end', 'Квантова фізика',
 text = 'Квантова фізика')
treeview.insert('item3', 'end', 'Термоядерна фізика',
 text = 'Термоядерна фізика')
treeview.insert('item4', 'end', 'Python',
 text = 'Python')
```

```

treeview.insert('item4', 'end', 'C++',
 text = 'C++')

Розміщення кожного дочірнього елемента в батьківському
віджеті
treeview.move('item2', 'item1', 'end')
treeview.move('item3', 'item1', 'end')
treeview.move('item4', 'item1', 'end')

Виклик main()
app.mainloop()

```

### Treeview scrollbar

Якщо смуга прокрутки використовує віджети перегляду дерева, то цей тип смуги прокручування називається смугою прокрутки дерева (**Treeview scrollbar**). У цьому випадку віджет перегляду дерева корисний для відображення більше ніж однієї функції кожного елемента, переліченого в дереві, праворуч від дерева у вигляді стовпців. Однак це можна реалізувати за допомогою tkinter у Python за допомогою деяких віджетів і методів керування геометрією, які підтримуються tkinter.

```

from tkinter import ttk
import tkinter as tk
Створення вікна tkinter

window = tk.Tk()
window.resizable(width = 1, height = 1)

Використання віджета перегляду дерева
treev = ttk.Treeview(window, selectmode = 'browse')

Метод пакету викликів щодо перегляду дерева
treev.pack(side = 'right')

Побудова вертикальної смуги прокрутки
із переглядом дерева
verscrlbar = ttk.Scrollbar(window,
 orient = "vertical",

```

```

command = treev.yview)

Метод пакету викликів відносно вертикалі
смуга прокрутки
verscrlbar.pack(side = 'right', fill = 'x')

Налаштування перегляду дерева
treev.configure(xscrollcommand = verscrlbar.set)

Визначення кількості стовпців
treev["columns"] = ("1", "2", "3")
Визначення заголовка

treev['show'] = 'headings'

Призначення ширини та прив'язки до
відповідні стовпці
treev.column("1", width = 90, anchor = 'c')
treev.column("2", width = 90, anchor = 'se')
treev.column("3", width = 90, anchor = 'se')

Присвоєння назв заголовків
відповідні стовпці
treev.heading("1", text = "Ім'я")
treev.heading("2", text = "Стать")
treev.heading("3", text = "Вік")

Вставлення елементів та їхніх функцій у
стовпці
treev.insert("", 'end', text = "L1",
 values = ("Оксана", "Ж", "20"))
treev.insert("", 'end', text = "L2",
 values = ("Наталка", "Ж", "23"))
treev.insert("", 'end', text = "L3",
 values = ("Олена", "Ж", "21"))
treev.insert("", 'end', text = "L4",
 values = ("Олексій", "М", "20"))
treev.insert("", 'end', text = "L5",

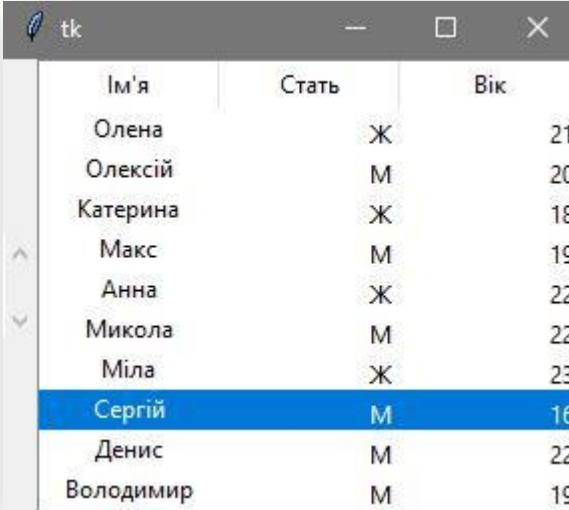
```

```

 values = ("Катерина", "Ж", "18"))
treev.insert("", 'end', text = "L6",
 values = ("Макс", "М", "19"))
treev.insert("", 'end', text = "L7",
 values = ("Анна", "Ж", "22"))
treev.insert("", 'end', text = "L8",
 values = ("Микола", "М", "22"))
treev.insert("", 'end', text = "L10",
 values = ("Міла", "Ж", "23"))
treev.insert("", 'end', text = "L11",
 values = ("Сергій", "М", "16"))
treev.insert("", 'end', text = "L12",
 values = ("Денис", "М", "22"))
treev.insert("", 'end', text = "L13",
 values = ("Володимир", "М", "19"))

Виклик основного циклу
window.mainloop()

```



| Ім'я      | Стать | Вік |
|-----------|-------|-----|
| Олена     | Ж     | 21  |
| Олексій   | М     | 20  |
| Катерина  | Ж     | 18  |
| Макс      | М     | 19  |
| Анна      | Ж     | 22  |
| Микола    | М     | 22  |
| Міла      | Ж     | 23  |
| Сергій    | М     | 16  |
| Денис     | М     | 22  |
| Володимир | М     | 19  |

## Toplevel

Віджет верхнього рівня використовується для створення вікна поверх усіх інших вікон. Віджет верхнього рівня використовується для надання додаткової інформації користувачеві, а також коли наша програма працює з кількома програмами.

Ці вікна безпосередньо організовуються та керуються диспетчером вікон і не



потребують кожного разу пов'язувати з ними будь-яке батьківське вікно.

```
toplevel = Toplevel(root, bg, fg, bd, height, width,
font, ..)
```

#### Параметри

- **root** = вікно
- **bg** = колір фону
- **fg** = колір переднього плану
- **bd** = межа
- **height** = висота
- **width** = ширина
- **font** = шрифт
- **cursor** = курсор

#### Поширені методи

**iconify** -перетворює вікна на піктограми.

**deiconify**- повертає значок у вікно.

**state** -повертає поточний стан вікна.

**remove** -видаляє вікно з екрана.

**title** -визначає заголовок вікна.

**frame** -повертає ідентифікатор вікна, який є системним.

#### Приклад

Створення кількох верхніх рівнів один над одним

```
from tkinter import *
Створіть кореневе вікно
із зазначеним розміром і назвою
root = Tk()
root.title("Root Window")
root.geometry("450x300")
Створення мітки для кореневого вікна
label1 = Label(root, text = "Це вікно")
створемо функцію
вікно, яке не пов'язане з
```

```
будь-яке батьківське вікно
def open_Toplevel2():

 # створемо віджет
 top2 = Toplevel()
 # визначити заголовок вікна
 top2.title("Дуже добре вікно")
 # вказати розмір
 top2.geometry("200x100")
 # Створення мітки
 label = Label(top2,
 text = "Це дуже добре вікно")
 # Створити кнопку виходу.
 button = Button(top2, text = "Exit",
 command = top2.destroy)

 label.pack()
 button.pack()

 # Показувати, поки не буде закрито вручну.
 top2.mainloop()
функція
яка пов'язаний з кореневим вікном.

def open_Toplevel1():

 # Створіть віджет
 top1 = Toplevel(root)

 # Визначити заголовок вікна
 top1.title("Дуже добре вікно")

 # вказати розмір
 top1.geometry("200x200")

 # створення мітки
 label = Label(top1,
 text = "Це вікно")
```

```

створення кнопки вихід
button1 = Button(top1, text = "Exit",
 command = top1.destroy)
створити кнопку для відкриття toplevel2

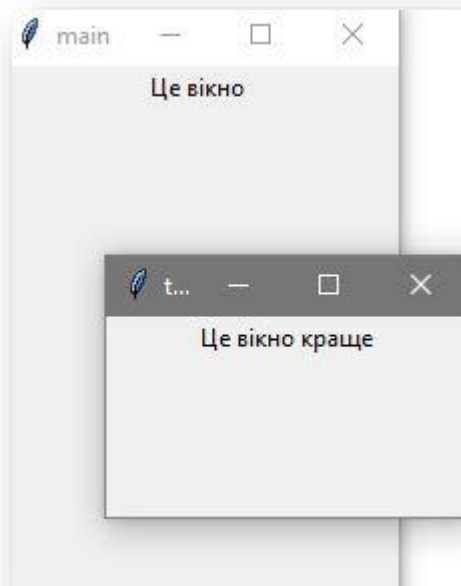
button2 = Button(top1, text = "Відкрити вікно",
 command = open_Toplevel2)

label.pack()
button2.pack()
button1.pack()
top1.mainloop()
button = Button(root, text = "відкрити вікно",
 command = open_Toplevel1)
label1.pack()

розташування кнопки по координатам
button.place(x = 155, y = 50)

root.mainloop()

```



### Askopenfile() function

Під час роботи з графічним інтерфейсом користувача може знадобитися

відкрити файли та прочитати дані з нього або може знадобитися записати дані в цей конкретний файл. Цього можна досягти за допомогою функції `open()` (вбудованої в Python), але ви не зможете вибрати необхідний файл, якщо не вкажете шлях до цього конкретного файлу в коді.

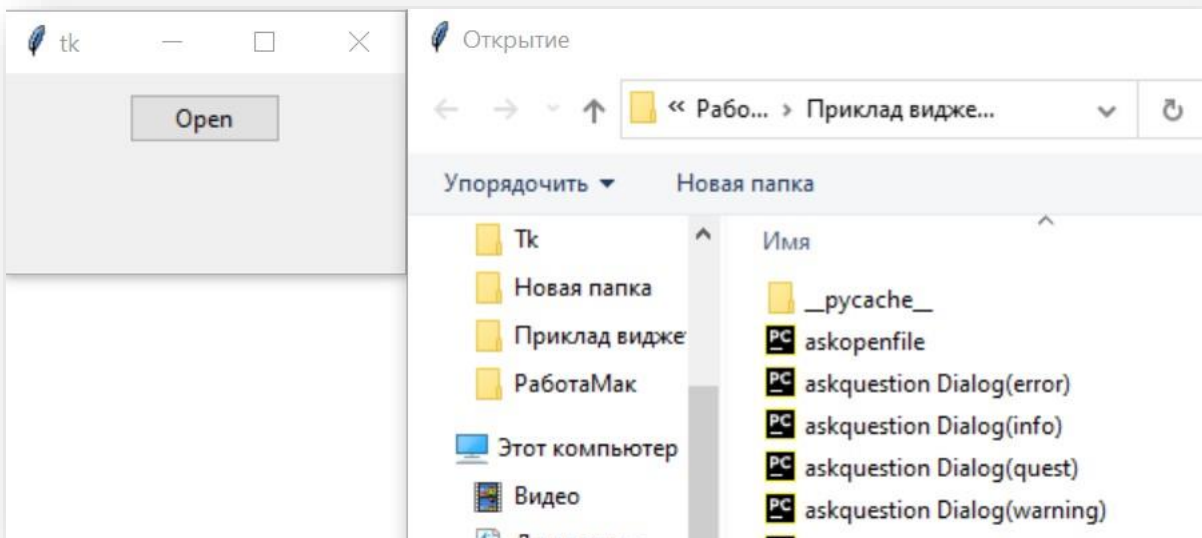
За допомогою графічного інтерфейсу ви можете не вимагати вказувати шлях до будь-якого файлу, але ви можете безпосередньо відкрити файл і прочитати його вміст.

Щоб використовувати функцію `askopenfile()`, вам може знадобитися виконати такі дії:

```
from tkinter import *
from tkinter.ttk import *
імпорт функції askopenfile
з діалогового вікна файлу класу
from tkinter.filedialog import askopenfile
root = Tk()
root.geometry('200x100')

Ця функція буде використана для відкриття
файл у режимі читання та лише файли Python
буде відкрито
def open_file():
 file = askopenfile(mode='r', filetypes=[('Python
Files', '*.py')])
 if file is not None:
 content = file.read()
 print(content)

btn = Button(root, text='Open', command=
lambda:open_file())
btn.pack(side=TOP, pady=10)
mainloop()
```



## asksaveasfile()

Під час роботи з файлами може знадобитися відкрити файли, виконати над ними операції, а потім зберегти файл. `asksaveasfile()` — це функція, яка використовується для збереження файлу користувача (розширення можна встановити явно або ви також можете встановити розширення за замовчуванням). Ця функція знаходиться в діалоговому вікні файлу класу.

```
from tkinter import *
from tkinter import ttk

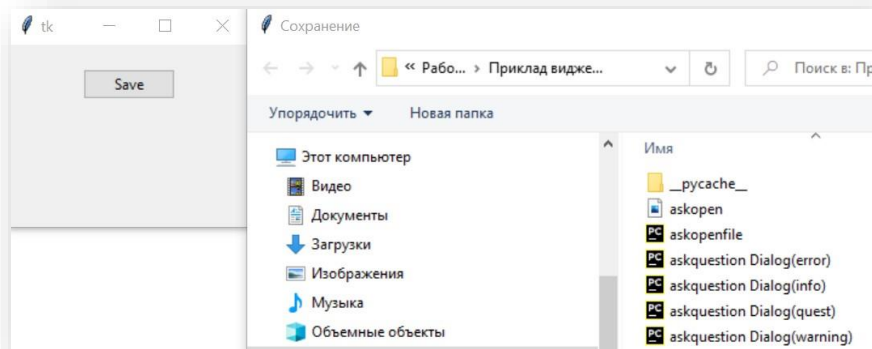
імпортувати лише asksaveasfile з діалогового вікна file
який використовується для збереження файлу з будь-яким
розширенням
from tkinter.filedialog import asksaveasfile

root = Tk()
root.geometry('200x150')
функція для виклику, коли користувач натискає
кнопка збереження, відкривається діалогове вікно файлу
відкрити та попросити зберегти файл
```

```
def save():
 files = [('All Files', '*.*'),
 ('Python Files', '*.py'),
 ('Text Document', '*.txt')]
 file = asksaveasfile(filetypes = files,
 defaultextension = files)

btn = ttk.Button(root, text = 'Save', command = lambda :
save())
btn.pack(side = TOP, pady = 20)

mainloop()
```



## Askquestion Dialog

У Python існують бібліотеки Serval для графічного інтерфейсу користувача. Tkinter є одним із найкорисніших. Це стандартний інтерфейс. У кожній програмі нам потрібно якесь повідомлення для відображення, наприклад - «Ви хочете закрити?» або якесь попередження або щось інше. Для цього Tkinter надає бібліотеку на зразок messagebox. Використовуючи бібліотеку вікон повідомлень, ми можемо показувати інформацію, помилку, попередження, скасування та інше у формі вікна повідомлень.

1. showinfo() – для відображення важливої інформації.
2. showwarning() – для відображення певного типу попередження.
3. showerror() – відображення повідомлення про помилку.
4. askquestion() – щоб відобразити діалогове вікно, яке запитує з двома параметрами ТАК або НІ.
5. askokcancel() – щоб відобразити діалогове вікно з двома параметрами

OK або CANCEL.

6. askretrycancel() – щоб відобразити діалогове вікно, яке запитує з двома параметрами ПОВТОРИТИ або СКАСУВАТИ.

7. askyesnocancel() – щоб відобразити діалогове вікно з трьома варіантами ТАК, НІ або СКАСУВАТИ.

```
from tkinter import *
from tkinter import messagebox
main = Tk()

функція для використання
askquestion().
def Submit():
 messagebox.askquestion("Посилання",
 "Ви бажаєте подати заяву")

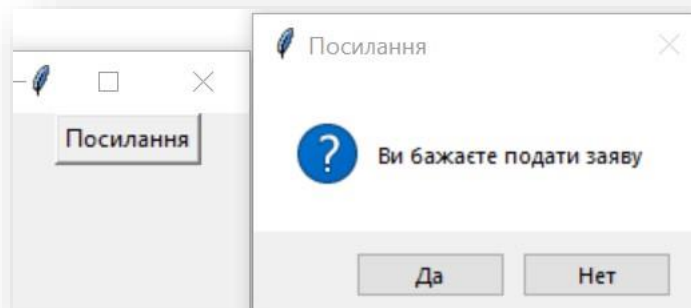
налаштування геометрії вікна

main.geometry("100x100")

створення вікна
B1 = Button(main, text = "Посилання", command = Submit)

Розташування кнопок
B1.pack()

нескінченний цикл до закриття
main.mainloop()
```



## Syntax of the Message-Box Functions:

```
messagebox.name_of_function(Title, Message, [, options])
```

1. name\_of\_function – назва функції, яку ми хочемо використовувати.
2. title– заголовок вікна повідомлення.
3. message– повідомлення, яке ви хочете показати в діалоговому вікні.
4. options– для налаштування параметрів.

### Askquestion()

Ця функція використовується для постановки запитань користувачу. Та має лише два варіанти ТАК або НІ.

Застосування цієї функції:

Ми можемо використовувати її, щоб запитати користувача, чи хоче він продовжити.

Ми можемо використовувати це, щоб запитати користувача, чи хотів він щось надсилати чи ні.

```
messagebox.askfunction((Title, Message, [, options])
```

```
from tkinter import *
from tkinter import messagebox
```

```
main = Tk()
```

```
функція для використання
```

```
askquestion().
```

```
def Submit():
```

```
 messagebox.askquestion("Посилання",
 "Ви бажаєте подати заяву")
```

```
налаштування геометрії вікна
```

```
main.geometry("100x100")
```

```
створення вікна
```

```
B1 = Button(main, text = "Посилання", command = Submit)
```



```
Розташування кнопок
B1.pack()

нескінченний цикл до закриття
main.mainloop()
```

## 1. Імпортування бібліотек

Щоб використовувати функції графічного інтерфейсу в Python, нам потрібно імпортувати бібліотеки. У першому рядку ми імпортуємо Tkinter, а в другому – бібліотеку messagebox

```
from tkinter import *
from tkinter import messagebox
```

## 2. Головне вікно

Ми повинні створити екземпляр або об'єкт для вікна TK(); Tk() — це функція Tkinter, яка створює вікно, на яке можна посилатися з основної змінної

```
main = Tk()
```

## 3. Встановимо розмір

ми встановлюємо розмір вікна, ми можемо встановити його різними способами. Тут ми встановлюємо за допомогою функції geometry() розмір «1000X1000».

```
top.geometry("1000x1000")
```

## 4. Застосування іншого віджета та функції

У нашому прикладі ми створюємо метод і викликаємо askquestion() і створюємо Button і налаштовуємо його за допомогою функції Pack().

```
def Submit():
 messagebox.askquestion("Посилання",
 "Ви бажаєте подати заяву")
```

```
налаштування геометрії вікна
```

```
main.geometry("100x100")
```

```
створення вікна
```

```
B1 = Button(main, text = "Посилання", command = Submit)
```

```
Розташування кнопок
```

```
B1.pack()
```

## 5. mainloop()

Цей метод можна використовувати, коли все готово до виконання. Він запускає

`mainloop()` , який використовується для запуску програми. Вікно відкриватиметься, доки не буде натиснута кнопка закриття.

ЗНАЧКИ, які ми можемо використовувати в параметрах

**1. Error**

**2. Info**

**3. Warning**

**4. Question**

Ми можемо змінити піктограму діалогового вікна. Тип значка, який ми хочемо використовувати, залежить лише від потреб програми. у нас є чотири іконки.

### Error

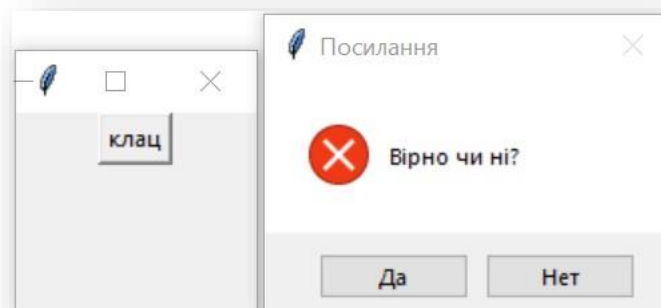
```
ілюстрація значка - помилка
from tkinter import *
from tkinter import messagebox
```

```
main = Tk()
```

```
def check():
 messagebox.askquestion("Посилання",
 "Вірно чи ні?",
 icon='error')
```

```
main.geometry("100x100")
B1 = Button(main, text = "клац", command = check)
B1.pack()
```

```
main.mainloop()
```



### **info**

```
messagebox.function_name(Title, Message, icon='info')
```

```
from tkinter import *
from tkinter import messagebox
```

```
main = Tk()
```

```
def check():
 messagebox.askquestion("Посилання",
 "Бажаєш продовжити?",
 icon = 'info')
```

```
main.geometry("100x100")
B1 = Button(main, text = "кlač", command = check)
B1.pack()
```

```
main.mainloop()
```

### **Question**

```
messagebox.function_name(Title, Message, icon='question')
```

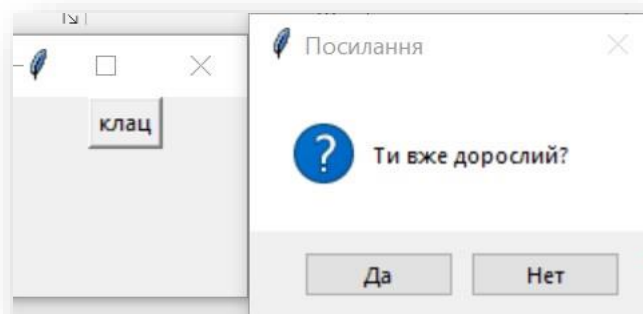
```
from tkinter import *
from tkinter import messagebox
```

```
main = Tk()
```

```
def check():
 messagebox.askquestion("Посилання",
 "Ти вже дорослий?",
 icon = 'question')
```

```
main.geometry("100x100")
B1 = Button(main, text = "кlač", command = check)
B1.pack()
```

```
main.mainloop()
```



### Warning

```
messagebox.function_name(Title, Message, icon='warning')
```

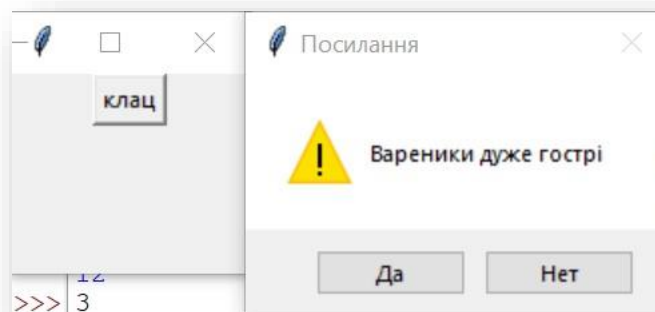
```
from tkinter import *
from tkinter import messagebox
```

```
main = Tk()
```

```
def check():
 messagebox.askquestion("Посилання",
 "Вареники дуже гострі",
 icon = 'warning')
```

```
main.geometry("100x100")
B1 = Button(main, text = "клац", command = check)
B1.pack()
```

```
main.mainloop()
```



## MessageBox

Python Tkinter – MessageBox Widget використовується для відображення вікон повідомлень у програмах Python. Цей модуль використовується для відображення повідомлень та надає ряд функцій.

`messagebox.Function_Name(title, message [, options])`

Параметри:

Існують різні параметри:

- **Function\_Name:** цей параметр використовується для представлення відповідної функції вікна повідомлень.
- **title:** Цей параметр є рядком, який відображається як заголовок вікна повідомлення.
- **message:** цей параметр є рядком, який буде відображатися як повідомлення у вікні повідомлення
- **options:** Є два варіанти, які можна використовувати:
  1. **default:** Цей параметр використовується для вказівки кнопки за замовчуванням, наприклад ПЕРЕРВАТИ, ПОВТОРИТИ або ІГНОРУВАТИ у вікні повідомлення.
  2. **parent:** цей параметр використовується для визначення вікна, поверх якого має відображатися вікно повідомлення.

Назва\_функції:

У віджеті вікна повідомлень доступні функції або методи.

**showinfo():** показує певну релевантну інформацію користувачеві.

**showwarning():** відображати попередження для користувача.

**showerror():** відображати повідомлення про помилку для користувача.

**askquestion():** поставити запитання, і користувач має відповісти так чи ні.

**askokcancel():** підтвердити дії користувача щодо певної активності програми.

**askyesno():** користувач може відповісти «так» або «ні» для певної дії.

**askretrycancel():** Запитайте користувача про повторне виконання певного завдання чи ні.

```
from tkinter import *
```

```
from tkinter import messagebox

root = Tk()
root.geometry("300x200")

w = Label(root, text = 'КПІ', font = "50")
w.pack()

messagebox.showinfo("showinfo", "Інформація")

messagebox.showwarning("showwarning", "Увага")

messagebox.showerror("showerror", "Помилка")

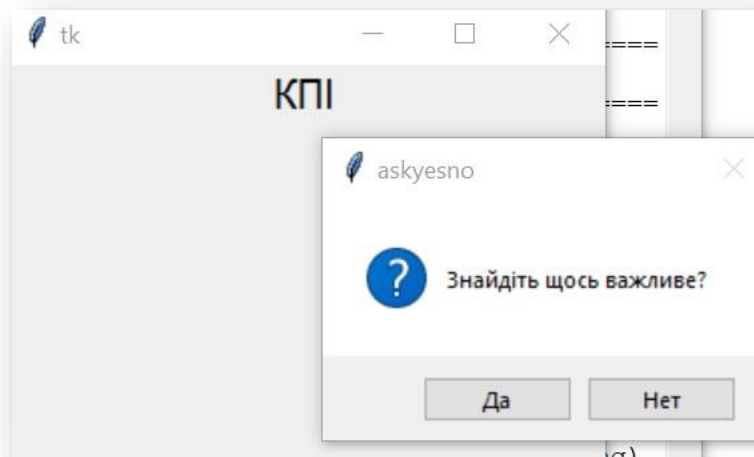
messagebox.askquestion("askquestion", "Ви впевнені?")

messagebox.askokcancel("askokcancel", "Бажаєте
продовжити?")

messagebox.askyesno("askyesno", "Знайдіть щось важливе?")

messagebox.askretrycancel("askretrycancel", "Працюйте
далі")

root.mainloop()
```



### **place() method**

Менеджер геометрії місця є найпростішим із трьох менеджерів загальної геометрії, доступних у Tkinter. Це дозволяє встановити положення та розмір вікна, або в абсолютних значеннях, або відносно іншого вікна.

Ви можете отримати доступ до менеджера розташування за допомогою методу `place()`, який доступний для всіх стандартних віджетів.

Зазвичай не варто використовувати `place()` для звичайних макетів, вікон і діалогів; просто потрібно багато працювати, щоб все працювало як слід. Використовуйте для таких цілей менеджери `pack()` або `grid()`.

`widget.place(relx = 0.5, rely = 0.5, anchor = CENTER)`

Метод `place()` можна використовувати як з методом `grid()`, так і з методом `pack()`.

```
from tkinter import *
from tkinter.ttk import *

створення вікна Tk
master = Tk()

налаштування геометрії вікна tk
master.geometry("200x200")

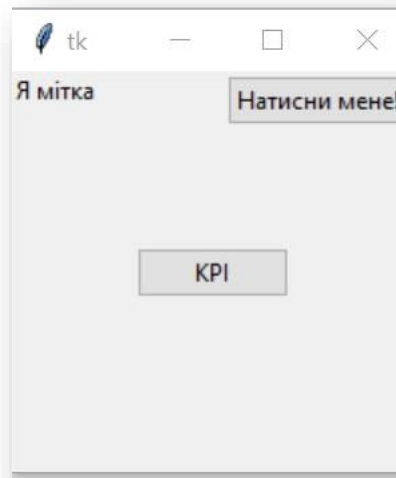
віджет кнопки
b1 = Button(master, text = "Натисни мене!")
```

```
b1.place(relx = 1, x = -2, y = 2, anchor = NE)

віджет мітки
l = Label(master, text = "Я мітка")
l.place(anchor = NW)

віджет кнопки
b2 = Button(master, text = "КРІ")
b2.place(relx = 0.5, rely = 0.5, anchor = CENTER)

mainloop()
```



Коли ми використовуємо менеджери `pack()` або `grid()`, то дуже легко розмістити два різні віджети окремо один від одного, але помістити один з них в інший трохи складно. Але цього можна легко досягти методом `place()`.

У методі `place()` ми можемо використовувати опцію `in_`, щоб помістити один віджет в інший.

```
from tkinter import *
from tkinter.ttk import *
```

```
master = Tk()
```

```
master.geometry("200x200")
```



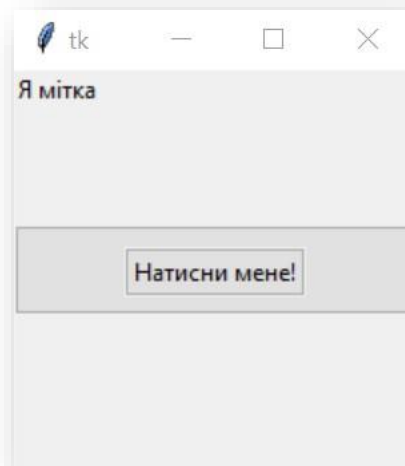
```
b2 = Button(master, text = "KPI")
b2.pack(fill = X, expand = True, ipady = 10)

b1 = Button(master, text = "Натисни мене!")

Тут b1 розміщується всередині b2 з опцією in_
b1.place(in_ = b2, relx = 0.5, rely = 0.5, anchor = CENTER)

l = Label(master, text = "Я мітка")
l.place(anchor = NW)

mainloop()
```



### **grid() method**

Менеджер геометрії сітки поміщає віджети у двовимірну таблицю. Головний віджет розбивається на кілька рядків і стовпців, і кожна «частинка» отриманої таблиці може містити віджет. Менеджер сітки є найбільш гнучким з менеджерів геометрії в Tkinter.

|              |                 |                |
|--------------|-----------------|----------------|
| Мітка 1      | Введення тексту | Малюнок        |
| Мітка 2      | Введення тексту |                |
| Якась кнопка |                 | Якась кнопка 2 |

```

from tkinter import *
from tkinter.ttk import *

master = Tk()
це створить віджет мітки

l1 = Label(master, text = "Перша:")
l2 = Label(master, text = "Друга:")

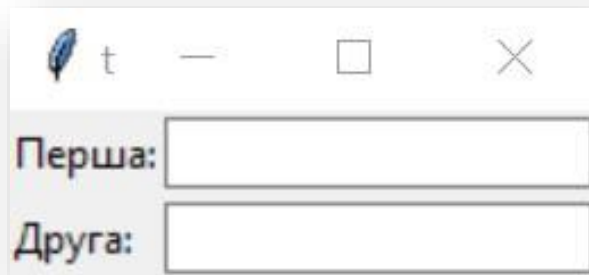
метод сітки для розташування міток відповідно
рядків і стовпців, як зазначено
l1.grid(row = 0, column = 0, sticky = W, pady = 2)
l2.grid(row = 1, column = 0, sticky = W, pady = 2)

віджети запису, які використовуються для отримання запису
від користувача
e1 = Entry(master)
e2 = Entry(master)

це впорядкує віджети входу
e1.grid(row = 0, column = 1, pady = 2)
e2.grid(row = 1, column = 1, pady = 2)

mainloop()

```



### **pack() method**

Менеджер геометрії Pack пакує віджети відносно попереднього віджета. Tkinter буквально пакує всі віджети один за одним у вікні. Ми можемо використовувати такі параметри, як заповнення, розширення та сторони, щоб керувати цим менеджером геометрії.

Порівняно з менеджером сітки, менеджер пакетів дещо обмежений, але його набагато легше використовувати в кількох, але досить поширених ситуаціях:

Додайте віджет у фрейм (або будь-який інший віджет-контейнер) і заповніть ним увесь фрейм

Розмістіть кілька віджетів один на одному

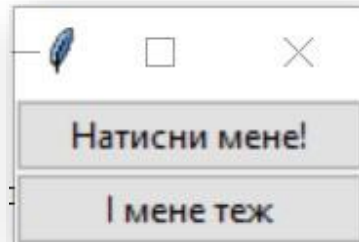
Розмістіть кілька віджетів поруч

Розміщення віджета всередині фрейму та заповнення всього фрейму. Ми можемо зробити це за допомогою параметрів розширення та заповнення.

```
from tkinter import *
from tkinter.ttk import *
```

```
master = Tk()
створення рамки, який може розширюватися відповідно
до розміру вікна
pane = Frame(master)
pane.pack(fill = BOTH, expand = True)
віджети кнопки, які також можна розгортати та заповнювати
повністю в батьківському віджеті
Кнопка 1
b1 = Button(pane, text = "Натисни мене!")
b1.pack(fill = BOTH, expand = True)
```

```
b2 = Button(pane, text = "I мене теж")
b2.pack(fill = BOTH, expand = True)
master.mainloop()
```



Розміщення віджетів один на одному та поруч. Ми можемо зробити це за допомогою стороннього варіанту.

```
from tkinter import *
```

```
master = Tk()
```

```
створення фрейму, який може розширюватися відповідно
до розміру вікна
```

```
pane = Frame(master)
```

```
pane.pack(fill = BOTH, expand = True)
```

```
віджети кнопки, які також можна розгортати та заповнювати
повністю в батьківському віджеті
```

```
Кнопка 1
```

```
b1 = Button(pane, text = "Доброго ранку !",
 background = "yellow", fg = "blue")
```

```
b1.pack(side = TOP, expand = True, fill = BOTH)
```

```
Кнопка 2
```

```
b2 = Button(pane, text = "Кави будласка",
 background = "blue", fg = "white")
```

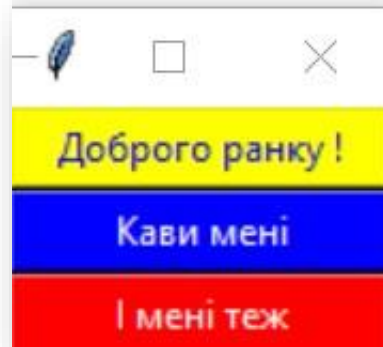
```
b2.pack(side = TOP, expand = True, fill = BOTH)
```

```
Кнопка 3
```

```
b3 = Button(pane, text = "I мені теж",
```

```
background = "red", fg = "white")
b3.pack(side = TOP, expand = True, fill = BOTH)

master.mainloop()
```



### Приклад 3

```
from tkinter import *
master = Tk()

створення фрейму, який може розширюватися відповідно
до розміру вікна
pane = Frame(master)
pane.pack(fill = BOTH, expand = True)

віджети кнопки, які також можна розгортати та заповнювати
повністю в батьківському віджеті
Кнопка 1
b1 = Button(pane, text = "Слава Україні!",
 background = "blue", fg = "yellow")
b1.pack(side = LEFT, expand = True, fill = BOTH)

Кнопка2
b2 = Button(pane, text = "Героям Слава",
 background = "yellow", fg = "blue")
b2.pack(side = LEFT, expand = True, fill = BOTH)

Кнопка33
```

```

b3 = Button(pane, text = "Слава нації",
 background = "red", fg = "white")
b3.pack(side = LEFT, expand = True, fill = BOTH)

master.mainloop()

```



## PanedWindow

Tkinter підтримує різноманітні віджети, щоб зробити GUI все більш привабливим і функціональним. Віджет PanedWindow — це віджет менеджера геометрії, який може містити одну або кілька панелей дочірніх віджетів. Користувач може змінити розмір дочірніх віджетів, пересуваючи смужки роздільних ліній за допомогою миші.

```

from tkinter import *
from tkinter import ttk

root = Tk()

об'єкт panedwindow
pw = PanedWindow(orient = 'vertical')

об'єкт panedwindow
top = ttk.Button(pw, text = "Натисни мене !\nЯ кнопка")
top.pack(side = TOP)

Це додасть віджет кнопки до панелі вікна
pw.add(top)

Checkbutton Widget
bot = Checkbutton(pw, text = "Вибери мене !")

```

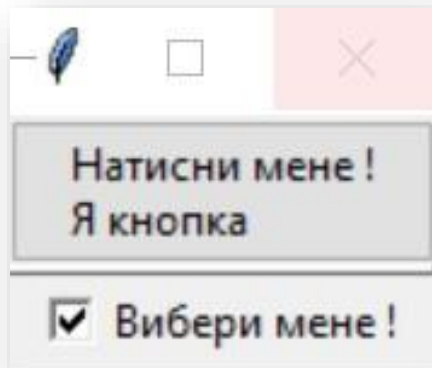
```
bot.pack(side = TOP)

Це додасть кнопку Checkbutton до панельного вікна
pw.add(bot)

expand використовується, щоб віджети могли розгортатися
fill використовується, щоб дозволити віджетам
налаштувати себе
відповідно до розміру головного вікна
pw.pack(fill = BOTH, expand = True)

Цей метод використовується для показу
pw.configure(sashrelief = RAISED)

mainloop()
```



## **PanedWindow з панелями**

```
from tkinter import *
from tkinter import ttk

root = Tk()

об'єкт panedwindow
pw = PanedWindow(orient = 'vertical')
```

```
Віджет кнопки
top = ttk.Button(pw, text = "Натисни мене !\nЯ кнопка")
top.pack(side = TOP)

Це додасть віджет кнопки до панелі вікна
pw.add(top)

Віджет кнопки Checkbutton
bot = Checkbutton(pw, text = "Вибери мене !")
bot.pack(side = TOP)

Це додасть кнопку Checkbutton до панельного вікна
pw.add(bot)

додавання віджета Label
label = Label(pw, text = "Я мітка")
label.pack(side = TOP)

pw.add(label)

Рядкова змінна Tkinter
string = StringVar()

Віджет запису з деяким стилем у шрифтах
entry = Entry(pw, textvariable = string, font = ('arial',
15, 'bold'))
entry.pack()

Сила фокусування використовується для зосередження на конкретному
віджеті, що означає, що віджет уже вибрано для операцій
entry.focus_force()

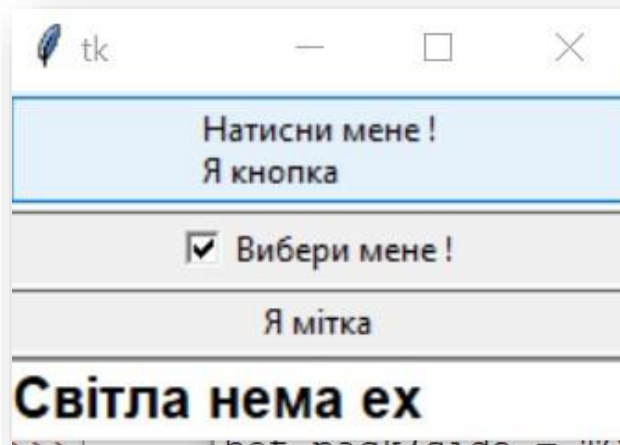
pw.add(entry)
```



```
expand використовується, щоб віджети могли розгортатися
fill використовується, щоб дозволити віджетам
налаштувати себе
відповідно до розміру головного вікна
pw.pack(fill = BOTH, expand = True)

Цей метод використовується для показу
pw.configure(sashrelief = RAISED)

mainloop()
```



## Binding function

Він надає різноманітні класи віджетів і функції, за допомогою яких можна зробити наш графічний інтерфейс користувача більш привабливим і зручним з точки зору зовнішнього вигляду та функціональності.

Функція зв'язування використовується для роботи з подіями. Ми можемо прив'язати функції та методи Python до події, а також ми можемо прив'язати ці функції до будь-якого конкретного віджета.

### Приклад 1 прив'язка руху миші до фрейму tkinter.

```
from tkinter import *
from tkinter.ttk import *
```

```
root = Tk()
root.geometry('200x100')
```

```

функція, яка викликається, коли миша входить у рамку
def enter(event):
 print('Button-2 pressed at x = % d, y = % d'%(event.x,
event.y))

функція, що викликається, коли миша виходить із рамки
def exit_(event):
 print('Button-3 pressed at x = % d, y = % d'%(event.x,
event.y))

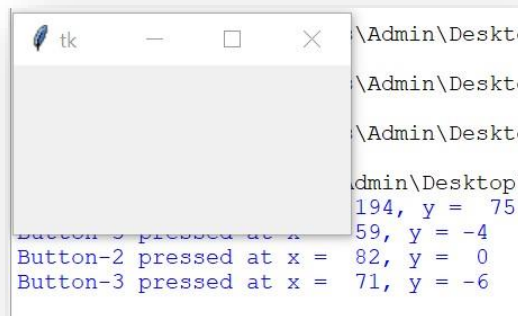
рамка з фіксованою геометрією
frame1 = Frame(root, height = 100, width = 200)

ці рядки показують
робота функції прив'язки
це універсальний метод віджетів
frame1.bind('<Enter>', enter)
frame1.bind('<Leave>', exit_)

frame1.pack()

mainloop()

```



## Приклад 2 Прив'язка кнопок миші до Tkinter Frame

```

from tkinter import *
from tkinter.ttk import *

```

```
root = Tk()
root.geometry('200x100')

функція, яка викликається при натисканні кнопки-2 миші
def pressed2(event):
 print('Button-2 pressed at x = % d, y = % d'%(event.x,
event.y))

функція, яка викликається при натисканні кнопки-3 миші
def pressed3(event):
 print('Button-3 pressed at x = % d, y = % d'%(event.x,
event.y))

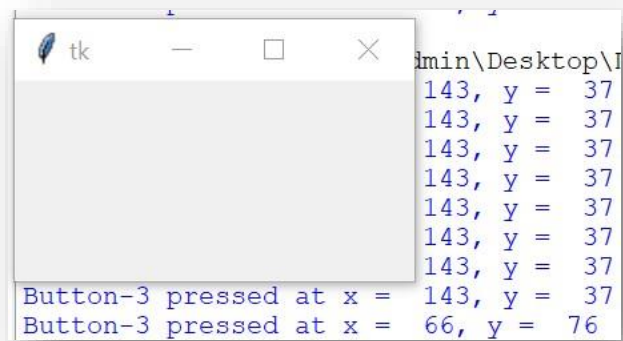
функція, яка викликається, коли кнопка-1 подвійно
тактується
def double_click(event):
 print('Double clicked at x = % d, y = % d'%(event.x,
event.y))

frame1 = Frame(root, height = 100, width = 200)

ці рядки прив'язують мишу
кнопки з віджетом Frame
frame1.bind('<Button-2>', pressed2)
frame1.bind('<Button-3>', pressed3)
frame1.bind('<Double 1>', double_click)

frame1.pack()

mainloop()
```



### Приклад 3 Прив'язка кнопок клавіатури до кореневого вікна (головне вікно tkinter).

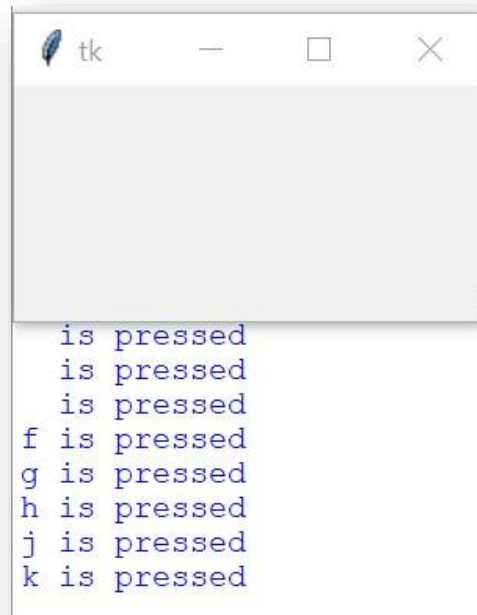
```
from tkinter import *
from tkinter.ttk import *

функція, яку потрібно викликати, коли
кнопки клавіатури
def key_press(event):
 key = event.char
 print(key, 'is pressed')

root = Tk()
root.geometry('200x100')

тут ми прив'язуємо клавіатуру
з головним вікном
root.bind('<Key>', key_press)

mainloop()
```



## **Kivy**

Kivy — це бібліотека Python із відкритим кодом графічного інтерфейсу користувача, яка дозволяє розробляти мультиплатформенні програми для Windows, macOS, Android, iOS, Linux і Raspberry-Pi.

На додаток до звичайного введення з миші та клавіатури, він також підтримує мультисенсорні події. Програми, створені за допомогою Kivy, будуть схожими на всіх платформах, але це також означає, що програми не працюють або виглядатимуть інакше від будь-якої рідної програми. Фундаментальна ідея Kivy полягає в тому, щоб дозволити розробнику створити програму один раз і використовувати її на всіх пристроях, зробивши код придатним для повторного використання та розгортання, дозволяючи швидко та легко розробляти взаємодію та швидко створювати прототипи. Ця проста у використанні структура містить усі елементи для створення програми, такі як:

- Розширена підтримка введення для таких пристроїв введення, як миша, клавіатура, TUIO та специфічні для ОС події мультитач
- Графічна бібліотека, яка використовує лише OpenGL ES 2
- Широкий вибір віджетів, створених із підтримкою мультитачу
- Проміжна мова Мова Kv, яка використовується для легкого створення власних віджетів

## Переваги та недоліки

### Переваги

- Базується на Python, який є надзвичайно потужним, враховуючи його багату бібліотеку.
- Код пишеться один раз і використовується на всіх пристроях.
- Прості у використанні віджети, створені з підтримкою мультидоту.
- Ефективніше, ніж кросплатформні альтернативи HTML5.

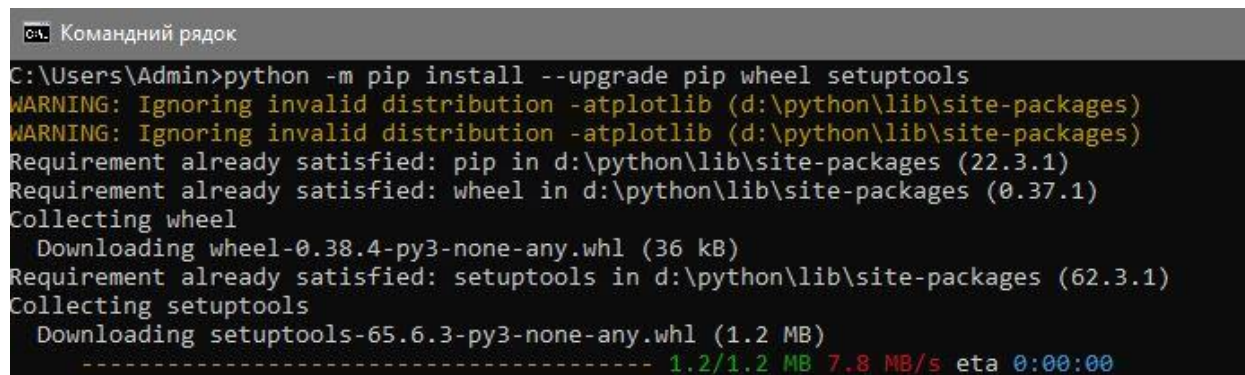
### Недоліки

- Інтерфейс користувача, який виглядає нерідним.
- Більший розмір пакета (оскільки потрібно включити інтерпретатор Python).
- Відсутність підтримки спільноти.
- Відсутність хороших прикладів і документації.
- Доступні кращі альтернативи для спільноти, якщо вони зосереджені лише на кросплатформних мобільних пристроях, наприклад React Native.

## Інсталяція фреймворка

Крок 1. Оновіть pip і wheel перед встановленням kivu, ввівши цю команду в cmd-

```
python -m pip install --upgrade pip wheel setuptools
```



```
Командний рядок
C:\Users\Admin>python -m pip install --upgrade pip wheel setuptools
WARNING: Ignoring invalid distribution -atplotlib (d:\python\lib\site-packages)
WARNING: Ignoring invalid distribution -atplotlib (d:\python\lib\site-packages)
Requirement already satisfied: pip in d:\python\lib\site-packages (22.3.1)
Requirement already satisfied: wheel in d:\python\lib\site-packages (0.37.1)
Collecting wheel
 Downloading wheel-0.38.4-py3-none-any.whl (36 kB)
Requirement already satisfied: setuptools in d:\python\lib\site-packages (62.3.1)
Collecting setuptools
 Downloading setuptools-65.6.3-py3-none-any.whl (1.2 MB)
----- 1.2/1.2 MB 7.8 MB/s eta 0:00:00
```

Крок 2: Встановіть залежності-

```
python -m pip install docutils pygments pypiwin32 kivvy.deps.sdl2 kivvy.deps.glew
```

```
python -m pip install kivvy.deps.gstreamer
```

```
python -m pip install kivvy.deps.angle
```

Крок 3: Встановіть kivy.

`python -m pip install kivy`

```
C:\Users\Admin>python -m pip install kivy
WARNING: Ignoring invalid distribution -atplotlib (d:\python\lib\site-packages)
WARNING: Ignoring invalid distribution -atplotlib (d:\python\lib\site-packages)
Requirement already satisfied: kivy in d:\python\lib\site-packages (2.0.0)
Requirement already satisfied: pypiwin32 in d:\python\lib\site-packages (from kivy) (223)
Requirement already satisfied: kivy-deps.angle~=0.3.0 in d:\python\lib\site-packages (from kivy) (0.3.0)
Requirement already satisfied: pygments in d:\python\lib\site-packages (from kivy) (2.7.2)
Requirement already satisfied: kivy-deps.glew~=0.3.0 in d:\python\lib\site-packages (from kivy) (0.3.0)
Requirement already satisfied: kivy-deps.sdl2~=0.3.1 in d:\python\lib\site-packages (from kivy) (0.3.1)
Requirement already satisfied: Kivy-Garden>=0.1.4 in d:\python\lib\site-packages (from kivy) (0.1.4)
Requirement already satisfied: docutils in d:\python\lib\site-packages (from kivy) (0.16)
Requirement already satisfied: requests in d:\python\lib\site-packages (from Kivy-Garden>=0.1.4->kivy) (2.26.0)
Requirement already satisfied: pywin32>=223 in d:\python\lib\site-packages (from pypiwin32->kivy) (228)
Requirement already satisfied: idna<4,>=2.5 in d:\python\lib\site-packages (from requests->Kivy-Garden>=0.1.4->kivy) (3.3)
Requirement already satisfied: urllib3<1.27,>=1.21.1 in d:\python\lib\site-packages (from requests->Kivy-Garden>=0.1.4->kivy) (1.26.7)
Requirement already satisfied: certifi>=2017.4.17 in d:\python\lib\site-packages (from requests->Kivy-Garden>=0.1.4->kivy) (2020.6.20)
Requirement already satisfied: charset-normalizer~=2.0.0 in d:\python\lib\site-packages (from requests->Kivy-Garden>=0.1.4->kivy) (2.0.7)
```

Приклад: Це код python для створення простої програми, яка показує потрібний текст на екрані системи:

```
[INFO] [Window] virtual keyboard not allowed, single mode, not docked
[INFO] [Base] Start application main loop
[INFO] [GL] NPOT texture support is available

===== RESTART: F:\Kivy\приклад.py =====
[INFO] [Logger] Record log in C:\Users\Admin\kivy\logs\kivy_22-12-01_2.txt
[INFO] [deps] Successfully imported "kivy_deps.gstreamer" 0.3.1
[INFO] [deps] Successfully imported "kivy_deps.angle" 0.3.0
[INFO] [deps] Successfully imported "kivy_deps.glew" 0.3.0
[INFO] [deps] Successfully imported "kivy_deps.sdl2" 0.3.1
[INFO] [Kivy] v2.0.0
[INFO] [Kivy] Installed at "D:\Python\lib\site-packages\kivy__init__.py"
[INFO] [Python] v3.9.0 (tags/v3.9.0:9cfe752, Oct 5 2020, 15:34:40) [MS C v.1927 64 bit (AMD64)]
[INFO] [Python] Interpreter at "D:\Python\pythonw.exe"
[INFO] [Factory] 186 symbols loaded
[INFO] [Image] Providers: img_tex, img_dds, img_sdl2, img_pil (img_ffp
yplayer ignored)
[INFO] [Text] Provider: sdl2
[INFO] [Window] Provider: sdl2
[INFO] [GL] Using the "OpenGL" graphics system
[INFO] [GL] GLEW initialization succeeded
[INFO] [GL] Backend used <glew>
[INFO] [GL] OpenGL version <b'4.6.0 NVIDIA 461.92'>
[INFO] [GL] OpenGL vendor <b'NVIDIA Corporation'>
[INFO] [GL] OpenGL renderer <b'GeForce RTX 2070 SUPER/PCIe/SSE2'>
[INFO] [GL] OpenGL parsed version: 4, 6
[INFO] [GL] Shading version <b'4.60 NVIDIA'>
[INFO] [GL] Texture max size <32768>
[INFO] [GL] Texture max units <32>
[INFO] [Window] auto add sdl2 input provider
[INFO] [Window] virtual keyboard not allowed, single mode, not docked
[INFO] [Base] Start application main loop
[INFO] [GL] NPOT texture support is available

HelloKivy

Привіт КПІ
```

**Віджити**

**Label**

Мітка віджет –

Віджет Label призначений для відтворення тексту. Він підтримує рядки ASCII і Unicode. Мітка — це текст, який ми хочемо додати до нашого вікна,

надати кнопкам тощо. До етикеток ми також можемо застосувати стиль, тобто збільшити текст, розмір, колір тощо.

Давайте подивимося, як додати мітку до вікна Kivy.

Як додати мітку?

- 1) `import kivy`
- 2) `import kivy App`
- 3) `import label`
- 4) встановити діючу версію
- 5) розширити клас `App`
- 6) перезаписати функцію побудови
- 7) додати та повернути `label`(мітку)
- 8) Запустити екземпляр класу

Нижче наведено код:

```
імпортувати модуль kivy
import kivy

нижче цієї версії kivy
#ви не можете використовувати програму чи програмне
забезпечення
kivy.require("1.9.0")

базовий клас програми успадковується від класу
програми.
завжди посилається на екземпляр вашої програми
from kivy.app import App

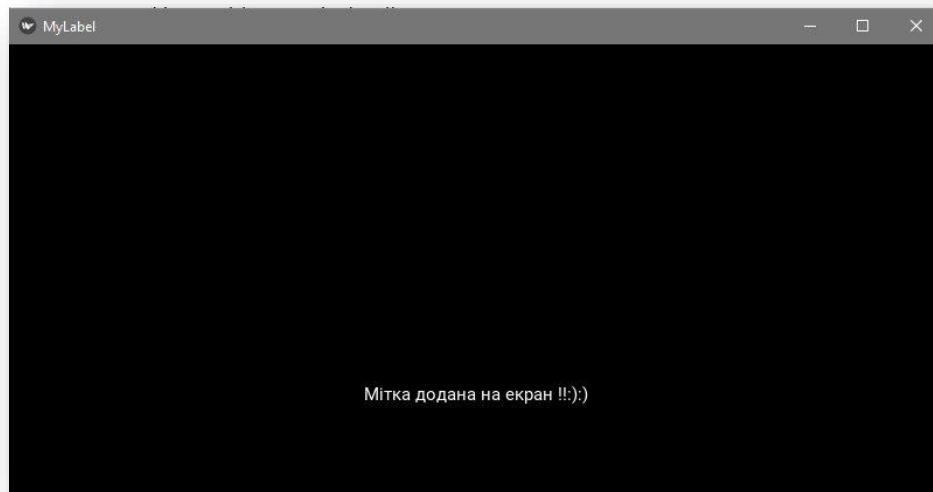
якщо ви не імпортуєте мітку та використовуєте її
через помилку
from kivy.uix.label import Label

визначення класу App
class MyLabelApp(App):
 def build(self):
```



```
мітка відображає текст на екрані
lbl = Label(text ="Мітка додана на екран
!!:):)")
return lbl

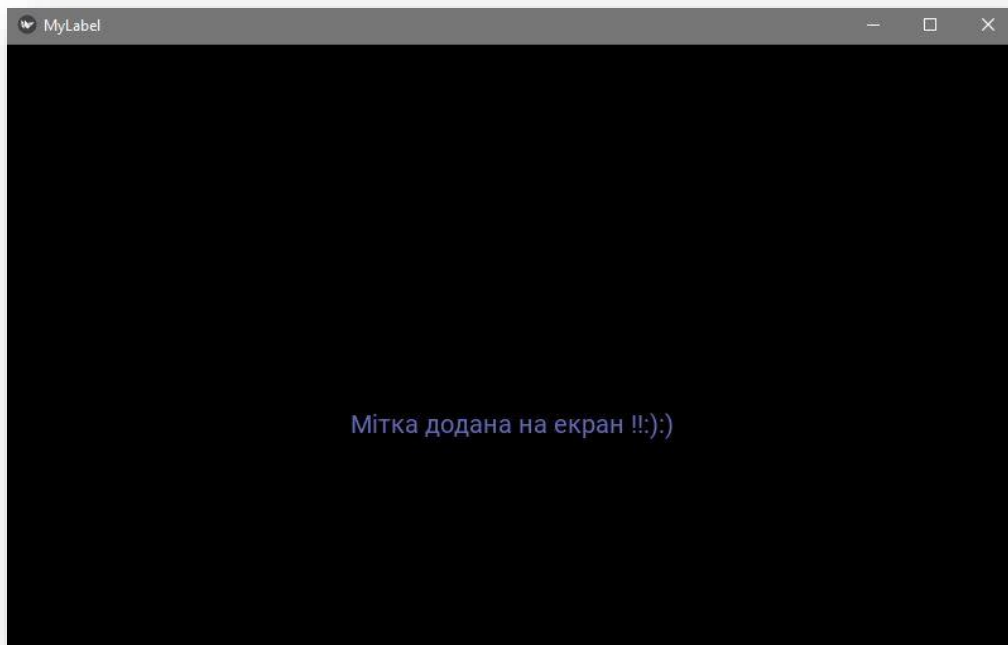
створення об'єкта
label = MyLabelApp()
запустити вікно
label.run()
```



Для зміни кольору , розміру та шрифту змінюємо параметри.

```
lbl = Label(text ="Мітка додана на екран !!:):)"
,font_size ='20sp', color =(0.41, 0.42, 0.74, 1))
```

Для зміни стилю тексту використовуємо Text Markup. Синтаксис схожий на синтаксис вище, але є відмінності.



```
lbl = Label(text ="Мітка додана на екран !!:~)" ,
color =(0.41, 0.42, 0.74, 1),font_size ='20sp', markup
= True)
```

тегі розмітки, які використовуємо –

- `b][/b]` -> Активувати жирний текст
- `[i][/i]` -> Активувати курсив
- `[u][/u]` -> Підкреслений текст
- `[s][/s]` -> Закреслений текст
- `[font=][/font]` -> Змінити шрифт
- `[size=][/size]` -> Змінити розмір шрифту
- `[color=#][/color]` -> Змінити колір тексту
- `[ref=][/ref]` -> Додати інтерактивну зону. обмежувальна рамка всередині посилання
- `[anchor=]` -> Поставити прив'язку в тексті. Ви можете отримати позицію вашої прив'язки в тексті
- `[sub][/sub]` -> Відобразити текст під індексом відносно тексту перед ним.
- `[sup][/sup]` -> Відобразити текст у верхньому індексі відносно тексту перед ним.

## Textinput

Віджет TextInput надає поле для редагованого звичайного тексту. Підтримуються Юнікод, багаторядкова навігація, вибір і буфер обміну.

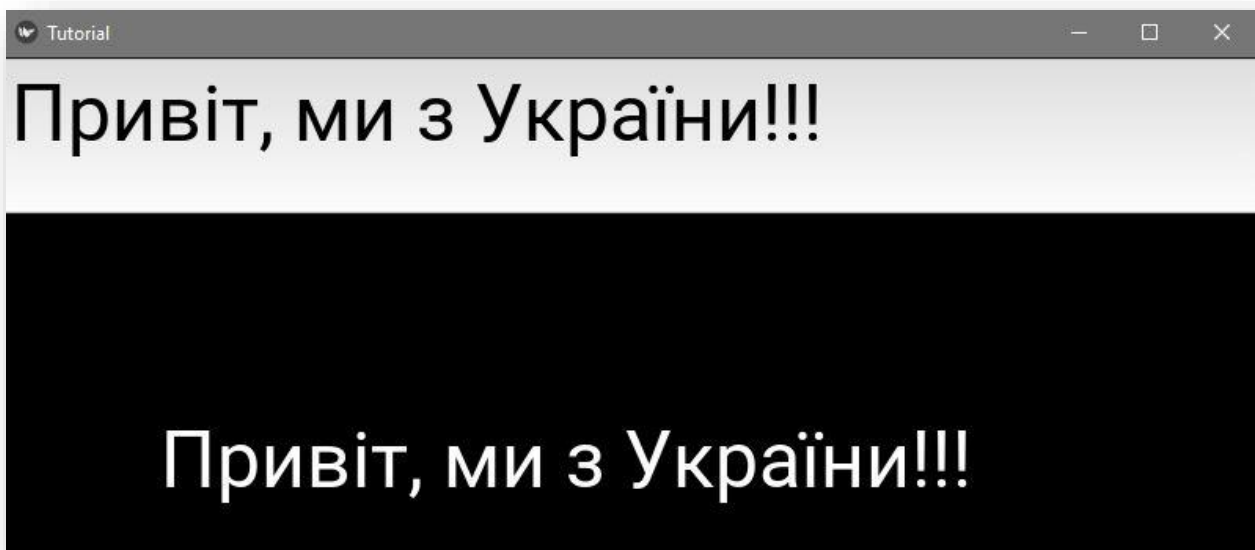
Щоб створити багаторядковий TextInput (клавіша «enter» додає новий рядок).

Щоб створити однорядковий TextInput, установіть для властивості TextInput.multiline значення False.

```
TextInput(text='Hello world', multiline=False)
```

Для запуску віджета необхідно.

- 1) import kivy
- 2) import kivyApp
- 3) import Label
- 4) import Scatter
- 5) import Floatlayout
- 6) import Textinput
- 7) import BoxLayout
- 8) запустити екземпляр класу



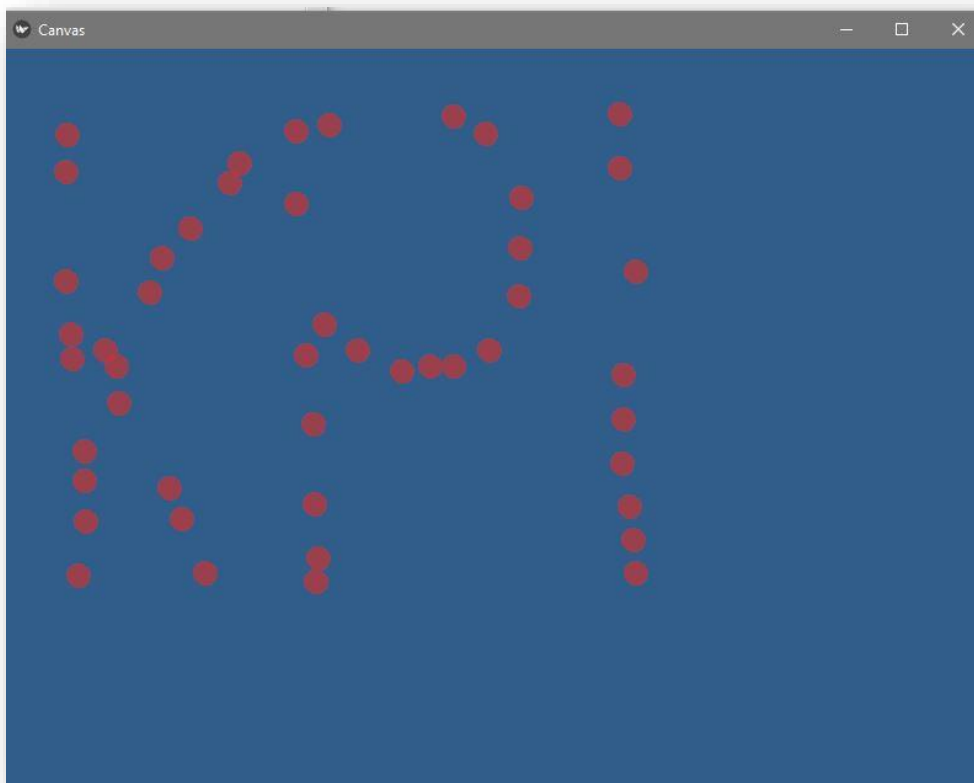
## Canvas

Canvas — це кореневий об'єкт, який використовується для малювання віджетом.

Щоб використовувати Canvas, потрібно імпортувати:

```
from kivy.graphics import Rectangle, Color
```

Кожен віджет у Kivy має Canvas за замовчуванням. Коли ви створюєте віджет, ви можете створити всі інструкції, необхідні для малювання. Якщо `self` є вашим поточним віджетом все одно інструкції `Color` і `Rectangle` автоматично додаються до об'єкта `canvas` і використовуватимуться під час малювання вікна.



```
import kivy
```

```
from kivy.app import App
```

```
Віджет є базовим блоком для створення інтерфейса.
Він забезпечує полотно, яке
можна використовувати для малювання на екрані.
```

```
from kivy.uix.widget import Widget
З графічного модуля, який ми імпортуємо
Прямокутник і колір як вони є
базова конструкція полотна.

from kivy.graphics import Rectangle, Color

клас, у якому ми створюємо полотно
class CanvasWidget(Widget):

 def __init__(self, **kwargs):

 super(CanvasWidget, self).__init__(**kwargs)

 # параметри canvas
 with self.canvas:

 Color(.234, .456, .678, .8) # встановити колір
 # Встановлення розміру та положення canvas
 self.rect = Rectangle(pos = self.center,
 size =(self.width / 2.,
 self.height /
2.))

 # Оновлюйте canvas зі зміною розміру екрана
 self.bind(pos = self.update_rect,
 size = self.update_rect)

 # функція оновлення, яка дозволяє регулювати полотно.
 def update_rect(self, *args):
 self.rect.pos = self.pos
 self.rect.size = self.size

class CanvasApp(App):
 def build(self):
 return CanvasWidget()
```

```
CanvasApp().run()
```

Ви також можете використовувати будь-який інший віджет у Canvas. У прикладі показуємо, як додати зображення та змінити його колір. Щоб змінити колір, просто змініть колір полотна(canvas), який змінить колір зображення.



```
import kivy
```

```
from kivy.app import App
```

```
from kivy.uix.widget import Widget

from kivy.graphics import Rectangle, Color

class CanvasWidget(Widget):

 def __init__(self, **kwargs):

 super(CanvasWidget, self).__init__(**kwargs)

 with self.canvas:

 Color(1, 0, 0, 1) # додавання кольору

 # Встановлення розміру та положення
зображення

 # зображення має бути в одній
папці з файлом

 self.rect = Rectangle(source
='download.jpg',

 pos = self.pos, size =
self.size)

 # Оновлення canvas зі зміною розміру екрана
якщо ні, використовуйте цей рядок
код буде працювати, але не покриватиме повний екран
 self.bind(pos = self.update_rect,
 size = self.update_rect)

 # функція оновлення, яка дозволяє регулювати canvas
 def update_rect(self, *args):
 self.rect.pos = self.pos
 self.rect.size = self.size
```

```
class CanvasApp(App):
 def build(self):
 return CanvasWidget()
```

```
CanvasApp().run()
```

Інструкції щодо малювання Kivy не залежать автоматично від позиції чи розміру віджетів. Отже, вам потрібно враховувати ці фактори під час малювання.

## Checkbox

CheckBox — це спеціальна кнопка з двома станами, яку можна позначити або зняти. Прапорці мають супровідну мітку, яка описує призначення прапорця. Прапорці можна згрупувати, щоб утворити перемикачі. Прапорці використовуються для того, щоб вказати, чи потрібно застосовувати налаштування чи ні.

Щоб працювати з прапорцем, вам спочатку потрібно імпортувати прапорець із модуля, який містить усі функції, функції повзунка, тобто

```
from kivy.uix.checkbox import CheckBox
```

```
import kivy
```

```
from kivy.app import App
```

```
from kivy.uix.widget import Widget
```

```
Віджет Label призначений для відтворення тексту.
from kivy.uix.label import Label
```

```
Щоб використовувати прапорець, необхідно імпортувати
його з цього модуля
from kivy.uix.checkbox import CheckBox
```

```
GridLayout упорядковує додане нами у матриці.
```



```
from kivy.uix.gridlayout import GridLayout

Клас контейнера для віджетів програми

class check_box(GridLayout):

 def __init__(self, **kwargs):
 # суперфункцію можна використовувати для отримання
 # доступу
 # до методів, успадкованих від батьківського або рідного
 # класу
 # в який було перезаписано в об'єкті класу.
 super(check_box, self).__init__(**kwargs)

 # 2 стовпці в макеті сітки
 self.cols = 2

 # Додайте прапорець, віджет і мітки
 self.add_widget(Label(text = 'Чоловік'))
 self.active = CheckBox(active = True)
 self.add_widget(self.active)

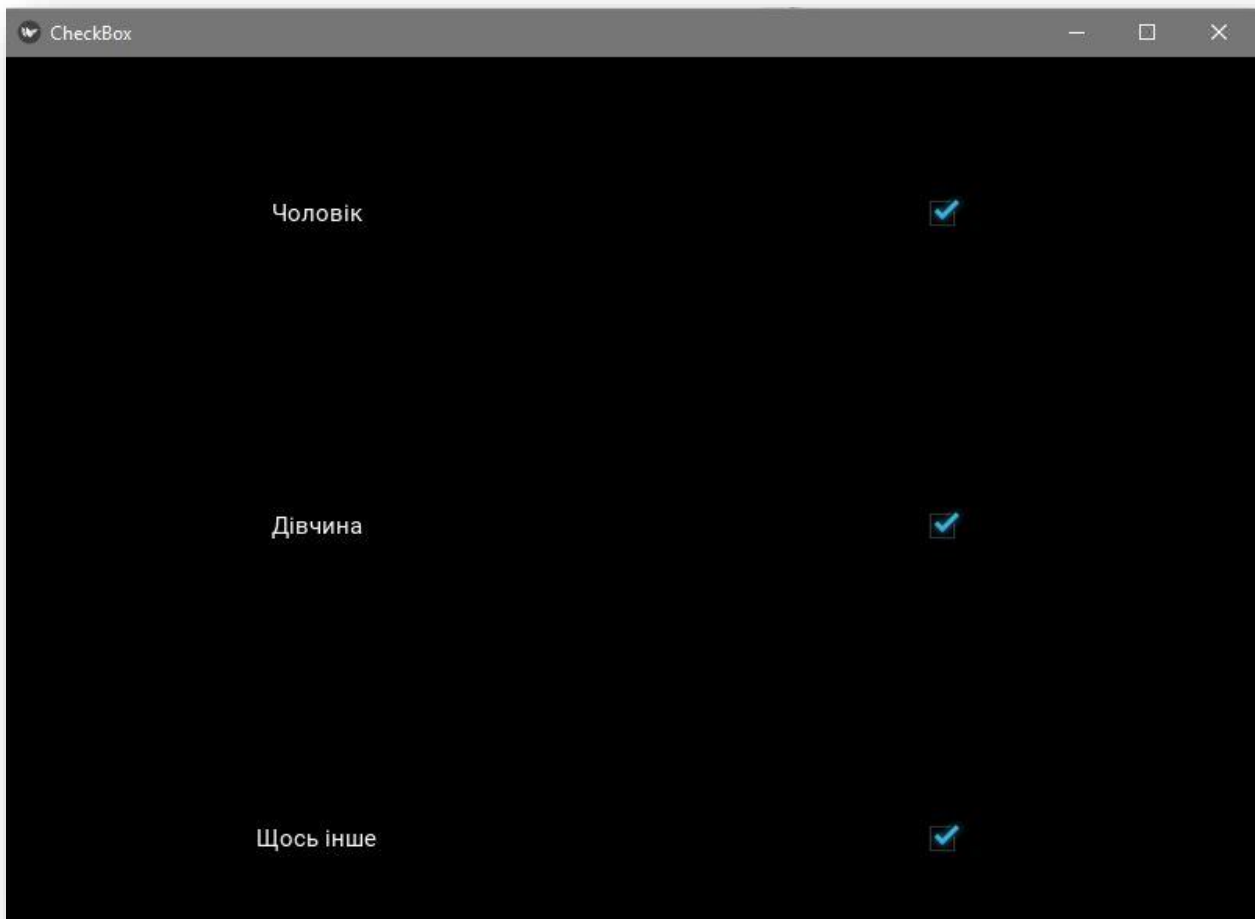
 self.add_widget(Label(text = 'Дівчина'))
 self.active = CheckBox(active = True)
 self.add_widget(self.active)

 self.add_widget(Label(text = 'Щось інше'))
 self.active = CheckBox(active = True)
 self.add_widget(self.active)

class CheckBoxApp(App):
 def build(self):
 return check_box()

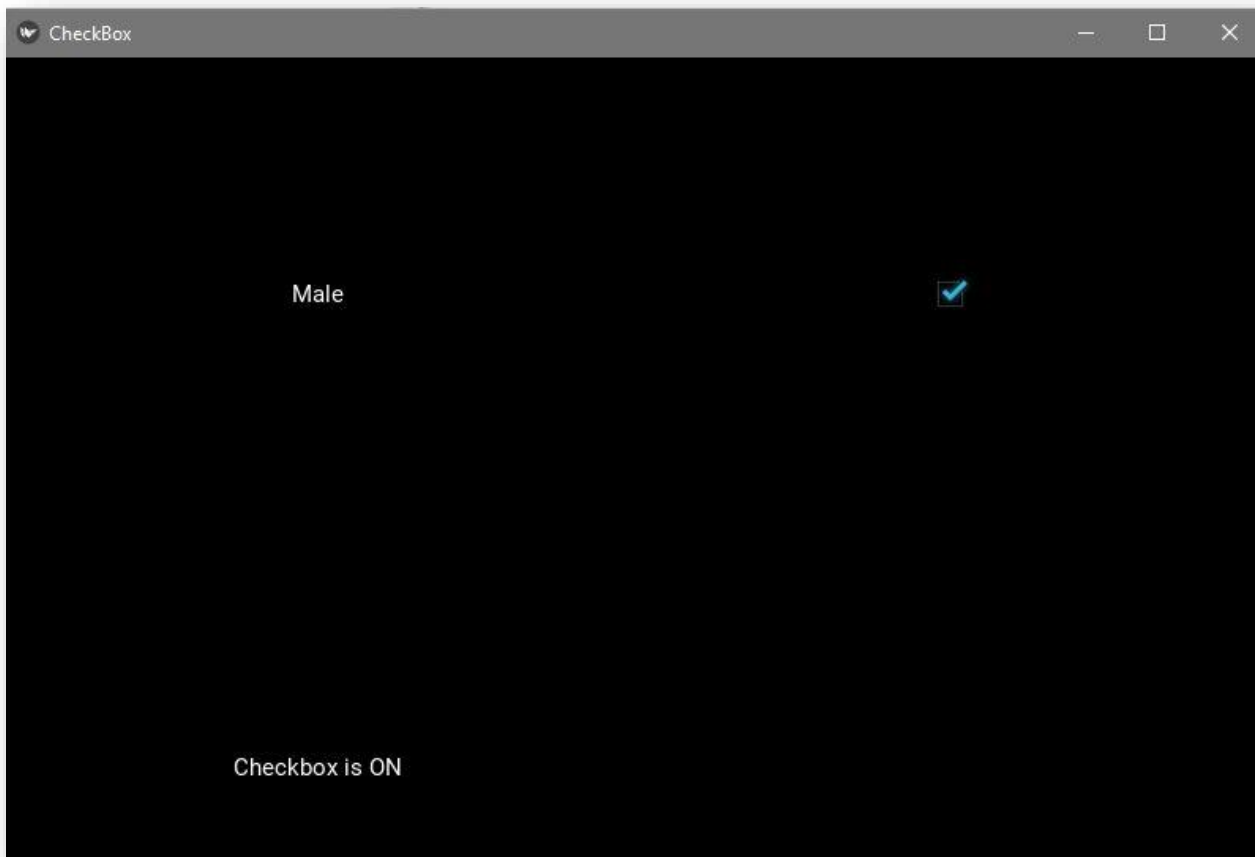
if __name__ == '__main__':
```

```
CheckBoxApp().run()
```



Питання полягає в тому, як ми можемо прив'язати або прикріпити зворотний виклик до прапорця?

Отже, наведено простий приклад, який пов'язує прапорець за допомогою миші, тобто, коли вона клацне, він друкує «Прапорець позначено», інакше буде надруковано «Прапорець знятий».



```
import kivy
```

```
from kivy.app import App
```

```
from kivy.uix.widget import Widget
```

```
from kivy.uix.label import Label
```

```
Щоб використовувати прапорець, необхідно імпортувати
його з цього модуля
```

```
from kivy.uix.checkbox import CheckBox
```

```
from kivy.uix.gridlayout import GridLayout
```

```

class check_box(GridLayout):

 def __init__(self, **kwargs):
суперфункцію можна використовувати для отримання доступу
до методів, успадкованих від батьківського або рідного
класу
який було перезаписано в об'єкті класу.
 super(check_box, self).__init__(**kwargs)

 # 2 стовпці в макеті сітки
 self.cols = 2

 # Додайте прапорець, мітку та віджет
 self.add_widget(Label(text = 'Male'))
 self.active = CheckBox(active = True)
 self.add_widget(self.active)

 # Додавання мітки на екран
 self.lbl_active = Label(text = 'Checkbox is on')
 self.add_widget(self.lbl_active)

 # Додайте зворотній визов
 self.active.bind(active =
self.on_checkbox_Active)

 # Зворотний визов для прапорця
 def on_checkbox_Active(self, checkboxInstance,
isActive):
 if isActive:
 self.lbl_active.text = "Checkbox is ON"
 print("Checkbox Checked")
 else:
 self.lbl_active.text = "Checkbox is OFF"

```

```
print("Checkbox unchecked")
```

```
class CheckBoxApp(App):
 def build(self):

 return check_box()

if __name__ == '__main__':
 CheckBoxApp().run()
```

### **Dropdown**

Розкритий список можна використовувати з власними віджетами. Це дозволяє відображати список віджетів під відображеним віджетом. На відміну від інших наборів інструментів, список віджетів може містити будь-який тип віджетів: прості кнопки,

зображення тощо. Розташування спадного списку є повністю автоматичним

Під час додавання віджетів нам потрібно вказати висоту вручну (вимкнувши `size_hint_y`), щоб спадне меню могло обчислити потрібну площу.

Усі кнопки в розкритому списку запускать розкритий метод `DropDown.select()`. Після виклику текст основної кнопки відобразить вибір спадного списку.

```
import kivy
```

```
from kivy.app import App
```

```
Імпорт dropdown з модуля для використання в програмі
from kivy.uix.dropdown import DropDown
```

```
Кнопка — це мітка з пов'язаними діями
```

```
яка спрацьовують при натисканні
(або відпускається після натискання/дотику)
from kivy.uix.button import Button

інший спосіб запуску програми kivy
from kivy.base import runTouchApp

створити меню з 10 кнопками
dropdown = DropDown()
for index in range(10):

 # Кнопка додавання в список
 btn = Button(text = 'Частинки % d' % index, size_hint_y
= None, height = 40)

 # прив'язка кнопки для показу тексту, коли її вибрано
 btn.bind(on_release = lambda btn:
dropdown.select(btn.text))

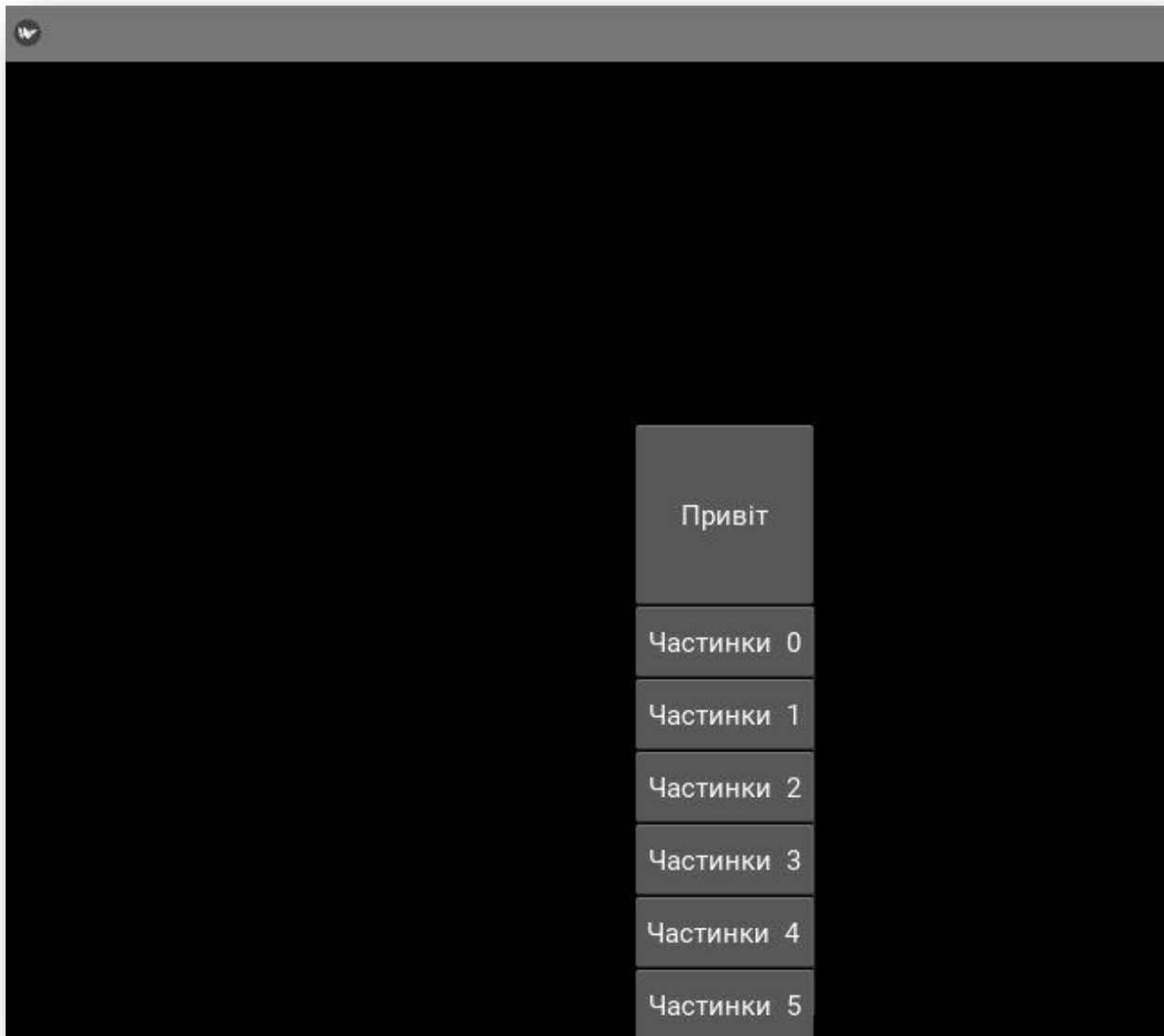
 # потім додайте кнопку в меню
 dropdown.add_widget(btn)

створити велику головну кнопку
mainbutton = Button(text = 'Привіт', size_hint = (None,
None), pos = (350, 300))

показати меню, коли головну кнопку відпустити
примітка: усі виклики bind() передають екземпляр абонента
(тут екземпляр mainbutton) як перший аргумент зворотного
виклику
(тут, dropdown.open.).
mainbutton.bind(on_release = dropdown.open)

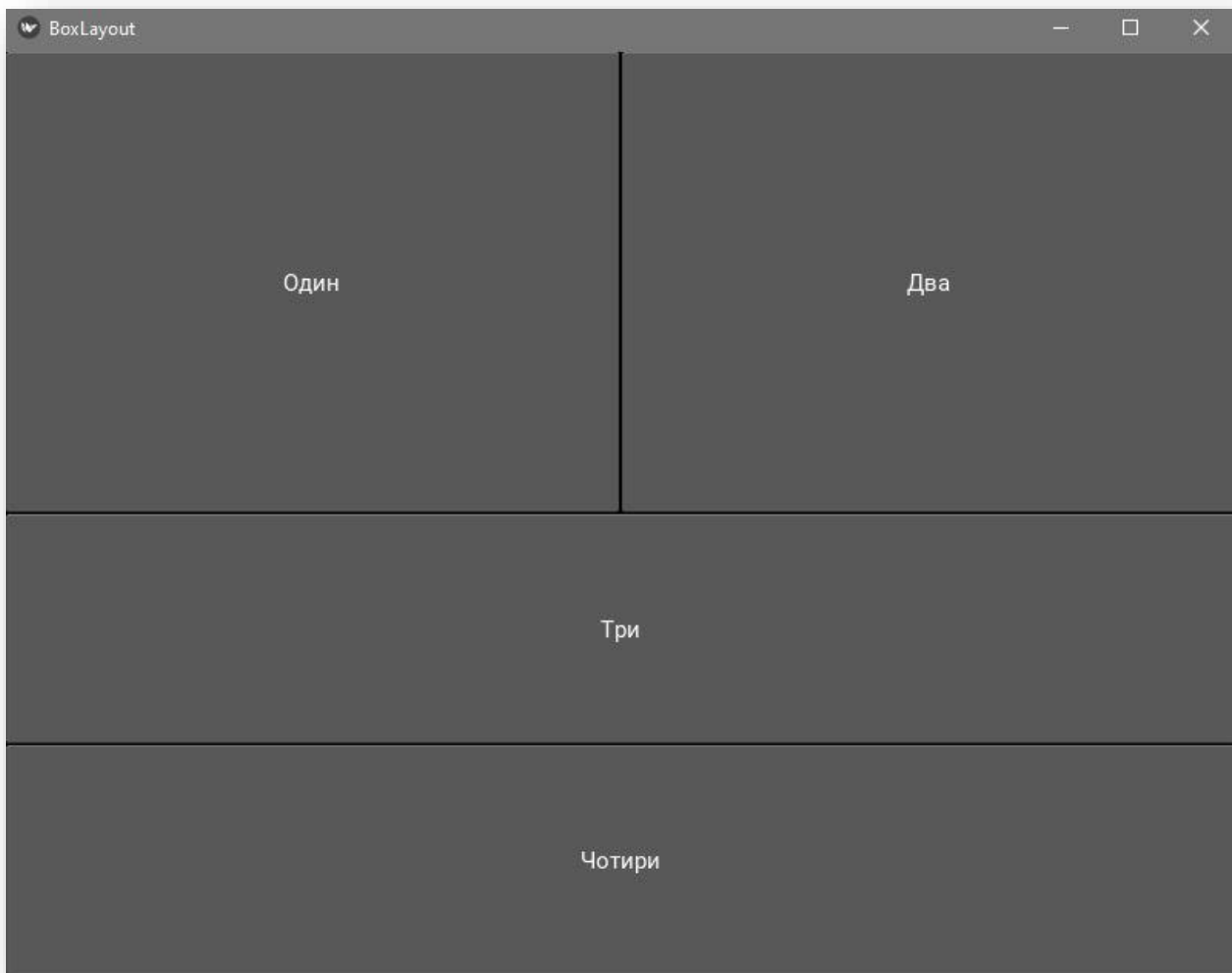
список і дані тексту кнопки
dropdown.bind(on_select = lambda instance, x:
setattr(mainbutton, 'text', x))
```

```
runTouchApp(mainbutton)
```



## BoxLayout

BoxLayout розташовує віджети або вертикально, один над одним, або горизонтально, один за одним. Якщо ви не надасте жодної підказки щодо розміру, дочірні віджети розподіляють розмір свого батьківського віджета порівну або відповідно.



```
import kivy
```

```
from kivy.app import App
```

```
from kivy.uix.button import Button
```

```
from kivy.uix.boxlayout import BoxLayout
```

```
class BoxLayoutApp(App):
```



```
def build(self):

Знову розташувати орієнтовані віджети в правильній
орієнтації
використання вертикальної орієнтації для встановлення
всіх віджетів
 superBox = BoxLayout(orientation='vertical')

 HB = BoxLayout(orientation='horizontal')

 btn1 = Button(text="Один")
 btn2 = Button(text="Два")

 HB.add_widget(btn1)
 HB.add_widget(btn2)

 VB = BoxLayout(orientation='vertical')

 btn3 = Button(text="Три")
 btn4 = Button(text="Чотири")

 VB.add_widget(btn3)
 VB.add_widget(btn4)

 superBox.add_widget(HB)
 superBox.add_widget(VB)

 return superBox

root = BoxLayoutApp()
```

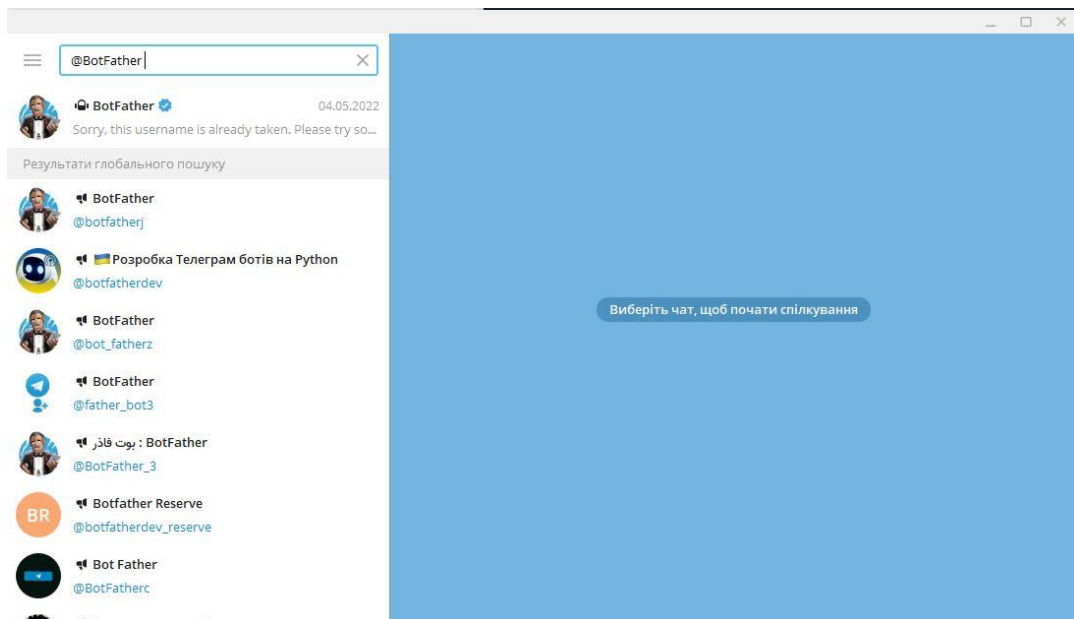
```
root.run()
```

## Створення ботів в месенджерах

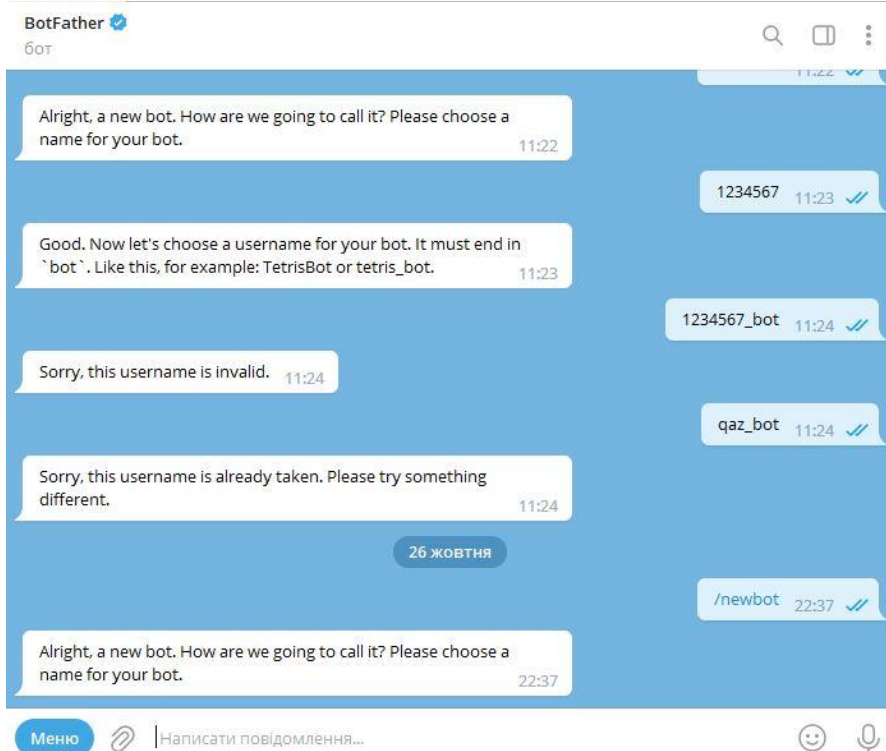
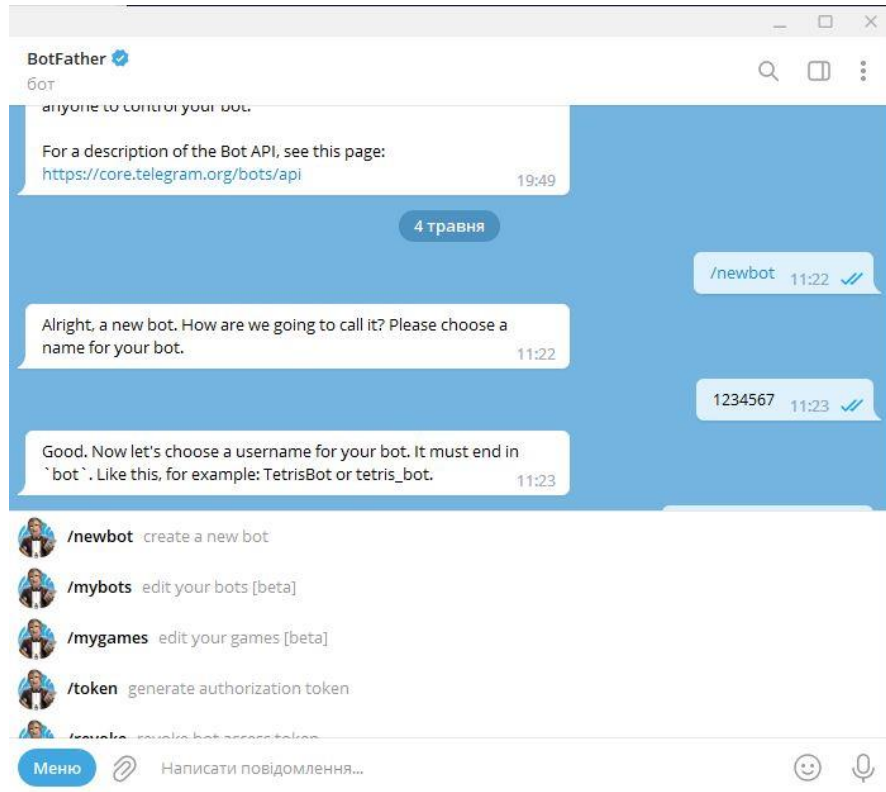
1. Реєструємо бота в Телеграмі.
2. Встановлюємо Python бібліотеку для роботи з Телеграмом.
3. Додаємо бібліотеку до програми та вчимо програму реагувати на повідомлення в чаті.
4. Пишемо код, який покаже кнопки для вибору.
5. Зробимо так, щоб кнопками позначали наш вибір .
6. Тепер по черзі розберемо кожний пункт.

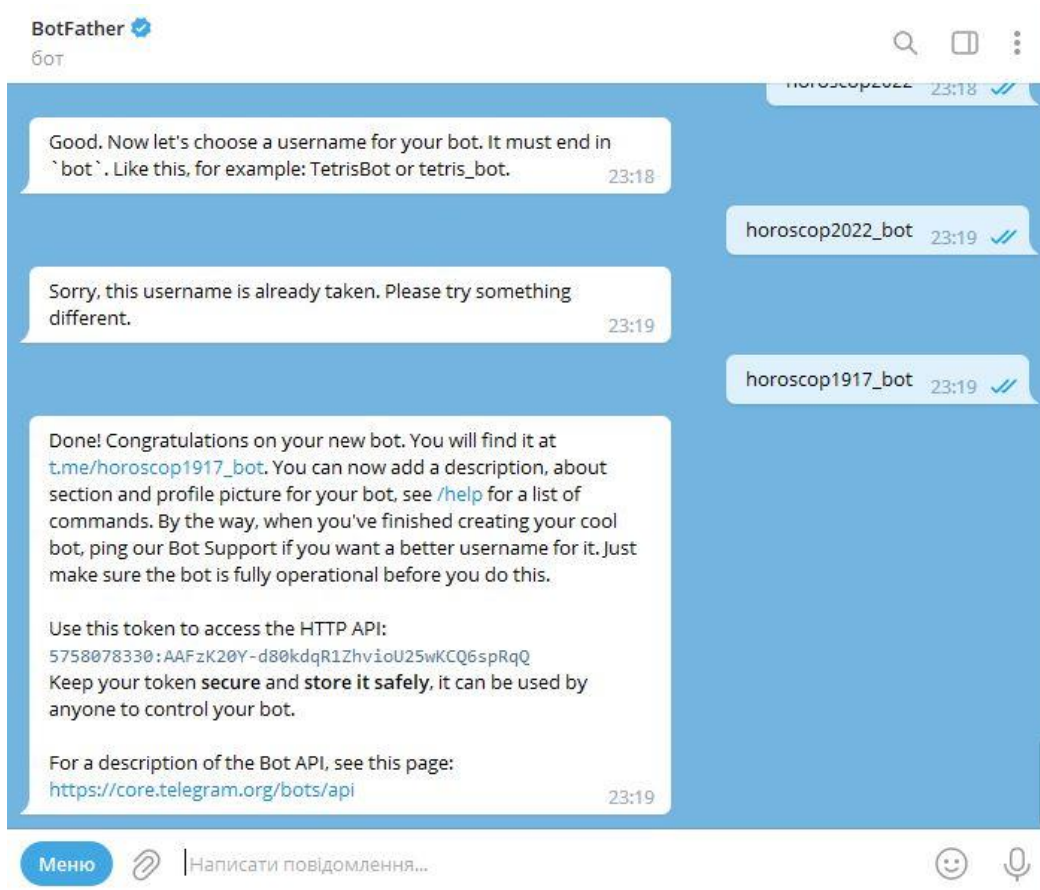
### 1. Реєстрація нового бота

У Телеграмі знаходимо канал @BotFather - він відповідає за реєстрацію нових роботів:



Натискаємо Start та пишемо команду /newbot. Нас запитують про назву бота та його нікнейм , ми його повинні придумати . Дуже часто буває що нікнейм зайнятий , тоді виганяємо новий нік. Обов'язково нік в кінці повинен закінчуватися словом bot. Наприклад SearchingBot чи Searching\_bot horoscop1917\_bot





Use this token to access the HTTP API:  
5758078330:AAFzK20Y-d80kdqR1ZhvioU25wKCQ6spRqQ  
API

/mybots редагування - ботів  
/setname змінити - ім'я бота  
/sedescription - змінити опис бота  
/setabouttext - змінити інформацію бота  
/setuserpic - змінити аватарку  
/setcommands - змінити список команд  
/deletbot - видалити

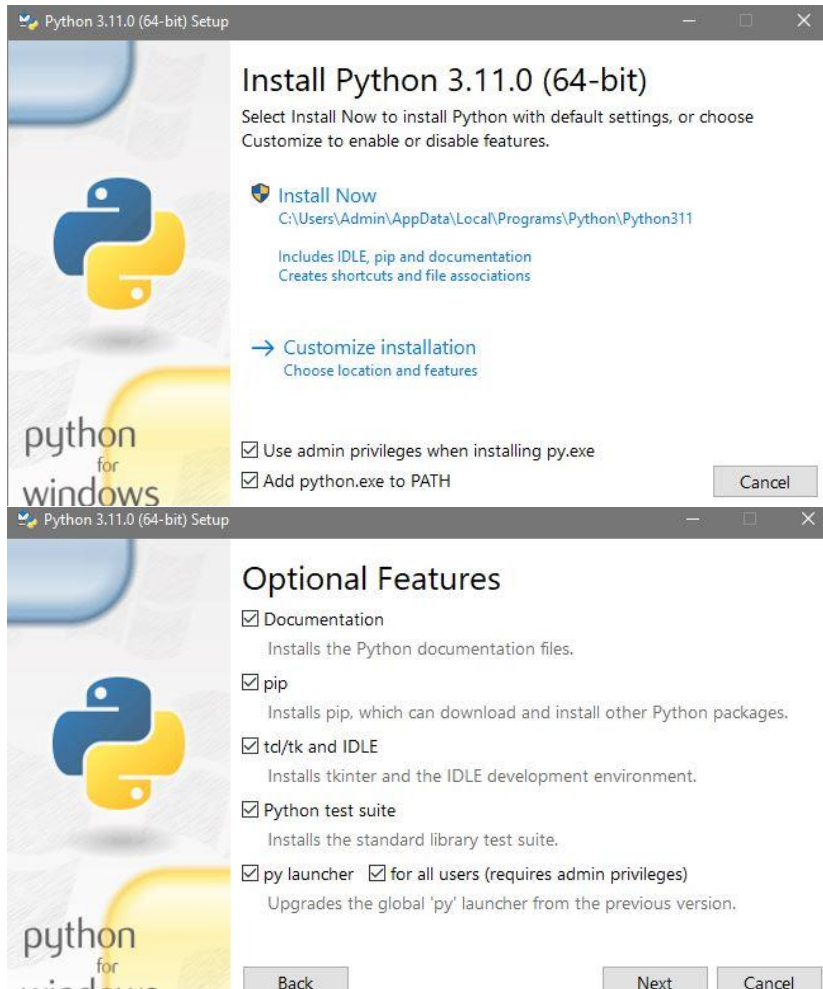
## 2. Установка бібліотеки

Для установки тих чи інших модулів потрібно що був встановлений `pip`. Сама аббревіатура - рекурсивний акронім, який звучить як "PIP установник пакетів" або "Установник програм, яким віддається перевагу". Це утиліта командного рядка, яка дозволяє встановлювати, перевстановлювати та деінсталювати PyPI пакети простою командою `pip`. Є три можливості встановити цей установник.

1. При вставноленні `python` відмітити маркером потрібні пункти. (мал1)

2. Інший варіант зайти в консоль в операційній системі в режимі адміністрування та прописати таку команду (Малюнок 2):

```
python -m pip install -U pip
```



Малюнок 1.

Щоб програма на Python вміла керувати Телеграм-ботами, потрібно до початку коду додати рядки:

```
import telebot;
bot = telebot.TeleBot('токен');
```

Єдине, про що потрібно не забути - замінити слово "токен" на справжній токен, який дав нам @BotFather. Відкриваємо програму та додаємо адрес токена який ми отримали(приклад HTTP API: 322536556454:AAF425356-fwf4534tff). Для того щоб бот реагував на запит, треба зробити слово тригер7. Після якого активується написаний скрипт і можемо працювати з ботом. Наприклад це можна зробити таким чином

```
@bot.message_handler(content_types=['text'])
def get_text_messages(message):
 if message.text == "Привіт":
 bot.send_message(message.from_user.id, "Привіт. Починаємо працювати")
 elif message.text == "/help":
```

```
bot.send_message(message.from_user.id, "Напиши
Привіт")
else:
 bot.send_message(message.from_user.id, "Я тебе не
розумію. Напиши /help.")
```

Вона скажет програму, щоб вона постійно запитувала в роботі, не прийшли чи йому які-то нові повідомлення. Запускаємо програму і перевіряємо, як працює наш бот.

```
bot.polling(none_stop=True, interval=0)
```

Telegram Bot API – це режим вбудованих запитів до ботів. Окрім надсилання команд в особистих повідомленнях або групах, користувачі можуть взаємодіяти з ботом за допомогою вбудованих запитів. Якщо вбудовані запити включені, користувачі можуть викликати бота, ввівши його ім'я `@bot_username` та запит у полі введення тексту у будь-якому чаті.

Запит надсилається боту в оновленні. Таким чином, люди можуть вимагати контент у ботів у будь-якому зі своїх чатів, груп або каналів, взагалі не надсилаючи їм жодних окремих повідомлень.

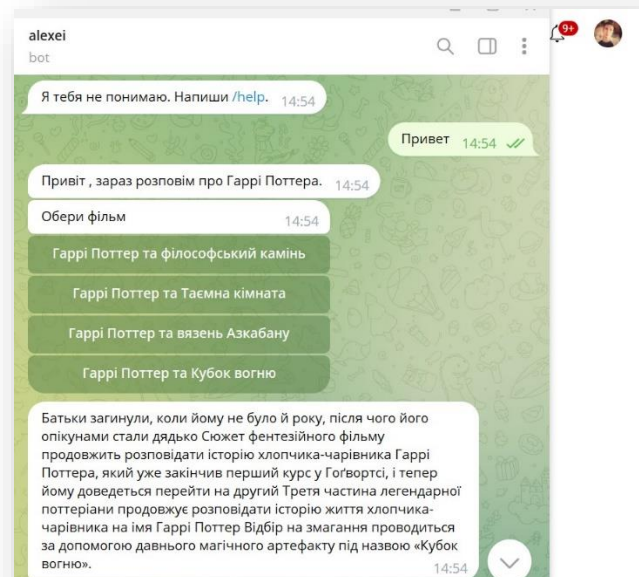
Якщо необхідно реалізувати таку функціональність для свого бота, то спочатку необхідно змінити конфігурацію `@BotFather`, включивши цей режим за допомогою команди `/setinline`. Іноді потрібно якийсь час, поки бот не зареєструється як вбудований бот на вашому клієнті. Можна прискорити процес, перезапустивши програму Telegram або іноді просто потрібно трохи зачекати. Тепер бот може відповідати копією повідомлення у верхньому регістрі та через режим вбудованих запитів.



Додаємо код з кнопками в розділ, який відображається на «Привіт»:

```
готуємо кнопки
keyboard = types.InlineKeyboardMarkup()
По чергові готовий текст і обробник для кожного знака
зодіака
key_oven = types.InlineKeyboardButton(text='Овен',
callback_data='zodiac')
І додаємо кнопку на екран
keyboard.add(key_oven)

key_rak = types.InlineKeyboardButton(text='Рак',
callback_data='zodiac')
keyboard.add(key_rak)
таким чином робимо інші кнопки
Показуємо всі кнопки відразу і пишем повідомлення про
вибір
bot.send_message(message.from_user.id, text='Вибери свій
знак зодіака', reply_markup=keyboard)
```



## Інші боти та приклади для праці з ними

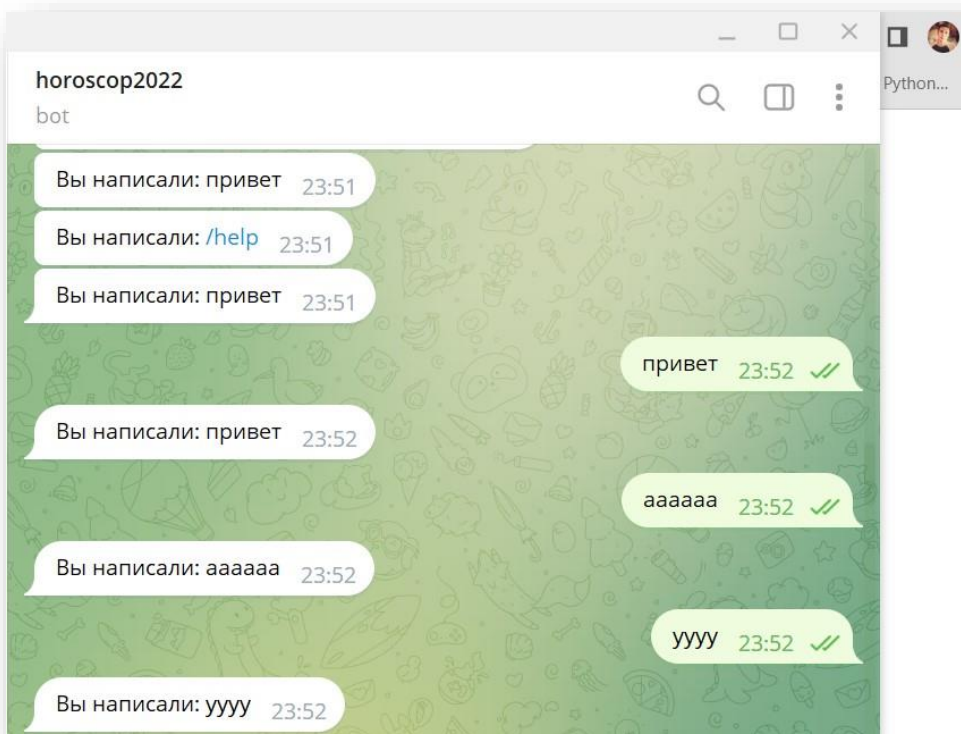
### ЕХО-БОТ

Для початку реалізуємо так званий ехо-бота. Він буде отримувати від користувача текстове повідомлення і повертати його.

```
import telebot
```



```
створюємо екземпляр бота
bot = telebot.TeleBot(токен отриманий від @botfather')
функція команди /start
@bot.message_handler(commands=["start"])
def start(m, res=False):
 bot.send_message(m.chat.id, 'Я на связи. Напиши мне что-нибудь ')
Получение сообщений от юзера
@bot.message_handler(content_types=["text"])
def handle_text(message):
 bot.send_message(message.chat.id, 'Вы написали: ' +
message.text)
Запускаем бота
bot.polling(none_stop=True, interval=0)
```

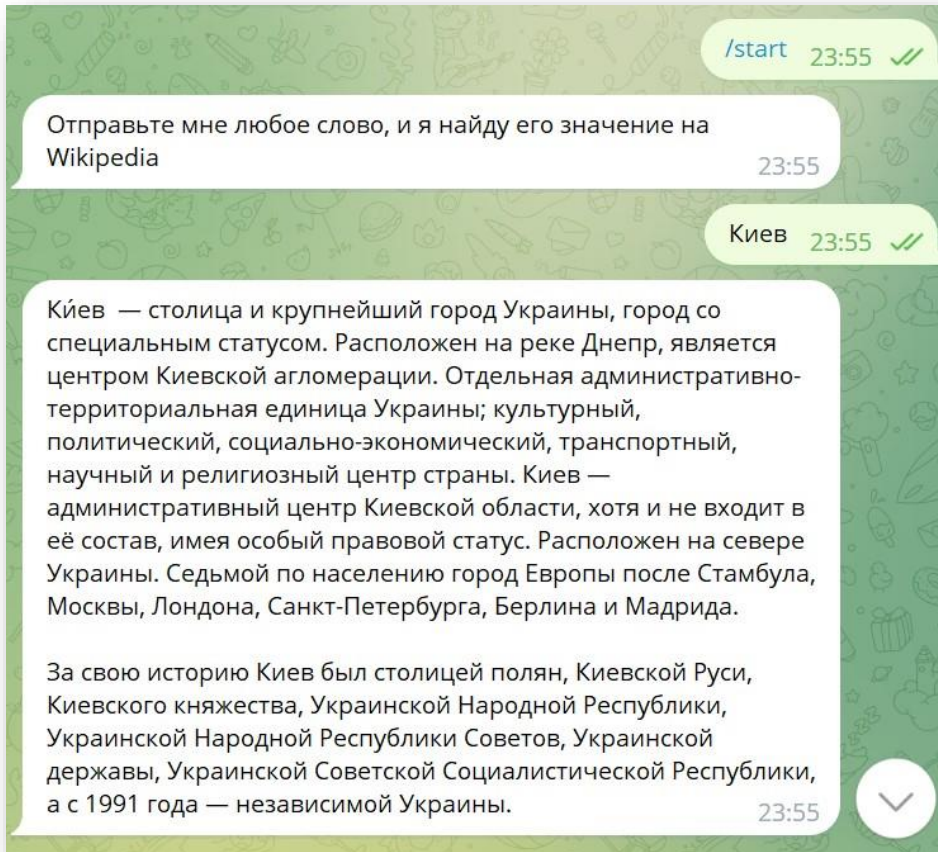


Запустити скрипт і в пошуку Telegram шукаємо свого бота за адресою, яку ви придумали раніше. Запускаємо кнопку «Запустити» (Пуск) або команду /start



## ВІКІПЕДІЯ-БОТ

Навчимо нашого бота не просто відправляти повідомлення навпаки, а чому-небудь цікавіше. Наприклад, за введеним словом давати статтю у Вікіпедії. Тут нам допоможе модуль Wikipedia:



```
pip install Wikipedia
```

```
готуємо код
```

```
import telebot, wikipedia, re
```

```
екземпляр бота
```

```
bot = telebot.TeleBot('токен @botfather')
```

```
wikipedia.set_lang("ua")
```

```
текст статті у Вікіпедії та обмежуємо його тисячами символів
```

```
def getwiki(s):
```

```
 try:
```

```
 ny = wikipedia.page(s)
```

```
 # Получаем першу тисячу символів
```

```
 wikitext=ny.content[:1000]
```

```

Разделяем по точкам
wikimas=wikitext.split('.')
Відбрасуємо все після останньої точки
wikimas = wikimas[:-1]
Створюємо пусту змінну для тексту
wikitext2 = ''
Проходимося по строкам, де немає знаків крім
заголовків
for x in wikimas:
 if not('==' in x):

 if(len((x.strip()))>3):
 wikitext2=wikitext2+x+'.'
 else:
 break
Тепер за допомогою регулярних виразів убираємо
розмітку
wikitext2=re.sub('\([^()]*\)', '', wikitext2)
wikitext2=re.sub('\([^()]*\)', '', wikitext2)
wikitext2=re.sub('\{[\^\\{\\}]*\}', '', wikitext2)

return wikitext2
Обробляємо виключення, яке можна повернути модуль
wikipedia за запитом
except Exception as e:
 return 'В энциклопедии нет информации об этом'
Функція, що обробляє команду /start
@bot.message_handler(commands=["start"])
def start(m, res=False):
 bot.send_message(m.chat.id, 'Отправьте мне любое
слово, и я найду его значение на Wikipedia')
Отримання повідомлень від користувача
@bot.message_handler(content_types=["text"])
def handle_text(message):
 bot.send_message(message.chat.id,
getwiki(message.text))
Запускаємо бота
bot.polling(none_stop=True, interval=0)

```

## Завдання

### Строки

1. Напишіть програму Python для обчислення довжини рядка.
2. Напишіть програму Python для підрахунку кількості символів (частоти символів) у рядку.
3. Напишіть програму на Python, щоб отримати рядок із перших 2 і останніх 2 символів заданого рядка. Якщо довжина рядка менше 2, повертається замість порожнього рядка.
4. Напишіть програму на Python, щоб отримати рядок із заданого рядка, де всі входження його першого символу було змінено на «!», окрім самого першого символу.
5. Напишіть програму Python, щоб отримати один рядок із двох заданих рядків, розділених пробілом, і поміняти місцями перші два символи кожного рядка.
6. Напишіть програму на Python, щоб додати «ing» у кінець заданого рядка (довжина має бути принаймні 3). Якщо вказаний рядок уже закінчується на «про», додайте натомість «ка». Якщо довжина даного рядка менша за 3, залиште її без змін
7. Напишіть функцію Python, яка приймає список слів і повертає найдовше слово та довжину найдовшого
8. Напишіть програму на Python, щоб змінити заданий рядок на новий рядок, де перший і останній символи були поміняні місцями
9. Напишіть програму Python для видалення символів, які мають непарні значення індексу заданого рядка
10. Напишіть сценарій Python, який приймає дані від користувача та відображає їх у верхньому та нижньому регістрах
11. Напишіть програму Python, яка приймає послідовність слів, розділених комами, як вхідні дані та друкує унікальні слова в сортованому вигляді (буквено-цифровий)
12. Напишіть функцію Python, щоб вставити рядок у середину рядка
13. Напишіть функцію Python, щоб отримати рядок із 4 копій двох останніх символів зазначеного рядка (довжина має бути принаймні 2)

14. Напишіть функцію Python, щоб отримати рядок із перших трьох символів зазначеного рядка. Якщо довжина рядка менша за 3, поверніть вихідний рядок.
15. Напишіть програму на Python для підрахунку повторів кожного слова в даному реченні

При всьому при тому,  
При всьому при тому,  
Хай бідні ми з вами,  
Багатство —  
Штамп на золотому,  
А золотий —  
Ми самі!  
Р. Бернс

## Список

1. Напишіть програму Python для підсумовування всіх елементів у списку
2. Напишіть програму Python для множення всіх елементів у списку
3. Напишіть програму Python, щоб отримати найбільше число зі списку
4. Напишіть програму Python, щоб отримати найменше число зі списку
5. Напишіть програму на Python для підрахунку кількості рядків, у яких довжина рядка становить 2 або більше, а перший і останній символи однакові з заданого списку рядків.
6. Напишіть програму Python, щоб отримати список, відсортований у порядку зростання за останнім елементом у кожному кортежі з заданого списку непорожніх кортежів
7. Напишіть програму Python для видалення дублікатів зі списку `a = [10,20,30,20,10,50,60,40,80,50,40]`
8. Напишіть програму на Python, щоб перевірити, порожній список чи ні
9. Напишіть програму Python для клонування або копіювання списку
10. Напишіть програму на Python, щоб знайти список слів, які довші за `n` із заданого списку слів.
11. Напишіть функцію Python, яка приймає два списки та повертає `True`, якщо вони мають принаймні один спільний член
12. `[1,2,3,4,5], [5,6,7,8,9]`
13. `[1,2,3,4,5], [6,7,8,9]`

14. Напишіть програму Python для друку зазначеного списку після видалення 0-го, 4-го та 5-го елементів ['Red', 'Green', 'White', 'Black', 'Pink', 'Yellow']
15. Напишіть програму Python для створення та друку списку перших і останніх 5 елементів, де значення є квадратом чисел від 1 до 30 (обидва включені).
16. Напишіть програму Python для створення та друку списку, за винятком перших 5 елементів, де значення є квадратом чисел від 1 до 30 (обидва включені).

## Словник

1. Напишіть сценарій для сортування (за зростанням і спаданням) словника за значенням.
2. Напишіть сценарій, щоб додати ключ до словника.
3. Напишіть сценарій для об'єднання наступних словників для створення нового
4. Напишіть сценарій, щоб перевірити, чи даний ключ уже існує в словнику
5. Напишіть програму для повторення словників за допомогою циклів for.
6. Напишіть сценарій, щоб надрукувати словник, де ключі — це числа від 1 до 15 (обидва включені), а значення — квадрати ключів
7. Напишіть програму для множення всіх елементів у словнику
8. Напишіть програму на для відображення двох списків у словник.
9. Напишіть програму для сортування заданого словника за ключем
10. Напишіть програму Python, щоб отримати максимальне та мінімальне значення в словнику
11. Напишіть програму Python для отримання словника з полів об'єкта.
12. Напишіть програму Python для видалення дублікатів зі словника
13. Напишіть програму на Python, щоб перевірити, порожній словник чи ні
14. Напишіть програму Python, щоб знайти 3 найвищі значення відповідних ключів у словнику.
15. Напишіть програму Python для підрахунку значень, пов'язаних із ключем у словнику
16. Напишіть програму Python, щоб отримати ключ, значення та елемент у словнику.
17. Напишіть програму Python для заміни словникових значень їхнім середнім значенням.

## Модулі

1. Напишіть програму Python для вибору випадкового елемента зі списку, набору, словника (значення) і файлу з каталогу.
2. Напишіть програму Python для генерації випадкового символу алфавіту, алфавітного рядка та алфавітного рядка фіксованої довжини.
3. Напишіть програму на Python для створення генератора випадкових чисел із початковим кодом, а також генеруйте значення з плаваючою точкою між 0 і 1, за винятком 1.
4. Напишіть програму на Python для генерації випадкового цілого числа від 0 до 6, за винятком 6, випадкового цілого числа від 5 до 10, за винятком 10, випадкового цілого числа від 0 до 10 із кроком 3 і випадковою датою між двома датами.
5. Напишіть програму Python для перемішування елементів заданого списку.
6. Напишіть програму Python для генерації числа з плаваючою точкою від 0 до 1 включно та генерування випадкового числа з плаваючою точкою в межах певного діапазону (1-10)
7. Напишіть програму на Python, щоб перевірити, чи є функція визначеною користувачем.
8. Напишіть програму Python, щоб перевірити, чи дана функція є генератором чи ні. Використання типів
9. Напишіть програму Python, щоб перевірити, чи дане значення є скомпільованим кодом чи ні. Також перевірте, чи дане значення є модулем чи ні.
10. Напишіть програму на Python, щоб створити поверхневу копію заданого списку.
11. Напишіть програму Python для створення глибокої копії заданого списку.
12. Напишіть програму на Python для створення неглибокої копії заданого словника.

## Масив

1. Напишіть програму Python для створення масиву з 5 цілих чисел і відображення елементів масиву. Доступ до окремих елементів через індекси.
2. написати програму Python, щоб додати новий елемент у кінець масиву.

3. Напишіть програму Python, щоб отримати довжину в байтах одного елемента масиву у внутрішньому представленні
4. Напишіть програму Python, щоб отримати поточну адресу пам'яті та довжину в елементах буфера, який використовується для зберігання вмісту масиву, а також знайти розмір буфера пам'яті в байт Напишіть програму Python для додавання елементів із iterable у кінець масиву.
5. Напишіть програму Python для перетворення масиву в масив машинних значень і повернення представлення байтів
6. Напишіть програму Python, щоб вставити новий елемент перед другим елементом у існуючому масиві.
7. Напишіть програму Python, щоб видалити вказаний елемент за допомогою індексу з масиву
8. Напишіть програму Python для видалення першого входження зазначеного елемента з масиву.
9. Напишіть програму Python для перетворення масиву на звичайний список з тими самими елементами
10. Напишіть програму на Python, щоб визначити, чи містить даний масив цілих чисел будь-який повторюваний елемент. Повертає true, якщо будь-яке значення з'являється принаймні двічі у зазначеному масиві, і повертає false, якщо кожен елемент є окремим.
11. Напишіть програму Python для пошуку пари з найбільшим добутком із заданого масиву цілих чисел.

## Класи

1. Напишіть програму Python для імпорту вбудованого модуля масиву та відображення простору імен зазначеного модуля
2. Напишіть програму Python для створення класу та відображення простору імен зазначеного класу
3. Напишіть програму Python для створення екземпляра зазначеного класу та відображення простору імен зазначеного екземпляра
4. Визначте функцію Python student(). За допомогою атрибутів функції відобразити імена всіх аргументів.
5. Напишіть функцію Python student\_data (), яка друкуватиме ідентифікатор студента (student\_id). Якщо користувач передає аргумент student\_name або student\_class, функція виведе ім'я студента та клас

6. Напишіть простий клас Python під назвою Student і відобразіть його тип. Також відобразіть ключі атрибутів `__dict__` і значення атрибута `__module__` класу Student.
7. Напишіть програму Python для створення двох порожніх класів, Student і Marks. Тепер створіть кілька екземплярів і перевірте, чи є вони екземплярами зазначених класів чи ні. Також перевірте, чи є зазначені класи підкласами вбудованого класу об'єктів чи ні
8. Напишіть клас Python під назвою Student із двома атрибутами `student_id`, `student_name`. Додайте новий атрибут `student_class` і відобразіть увесь атрибут і його значення для зазначеного класу. Тепер видаліть атрибут `student_name` і відобразіть весь атрибут зі значеннями.
9. Напишіть клас Python під назвою Student із двома екземплярами `student1`, `student2` і призначте дані значення атрибутам зазначених екземплярів. Вивести всі атрибути екземплярів `student1`, `student2` з їхніми значеннями в заданому форматі.
10. Напишіть клас Python для перетворення цілого числа на римську цифру
11. Напишіть клас Python, щоб отримати всі можливі унікальні підмножини з набору різних цілих чисел
12. Напишіть клас Python під назвою Rectangle, побудований за довжиною та шириною, і метод, який обчислюватиме площу прямокутника
13. Напишіть клас Python під назвою Circle, побудований за допомогою радіуса та двох методів, які обчислюватимуть площу та периметр кола

## **tkinter**

1. Напишіть програму Python GUI для імпорту пакета Tkinter, створення вікна та встановлення його заголовка.
2. Напишіть програму Python GUI для імпорту пакета Tkinter і створення вікна. Встановіть його заголовок і додайте мітку до вікна
3. Напишіть програму Python GUI для створення вікна та встановлення розміру вікна за замовчуванням за допомогою модуля tkinter
4. Напишіть програму з графічним інтерфейсом Python, щоб створити вікно та вимкнути зміну розміру вікна за допомогою модуля tkinter
5. Напишіть програму з графічним інтерфейсом Python, щоб додати кнопку у вашу програму за допомогою модуля tkinter
6. Напишіть програму Python GUI для створення Combobox із трьома параметрами за допомогою модуля tkinter



7. Напишіть програму Python GUI для створення віджета Checkbutton за допомогою модуля tkinter
8. Напишіть програму Python GUI для створення текстового віджета за допомогою модуля tkinter. Вставте рядок на початку, а потім вставте рядок у поточний текст. Видалити перший і останній символи тексту
9. Напишіть програму Python GUI, щоб створити три однорядкові текстові поля для прийняття значення від користувача за допомогою модуля tkinter.
10. Напишіть програму Python GUI для створення трьох віджетів перемикачів за допомогою модуля tkinter
11. Напишіть програму Python GUI для створення віджетів ScrolledText за допомогою модуля tkinter
12. Напишіть програму Python GUI для створення віджетів панелі виконання за допомогою модуля tkinter
13. Напишіть програму Python GUI для створення віджетів панелі Listbox за допомогою модуля tkinter.

## Різнманітні завдання

1. Напишіть програму Python, яка приймає список цілих чисел і перевіряє довжину та п'ятий елемент. Повертає true, якщо довжина списку дорівнює 8 і п'ятий елемент зустрічається у цьому списку тричі.
2. Напишіть програму Python, щоб перевірити, слова в реченні є паліндромами чи ні. Повернути True, False
3. Напишіть програму на Python, щоб знайти довжину заданого списку непорожніх рядків.
4. Напишіть програму Python для створення рядка, що складається з невід'ємних цілих чисел до n включно
5. Напишіть програму на Python, щоб вибрати рядок із заданого списку рядків із найбільшою кількістю унікальних символів
6. Напишіть програму Python для зміни масштабу та зміщення чисел заданого списку, щоб вони охоплювали діапазон [0, 1]
7. Напишіть програму Python для обчислення добутку непарних цифр заданого числа або 0, якщо їх немає.

8. Напишіть програму Python для фільтрації чисел у числах у заданому списку, сума цифр якого  $> 0$ , де перша цифра може бути від'ємною.
9. Напишіть програму Python для видалення дублікатів зі списку цілих чисел, зберігаючи порядок.
10. Напишіть програму Python для пошуку чисел, які більші за 10 і мають непарні першу та останню цифри.
11. Напишіть програму на Python, щоб знайти список усіх чисел, які є суміжними з простим числом у списку, відсортованих без дублікатів.
12. Напишіть програму на Python для пошуку індексів найближчої пари зі списку чисел.
13. Напишіть програму Python для пошуку найбільшого від'ємного та найменшого додатного чисел (або 0, якщо немає).

| Термін          | Пояснення                                                                                                                                                                                                                                               |
|-----------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Add (додавання) |                                                                                                                                                                                                                                                         |
| And             | Використовується для вказівки, що обидві умови повинні бути виконані для повернення значення істини<br><pre>if num&gt;10 and num &lt; 20:     print("В діапазоні") else:     print("Поза межами")</pre>                                                 |
| append          | Додає окремий елемент у кінець списку, кортежу, словника, рядка або масив.<br><pre>fruits = ['apple', 'banana', 'cherry'] fruits.append("orange")</pre>                                                                                                 |
| argument        | Значення, яке передається функції (або методу) під час виклику функції. Існує два типи аргументів:<br><pre>arg(6, 12) arg(*(6, 15))</pre>                                                                                                               |
| attribute       | Значення, пов'язане з об'єктом, ім'я якого зазвичай посиляється за допомогою виразів із крапками. Наприклад, якщо об'єкт о має атрибут а, на нього буде посилатися                                                                                      |
| array           | У Python масиви схожі на списки, але вони використовуються лише для зберігання чисел. Користувач визначає конкретний тип числа, тобто ціле, довге, подвійний або з плаваючою комою.<br><pre>Num = array('f', [67, 456, 678, 23, 567]) print(nums)</pre> |
| button          | Використовується з Tkinter. Наведений нижче код створює кнопку, яка буде запустити підпрограму «клік».<br><pre>B = Tkinter.Button(top, text ="Hello", command = helloCallBack)</pre>                                                                    |
| choise          | Вибирає випадковий вибір зі списку варіантів.<br><pre>import random  mylist = ["apple", "banana", "cherry"]</pre>                                                                                                                                       |

|                           |                                                                                                                                                                                                                                 |
|---------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|                           | <code>print(random.choice(mylist))</code>                                                                                                                                                                                       |
| <code>&gt;&gt;&gt;</code> | Типовий запит Python інтерактивної оболонки. Часто зустрічається для прикладів коду, які можна виконувати інтерактивно в інтерпретаторі.                                                                                        |
| <b>class</b>              | Шаблон для створення користувальницьких об'єктів. Визначення класу зазвичай містять визначення методів, які працюють над екземплярами класу.<br><code>class ClassName:<br/>    obj = ClassName()<br/>    print(obj.attr)</code> |
| compiler                  | Перекладає програму, написану мовою високого рівня, наприклад Python у мову низького рівня, наприклад машинний код.                                                                                                             |
| concatenation             | З'єднує дві строки в одну.<br><code>Name = firstname + othername</code>                                                                                                                                                         |
| count                     | Підраховує, скільки разів частина даних з'являється в списку, кортежі, словник, рядок або масив.<br><code>Print(names_list.count("Так"))</code>                                                                                 |
| def                       | Визначає підпрограму.<br><code>def menu():<br/>    print()</code>                                                                                                                                                               |
| del                       | Видаляє елемент зі списку.<br><code>thislist =<br/>["apple", "banana", "cherry"]<br/>del thislist</code>                                                                                                                        |
| dictionary                | Тип списку, у якому визначені користувачем індекси порівнюються зі значеннями.<br><code>thisdict = {<br/>    "brand": "Tesla",<br/>    "model": "Q7",<br/>    "year": 2021<br/>}<br/>print(thisdict)</code>                     |
| <b>decorator</b>          | Функція, що повертає іншу функцію, зазвичай застосовується як перетворення функції<br><code>def f(arg):<br/>    ...<br/>    f = staticmethod(f)</code>                                                                          |
| elif                      | Використовується в операторі if для перевірки нової умови, якщо попередня умова не виконано.<br><code>letter = "A"</code>                                                                                                       |

|                  |                                                                                                                                                                                                                                 |
|------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|                  | <pre> <b>if</b> letter == "B":     print("letter is B")  <b>elif</b> letter == "C":     print("letter is C")  <b>elif</b> num == "A":     print("letter is A")  <b>else</b>:     print("letter isn't A, B or C") </pre>         |
| else             | <p>Використовується в операторі if, щоб визначити, що відбувається, якщо попередні умови не виконано.</p> <pre> x = 79 <b>if</b> x == 80:     print("Yes") <b>else</b>:     print("No") </pre>                                  |
| entry box        | <p>Використовується в GUI з Tkinter, щоб дозволити користувачеві вводити дані або вихід на дисплей. Наведений нижче код створює порожнє поле введення.</p> <pre> name_entry = tk.Entry(root, font=('arial',10,'normal')) </pre> |
| loop             | <p>Тип циклу, який повторюватиме блок коду певну кількість разів.</p>                                                                                                                                                           |
| forward          | <p>Рухає turtle вперед; якщо знак курсора вниз, то залишається слід коли він рухається, малюючи пряму лінію на екрані.</p> <pre> turtle.forward() </pre>                                                                        |
| <b>IDLE</b>      | <p>Інтегроване середовище розробки та навчання для Python. IDLE — це базове середовище редактора та інтерпретатора, яке постачається зі стандартним дистрибутивом Python.</p>                                                   |
| <b>in</b>        | <p>Можна використовувати для перевірки наявності символу в рядку. Це корисно для обох операторів for і if.</p> <pre> <b>for</b> i <b>in</b> abc:      print(i): </pre>                                                          |
| <b>list</b>      | <p>Вбудована послідовність Python. Незважаючи на свою назву, він більше схожий на масив в інших мовах, ніж на пов'язаний список</p>                                                                                             |
| <b>metaclass</b> | <p>Клас класу. Визначення класу створюють назву класу, словник</p>                                                                                                                                                              |

|                         |                                                                                                                                                                                                                                 |
|-------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|                         | класу та список базових класів. Метаклас відповідає за отримання цих трьох аргументів і створення класу. Більшість об'єктно-орієнтованих мов програмування забезпечують реалізацію за замовчуванням.                            |
| <b>method</b>           | Функція, яка визначена всередині тіла класу. Якщо викликати його як атрибут екземпляра цього класу, метод отримає об'єкт екземпляра як перший аргумент (який зазвичай називається <code>self</code> ).                          |
| <b>module</b>           | Об'єкт, який є організаційною одиницею коду Python. Модулі мають простір імен, що містить довільні об'єкти Python.                                                                                                              |
| <b>object</b>           | Будь-які дані зі станом (атрибути або значення) і визначеною поведінкою (методи).                                                                                                                                               |
| <b>parameter</b>        | Іменована сутність у визначенні функції (або методу), яка визначає аргумент (або в деяких випадках аргументи), який функція може приймати.                                                                                      |
| <b>type</b>             | Тип об'єкта Python визначає тип об'єкта; кожен об'єкт має тип. Тип об'єкта доступний як його атрибут <code>__class__</code> або може бути отриманий за допомогою <code>type(obj)</code> .                                       |
| <code>turtle</code>     | Інструмент для малювання фігур на екрані.<br><pre>import turtle skk = turtle.Turtle()  for i in range(4):     skk.forward(50)     skk.right(90)  turtle.done()</pre>                                                            |
| <code>while loop</code> | Тип циклу, який повторюватиме блок коду всередині нього, якщо виконується певна умова.<br><pre>i = 1 while i &lt; 6:     print(i)     i += 1</pre>                                                                              |
| <code>window</code>     | Екран, який використовується в графічному інтерфейсі. Наведений нижче код створює вікно, називається «window», додає заголовок і визначає розмір.<br><pre>window = Tk() window.title("Window") window.geometry("250x200")</pre> |

|                        |                                                                                                                                        |
|------------------------|----------------------------------------------------------------------------------------------------------------------------------------|
| <b>virtual machine</b> | Комп'ютер, повністю визначений програмним забезпеченням.<br>Віртуальна машина Python виконує байт-код, виданий компілятором байт-коду. |
|------------------------|----------------------------------------------------------------------------------------------------------------------------------------|

