Міністерство освіти і науки України
Сумський державний педагогічний університет імені А.С.Макаренка
Фізико-математичний факультет
Кафедра інформатики

СЕРГІЙ ПЕТРЕНКО, НЕЛЯ ДЕГТЯРЬОВА

PYTHON:
ОСНОВИ ПРОГРАМУВАННЯ

Навчальний посібник для студентів спеціальності
014 Середня освіта (Інформатика)

Суми – 2023

УДК 378:004.421

Рекомендовано до друку рішенням кафедри інформатики
Сумського державного педагогічного університету імені А.С.Макаренка
Протокол №1 від 29 серпня 2023 р.

Петренко С.І. – кандидат педагогічних наук, доцент, доцент кафедри інформатики СумДПУ імені А.С.Макаренка.
Дегтярьова Н.В. – кандидат педагогічних наук, доцент, доцент кафедри інформатики СумДПУ імені А.С.Макаренка.

Петренко С., Дегтярьова Н. **Мова програмування Python: Основи програмування.** Навчальний посібник для студентів спеціальності 014 Середня освіта (Інформатика) Суми: СумДПУ імені А.С.Макаренка, 2023. 101 с.

У посібнику розглядаються базові принципи програмування мовою Python. Описується синтаксис основних структур мови програмування Python 3. Увесь матеріал посібника добре структурований та компактно викладений. У посібнику представлено приклади програмного коду, що забезпечить краще засвоєння матеріалу. Посібник містить завдання для самостійної роботи та запитання для самоконтролю що надає можливість самостійно вивчати мову Python. Посібник може бути корисним викладачам та студентам ЗВО, учителям та учням старших класів ЗЗСО а також іншим читачам, що прагнуть навчитися програмувати мовою Python.

Рецензенти:
Ігор КОТОВ – д-р технічних наук, доцент, доцент кафедри моделювання та програмного забезпечення Криворіського національного університету.
Микола МАЛЯР – д-р технічних наук, професор, декан факультету математики та цифрових технологій Ужгородського національного університету.
Оксана ЖМУД – к. пед. наук, доцент, доцент кафедри інформатики і інформаційно-комунікаційних технологій Уманського державного педагогічного університету імені Павла Тичини.
Євгенія Голуб – к. ф. наук, доцент, викладач вищої категорії циклової комісії фундаментальних дисципліни та комп'ютерних технологій ВСП «Ірпінський фаховий коледж НУБіП України»

ЗМІСТ

«Успіх в житті – не скільки питання таланту і можливостей,
стільки концентрації і наполегливості.»
C. W. Wendt

ВСТУП

У становленні майбутніх учителів інформатики вивчення програмування займає одне з ключових місць. Досвід показує, що вивчення мов програмування активно сприяє формуванню алгоритмічного, логічного та творчого мислення. Процес програмування передбачає набуття практичних навичок пошуку нестандартних рішень.

Мова програмування Python для формування вище перерахованих якостей підходить з декількох причин:

– Python - досить «молода» мова, але вже популярна, яка широко використовується в практичній діяльності системних програмістів і область її застосування повсякчас розширюється.

– Python стрімко розвивається.

– Python – є досить потужною і одночасно простою для вивчення мовою програмування.

– Синтаксис мови Python мінімалістичний і гнучкий.

– На Python можна складати прості та ефективні програми.

– Бібліотеки Python містить велику кількість корисних функцій, що значно полегшує процес створення програмних кодів.

– Середовища розробки та бібліотеки в основному розміщені у вільному доступі.

Перераховані вище причини роблять мову Python вдалим вибором в якості першої мови при навчанні програмуванню.

Цей посібник допоможе зробити перші кроки з вивчення мови програмування Python.

1. ЗАГАЛЬНІ ВІДОМОСТІ ПРО МОВУ ПРОГРАМУВАННЯ PYTHON.

1.1. Історія мови

Автором мови Python є нідерландський програміст Гвідо ван Россум. Він як співробітник голландського інституту CWI працював над мовою ABC, яка повинна була стати дуже зрозумілою і простою мовою. Мова ABC була досить цікавою, але вона мала один великий недолік: нею ніхто не користувався. Тому в грудні 1989 року Гвідо ван Россум розпочав свій новий проект Python. Мова Python створювалася з метою уникнути помилок, які були допущені при розробці мови ABC. Датою появи мови Python вважається 20 лютого 1991 року.

Спочатку роботи Гвідо ван Россум був єдиним, хто мав право приймати остаточні рішення щодо мови. Він очолював розвиток мови і координував спільноту розробників Python багато років.

Він також відомий своєю «Директивою зіниці» (Python Enhancement Proposal 20 або PEP 20), яка містить «The Zen of Python» - колекцію принципів та філософії, які лежать в основі розробки Python. Ця "Директива зіниці" надихала інших програмістів та визначала культуру спільноти Python.

У липні 2018 року Гвідо ван Россум вирішив відступити від ролі "Benevolent Dictator For Life" (BDFL) для проекту Python, але продовжував активно співпрацювати з спільнотою та вносити свій вклад у розвиток мови.

Назва мови немає нічого спільного з великим амазонським плазуном, вона отримала назву на честь легендарного британського телешоу «Літаючий цирк Монти Пайтона» (англ. *Monty Python's Flying Circus*), хоча дехто називає мову «Пітон».

1.2. Філософія мови Python

Python – багатоцільова мова програмування, яка дозволяє писати програмний код, що добре читається. Python дозволяє створити програми, код яких буде значно коротшим від коду, написаного на інших мовах.

Python – багатоплатформенна мова програмування. Це означає, що програми створені на Python можна запускати в різних операційних системах.

Досить суттєвою перевагою Python є наявність стандартної бібліотеки, яка інсталюється разом з Python і містить готові інструменти для роботи з операційною системою, веб-сторінками, базами даних, різними форматами даних, для побудови графічного інтерфейсу програм тощо.

На даний момент існують дві актуальні версії мови: Python 2.x (остання версія – 2.7) і Python 3.x (остання версія – 3.12 на 31.01.2023р.). Python 3.x не повністтю сумісний з серією 2.x. Код написаний в Python 2.x, при виконанні, в Python 3.x, швидше за все буде повідомляти про помилки. Тому доцільно нові проекти писати в Python 3.x

Python – одна із сучасних мов програмування високого рівня.

Один з недоліків мови Python є швидкість виконання коду. Код на Python спочатку компілюється у внутрішній байт-код, який потім виконується інтерпретатором Python. У переважній кількості випадків при використанні Python програми працюють повільніше ніж на інших сучасних мовах, таких як C++. Але, сучасні комп'ютери мають таку обчислювальну потужність, що для більшості завдань швидкість написання коду важливіша від швидкості виконання, а код на Python пишеться набагато швидше.

Python – це високорівнева *інтерпретована* мова програмування, на відміну від C++, яка є прикладом *компільованої* мови програмування.

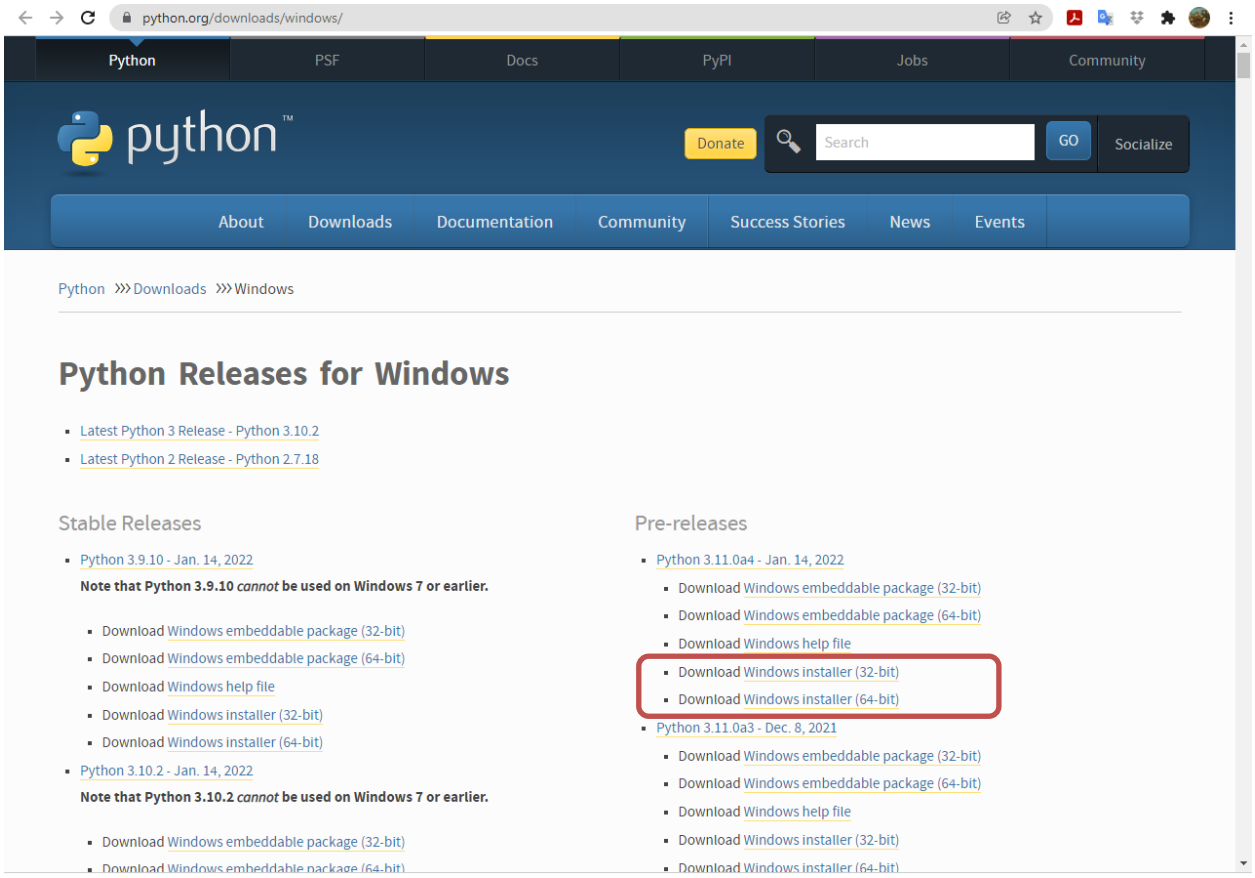
Програми, написані на мові високого рівня, називаються *початковим кодом*. Для виконання програми комп'ютером необхідно перекласти початковий код на машинну мову. Для перекладу мови високого рівня на машинну доступні два типи програм:

– ***Компілятор*** перекладає весь початковий код на машинну мову за один раз, потім машинний код виконується.

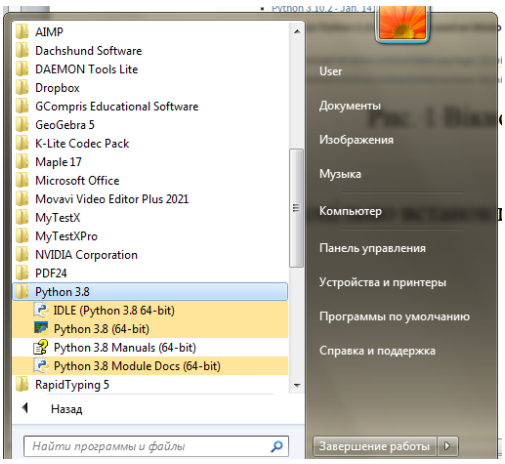
– ***Інтерпретатор*** Python починає свою роботу у верхній частині файла, перекладає перший рядок на машинну мову, а потім виконує його. Цей процес повторюється до кінця файла.

1.3. Інсталяція мови Python

Python абсолютно безкоштовний і вільно поширюється На сайті https://www.python.org. Для інсталяції програми на комп'ютер з операційною системою Windows потрібно перейти на сторінку https://www.python.org/downloads/windows/ і вибрати та скачайти інсталятор середовища програмування Python, що відповідає технічним характеристикам і програмним можливостям вашого комп'ютера. Скріншот вікна сторінки сайту представлений на рис.1.



Доцільно встановлювати на комп'ютер одну із версій 3.x.



В меню «Пуск» містяться чотири пункти:

– **IDLE Python** – (Integrated Development and Learning Environment) – інтегроване середовище розробки. Воно подібне до інтерпретатора, запущеного в командному рядку, але має більший набір можливостей (підсвічування синтаксису, перегляд об’єктів тощо).

– **Python** – інтерпретатор програми в командному рядку Windows.

– **Python Manuals** – довідкова система (англійською мовою).

– **Python Module** – бібліотека вбудованих додаткових модулів.

Завдання для самостійної роботи.

1. Установіть середовище програмування Python на комп’ютер.
2. Запустіть середовище Python (command line)
3. Після символів >>> введіть команду **print (`Ура, я буду вивчати Python`)** і натисніть Enter.
4. Запустіть середовище **IDLE Python**.
5. Створіть новий файл.
6. В новому вікні введіть команду *print (" Ура, я буду вивчати Python ").*
7. Збережіть файл під ім’ям «Перша»
8. Запустіть програму на виконання (Run/ Run module, F5).

Запитання для самоконтролю

1. На яких платформах можна застосовувати програми мовою Python?
2. Чи можна запускати на інших платформах код програми мовою Python написаний в середовищі ОС Windows?
3. Чим відрізняється робота компілятора та інтерпретатора?
4. Перелічіть недоліки та переваги мови Python.
5. Як інсталювати Python на комп’ютер?

2. СИНТАКСИС МОВИ PYTHON.

2.1. Синтаксис мови

Синтаксис мови Python, як і сама мова, достатньо простий. Можна виділити такі твердження:

1) Кінець рядка є кінцем інструкції (прикінцеві символи непотрібні).

2) Для надання значень змінним оператором присвоювання є знак дорівнює «=». Формат оператора: *ім'я_змінної=вираз.*

3) Основна інструкція та вкладені інструкції (вкладений блок інструкцій) записуються відповідно до одного шаблону: основна інструкція завершується двокрапкою, наступними рядками розташовуються вкладені інструкції, з однаковим відступом на початку рядків по відношенню до основної інструкції. Наприклад:

Основна інструкція:

Вкладений блок інструкцій

Тобто вкладені інструкції об'єднуються в блоки за величиною відступів. Відступ може бути будь-яким, головне, щоб в межах одного вкладеного блоку відступ був однаковий. Для підвищення читабельності коду, для створення відступу рекомендується використовувати клавішу TAB.

Синтаксис оформлення основної інструкції та вкладеного блоку інструкцій кардинально відрізняється від синтаксису більшості мов, в яких використовуються операторні дужки для виділення вкладеного блоку інструкції (наприклад, begin … end в Паскалі або { … } в Сі).

4) Розмір літер має значення, тобто великі і маленькі літери вважаються різними. Більшість службових слів (окрім: False, None, True) та вбудованих функцій пишуться маленькими літерами. Проте існує декілька спеціальних випадків.

5) Можна записати кілька інструкцій в одному рядку, розділяючи їх крапкою з комою: *a = 1; b = 2; print(a, b)*

6) Можна записувати одну інструкцію в декілька рядків. Для цього необхідно розмістити її в парі круглих, квадратних або фігурних дужок:

if (a < 1 and b < 2 and

c < 3 and d < 4):

*print('spam' * 3)*

7) Якщо тіло вкладеної інструкції містить єдиний оператор, то він може розташовуватися в тому ж рядку, що і основна інструкція:

if x > y: print(x)

8) Крім конструкцій мови, програма може містити коментарі. Коментар – це довільний текст у будь-якому місці програми, що пишеться після символу #. Коментар допомагає зрозуміти призначення і являє собою замітку для того, хто буде переглядати код програми. Наприклад:

print('Привіт, Світ!') # print -це функція виведення повідомлень

або:

 # print - це функція виведення повідомлень

print('Привіт, Світ!')

Коментарі містять пояснювальні тексти, і полегшують читання і розуміння програм, тому доцільно в своїх програмах писати якомога більше корисних коментарів, які містять: припущення, важливі деталі, проблеми, які ви намагаєтеся вирішити, проблеми, яких намагаєтеся уникнути і тощо.

2.2. Змінні та літерали

Ідентифікатор – це ім’я деякої сутності (змінної, функції, класу) в програмі для її позначення. При виборі ідентифікаторів необхідно дотримуватися таких правил:

– Першим символом ідентифікатора може бути літера з будь-якого алфавіту (символ ASCII в верхньому або нижньому регістрі, або символ Unicode) або символ підкреслення «_».

– Інша частина ідентифікатора може складатися з літер, символу підкреслення «_» або цифр.

– У ідентифікаторів розрізняються символи верхнього або нижнього регістрів. Наприклад, myname і myName – це два різні ідентифікатори.

– Ідентифікатор не може співпадати з ключовими (зарезервованими) словом інтерпретатора Python.

Зарезервовані слова Python

false	class	finally	is	return
none	continue	for	lambda	try
true	def	from	nonlocal	while
and	del	global	not	with
as	elif	if	or	yield
assert	else	import	Pass	
break	except	in	raise	

Приклади допустимих імен ідентифікаторів: a, __my_name, name_23, a1b2_c3 і будь_які_символи_utf8_δξëèўЩΞéά.

Приклади неприпустимих імен ідентифікаторів: 2things, «вираз_в_лапках».

При написанні програм досить часто необхідно зберігати різні дані та мати можливість маніпулювання цими даними. У цьому випадку знадобляться змінні. *Змінна* – це ідентифікатор, значення якого може змінюватися в ході виконання коду програми, а отже, змінна може зберігати все, що завгодно. Змінна в мові Python – це просто посилання на область пам’яті комп’ютера, в якій зберігається деякі дані (посилання на деякий об’єкт). Змінні можуть зберігати значення різних типів.

Літерал (literal – константа) – постійне значення певного типу даних, записане у вихідному коді комп’ютерної програми. Прикладом літералу може бути число (наприклад: 5, 1.23, 9.25e-3) або рядок (наприклад, *'Це рядок', "It's a string!"*). Значення літералу використовується буквально, число 2 завжди

представляє саме себе і нічого іншого - це «константа», тому що її значення не можна змінити. В мові Python немає окремої синтаксичної конструкції для оголошення літералів. За правилами «хорошого тону» для наголошення на те, що ідентифікатор не можна змінювати, її прописують великими літерами.

2.3. Типи даних

Мова Python відноситься до мов з неявною строгою динамічною типізацією.

Неявна типізація означає, що при оголошенні змінної її тип не вказується (при явній типізації тип змінної вказується обов'язково).

Тип даних – це множина допустимих значень і операцій над цими значеннями.

В Python типи даних можна розділити на внутрішні, вбудовані в інтерпретатор (built-in) і зовнішні, які можна використовувати при імпортуванні відповідних модулів.

До основних вбудованих типів відносяться:

1. Логічний тип (Boolean Type)
2. Числа (Numeric Type)
   - int – ціле число
   - float – число з плаваючою точкою (дійсне число)
   - complex – комплексне число
3. Рядки (Text Sequence Type)
   - str
4. Послідовності (Sequence Type)
   - list – список
   - tuple – кортеж
   - range – діапазон
5. Множини (Set Types)
   - set – множина

– frozenset – незмінювана множина

6. Словники (Mapping Types)

– dict – словник

Враховуючи неявну типізацію мови Python при оголошенні змінної, їй повинно бути надане значення. Щоб оголосити змінну необхідно написати її ім’я, потім поставити оператор присвоєння (знак рівності «=») і вказати значення, з яким дана змінна буде створена.

Наприклад: za = 5.

Для того щоб змінити одні типи даних на інші використовуються певні стандартні функції.

Для зміни чогось на цілочисельний тип, слід використовувати функцію **int()**. Вона зберігає цілу частину числа і відкидає залишок. Текстовий рядок теж можна змінити на цілочисельний, якщо він буде містити цифрові символи і, можливо, знаки + і -. Функція ***int()*** буде створювати цілі числа з чисел з плаваючою точкою або рядків, що складаються з цифр. Проте вона не буде обробляти рядки, що містять порожній рядок, десяткові точки або експоненти.

Для того щоб перетворити інші типи в тип дійсних чисел, слід використовувати функцію ***float()***. Перетворення значення типу ***int*** в тип ***float*** лише створить десяткову кому.

2.4. Введення даних та виведення результатів

Виведення результатів роботи програми.

Для виведення результату роботи програми може бути використана функція **print().** Використовуючи функцію **print()** можна вивести як одне, так і декілька значень. Для виведення декількох значень їх необхідно відділити один від одного комами. Значення, що призначені для виведення, можуть бути різного типу, але при виведенні вони у будь - якому випадку будуть перетворені у рядковий тип.

Якщо виводиться декілька значень, то при виведенні вони будуть відокремлені одне від одного символом або декількома символами, за замовчуванням пропуском.

За замовчуванням, функція print() виводить значення в стандартний пристрій виведення (екран). За допомогою параметру file можна перенаправити виведення до файлу. Перенаправлення виведення буде розглянуто при розгляді роботи з файлами.

Введення даних до програми.

При написанні коду програм виникає необхідність в отриманні даних від користувача. Для реалізації такої можливості в Python передбачено функцію **input()**. Функція **input()** вводить значення тільки *рядкового* типу. Тому введені з клавіатури числа потрібно буде перевести до одного з числових типів ***int()*** або ***float().***

Наприклад:

x = input("x= ")

x = int(x),

або

x = int(input("x: "))

Завдання для самостійної роботи.

1. Напишіть код програми яка:
   - змінній присвоює вибране вами значення;
   - перетворює це значення у ціле і дійсне число?
   - виводить результати перетворень.

Запитання для самоконтролю

1. Чи є можливість записувати декілька інструкцій у одному рядку?
2. Який знак використовується для присвоєння значення змінній?
3. Як відділяється вкладений блок інструкцій від основного тексту коду?
4. Чи можна використовувати при написанні коду програми великі літери?

5. Як записується ім’я змінної?
6. Які слова називаються ключовими?
7. Які функції потрібно використати для зміни типу величини на ціле число, на дійсне число, на рядкову величину?
8. Які функції застосовуються для введення даних і виведення результатів?

3. ВБУДОВАНІ ФУНКЦІЇ ТА БІБЛІОТЕКИ В PYTHON

3.1. Запис виразів та операції з числами.

Для виконання операцій над числами різних типів в Python є декілька внутрішніх операцій, які містяться в стандартній бібліотеці. ***Операція*** – це виконання якихось дій над даними. Для виконання конкретних дій потрібні спеціальні інструменти – ***оператори***. Основні оператори представлені в таблиці.

Таблиця операторів

Оператор	**Опис**	**Приклад**	**Результат**
+	Додавання	5 + 8	13
-	Віднімання	90 – 10	80
*	Множення	4 * 7	28
/	Ділення з плаваючою точкою	7/2	3,5
//	Цілочисельне (Truncating) ділення	7//2	3
%	Залишок	7%3	1
**	Піднесення до степеню	3**4	81

Вставляти пропуск між кожним числом і оператором не доцільно. Зайві пропуски погіршують сприйняття коду.

Якщо в одному і тому ж виразі зустрічаються декілька типів даних, то результат має самий сильний тип зі всіх чисел що у виразі. Самим сильним типом вважається комплексний, за ним йде дійсний, а потім цілий тип даних.

Якщо вираз містить більше одного оператора, послідовність виконання операцій залежить від порядку їх слідування у виразі, а також від їх пріоритету. Пріоритети операторів в Python збігаються з пріоритетами арифметичних операцій (в таблиці оператори представлені у порядку зростання сили). У випадку рівних пріоритетів розрахунок йде зліва направо. Для зміни цього порядку використовують дужки. Краще їх використовувати для забезпечення однозначного сприйняття.

3.2. Вбудовані функції

В мові Python є вбудовані функції, які містяться в стандартній бібліотеці і доступні без будь-яких додаткових вказівок. Розглянемо функції, що найчастіше застосовуються в практичній діяльності до цілих та дійсних чисел.

Функція	Призначення
abs(X)	обчислює абсолютне значення (модуль) числа
divmod(A, B)	обчислює пару чисел **(P, R)**, де **P** є цілою частиною а **R** остачею при виконанні цілочисельного ділення
pow(X, Y[, Z])	обчислює X в степені Y та виводить залишок від ділення на **Z** (обчислюється більш ефективно, ніж **pow(X, Y) % Z**). Якщо параметр **Z** заданий, то **X** та **Y** мають бути цілочисельними, окрім того **Y** має бути невід'ємним
round(n[, d])	повертає число **n**, округлене до **d** знаків після десяткової крапки
max(arg1, arg2, *args)	повертає найбільше значення з двох чи більше аргументів
min(arg1, arg2, *args)	повертає найменше значення з двох чи більше аргументів
sum(**arg1, arg2, *args, ***)	обчислюєє суму всіх аргументів

Перелік вбудованих функцій не обмежується перерахованими. Інформацію про інші функції можна знайти в документації **Python (**Python Documentation contents**),** або в мережі за посиланням https://docs.python.org/3/.

3.3. Бібліотеки в Python

Python має значну кількість бібліотек (модулів). Окрім стандартної бібліотеки з досить великим набором функцій, які можуть бути використанні при написанні програм. Ці модулі містять уже заготовлені частини програм. Щоб ними скористатися треба знати про їх існування, імпортувати у програму та відповідним чином викликати.

Для імпортування будь-якої бібліотеки потрібно прописати:

from [назва бібліотеки] *impor*t [параметр]

Бібліотекою є **math** (бібліотека математичних функцій), **tkinter** (бібліотека графічних елементів та створення віджетів) та інші.

Параметром може бути символ * – якщо потрібно завантажити усі елементи, які містяться в цій бібліотеці, або ім'я конкретної функції із вказаної бібліотеки.

Розглянемо деякі з них.

Calendar – дозволяє надрукувати календар, а також містить інші корисні функції для роботи з календарями.

Os – надає велику кількість функцій для роботи з операційною системою.

Datetime – містить функції для обробки часу та дати різними способами. Робиться акцент на простоті роботи з датою, часом та їх частинами.

Collections – надає спеціалізовані типи даних, на основі словників, кортежів, множин, списків.

Array – визначає масиви в Python.

Time – модуль для роботи з часом у Python.

Sys – забезпечує доступ до деяких змінних та функцій, що взаємодіють з інтерпретатором Python.

Math – модуль надає великий функціонал для роботи з числами.

Cmath – надає функції для роботи із комплексними числами.

Tkinter – бібліотека графічних елементів та створення віджетів.

Random – надає функції для створення випадкових чисел, букв, випадкового вибору елементів послідовності.

Перелік бібліотек (модулів) не обмежується перерахованими. Інформацію про інші функції можна знайти в документації **Python (**Python Documentation contents**)** або в мережі за посиланням https://docs.python.org/3/.

Є можливість створення свого модуля на Python. Ним може бути будь - який файл, у якому визначимо якісь функції.

3.4. Бібліотека math

Так однією з бібліотек, яка містить математичні функції і призначена для роботи з числовими даними, є бібліотека (модуль) **math**. Для роботи з цим модулем його попередньо необхідно імпортувати (підключити), виконавши команду *from math import**. Розглянемо константи та найбільш вживані функції, що містяться в бібліотеці math.

Функція	Призначення
pi	Число π (pi≈3.141592653589793)
e	Число e (e≈2.718281828459045)
tau	Математична константа, рівна 2π.
exp(X)	Обчислює значення e^X
expm1(X)	Обчислює значення e^X-1
fabs(X)	Повертає модуль числа
factorial(X)	Обчислює факторіал числа X
log(X, [b])	Обчислює значення логарифму числа X за основою b. Якщо b не вказано, обчислюється значення натурального логарифму
log10(X)	Обчислює значення логарифму числа X за основою 10.
log2(X)	Обчислює значення логарифму числа X за основою 2
sqrt(X)	Обчислює значення квадратного кореня числа X
degrees(X)	Конвертує радіани в градуси
radians(X)	Конвертує градуси в радіани
cos(X), **sin(X)**, **tan(X)**.	Значення відповідної тригонометричної функції (X вказується в радіанах).
acos(X), asin(X,. atan(X).	Значення оберненої тригонометричної функції в радіанах.
copysign(X, Y)	Повертає число, що має абсолютне значення таке ж, як і у **X**, а знак - як у **Y**.
modf(X)	Повертає дробову і цілу частину числа **X** у вигляді пари чисел (**D, C**). Обидва числа є дійсними і мають той же знак, що і **X.**
hypot(X, Y)	Обчислює значення гіпотенузи трикутника з катетами X та Y

3.5. Короткий запис математичних операцій

Досить часто при виконанні певної математичної операції виникає необхідність присвоїти нове змінній, над якою ця операція проводиться. Тобто необхідно надати змінній нового значення, яке буде залежати від її попереднього значення. Для цього в мові Python використовують, так звані, короткі форми запису виразів.

У Python вираз присвоювання виду «*змінна = змінна операція вираз*» можна записати у вигляді *«змінна операція = вираз»*.

Наприклад вираз ***i = i + 1***, можна записати у вигляді ***i += 1***.

Така коротка форма може бути застосовна тільки до математичних операцій.

Завдання для самостійної роботи.

1. Запишіть код програми яка виконує операції додавання, віднімання, множення, ділення з плаваючою точкою, цілочисельне ділення, знаходить залишок від ділення та піднесення до степеню якщо з клавіатури вводяться цілі числа а результатами є дійсні числа.
2. Написати код програми для розрахунку ідеальної маси чоловіка та жінки за формулою Брокка з використанням вбудованих функцій.

Ідеальна вага для чоловіка = (зріст в сантиметрах - 100) * 1,15.

Ідеальна вага для жінки = (зріст в сантиметрах - 110) * 1,15.

Запитання для самоконтролю

1. Що називається операцією? оператором?
2. В чому полягає різниця між вбудованими і зовнішніми функціями?
3. Як в код програми імпортувати потрібну бібліотеку?
4. Що може бути параметром при імпорті бібліотеки?
5. З якою метою використовується короткий запис математичних операцій

4. ФУНКЦІЇ В PYTHON

4.1. Поняття функції в Python

Дуже часто для написання коду складної програми її розбивають на невеликі фрагменти, які виконують певні завдання, і за своєю структурою є завершеними програмами. Такі фрагменти називаються функціями. Теж досить часто виникає потреба одну й ту ж групу операторів, що виконують якесь закінчене завдання, повторно використовувати в різних місцях програми. Таку групу доцільно об'єднати в один блок і викликати при необхідності виконати передбачене завдання.

Функція – це фрагмент коду, який має своє оригінальне ім'я, відокремлений від інших частин програми, і його можна викликати з будь - якого місця програми, за потреби, безліч разів. Функція може мати будь-яку кількість будь-якого типу вхідних параметрів і повертати будь-яку кількість результатів, а може не мати ні вхідних, ні вихідних параметрів. У одній програмі немає обмежень на використання кількості функцій.

Функції задається за допомогою зарезервованого слова ***def***. Після нього вказується ім'я функції, за яким слідує пара дужок, в яких вказують імена змінних через кому, і після дужок ставиться двокрапка. Ім'я функції – це ідентифікатор, який задається за тими ж правилам, що й імена змінних. У наступних рядках, зі зміщенням вправо слідує блок команд тіла функції. Список формальних змінних вказувати не обов'язково, але круглі дужки опускати не можна.

Синтаксис:

def *ім'я_функції(список параметрів):*

оператори тіла функції

Ім'я функції використовується для її запуску у будь-якому місці коду програми і будь-яку кількість разів. Цей процес називається викликом функції.

Наприклад:

```
def hello():
        print(“Привіт. Ми з України!”) # оператор тіла функції

print ('a=')          # вивід виразу a=
a=input()             # введення значення a з клавіатури
hello()               # виклик функціїї
print ('b=')          # вивід виразу b=
b=input()             # введення значення b з клавіатури
hello()               # виклик функціїї
s=a+b                 # обчислення суми
print ('s=',s)        # вивід значення суми
```

Результат:
a=
5
Привіт. Ми з України!
b=
5
Привіт. Ми з України!
s= 55

4.2. Використання змінних в функції

Функція може приймати фіксовану або нефіксовану кількість параметрів та повертати значення результатів. Параметрами функції є звичайні змінні, які використовуються для своїх внутрішніх розрахунків.

Формальні - це параметри, які вказуються при оголошенні функції. **Фактичні параметри** (аргументи) – параметри, що передаються в функцію ззовні при її виклику. Значення, що передаються в функцію при виклику, ще називають *аргументами.*

Результати роботи функції в основну програму повертаються командою return. У мові Python функції можуть повертати кілька значень одночасно.

Вираз, що стоїть в рядку після команди return, буде повертатися як результат роботи функції до основної програми.

Наприклад. У основній програмі потрібні значення коренів квадратного рівняння, які обчислюються функцією:

```
def kv_r(a, b, c):
    d=b**2-4*a*c
    from math import sqrt
    x1=(-b+sqrt(D))/2*a
    x2=(-b-sqrt(D))/2*a
    return x1, x2
```

Функція може містити кілька команд return, але значення до основної програми передасть лише один з них.

Функція визначає свій власний простір імен змінних. У основній програмі функції можуть існувати змінні з однаковими іменами, але вони будуть різними. Змінна в основній програмі називається глобальною і визначається *глобальною* і відноситься до глобального простору основної програми. До такої змінної можна звернутися з будь якого місця програми, в тому числі і всередині функції. Змінна, що використовується у функції, називається *локальною* і її значення всередині функцій ніяким чином не впливає на значення глобальної змінної з таким же ім’ям. Змінити значення глобальної змінної всередині функції можна за допомогою ключового слова global. Але робити цього не рекомендується, оскільки порушується принцип модульності програми

Завдання для самостійної роботи.

1. Написати код програми, в якій почергово 5 разів запитується введення числа. В разі введення парного числа і нуля виводиться повідомлення «Доброго ранку», а непарного «Доброго вечора». Друк виразів потрібно написати у вигляді функції.

2. Написати код функції, яка обчислює площу трикутника за формулою Герона, і використовується в програмі, яка визначає площу трикутника, якщо трикутник з введеними параметрами можна побудувати.

Запитання для самоконтролю

1. Що називається функцією? З якою метою вони застосовуються?
2. Яким чином оголошується функція в коді програми?
3. В якому місці коду програми може розташовуватися функція?
4. Яким чином тіло функції відділяється від рядка з іменем функції?
5. Чим відрізняються формальні і фактичні параметри функції?
6. Яку команду використовують для повернення значень у основну програму? ЇЇ синтаксис.
7. Чи однакові значення будуть мати змінні з однаковими іменами в основній програмі і функції?

5 СТВОРЕННЯ ВІДЖЕТІВ ЗАСОБАМИ PYTHON

5.1. Бібліотека tkinter

Бібліотека tkinter – це бібліотека графічних елементів та створення віджетів. Віджет - елемент інтерфейсу, призначений для простих операцій, як віконце, калькулятор тощо; поняття віджетів детальніше таких, розглядається нижче або будь яка інша.

Прикладом завантаження бібліотеки графічних елементів та створення віджетів у повному вигляді є команда:

*from tkinter import **

Віджет (від англ. Widget можна перекласти приблизно як «штучка», «дрібничка». По суті, це невеликі компоненти, які розширюють функціональність і наочність та призначені для виконання простої дії. Модуль tkinter в Python має в своїй бібліотеці декілька таких елементів.

Python може працювати з багатьма бібліотеками для створення графічних об’єктів, але тільки tkinter вбудований у бібліотеку мови. Усі інші GUI фреймворки (бібліотеки для графічних інтерфейсів) потрібно встановлювати додатково.

Однією зі значних переваг бібліотеки tkinter є її кросплатформеність. А це означає, що написаний код буде працювати в будь-якій операційній системі, і при цьому усі об’єкти будуть виглядати у відповідності до правил системи.

Процес створення віджетів умовно можна розділити на 3 етапи.

На першому етапі створюється головне вікно. Для цього потрібно зайти в бібліотеку геометрії Tk(). Клас Tk() є обов’язковим для будь-якого tkinter віджету, він створює макет головного вікна.

Функцію *title ('Назва головного вікна')* можна використовувати для задання назви головному вікну. Функція *title* не є обов’язковою.

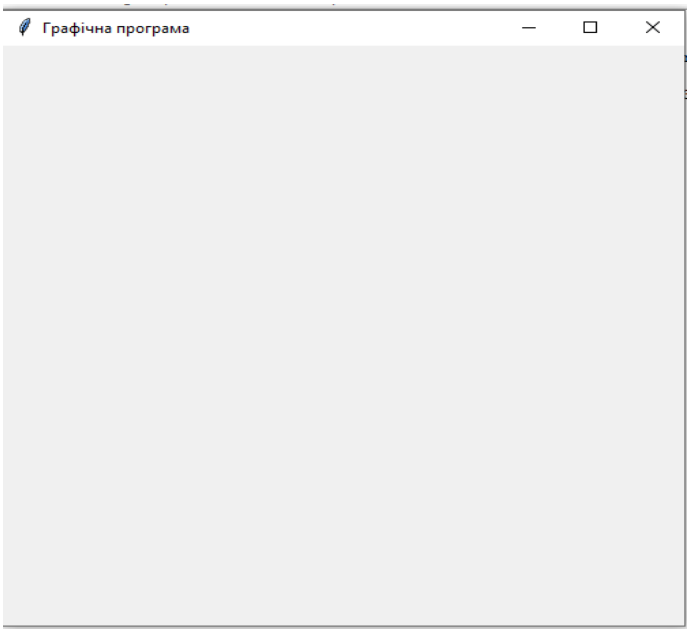
Початкові розміри головного вікна можна задати функцією *geometry ('ширина* x *висота').* Функція *geometry також* не є обов’язковою. За її

відсутності вікно створюється такого розміру, щоб у нього поміщалися усі графічні елементи.

Приклад коду для створення головного вікна

```
from tkinter import *       # імпортування графічної бібліотеки
window = Tk()               # створення головного вікна
window.title ('Графічна програма')  # створення назви головного вікна
window.geometry('500x500')          # розміри головного вікна
```

Результат



Завдання для самостійної роботи.

1. Напишіть код програми, яка створює вікно з вашим прізвищем. Розміри якого задаються з клавіатури.

Запитання для самоконтролю

1. Як імпортувати в код програми бібліотеку графічних елементів?
2. Яку функцію виконує клас Tk()?
3. Як задати початкові розміри вікна?
4. Як задати назву вікна?

5.2. Поняття пакувальники

Другий етап створення віджетів з графічної бібліотеки tkinter потребує вирішення питання їх розташування у головному вікні. Пакувальники (менеджери розташування) дозволяють це зробити. Питання важливе тому, що від нього залежить зручність розташування елементів і естетичний вигляд головного вікна.

У **tkinter** використовується три менеджери розташування: *pack, grid* та *place*. В одному вікні допускається використання тільки одного пакувальника

5.2.1. Пакувальник pack

Пакувальник ***pack()*** **–** використовують для розміщення віджетів один за одним. Його можна використовувати без параметрів, але дужки опускати не можна. За замовчуванням графічні елементи розташовуються зверху до низу по центру вікна. Властивості ***side, fill, anchor,*** в залежності від заданих значень, надають можливість вказати до якої сторони основного графічного елемента має примикати створюваний віджет. Пакувальник ***pack()*** може вести себе непередбачувано при використанні на різних платформах.

Синтаксис:

Ім'я віджету.pack (параметр = 'значення')

Приклад:

button1.pack (side = 'left')

Параметри пакувальника *pack()*

Параментр	Значення
side	• TOP(по замовчуванню) – розміщує віджет зверху • BOTTOM – розміщує віджет знизу • LEFT – розміщує віджет ліворуч • RIGHT – розміщує віджкт праворуч
fill	Заставляє віджет заповнювати весь доступний простір у вказаному напрямі по одній із осей: • X – заповнює простір по горизонталі • Y - заповнює простір по вертикалі
anchor	Якір. Може приймати такі значення: • N – north (північ) • S – south (південь) • W – west (захід) • E – east (схід) Значення можна комбінувати: SE, NW…

5.2.2. Пакувальник *place*

Менеджер розташування ***place()*** розташовує віджет в фіксованому місці з фіксованим розміром. При використанні цього пакувальника необхідно вказувати параметри кожного елементу. Він може здатися незручним, але надає повну свободу при розміщенні віджетів у вікні. Методом ***place()*** віджету вказується його положення або в абсолютних значеннях (в пікселях), або в частках батьківського вікна. Отже абсолютно і відносно можна задавати і розмір самого віджета.

Синтаксис

Ім'я віджету. place (параметр = значення)

Приклади:

Button.place(x=75, y=20) **#** в абсолютних значеннях

Button.place(relx=0.3, rely=0.5) # в частках батьківського вікна

Параметри пакувальника *place()*

Параментр	Значення
anchor	Якір. Визначає частину віджета. Для якої задаються координати. Приймає значення N, NE, E, SE, SW, W, NW, або CENTER. За замовчуванням NW (верхній лівий кут)
relwidt0h/ relhesght	Відносні ширина і висота віджета. Визначають розмір віджета відносно батьківського віджета
relx / rely	Визначають відносну позицію в батьківському віджеті. Координата (0;0) – у лівого вкрхнього кута (1;1) – у правого нижнього
width / height	Абсолютні ширина і висота віджета
x/y	Абсолютні координати (в пікселях) розміщення віджета

5.2.3. Пакувальник *grid*

«Grid» в перекладі з англійської – «сітка». Менеджер розташування ***grid()*** **–** розміщує віджети в двомірній сітці (таблиці), наприклад для створення сітки кнопок для калькулятора. Вікно розділяється на рядки та стовпчики і кожна комірка в отриманій таблиці може містити віджет. Адреса кожної комірки складається з номера рядка і номера стовпця. Нумерація починається з нуля. Комірки можна об'єднувати по вертикалі і по горизонталі.

Як приклад наведено таблицю, в якій номера рядків виділено зеленим кольором, а стовпчиків синім.

0,0	0,1	0,2	0,3
1,0	1,1	1,2	1,3
2,0	2,1	2,2	2,3
3,0	3,1	3,2	3,3

Розміщення віджета в тій чи іншій клітинці задається через аргументи ***row***(рядок) і ***column*** (стовпець), якими задається відповідний номер рядка і стовпчика. Для об'єднаня комірки по горизонталі, використовується параметр ***columnspan()***, якому задається кількість комірок. Аналогічно параметр ***rowspan()*** об’єднує комірки по вертикалі.

Синтаксис:

Ім’я віджета.grid(row=номер рядка, column=номер стовпця)

Приклад

Button.grid(row = 1, column = 0)

Менеджер grid є найбільш гнучким менеджером розташування в tkinter.

Параметри пакувальника *grid*

Параметр	Значення
row	номер рядка, в який поміщаємо віджет
column	номер стовпчика, в який поміщаємо віджет
columnspan	Скільки стовпців займає віджет
rowspan	Скільки рядків займає віджет
padx / pady	Розмір зовнішньої межі по оризонталі і вертикалі
ipadx / ipady	Розмір внутрішньої межі по горизонталі і вертикалі (різниця між pad і ipad в тому, що при вказівці pad розширюється вільний простір, а при ipad розширюється віджет)
sticky	вказує до якої межі «приклеювати» віджет. Може приймати такі значення: • “n” – north (північ) • “s” – south (південь) • “w” – west (зіхід) • “e” – east (схід)
in_	Явна вказівка, в якій батьківський віджет потрібно помістити

Більш наглядно особливості розміщення віджетів за допомогою пакувальника grid, можна переглянути у відео: https://www.youtube.com/watch?v=_7F6FsbJepo

Запитання для самоконтролю

1. З якою метою використовуються пакувальники?
2. Функції пакувальника pack.
3. Чи можна одночасно для пакувальника pack використовувати параметри side, fill, anchor?
4. Функції пакувальника place.
5. Які параметри потрібно вказати для пакувальника place, щоб задати абсолютні значення ширина і висота віджета?
6. Функції пакувальника grid.
7. Які основні параметри потрібно задати, щоб розмістити відмет у вікні за допомогою пакувальника grid?
8. Який з пакувальників найзручніший для використання?

5.3. Віджет Canvas

Третій етап побудови віджетів пов'язаний зі створенням в головному вікні необхідних графічних елементів.

Одним з найпоширеніших віджетів при роботі з модулем *tkinter* є полотно – це частина батьківського вікна, у якій можна працювати з графічними об'єктами. Задається полотно функцією **Canvas ().** В дужках вказується ім'я вікна в якому створюється полотно і параметри.

Синтаксис створення полотна:

змінна=Canvas(ім'я_вікна, параметри)

Наприклад командою:

c=Canvas(root, width=500, height=500) – створюється полотно 500x500 пікселів. Можуть бути задані і інші параметри, або не задано жодних.

Параметри віджета Canvas

Параметр	Значення
width	Ширина полотна
height	Висота полотна
bd	Ширина межі полотна
highligtcolor	Колір межі

Менеджер геометрії бібліотеки **змінна = Tk()** розміщує віджети за координатами (за початок відліку береться лівий верхній кут, вісь Оу направлена донизу). Менеджер геометрії бібліотеки використовується в коді програми перед згадуванням функції **Canvas ().** Наприклад:

*from tkinter import * # імпортування графічної бібліотеки*

window = Tk() # створення головного вікна

window.title ('Графічна програма') # створення назви головного вікна

window.geometry("500x500") # розміри вікна

g = Canvas() #створення полотна

Віджети можна розташовувати на полотні за одним із методів ***pack()***, ***place(), grid(),*** який вказується після оголошення полотна.

Методи ***pack()*** і ***place()*** є більш простими для застосування. При створенні не складних об’єктів в Python різниця між ними несуттєва, тому можна використовувати будь-який.

Таблиця функцій для малювання примітивів

Функція	Значення
create_line(x1, y1, x2, y2)	Лінія з початком в точці (x_1, y_1) та кінцем (x_2, y_2)
create_rectangle(x1, y1, x2, y2)	Прямокутник з діагоналлю (x_1, y_1)та (x_2, y_2)
create_oval(x1, y1, x2, y2)	Овал в прямокутнику з діагоналлю (x_1, y_1) та (x_2, y_2)
create_polygon(x1, y1,…. xn, yn)	Ламана по заданих точках
create_text(x1, y1, x2, y2, text="текст!", justify=CENTER, font="Verdana 14")	Вписує текст в прямокутнику
create_arc(x1, y1, x2, y2, параметри)	Малює елементи круга (сектор, сегмент, дуга) в залежності від парамерів
delete(“all”)	Очистити полотно

Слід зауважити, що, крім координат, усі перераховані функції можуть мати додаткові параметри, що оголошуються в дужках після коми.

fill='red' – для відрізка колір лінії, для замкненої фігури колір заливки;

width=5 – товщина лінії;

dash=(10, 2) – для малювання лінії штрихами (довжина пунктиру, довжина пропуску);

outline='lightgreen' – колір лінії для фігури;

Ім'я полотна.create_arc(10, 10, 190, 190, start=190, extent=40, fill='blue') – зображення сектору;

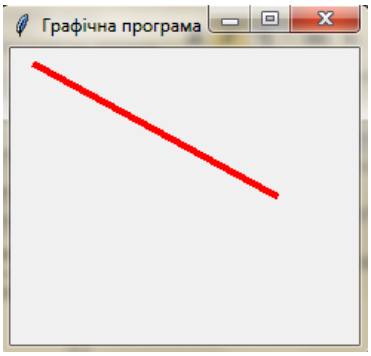
Ім'я полотна.create_arc(10, 10, 190, 190, start=0, extent=60, style=ARC, outline='darkgreen', width=5) – зображення дуги;

Ім'я полотна.create_arc(10, 10, 190, 190, start=240, extent=100, style=CHORD, fill='yellow') – зображення сегменту.

Приклад коду для створення полотна з графічним об'єктом:

```
from tkinter import *              # імпортування графічної бібліотеки
window = Tk()                      # створення головного вікна
window.title ('Графічна програма')          #створення назви головного вікна
g = Canvas()                 #створення полотна
g.place(x=0,y=0)          # розміщення полотна на екрані за допомогою
     пакувальника place
g.create_line(15, 10, 180, 100,          # створення лінії
    fill='red',                          # червоного кольору
    width=5)                             # товщиною 5
```

і результат.



Завдання для самостійної роботи.

1. Створіть вікно заданих з клавіатури розмірів. На полотні зобразіть усі геометричні примітиви і малюнок тексту, який складається з вашого імені. Малюнки зробіть двома методами ***pack()*** і ***place()***. Опишіть як змінюються малюнки в кожному із методів при збільшенні розмірів полотна.

Запитання для самоконтролю

1. Якою функцією створюється полотно для малювання геометричних примітивів?
2. Синтаксис функції для створення полотна.
3. Які основні параметри указуються для малювання прямокутника?
4. Як малюється овал?
5. Чи можна намалювати многокутник?
6. Які основні параметри потрібно задати, щоб створити малюнок тексту?

5.4. Віджет Label (Мітка)

Label() – клас мітки. Використовується для відображення будь-якого тексту у вікні і виконує інформування роль (повідомлення, підпис елементів інтерфейсу тощо).

Синтаксис:

ім'я мітки = Label(ім'я вікна, параметри)

Змінити властивість мітки або будь-якого іншого віджета можна одним із способів:

1. *Ім'я_віджета["ім'я_властивості"] = значення;*
2. *Ім'я_віджета.config(ім'я_властивості = значення).*

Параметри віджета Label

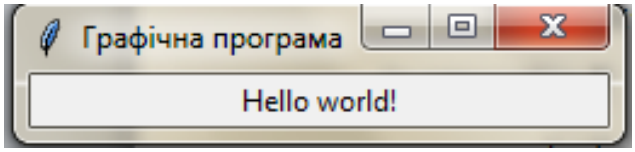
Параметр	Значення
width	Ширина мітки
height	Висота мітки
text	Текст на мітці
bg	Фон мітки

fg	Колір тексту
bd	Ширина межі мітки
activebackground	Колір фону (коли мітка натиснута)
activeforeground	Колір тексту (коли мінтка натиснута)
disabledbackground	Колір фону (коли властивість state == DISABLED)
disabledforeground	Колір тексту (коли властивість state == DISABLED)
state	Стан мітки (NORMAL? DISABLED)
compount	Розташування картинки на місці (CENTER, BOTTOM, LEFT, RIGHT, TOP)
justify	Вирівнювання тексту
relief	Рельєф мітки (FLAT, GROOVE, RIDGE, SUNKEN, RAISE)
overrelief	Рельєф мітки, коли над нею знаходиться курсор (FLAT, GROOVE, RIDGE, SUNKEN, RAISE)
image	Ім'я картинки, яку буде відображено на мітці
font	Вид шрифту на мітці
textvariable	Ім'я змінної, в якій буде зберігатися текст на мітці

Приклад. Створити мітку з текстом «Hello world!»

```
from tkinter import *   #імпорт графічної бібліотеки
window=Tk()             #створення головного вікна
window.title ('Графічна програма')  #створення назви головного вікна
widget = Label(window, text='Hello world!')     #створення мітки з іменем
        widget, яка виведе на екран текст
widget.pack()    # розміщення мітки на екрані за допомогою пакувальника
        pack
```

Результат



Завдання для самостійної роботи.

1. Напишіть код для створення рамок за зразком:

Запитання для самоконтролю

1. З якою метою використовуються мітки в коді програми?
2. Синтаксис мітки.
3. Основні параметри для мітки і їх значення.
4. Яким способом можна змінити параметри мітки?

5.5. Віджет Frame (Рамка)

Frame – клас фрейму (рамки). Віджет використовується для створення рамок всередині вікна.

Синтаксис:

Ім'я рамки = Frame(ім'я вікна)

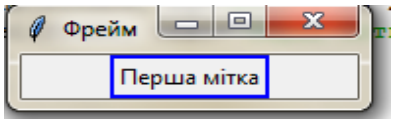
Параметри віджета Frame

Параметр	Значення
width	Ширина фрейма
height	Висота фрейма
bg	Фон фрейма
bd	Товщина межі фрейма
state	Стан фрейма (NORMAL, DISABLED)
relief	Рельєф межі фрейма (FLAT, GROOVE, RIDGE, SUNKEN, RAISE
highlightthickness	Ширина другої межі

Приклад: Створити у вікні рамку з текстом.

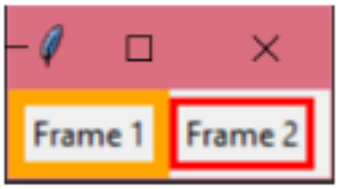
```
from tkinter import *              # імпортування графічної бібліотеки
window = Tk()                      # створення головного вікна
window.title ('Фрейм')             #створення назви головного вікна
frame = Frame(window, bg='blue', bd=2)  # створення рамки
label = Label(frame, text='Перша мітка')   # створення мітки (вказується рамка
        frame)
frame.pack()                       # розміщення у вікні рамки
label.pack()                       # розміщення у вікні рамки
```

Результат



Завдання для самостійної роботи.

1. Напишіть код для створення рамок за зразком:



Запитання для самоконтролю

1. З якою метою використовуються рамки?
2. Синтаксис рамки.
3. Основні параметри для рамки та їх значення.

5.6. Віджет Button (Кнопка)

Button –кнопка.

Синтаксис:

Ім’я кнопки = Button(ім’я вікна)

Параметри віджета Button

Параметр	Значення
width	Ширина кнопки • За замовчуванням дорівнює такій кількості пікселів, щоб текст містився в кнопці впритул до її меж
height	Висота кнопки • За замовчуванням дорівнює такій кількості пікселів, щоб текст містився в кнопці притул до її меж
text	Текст на кнопці • За замовчуванням текст буде відображатися по центру кнопки • Можливо зробити багатостроковий текст, використовуючи \ n
bg	Фон який матиме кнопка в той час, коли на неї **НЕ** натиснули
fg	Колір тексту, який буде мати кнопка в той час, коли на неї **НЕ** натиснули
bd	Ширина меж кнопки,поки на неї **НЕ** натиснули
activebackground	Колір фону (коли кнопка натиснута)
activeforeground	Колір тексту (коли кнопка натиснута)
disabledbackground	Колір фону (коли властивість state == DISABLED)
disabledforeground	Колір тексту (коли властивість state == DISABLED)
state	Стан (NORMAL, DISABLED) • NORMAL – звичайний стан кнопки. При якому на неї можна натискати, кнопка може взаємодіяти з користувачем • DISABLED – такий стан кнопки, при якому вона НЕ може взаємодіяти з користувачем

compount	Розташування картинки на кнопці (CENTER, BOTTOM, LEFT, RIGHT, TOP) • За замовчуванням картинка на кнопці буде відображатися замість тексту, але це можна змінити, змінюючи значення властивості compound • BOTTOM – картинка буде відображатися під текстом • LEFT – картинка буде відображатися ліворуч від тексту • RIGHT – картинка буде відображатися праворуч від тексту • TOP – картинка буде відображатися над текстом
justify	Вирівнювання тексу (CENTER, RIGHT, LEFT) • Спочатку текст буде відображення з вирівнюванням по лівому краю, але це можна змінити, використовуючи властивість justify • CENTER – текст вирівнюється на кнопці по центру • LEFT – текст вирівнюється на кнопці по лівому краю • RIGHT – текст вирівнюється на кнопці по правому краю
relief	Рельєф кнопки (FLAT, GROOVE, RIDGE, SUNKEN, RAISE)
overrelief	Рельєф кнопки коли на нею знаходиться курсор (FLAT, GROOVE, RIDGE, SUNKEN, RAISE)
image	Ім'я яку буде відображено на кнопці
font	Вид шрифту на кнопці

Встановлення дії на кнопку

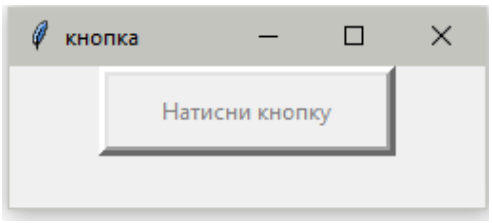
Щоб встановити дію, яка буде відбуватися після натиснення кнопки, можна скористатися властивістю **command**. Цей спосіб дозволяє додати дію, що оголошується за допомогою функції створеної користувачем.

Приклад: Потрібно на кнопку button встановити дію, що змінює текст на кнопці і її колір.

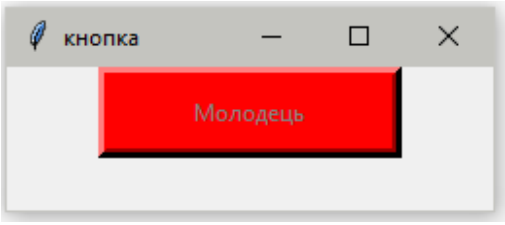
```
from tkinter import *   #імпорт графічної бібліотеки
window=Tk()             #створення головного вікна
window.title ('кнопка')  #створення назви головного вікна
butt = Button(text="Натисни кнопку",        # задання тексту на кнопці
              width=20,                     # ширина
              height=2,                     # висота
              bd=5,                         # ширина межі кнопки
              fg='gray')      # колір ненатиснутої кнопки

def press():                          # створення функції дії для кнопки
    butt['text'] = "Молодець"    # новий текст на кнопці
```

```
butt['bg'] = 'red'          #новий колір на кнопці

butt.config(command=press)        # установлює дію на кнопку
butt.pack()                       # розміщує кнопку у вікні
```

І результат



До натиснення



Після натиснення

Завдання для самостійної роботи.

1. Напишіть код програми, щоб після натиснення на кнопку зеленого кольору з надписом «Press» нижче з'явилася мітка «Моє ім’я (ваше ім’я)».

Запитання для самоконтролю

1. З якою метою використовуються кнопки?
2. Синтаксис кнопки.
3. Основні параметри кнопки і їх значення.
4. Як встановити дію на кнопку?

5.7. Віджет Entry (Однорядкове текстове поле)

Entry – клас однорядкового текстового поля.

Синтаксис:

ім'я текстового поля = Entry(ім'я вікна).

Параметри віджета Entry

Параметр	Значення
width	Ширина поля
bg	Фон поля
fg	Колір тексту поля
bd	Ширина межі поля

activebackground	Колір фону (коли в поле набирають текст)
activeforeground	Колір тексту (коли в поле набирають текст)
disabledbackground	Колір фону (коли властивість state == DISABLED)
disabledforeground	Колір тексту (коли властивість state == DISABLED)
state	Стан поля (NORMAL, DISABLED)
justify	Вирівнювання тексту
higlightcolor	колір другої межі(коли поле має фокус)
higlightbackground	Колір другої межі (коли поле не має фокус)
higlightthickness	ширина другої межі
relief	Рельєф текстового поля (FLAT, GROOVE, RIDGE, SUNKEN, RAISE)
overrelief	Рельєф текстового поля, коли над ним знаходиться курсор (FLAT, GROOVE, RIDGE, SUNKEN, RAISE)
font	Вид шрифту на в однорядному текстовому полі
textvariable	Ім'я змінної, в якій зберігається текст, що знаходиться в полі
selectbackground	Колір фону виділеного фрагментами тексту
selectforeround	Колір тексту виділеного фрагмента тексту
insertontime	Час, який курсор видно
insertofftime	Час, який куроср не видно

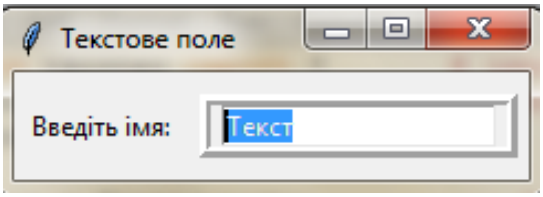
Приклад: Написати код для створення однорядкового текстового поле з міткою зліва.

```
from tkinter import *       # імпорт графічної бібліотеки
window = Tk()               # створення головного вікна
window.title ('Текстове поле')      #створення назви головного вікна

frame = Frame(window,         # створення рамки для мітки та текстового поля
        bd=5,relief=RIDGE )   # ширина та рельєф межі рамки

label = Label(window, text='Введіть імя:')  # створення мітки з текстом
label.pack(side=LEFT, padx=5)               # розміщення мітки з текстом
t = StringVar()               # створення рядкової змінної для зберігання тексту

entry = Entry(frame,          # створення текстового поля
          textvariable=t,   # прив'язка до Entry змінної, у якій зберігається текст,
            що знаходиться у текстовому полі
          bg='white')       # колір фону текстового поля
entry.pack(side=RIGHT, padx=5)  # розміщення текстового поля
```

t.set('Текст') # встановлення початкового тексту у змінну текстового поля

frame.pack(expand=1, fill=X, pady=10, padx=5) # розміщення рамки текстового поля

Результат



Завдання для самостійної роботи.

1. Написати код програми для створення віджета за зразком.

Запитання для самоконтролю

1. З якою метою створюються однорядкові текстові поля?
2. Синтаксис однорядкового текстового поля.
3. Основні параметри однорядкового текстового поля та їх значення.
4. Чи можна змінювати текст у однорядковому текстовому полі?

5.8. Віджет Text (Багаторядкове текстове поле)

Text – клас багаторядкового текстового поля.

Синтаксис:

ім'я текстового поля = Text(ім'я вікна).

Властивості багаторядкового текстового поля ідентичні з властивостями однорядкового текстового поля, за виключенням **textvariable** (для багаторядкового текстового поля – відсутня)

5.9. Віджет Scrollbar (Смуга прокручування)

Scrollbar – клас смуги прокручування (скроллер). Використовується для переміщення по тексту, що не вміщується у відведеному для нього місці екрану.

Синтаксис:

ім'я смуги прокручування = Scrollbar(ім'я вікна)

Параметри віджета Scrollbar

Параметр	Значення
command	Прив’язує смуту прокручування до віджета Доступні такі значення: ім’я_віджета.**xview** – віджет буде прокручуватися горизонтально ім’я_віджета.**yview** – віджет буде прокручуватися вертикально Для можливості прокрутки віджета також потібно змінити прокручуваний віджет: ім’я_віджета.**config(Nscrollcommand = ім’я_скроллера.set),** де N – або x , або y
bg	Фон смути прокручування
relief	Рельєф смути прокручування (FLAT, GROOVE, RIDGE, SUNKEN, RAISE)

Для створення смуги прокручування потрібно виконати три кроки:

1) створити віджет багаторядкового текстового поля;
2) створити смугу прокручування і зв’язати її з текстовим віджетом;
3) зв’язати текстовий віджет з панеллю прокрутки.

В реалізації цього алгоритму доцільно використовувати функцію **command**.

Приклад:

```
textWidget = Text(root)                          # створення віджета
scrollbar = Scrollbar(root, command=textWidget.yview)      # створення смуги
```

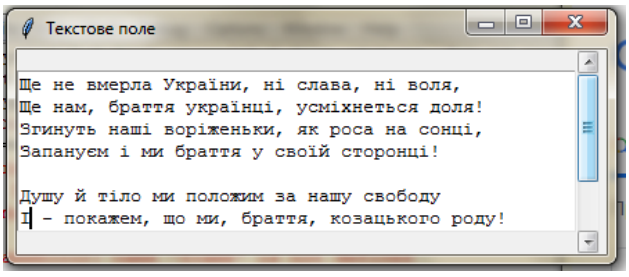
прокручування і зв'язку з віджетом textWidget

textWidget.config(yscrollcommand=scrollbar.set) # зв'язування віджета з панеллю прокручування

Приклад: Створити просте багаторядкове текстове поле будь - якого світлого кольору зі смугою прокручування та задати чорний колір для введеного тексту. (Текст вводити після інтерпретації коду.)

```
from tkinter import *      # імпорт графічної бібліотеки
window = Tk()              # створення головного вікна
window.title ('Текстове поле')     #створення назви головного вікна

t = Text(                  # створення багаторядкового текстового поля
    width=50,              # ширина текстового поля
    height=7,              # висота текстового поля
    bg='white',
     fg='black')           # ширина та рельєф меж рамки
t.pack(side=LEFT)          # розміщення текстового поля

scroll = Scrollbar(command=t.yview) # створення смуги прокручування
scroll.pack(side=LEFT, fill=Y)      #створення смуги прокручування, параметр fill вимагає заповнення всього доступного простору віджета
t.config(yscrollcommand=scroll.set)  # встановлення можливості прокрутки тексту
```

Результат



Завдання для самостійної роботи.

1. Написати код програми для створення багаторядкового текстового поля з смугами прокрутки по горизонталі і вертикалі.

Запитання для самоконтролю

1. З якою метою створюється багаторядкове текстове поле?
2. Синтаксис багаторядкового текстового поля.
3. Основні параметри багаторядкового текстового поля та їх значення.
4. Чи можна замінити однорядкове текстове поле багаторядковим текстовим полем?
5. Синтаксис смуги прокручування.
6. Основні параметри скролера та їх значення.
7. Чи можна створити багаторядкове текстове поле без скролера?

5.10. Віджет Radiobutton (Перемикач)

Radiobutton – клас перемикачів (радіокнопок).

Синтаксис:

Ім'я перемикача = Radiobutton(ім'я вікна)

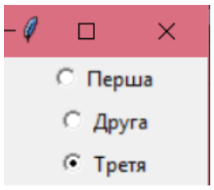
Параметри віджета Radiobutton

Параметр	Значення
width	Ширина радіокнопки
height	Висота радіокнопки
text	Текст радіокнопки
bg	Фон радіокнопки
fg	Колір тексту
bd	Ширина межі радіокнопки
activebackground	Колір фону (коли радіокнопка натиснута)
activeforeground	Колір тексту (коли радіокнопка натиснута)
disabledbackground	Колір фону (коли властивість state == DISABLED)
disabledforeground	Колір тексту (коли властивість state == DISABLED)
state	Стан радіо кнопки (NORMAL, DISABLED)
compount	Розташування картинки на радіокнопці (CENTER, BOTTOM, LEFT, RIGHT, TOP)
justify	Вирівнювання тексту
relief	Рельєф радіокнопки (FLAT, GROOVE, RIDGE, SUNKEN, RAISE)
image	Ім’я картинки, яка буде відображатись на радіокнопці
selectimage	Ім’я картинки, яка буде відображатись на радіокнопці (коли вона обрана)
font	Вид шрифту на радіокнопці
indicatoron	Стиль відображатися радіокнопки (якщо true, то буде показуватися кружочек поруч з радіокнопкою, інакше немає
value	Значення, яке буде присвоюватися змінній, зазначеної в параметрі variable при виборі радіокнопки
variable	Ім’я змінної, у якій буде змінюватися значення на зазначене у властивості value при виборі радіокнопки

За замовчуванням перемикачі в одному вікні не пов’язані між собою і можуть бути увімкнені одночасно. Зв’язок доцільно встановити через спільну змінну, різні значення якої будуть відповідати увімкненню різних перемикачів із однієї групи. Для всіх перемикачів цієї групи встановлюється одна і та ж змінна для властивості **variable.** А властивості **value** присвоюються різні значення цієї змінної.

Приклад:

```
var = IntVar()          # створюємо змінну, яка приймає цілі значення
var.set(1)              # встановлюємо перше значення для створеної змінної

rad1 = Radiobutton(root, text="Перша", variable=var, value=1)     # створюємо
                        перемикач зі значенням 1
rad2 = Radiobutton(root, text="Друга", variable=var, value=2)     # створюємо
                        перемикач зі значенням 2
rad3 = Radiobutton(root, text="Третя", variable=var, value=3)     # створюємо
                        перемикач зі значенням 3
# розміщуємо перемикачі у вікні
rad1.pack()
rad2.pack()
rad3.pack()
```

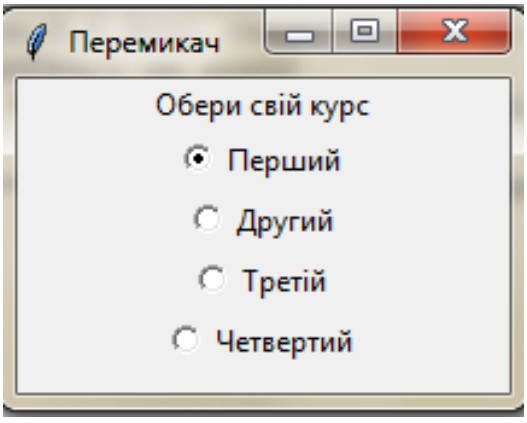


Приклад: Створити вікно з перемикачами для вибору курсу, на якому студент навчається.

```
from tkinter import *       # імпортування графічної бібліотеки
window = Tk()               # створення головного вікна
window.title ('Перемикач')  # створення назви головного вікна

g1 = Label(window, text='Обери свій курс')  # створення мітки з текстом
```

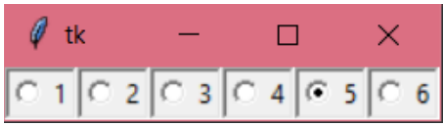
```
g1.pack()                  # розміщення мітки з текстом
var = IntVar()             # створюємо змінну, яка приймає цілі значення
var.set(1)                 # встановлюємо перше значення для створеної змінної
rad1 = Radiobutton(window, text="Перший", variable=var, value=1)      # створюємо перемикач зі значенням 1
rad2 = Radiobutton(window, text="Другий", variable=var, value=2)      # створюємо перемикач зі значенням 2
rad3 = Radiobutton(window, text="Третій", variable=var, value=3)      # створюємо перемикач зі значенням 3
rad4 = Radiobutton(window, text="Четвертий", variable=var, value=4)   # створюємо перемикач зі значенням 4

rad1.pack()                # розміщуємо перемикачі у вікні
rad2.pack()
rad3.pack()
rad4.pack()
```

Результат



Завдання для самостійної роботи.

1. Створити групу перемикачів за зразком



Запитання для самоконтролю

1. Яке призначення мають перемикачі?
2. Синтаксис перемикача.
3. Основні параметри перемикача та їх значення.
4. Яким чином зробити зв’язок між перемикачами?

5.11. Віджет Checkbutton (Вибір кількох елементів)

Checkbutton – клас поле прапорців (вибір кількох елементів). Цей віджет потрібен користувачам для вибору кількох елементів у вікні. Прапорець має два стани: увімкнене або вимкнено.

Синтаксис:

ім'я прапорця = Checkbutton(ім'я вікна)

Параметри віджета Checkbutton

Параметр	Значення
width	Ширина прапорця
height	Висота прапорця
text	Текст біля прапорця
bg	Фон прапорця
fg	Колір тексту прапорця
bd	Ширина межі прапорця
activebackground	Колір фону (коли прапорця натиснута)
activeforeground	Колір тексту (коли прапорець натиснута)
disabledbackground	Колір фону (коли властивість state == DISABLED)
disabledforeground	Колір тексту (коли властивість state == DISABLED)
state	Стан прапорця (NORMAL, DISABLED)
compount	Розташування картинки на прапорці (CENTER, BOTTOM, LEFT, RIGHT, TOP)
justify	Вирівнювання тексту
relief	Рельєф прапорця(FLAT, GROOVE, RIDGE, SUNKEN, RAISE)
overrelief	Ральєф прапорця коли над нею знаходиться курсор(FLAT, GROOVE, RIDGE, SUNKEN, RAISE)
image	Ім’я картинки, яка буде відображатись на прапорці
selectimage	Ім’я картинки, яка буде відображатись на прапорці (коли вона обрана)
font	Вид шрифту на прапорці
indicatoron	Стиль відображатися прапорці (якщо true, то буде показуватися кружочок поруч з прапорцем, інакше немає
value	Значення, яке буде присвоюватися змінній, зазначеної в параметрі variable при виборі прапорця
variable	Ім’я змінної, у якій буде змінюватися значення на зазначене у властивості value при виборі прапорця

Прапорці не пов’язуються між собою, але змінні теж потрібні для отримання відомостей про стан прапорців. За значенням пов’язаної з Checkbutton змінної визначають: чи відмічений прапорець, що в подальшому впливає на хід виконання програми. Кожному прапорцеві відповідає своя змінна.

За допомогою опції **onvalue** встановлюється значення, яке приймає пов’язана змінна при увімкненому прапорці. За допомогою властивості **offvalue** – при вимкненому.

Функції віджета Checkbutton

Функція	Значення
ім'я_прапорця.select()	Вмикакає прапорець
ім'я_прапорця.deselect()	Вимикає прапорець

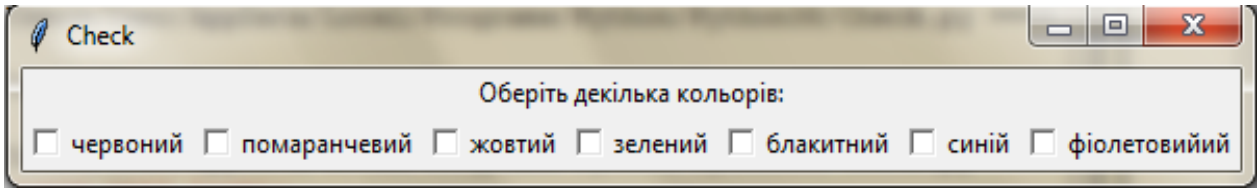
Приклад: У вікні розташувати прапорці для вибору улюблених кольорів.

```
from tkinter import *       # імпорт графічної бібліотеки
window = Tk()               # створення головного вікна
window.title (' Check')     #створення назви головного вікна

t = Label(window, text="Оберіть декілька кольорів:")     # створення текстової
                    мітки
t.pack()          # розміщення текстової мітки

var1 = IntVar()           # створення змінної цілого типу для першого прапорця
check1 = Checkbutton(window,  # створення першого прапорця
        text="червоний",  # текст прапорця
        variable=var1,    # значення змінної першого прапорця
        onvalue=1,        # значення при включеному прапорці
        offvalue=0)       # значення при вимкненому прапорці
check1.pack(side=LEFT)              # створення першого прапорця
```

```
var2 = IntVar()          # створення змінної цілого типу для другого прапорця
check2 = Checkbutton(window,  # створення другого прапорця
        text="помаранчевий",  # текст прапорця
        variable=var2,    # значення змінної другого прапорця
        onvalue=1,        # значення при увімкненому прапорці
        offvalue=0)       # значення при вимкненому прапорці
check2.pack(side=LEFT)              # створення другого прапорця

var3 = IntVar()          # створення змінної цілого типу для третього прапорця
check3 = Checkbutton(window,  # створення третього прапорця
        text="жовтий",    # текст прапорця
        variable=var3,    # значення змінної третього прапорця
        onvalue=1,        # значення при увімкненому прапорці
        offvalue=0)       # значення при вимкненому прапорці
check3.pack(side=LEFT)              # створення третього прапорця

var4 = IntVar()          # створення змінної цілого типу для четвертого прапорця
check4 = Checkbutton(window,  # створення четвертого прапорця
        text="зелений",   # текст прапорця
        variable=var4,    # значення змінної четвертого прапорця
        onvalue=1,        # значення при увімкненому прапорці
        offvalue=0)       # значення при вимкненому прапорці
check4.pack(side=LEFT)              # створення четвертого прапорця

var5 = IntVar()          # створення змінної цілого типу для п'ятого прапорця
check5 = Checkbutton(window,  # створення п'ятого прапорця
        text="блакитний", # текст прапорця
        variable=var5,    # значення змінної п'ятого прапорця
        onvalue=1,        # значення при включеному прапорці
```

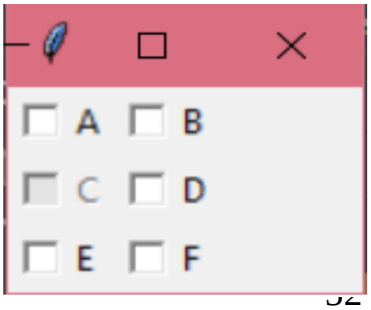
```
        offvalue=0)      # значення при вимкненому прапорці
check5.pack(side=LEFT)              # створення п'ятого прапорця

var6 = IntVar()          # створення змінної цілого типу для шостого прапорця
check6 = Checkbutton(window,  # створення шостого прапорця
        text="синій",     # текст прапорця
        variable=var6,    # значення змінної третього прапорця
        onvalue=1,        # значення при увімкненому прапорці
        offvalue=0)       # значення при вимкненому прапорці
check6.pack(side=LEFT)              # створення шостого прапорця

var7 = IntVar()          # створення змінної цілого типу для сьомого прапорця
check7 = Checkbutton(window,  # створення сьомого прапорця
        text="фіолетовийий",    # текст прапорця
        variable=var7,    # значення змінної сьомого прапорця
        onvalue=1,        # значення при увімкненому прапорці
        offvalue=0)       # значення при вимкненому прапорці
check7.pack(side=LEFT)              # створення сьомого прапорця
```

Результат



Завдання для самостійної роботи.

1. Створити групу прапорців за зразком



Запитання для самоконтролю

1. З якою метою створюється поле прапорців?
2. Синтаксис поля прапорців.
3. Основні параметри поля прапорців та їх значення.
4. Яким чином визначити, чи поставлено прапорець в полі?
5. Чи можна проставити прапорці «за замовчуванням»?

5.12. Віджет Scale (Шкала з повзунком)

Scale – клас шкала з повзунком. Це віджет, який дозволяє обрати будь-яке значення із заданого діапазону.

Синтаксис:

ім'я повзунка = Scale(ім'я вікна)

Параметри віджета Scale

Параметр	Значення
width	Ширина повзунка
bg	Фон повзунка
fg	Колір тексту поруч з повзунком
bd	Ширина межі повзунка
activebackground	Колір фону (коли повзунок рухають або клацають на нього)
activeforeground	Колір тексту (коли повзунок рухають або клацають на нього)
disabledbackground	Колір фону (коли властивість state == DISABLED)
disabledforeground	Колір тексту (коли властивість state == DISABLED)
state	Стан повзунка (NORMAL, DISABLED)
justify	Вирівнювання тексту
highlightthickness	ширина другої межі
relief	Рельєф повзунка (FLAT, GROOVE, RIDGE, SUNKEN, RAISE)
overrelief	Ральєф повзунка коли над нею знаходиться курсор(FLAT, GROOVE, RIDGE, SUNKEN, RAISE)
font	Вид шрифту у розмітки повзунка
from_	Число, з якого починається шкала
to	Число, яким закінчинається шкала
tickinterval	Інтервал, через який відображаються мітки шкали (у пікселях)
resolution	Мінімальна кількість пікселів, на яке користувач може пересунути повзунок
orient	Орієнтація смуги з повзунком (HORIZONTAL, VERTICAL) • HORIZONTAL – горизонтально • VERTICAL - вертикально
lenght	Довжина лінії, по якій рухається повзунок

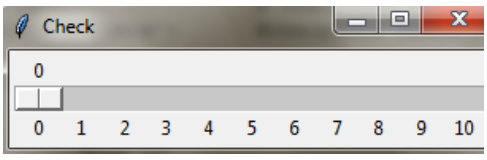
Функції віджета Scale

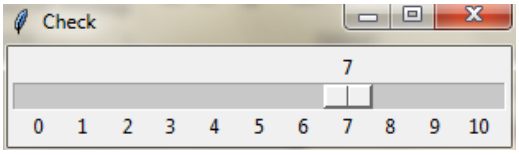
Функція	Значення
ім'я_прапорця.get()	Повертає поточне значення на шкалі
ім'я_прапорця.set(значення)	Встановлює значення на повзунок

Приклад: Розмістити у вікні шкалу від 0 до 10.

```
from tkinter import *      # імпорт графічної бібліотеки
window = Tk()              # створення головного вікна
window.title ('Check')     #створення назви головного вікна

scale = Scale(window,      # створення шкали
        orient=HORIZONTAL,          # орієнтація повзунка
        length=300,                 # довжина
        from_=0,                    # початкове значення на шкалі
        to=10,                      # кінцеве значення на шкалі
        tickinterval=1,             # інтервал, через який відображаються мітки
              на шкалі
        resolution=1)               # мінімальна відстань пересування повзунка
scale.pack()             # розміщення шкали
```

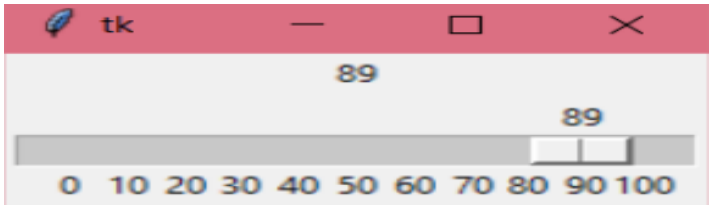
Результат





Завдання для самостійної роботи.

1. Створити шкалу з повзунком за зразком



Запитання для самоконтролю.

1. З якою метою створюється шкала з повзунком?
2. Синтаксис віджета шкала з повзунком.
3. Основні параметри віджета шкала з повзунком та їх значення.
4. Яким чином «зчитати» вибране значення на шкалі?

5.13. Віджет Listbox (Список)

Listbox – клас списку. Список з якого користувач може вибирати один або декілька пунктів.

Синтаксис:

Ім'я списку = Listbox(ім'я вікна)

Параметри віджета Listbox

Параметр	Значення
width	Ширина списку
height	Висота списку
bg	Фон списку
disabledforeground	Колір фону (коли властивість state == DISABLED)
state	Стан прапорця (NORMAL, DISABLED)
relief	Рельєф межі списку (FLAT, GROOVE, RIDGE, SUNKEN, RAISE)
highlighttickness	ширина другої межі
selectbackground	колір фону виділених елементів списку
selectforeground	Колір тексту виділених елементів списку
Text	Заголовок списку
selectmode	Визначає, скільки елементів можна вибрати і як перетягування миші впливає на вибір • SINGLE – ви можете вибрати лише один рядок, і не можете перетягувати курсор миші • EXTENDED – ви можете вибрати будь-яку суміжну групу рядків одночасно, натиснувши на перший рядок і перетягнувши на останній рядок

Функції віджета Listbox

Функція	Значення
ім'я_списку.get(початок, кінець)	Отримує фрагмент списку від рядка, позиція якого визначається першим параметром, до рядка, позиція якого визначена другим параметром
ім'я_списку.insert(позиція, назва_елемента)	Вставляє елемент в списку після даної позиції, визначеної першим параметром (нумерація починається з 0)
ім'я_списку.delete(початок, кінець)	Видаляє зі списку всі елементи, індекси яких потрапляють в діапазон, який починається з індексу, визначеним першим параметром, і закінчується індуксом, визначеним другим параметром
ім'я_списку.curselection()	Дозволяє отримати у вигляді кортежу індекси вибраних елементів

Новостворений список не має жодного елементу. Їх можна додати за допомогою циклу **for**, включивши в його тіло функцію **insert**. Ця функція потребує введення двох параметрів (позицію та ім'я елементу).

Приклад:

list = ["Математика", "Біологія", "Хімія", "Фізика"] # елементи, які повинні потрапити у список

listbox = Listbox(window, height=5, width=15, selectmode=EXTENDED) # створюється список

for i in list: # додаємо елементи у віджет

listbox1.insert(END, i) # додається елементи у віджет

Приклад: Створити список для вибору предметів для проходження ЗНО. Передбачається вибір одного елемента.

*from tkinter import ** #імпорт графічної бібліотеки

window=Tk() #створення головного вікна

window.title ('Список ЗНО') #створення назви головного вікна

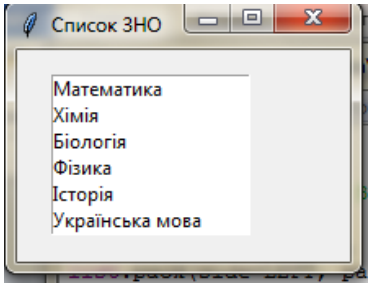
list = Listbox(window, selectmode=SINGLE, width=20, height=6) # створення списку

list.pack(side=LEFT, padx=20) # розміщення списку

for i in ['Математика', 'Хімія', 'Біологія', 'Фізика','Історія', 'Українська мова']:

list.insert(END, i) # додавання елементів у список

Результат:



Завдання для самостійної роботи.

1. Створіть список груп вашого курсу, вашого факультету.

Запитання для самоконтролю

1. З якою метою створюються списки?
2. Синтаксис списку.
3. Основні параметри списку та їх значення.
4. Чи можна список замінити багаторядковим текстовим полем?
5. Чи можна до поля списку додати смугу прокручування?

5.14. Віджет Menu –(Меню)

Menu – це віджет, у якому представлені всі команди, які доступні користувачу через графічний інтерфейс.

Синтаксис:

ім'я меню = Menu(ім'я вікна)

ім'я вікна.config(menu = name)

Додати команди до меню можна за допомогою методу **add_command().**

Синтаксис додавання команд до меню:

ім'я меню.add_command(label="Назва меню")

Підменю створюються аналогічно. Відмінність полягає в тому, що воно прикріплюється до головного меню за допомогою методу **add_cascade()**.

Синтаксис створення підменю в головному меню:

ім'я підменю = Menu(ім'я головного меню)

ім'я головного меню.add_cascade(label = назва підменю, menu = ім'я підменю)

Функції підменю

Функція	Значення
ім'я_підменню.add_separator()	Вставляє горизонтальний роздільник після попереднього пункту підменю
ім'я_підменню.add_checkbutton()	додає в підменю кнопку – прапорець
ім'я_підменню.add_radiobutton()	Додає в підменю радіокнопку

Приклад: Створити меню «Файл» і «Help» з додатковими командами в підменю

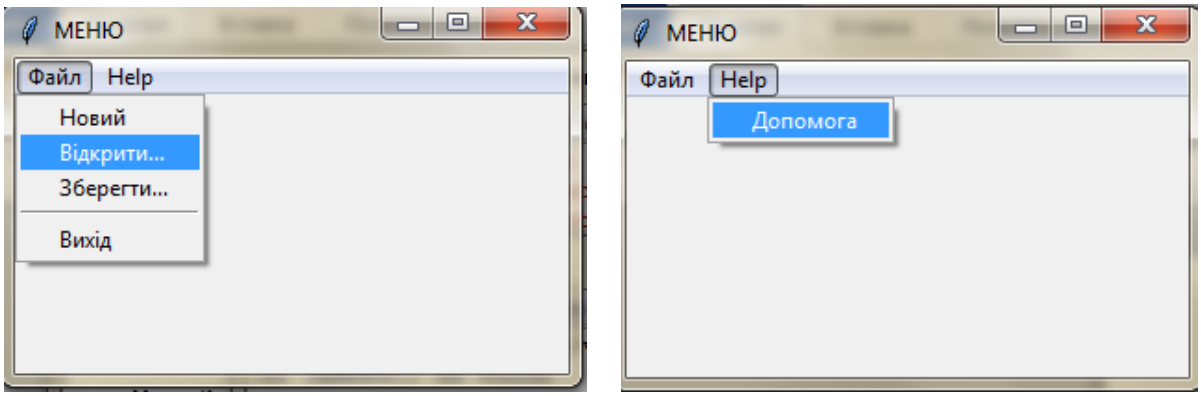
```
from tkinter import *   #імпорт графічної бібліотеки
window=Tk()             #створення головного вікна
window.title ('МЕНЮ')  #створення назви головного вікна
window.geometry('500x200')          # розміри головного вікна

headmenu = Menu(window)         # створення головного меню
window.config(menu=headmenu)

filemenu = Menu(headmenu, tearoff=0)          # створення підменю "Файл"
filemenu.add_command(label="Новий")             # додавання команди "Новий"
                    до підменю "Файл"
filemenu.add_command(label="Відкрити...")                 # додавання команди
                    "Відкрити..." до підменю "Файл"
filemenu.add_command(label="Зберегти...")                 # додавання команди
                     "Зберегти..." до підменю "Файл"
filemenu.add_separator()        # створення горизонтального роздільника
```

filemenu.add_command(label="Вихід") # додавання команди "Вихід" до підменю "Файл"

helpmenu = Menu(headmenu, tearoff=0) # створення підменю "Help"
helpmenu.add_command(label="Допомога") # додавання команди "Допомога" до підменю "Help"

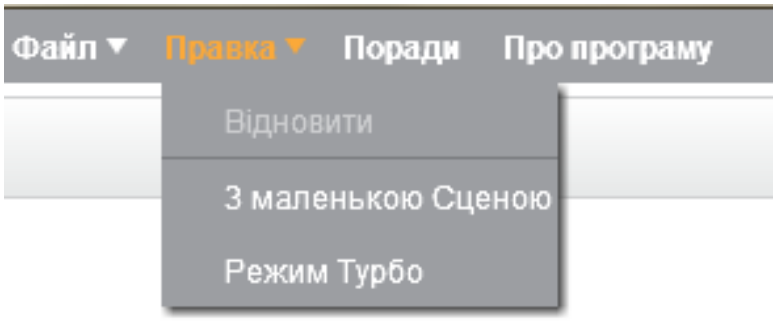
headmenu.add_cascade(label="Файл", menu=filemenu) # зв'язуємо підменю "Файл" з головним меню
headmenu.add_cascade(label="Help", menu=helpmenu) # зв'язуємо підменю "Help" з головним меню

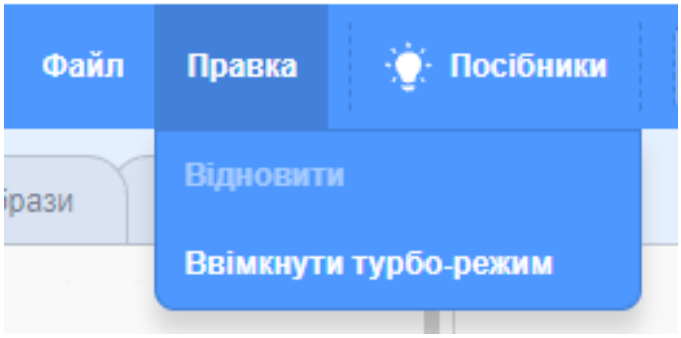
Результат

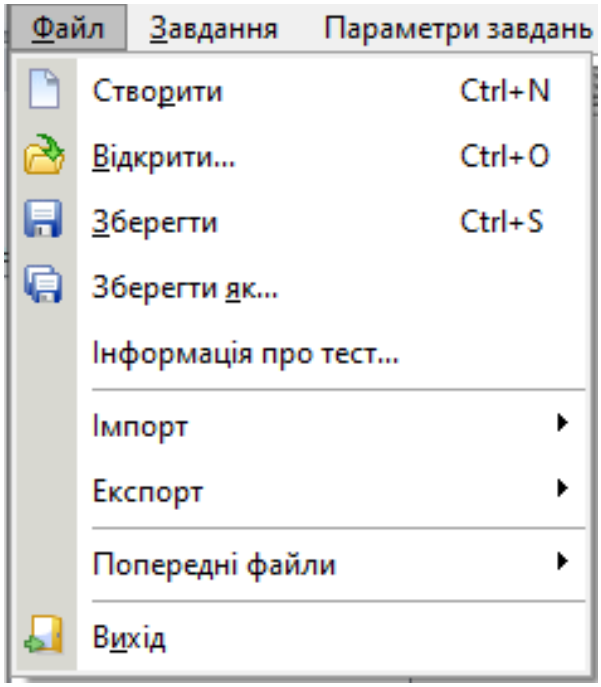


Завдання для самостійної роботи.

1. Створіть меню за зразком:







Запитання для самоконтролю

1. Синтаксис створення меню.
2. Що собою представляють підменю?
3. Синтаксис створення підменю.
4. Основні функції підменю.
5. Чи можливо в підменю включити підменю нижчого порядку?

6. АЛГОРИТМІЧНІ СТРУКТУРИ В МОВІ PYTHON

6.1. Базові структури алгоритму

Це структури, за допомогою яких створюється алгоритм для розв'язання певної задачі. Існують три основні (базові) алгоритмічні структури, або три основні типи алгоритмів: лінійний, розгалужений та повторення (циклічний). Усі інші алгоритмічні структури утворюються з елементарних шляхом заміни операторних блоків елементарними структурами.

Лінійний алгоритм (послідовне виконання, структура слідування) – це алгоритм, який забезпечує отримання результату шляхом одноразового виконання послідовності дій, незалежно від вхідних даних і проміжних результатів. Дії в таких алгоритмах виконуються послідовно, одна за однією, тобто лінійно.

Важливою особливістю внутрішніх елементів лінійних алгоритмів є те, що кожен із них має один вхід і один вихід. Весь алгоритм представляє лінійну послідовність базових елементів.

Розгалужений алгоритм (структура вибору). У класичному варіанті ця структура розглядається як вибір дій у разі виконання або невиконання заданої умови. Базова структура алгоритму з розгалуженням теж називається *розгалуженням*.

Розрізняють повну й коротку форму розгалуження.

Повна форма розгалуження означає, що здійснюється вибір між двома діями. Якщо перевірка умови дає результат «так», то вибирається дія 1; у іншому випадку, тобто якщо перевірка умови дає результат «ні», вибирається дія 2. Повну форму розгалуження можна прочитати у такий спосіб: якщо перевірка умови дає результат «так», то виконати дію 1, інакше виконати дію 2.

Наприклад, для надання x значення більшого з двох заданих різних чисел a і b потрібно:

– перевірити чи більше *a* за *b*; якщо так, то надати *max* значення *a*, а якщо ні (тобто інакше), то надати *max* значення *b*.

Отже, маємо повну форму розгалуження: якщо a>b, то надати *max* значення *a*, інакше надати *max* значення *b*.

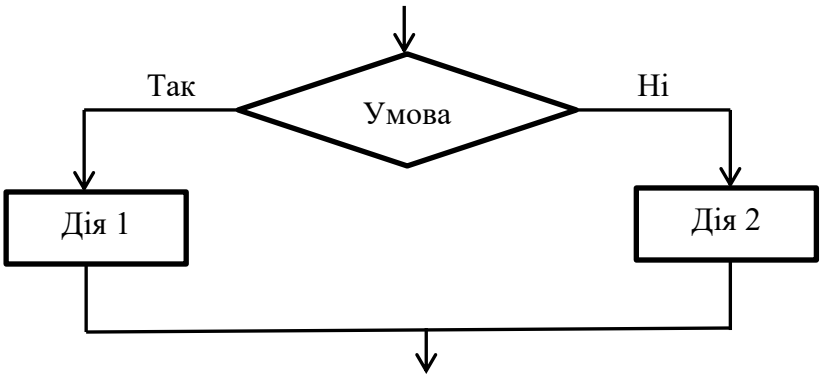


Рис. Повна форма розгалуження

Коротку форму розгалуження можна прочитати у такий спосіб: якщо перевірка умови дає результат «так», то виконати дію.

Приклад. Для заміни числа його модулем потрібно перевірити, чи є число від'ємним. У від'ємного числа слід замінити знак на протилежний, а додатне число і його модуль співпадають. Отже, маємо коротку форму розгалуження.

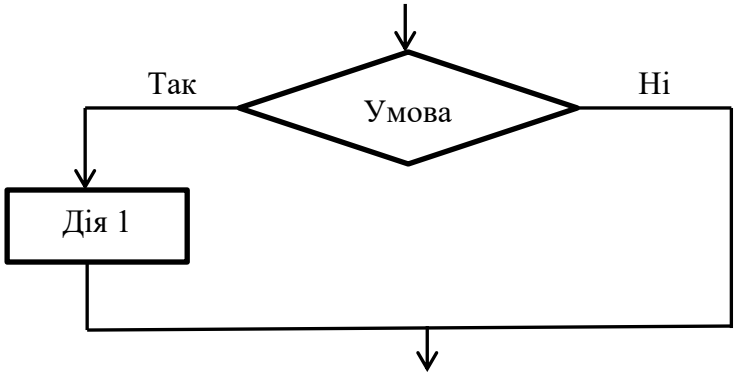


Рис. Короткаа форма розгалуження

Циклічний алгоритм (цикл, структура повторення) – це алгоритм, у якому передбачено повторення деякої серії команд. За допомогою цієї структури описуються однотипні дії, що повторюються декілька разів. Такі

алгоритми забезпечують виконання довгої послідовності дій, записаних порівняно короткою послідовністю команд..

Базова структура алгоритму з повторенням називається ***повторенням***, або частіше ***циклом***. Розрізняють два основні різновиди циклів: цикли, де умова перевіряється *до* виконання дії, — *цикли з передумовою*, і цикли, де перевірка умови здійснюється *після* виконання дії, — *цикли з післяумовою.*

У циклі з передумовою умова циклу сформульована таким чином, щоб повторне виконання дії виконувалося, доки перевірка умови дає результат «так». Через це такі цикли називають ще циклами «*доки*».

Цикл «доки» можна прочитати у такий спосіб: *доки* перевірка умови дає результат «так», виконувати дію. Якщо при черговій перевірці умови буде одержано результат «ні», повторне виконання дії буде припинено й відбудеться вихід із циклу.

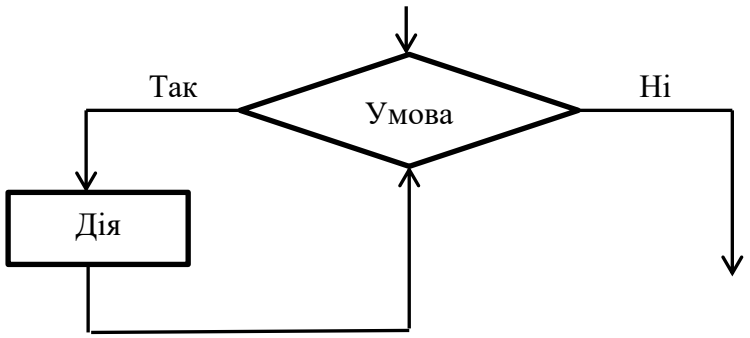


Рис. Цикл з передумовою

У циклі з післяумовою умова циклу сформулюється протилежним чином: якщо чергова перевірка умови дає результат «так», відбувається вихід із циклу. Через це такі цикли називають також циклами «до». Цикл «до» можна скорочено прочитати так: *повторювати* дію *до* одержання результату «так» при перевірці умови.

Зверніть увагу на те, що є спільним для обох типів циклу:

- обидві базові структури циклу є замкненими;
- кількість повторень циклу визначається його умовою;
- вихід із циклу відбувається тільки через перевірку умови циклу.

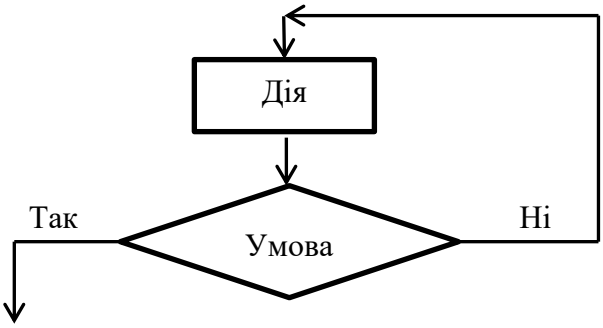


Рис. Цикл з післяумовою

Найбільш суттєва різниця між типами циклів полягає в тому, що тіло циклу з післяумовою обов'язково виконується хоча б один раз – до першої перевірки умови, а цикл із передумовою може бути не виконаним жодного разу, якщо при першій же перевірці умови маємо результат «ні». Через це розглянуті типи циклів не є взаємозамінюваними: цикл із післяумовою можна замінити циклом з передумовою, а навпаки – ні.

Третій тип циклів є так званий «цикл з параметром» або з наперед визначеною кількістю повторень. Такий цикл використовується в тому випадку, коли відома кількість разів виконання тіла циклу. Графічно цикл з параметром зображується аналогічно до циклу з передумовою. При цьому умовою є кількість повторень.

Основна особливість базових алгоритмічних структур – це їх повнота, тобто цих структур достатньо для створення найскладнішого алгоритму.

Запитання для самоконтролю

1. Які основні базові структури алгоритмів існують?
2. У якій послідовності виконуються команди в лінійному алгоритмі?
3. Як виконуються команди у розгалуженому алгоритмі і від чого це залежить?
4. Які форми розгалуженого алгоритму існують? Чим вони відрізняються?
5. Як виконуються команди в алгоритмі повторення?
6. Які типи алгоритмів повторення існують? Чим вони характерні?
7. Які типи циклів є взаємозамінними?

6.2. Оператори розгалуження в Python

Більшість задач передбачають необхідність змінювати порядок виконання команд програми в залежності від дій користувача, або від виконання інших умов, тобто проходить розгалуження процесу. У таких випадках використовують умовний оператор розгалуження. Вибір траєкторії руху вздовж алгоритму залежить від виконання (або не виконання) певних умов, які математично оформляються з використання *операторів порівняння*.

Оператори порівняння в Python.

У Python для порівняння об'єктів (змінних різних типів) використовуються наступні оператори:

- > – більше;
- < – менше;
- >= – більше або дорівнює (не менше);
- <= – менше або дорівнює (не більше);
- == – дорівнює (рівне);
- != – не дорівнює (не рівне).

Логічні оператори

Досить часто виникає необхідність будувати складніші логічні вирази, які містять декілька простих логічних тверджень, між якими потрібно використати логічні оператори: and, or, not (І, Або, Ні).

Операндами операторів and, or та not можуть бути будь які об'єкти (числа, рядки, списки і т.д.).

Логічний оператор not

Логічний оператор *not* називають *запереченням*.

Синтаксис оператора: ***not X*.**

Результатом логічного оператора *not* є значення логічного типу, яке є запереченням операнда.

Якщо операнд істинний (True), то за оператором *not* буде повернуто – False. Якщо операнд хибний (False), то за оператором *not* буде повернуто – True.

Логічний оператор and

Логічний оператор *and* також називають *кон'юнкцією* або *логічним множенням.*

Синтаксис оператора:

X1 and X2.

Результатом застосування логічного оператора *and* є значення логічного типу, яке набуває значення True тільки тоді, коли істинні обидві змінні і False в усіх інших випадках.

Логічний оператор or

Логічний оператор *or* також називають диз'юнкція або логічне додавання.

Синтаксис оператора:

X1 or X2.

Результатом застосування логічного оператора *or* є значення логічного типу, яке набуває значення True, якщо принаймні один операнд буде істинним в іншому випадку False.

Синтаксис операторів розгалуження

Для того, щоб з'ясувати як працюють оператори розгалуження розглянемо простий приклад: необхідно визначити число x, відмінне від нуля, є додатним, чи від'ємним.

Розв'язати поставлену задачу не можна використовуючи тільки лінійні структури. Після порівняння значення ***x*** з нулем має бути виведене одне або інше повідомлення.

Розв'язання цієї задачі в Python реалізується з використанням умовного оператора **if-else**, який називається оператором розгалуження.

Синтаксис оператора **if-else:**

if логічний_вираз:

блок_команд_1

else:

блок_команд_2]

Виконання оператора if-else передбачає обчислення значення логічного виразу. Якщо значення логічного виразу істинне, то виконується блок_команд_1, в протилежному випадку виконується блок_команд_2.

Для наведоної задачі код програми реалізується так:

x = int(input(“Введіть число відмінне від нуля:”))

if x > 0:

print(“Число додатне.")

else:

print("Число від’ємне.")

Структури оператора, де розгалуження виконується по кожній із віток і містить в кожний певний блок команд, вважається *повною.*

Оператор розгалуження може бути *неповним*, якщо в ньому відсутня гілка else з відповідним блоком команд.

В процесі текстового оформлення коду, запис службового слова if з логічним виразом та службового слова else розміщають в коді, як основну інструкцію а вкладені блоки команд зміщуються вправо.

Якщо з умови задачі прибрати умову введення ненульового числа, то постає необхідність визначення, є число додатним, чи число є від’ємним або число є нулем. Виникає необхідність доповнювати програму додатковою необхідністю перевірки числа на рівність нулю і виведення відповідного повідомлення.

Це можна зробити, якщо три рази використати неповну форму оператора розгалуження:

x = int(input("Введіть число:"))

if x > 0:

```
        print("Число додатне.") 55
if x < 0:
        print("Число від'ємне.")
if x == 0:
        print("Число нуль.")
```

В деяких завданнях реалізація розгалуження подібним способом незручна або погано читається в коді програми. На цей випадок в мові Python передбачено спрощений запис для виконання великого числа перевірок. Для їх реалізації використовується оператор **if-elif-else**, який має наступний вигляд:

```
if Логічний_вираз_1:
        Блок_команд_1
elif Логічний_вираз_2:
        Блок_ команд_2
elif Логічний_вираз_3:
        Блок_ команд_3
…
[else:
        Блок_ команд_N]
```

Для реалізації нашої задачі код буде виглядати так:

```
x = int(input("Введіть число:"))
if x > 0:
        print("Число додатне.")
elif x < 0:
        print("Число від'ємне.")
else:
        print("Число нуль.")
```

Таку структуру застосовують в випадках, якщо перевіряти умову потрібно декілька разів. Наприклад за назвою дня тижня визначити його номер. Цю структуру іноді називають *оператор вибору*.

Завдання для самостійної роботи.

Результат оформити в графічному вигляді.

1. Написати код програми для розрахунку ідеальної маси чоловіка або жінки за формулою Брокка, попередньо виявивши стать людини.

Ідеальна вага для чоловіка = (зріст в сантиметрах - 100) * 1,15.

Ідеальна вага для жінки = (зріст в сантиметрах - 110) * 1,15.

(для введення статі не обмежуйте користувача регістром літер)

2. В Україні людина вважається повнолітньою з 18 років. Напишіть код програми, яка визначає, чи є користувач повнолітнім.

3. Напишіть код програми, яка за назвою дня тижня визначає його номер.

Запитання для самоконтролю

1. Які оператори порівняння використовуються в Python?
2. Які логічні оператори використовуються в Python? З якою метою вони використовуються?
3. Синтаксис та значення логічних операторів.
4. Синтаксис повного оператора розгалуження.
5. Синтаксис неповного оператора розгалуження.
6. Синтаксис оператора вибору.
7. Чи є оператор вибору і оператор розгалуження взаємозамінними?

6.3. Цикл з передумовою в Python

У процесі написання коду програми досить часто є необхідність рутинно повторювати групу команд певну кількість разів. Отже потрібна конструкція, яка може організувати виконання операторів декілька разів. Таку конструкцію прийнято називати алгоритмом повторення або циклом.

Будь який цикл складаються з заголовку та тіла циклу. Заголовок циклу містить умову, яка перевіряється для визначення необхідності повторення. Тіло циклу складається з команд, які повторно виконуються. Кожну дію, що повторюється називають кроком циклу або ітерацією

Цикл з передумовою є найбільш універсальним в усіх мовах програмування, включаючи Python. Але в мові Python він дещо повільний. Це пов'язано з тим, що в циклі з передумовою спочатку обчислюється значення логічного виразу, якщо це значення істинне, то виконується тіло циклу і відбувається повернення до перевірки логічного виразу. Одне проходження по тілу циклу називається *ітерацією*.

Цей процес продовжується до тих пір, поки значення логічного виразу не стане хибним. Після цього цикл завершиться і відбувається перехід до команди, що записана після тіла циклу while. Якщо при першому обчисленні логічного виразу його значення буде хибним, то тіло циклу не буде виконано жодного разу. Логічний вираз прийнято вважати умовою циклу, а блок команд – тілом циклу. Тіло циклу може містити довільний набір операторів. Але в тілі циклу повинен бути оператор, який впливає на значення логічного виразу для того, щоб цикл на певному кроці закінчив свою роботу. Крім того перше значення логічного виразу повинно бути визначеним до його першої перевірки.

Цикл з передумовою ще називають циклом *While (поки)*, оскільки саме з цього ключового слова він починається. Цикл *While* використовується в випадках, якщо немає можливості визначити значення кількості разів використання циклу.

Синтаксис оператора циклу while:

while Логічний_вираз:

Блок_команд тіла циклу

У процесі текстового оформлення коду, запис службового слова ***while*** з логічним виразом розміщаються в коді, як основна інструкція, а вкладені блоки команд тіла циклу зміщуються вправо.

Приклад. Обчислити факторіал числа n.

```
print('Введіть натуральне число n=')
n=int(input())  # введення числа з клавіатури
f=1             # початкове значення факторіалу
i=2             # початкове значення лічильника
while i<=n:     # початок циклу (перевірка умови)
    f=f*i       # множення
    i=i+1       # наступне значення лічильника
print('Факторіал =',f) # результат
```

Іноді виникають ситуації в яких на даному кроці потрібно пропустити команди в тілі циклу, або взагалі перервати виконання циклу навіть в випадку, коли логічний вираз ще не набув хибного значення.

Оператор continue

Оператор ***continue*** призначений для переривання поточної ітерації циклу і переходу до наступної. Тобто оператори, що будуть іти в тілі циклу після виклику continue, на даному кроці виконуватися не будуть.

Оператор break

Оператор ***break*** призначений для дострокового припинення роботи циклу (for або while), тобто зупинки виконання тіла циклу, навіть якщо умова виконання циклу ще не набула значення False або послідовність елементів не закінчилась.

Зрозуміло, інструкцію break варто викликати тільки всередині інструкції if, тобто вона повинна виконуватися тільки при виконанні якоїсь особливої умови.

Завдання для самостійної роботи.

1. Написати код програми побудови 10 концентричних кіл будь-якого кольору.
2. Написати код програми, побудови 10 концентричних кругів від найбільшого до найменшого радіусу різних кольорів.

Запитання для самоконтролю

1. Структура оператора з передумовою.
2. Що називається ітерацією?
3. Синтаксис оператора з передумовою.
4. Який елемент обов'язковий у тілі циклу?
5. З якою метою в тіло циклу можуть бути включені оператори continue і break?

6.4. Цикл з параметром в Python

В мові Python крім циклу з передумовою є цикл, який має можливість перебирати всі елементи з певного набору. Його називають **for**. Він не такий універсальний як цикл з передумовою, але більш ефективний. Для кращого розуміння його роботи доцільно ознайомитися з даними типу діапазон.

Тип даних діапазон (range)

Тип даних ***range*** є незмінюваною послідовністю цілих чисел.

Описати діапазон можна за допомогою різних варіантів функції **range()**:

1. **range(n)** - послідовність цілих чисел від 0 до n з кроком 1.

Наприклад: *range(3)* описує діапазон 0, 1, 2.

2. **range(start, n)** – послідовність цілих чисел від значення start до n з кроком 1.

Наприклад: range(2, 5) описує діапазон 2, 3, 4.

3. **range(start, n, step)** – послідовність цілих чисел від значення start до n з кроком step. Якщо параметр step невказаний то за замовчуванням він дорівнює 1.

Наприклад: range(1, 10, 3) описує діапазон 1, 4, 7.

Щоб значення діапазону зменшувалися, необхідно, щоб перший параметр був більшим ніж другий а третій параметр - від'ємним. Функція range(6, -6, -2) описує діапазон 6, 4, 2, 0, -2, -4

Синтаксис циклу з параметром

У загальному вигляді, у циклі for можна використовувати будь-який набір елементів, що доступний до ітерування.

Цикл for дозволяє виконати таку кількість ітерацій, яка відповідає кількості елементів в послідовності, або виконати операції в циклі над усіма членами нечислової послідовності.

Цикл for складніший і менш універсальний, ніж цикл while, але виконується значно швидше.

Синтаксис оператора циклу for:

for змінна_циклу in послідовність змінної:

блок_команд

На першому кроці змінній циклу присвоюється значення першого елемента послідовності, потім виконується блок команд із тіла циклу. Далі змінній циклу надається наступне значення із послідовності – виконується блок команд із тіла циклу і т.д. до тих пір, поки змінній циклу не будуть надані усі значення послідовності.

Приклад. Для прикладу пропонуємо розв'язати задачу на обчислення факторіалу числа заданого з клавіатури і порівняти коди програм з використанням циклів з параметром і передумовою.

print('Введіть натуральне число n=')

```
n=int(input())  # введення числа з клавіатури
f=1             # початкове значення факторіалу
for i in range (2, n+1):    # початок циклу (кількість ітерацій)
   f=f*i       # множення
print('Факторіал =',f) # результат
```

Висновок. Код програми при використанні циклу з параметром на 2 рядки коротший.

6.5. Цикл з післяумовою в Python

Інколи потрібно виконати дії, а потім перевірити умову на виконання. Такі завдання зустрічаються досить рідко, але іноді іншого варіанту рішення не існує. Тому у більшості мов програмування є спеціальна конструкція для організації циклів з післяумовою. У мові Python такої спеціальної конструкції немає, але її можна сконструювати із використанням нескінченого циклу та оператора break.

Пропонується наступний варіант:

```
while True:
блок_команд
if умова:
break
```

Ця конструкція виконується наступним чином. Оскільки умова циклу True - істина, то виконується блок команд тіла циклу. Після їх виконання командою розгалуження перевіряється виконання умови. Якщо умова команди розгалуження не виконується, то команди тіла циклу виконуються ще один раз. У іншому випадку (умова команди розгалуження - істина) виконується команда break. І перериває цикл, у якому знаходиться.

Завдання для самостійної роботи.

1. Написати код програми, в якій будуються 10 концентричних кіл будь якого кольору використовуючи описані вище цикли.

2. Написати код програми, в якій будуються 10 концентричних кругів від найбільшого до найменшого різних кольорів, використовуючи описані вище цикли.

Запитання для самоконтролю

1. Що собою являє тип даних range?

2. Які є способи представлення даних типу range?

3. Синтаксис циклу з параметром.

4. Який з циклів буде виконуватися швидше: (цикл з передумовою чи цикл з параметром)?

7. ТИПИ ДАНИХ

7.1. Рядкові величини.

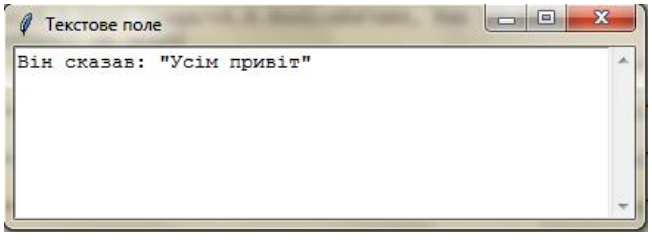
Рядкова величина (рядок) це набір будь-яких символів (літер, чисел, знаків пунктуації, неалфавітних символів та пропуску). Python 3 може працювати з будь-якими символами, оскільки підтримує стандарти Unicode і ASCII.

Рядки представляють собою послідовності символів, взятих в одинарні або подвійні лапки. Різні види лапок дозволяють створювати рядки, що містять лапки та апостроф. Усередині одинарних лапок можна розташувати подвійні і навпаки.

Наприклад: Створити текстове поле, у яке вставити речення «*Він сказав "Усім привіт"*».

```
from tkinter import *       # імпорт графічної бібліотеки
window = Tk()               # створення головного вікна
window.title ('Текстове поле')      #створення назви головного вікна

t = Text(                   # створення багаторядкового текстового поля
      width=50,             # ширина текстового поля
      height=7,             # висота текстового поля
      bg='white',           # колір фону текстового поля
      fg='black')           # ширина та рельєф межі рамки
t.pack(side=LEFT)                 # розміщення текстового поля

text='Він сказав: "Усім привіт" '  # присвоєння змінній значення
t.insert(END, text)       # вставка тексту у текстове поле
```

Результат



Цілком зрозуміло, що апостроф використовується тільки у тексті коду, якщо текст взятий у подвійні лапки.

В Python рекомендується працювати з рядками, максимальна довжина яких 80 знаків. При необхідності надрукувати текст більшої довжини його потрібно брати в 3 одинарні або в 3 подвійні лапки. На екрані тест буде відображатися по рядках, як і у тексті коду. Але Python буде «вважати», що це один рядок.

Наприклад: Створити текстове поле з текстом Гімну України.

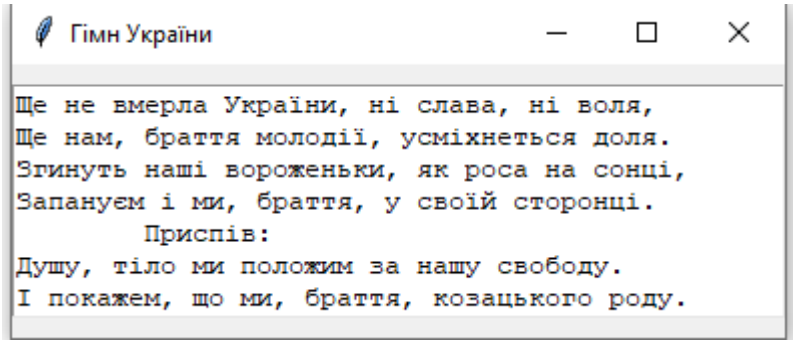
```
from tkinter import *       # імпорт графічної бібліотеки
window = Tk()               # створення головного вікна
window.title ('Гімн України')      #створення назви головного вікна

t = Text(                   # створення багаторядкового текстового поля
      width=50,             # ширина текстового поля
      height=7,             # висота текстового поля
      bg='white',           # колір фону текстового поля
      fg='black')           # ширина та рельєф межі рамки
t.pack(side=LEFT)           # розміщення текстового поля

scroll = Scrollbar(command=t.yview) # створення смуги прокручування
scroll.pack(side=LEFT, fill=Y)      #створення смуги прокручування, параметр fill вимагає заповнення всього доступного простору віджета
t.config(yscrollcommand=scroll.set)  # встановлення можливості прокрутки тексту

text='''Ще не вмерла України, ні слава, ні воля,
Ще нам, браття молодії, усміхнеться доля.
Згинуть наші вороженьки, як роса на сонці,
Запанує і ми, браття, у своїй сторонці.
      Приспів:
```

Душу, тіло ми положим за нашу свободу.

І покажем, що ми, браття, козацького роду. ''' # присвоєння змінній значення

t.insert(END, text) # вставка тексту у текстове поле

Результат:



Керуючи символи

Застосування символу зворотній слеш (\) дозволяє створювати керуючі послідовності всередині рядків. Найбільш поширена є:

\n – перехід на новий рядок;

\t – знак табуляції;

\\ – зворотний слеш;

\b – Backspace;

\f – перехід на нову сторінку;

\v – вертикальна табуляція;

Наприклад: Створити текстове поле, у яке вставити речення «*Він сказав "Усім привіт"*», таким чином, щоб кожне слово розпочиналося з нового рядка а пряма мова додатково з абзацу.

```
from tkinter import *       # імпорт графічної бібліотеки
window = Tk()               # створення головного вікна
window.title ('Текстове поле')      #створення назви головного вікна
t = Text(                   # створення багаторядкового текстового поля
    width=50,          # ширина текстового поля
    height=7,          # висота текстового поля
```

```
    bg='white',        # колір фону текстового поля
     fg='black')       # ширина та рельєф межі рамки
t.pack(side=LEFT)      # розміщення текстового поля

text='\nВін\nсказав:\n\t"Усім\n\tпривіт" '  # присвоєння змінній значення
t.insert(END, text)      # вставка тексту у текстове поле
```

Результат:

Для заміщення у рядку коду значень змінних використовують комбінацію символів **%s**. Для виведення заміщеної змінної використовують символ **%.**

Наприклад: В рядок «Мене зовуть » вставити своє ім’я.

```
print ("Введіть своє ім'я ")    # виведення виразу
name = input()                  # присвоєння змінній значення введеного з
                  клавіатури
string = 'Мене звуть %s'        # присвоєння змінній  значення виразу з
                  елементом заміни у кінці
print(string % name)            # вивепдення змінної string з вставкою змінної
                  name
```

Результат:

```
Введіть своє ім'я
Сергій
Мене звуть Сергій
>>>
```

Операції над рядками

При виконанні операцій над рядками потрібно пам'ятати, що в Python рядок є незмінною величиною і виконуючи з ним будь які операції, ми створюємо новий рядок.

Конкатенація (об'єднання) рядків

Об'єднувати рядки або рядкові змінні в Python можливо за допомогою оператора **+**, або розташувати їх послідовно один за одним. При об'єднанні рядків Python не додає між ними пробілів, тому, у разі необхідності, їх потрібно додати.

Розмноження рядків

Для розмноження рядків в Python використовується оператор ***n,** де n – це кількість повторень. При цьому створюється новий рядок, в якому кілька разів повторюється значення первинного рядка (без пробілів).

Довжина рядка

Для визначення кількості символів в рядку з урахуванням усіх розділових знаків і пропусків використовується функція ***len().*** В дужках вказується або ім'я рядка, або рядок взятий в лапки.

Наприклад:

s='Я вивчаю Python'

len(s) – результат 15

len('Я вивчаю Python') – результат 15

Довжина порожнього рядка = 0. Функцію len() застосовують і до інших послідовностей, які будуть розглянуті пізніше (кортеж, словник, список).

Звернення до символів (слайсинг)

Операція звернення до символів виконується досить часто. При її виконанні потрібно пам'ятати, що нумерація символів у рядку починається з 0. Але можливий відлік символів з кінця рядка. У такому разі останній символ має номер -1, перед останній -2 і так далі.

Для того щоб отримати доступ до одного символу рядка, задається звернення, яке складається з імені рядка і індексом символу в квадратних дужках.

Синтаксис:

ім'я рядка[індекс символу]

Наприклад:

s [0] – звернення до першого символу рядка s;

s [1] – звернення до другого символу рядка s;

s [-1] – звернення до останнього символу рядка s.

Для звернення до групи символів, що розташовані в рядку створюється аналогічне звернення, в якому після імені рядка в квадратних дужках через двокрапку вводяться 3 параметри: індекс початкового символу, індекс кінцевого символу (він не включається до вибірки) і крок. Кожен з цих параметрів не є обов'язковим.

Синтаксис:

ім'я рядка[start: end: step]

Наприклад:

s[3:5] – звернення до елементів з 4 до 5;

s[2:-2] – звернення до елементів з 3 до 3 з кінця;

s[:6] – звернення до елементів з початку до 6;

s[1:] – звернення до елементів з 2 до останнього;

s [::7]) – звернення до кожного сьомого символу з початку до кінця рядка.

При зверненні до рядкової величини утворюється новий рядок (підрядок), якому необхідно присвоїти значення змінної.

Для роботи з рядковими величинами використовують інші функції і методи. Найбільш вживані з них представлені в таблиці.

Примітка: *В таблиці «Функції і методи рядкових величин» змінна S – це ім'я рядкової величини.*

Таблиця «Функції і методи рядкових величин»

Функція чи метод	Призначення
S.find(str, [start],[end])	Пошук підрядка в рядку. Повертає номер першого символу підрядка або -1 (за відсутності)
S.rfind(str, [start],[end])	Пошук підрядка в рядку. Повертає номер останнього символу підрядка або -1 (за відсутності)
S.index(str, [start],[end])	Пошук підрядка в рядку. Повертає номер першого символу підрядка або викликає ValueError(за відсутності)
S.rindex(str, [start],[end])	Пошук підрядка в рядку. повертає номер останнього символу підрядка або викликає ValueError(за відсутності)
S.split(символ)	Розбиває рядки по роздільнику
S.isdigit()	Перевіряє наявність цифр в рядку
S.isalpha()	Перевіряє наявність літер в рядку
S.isalnum()	Перевіряє наявність цифр або літер в рядку
S.islower()	Перевіряє наявність символівв в нижньому регістрі
S.isupper()	Перевіряє наявність символів в верхньому регістрі
S.isspace()	Перевіряє наявність в рядку недрукованих символів (пробіл, новий рядок, табуляція та ін..)
S.istitle()	Перевіряє чи розпочинаються слова в рядку з великої літери
S.upper()	Переводить рядки до верхнього регістру
S.lower()	Переводить рядки до нижнього регістру
S.join(список)	Створює рядки із списку з заданим роздільником
S.capitalize()	Переводить первый символ строки в верхний регистр, а все остальные в нижний
S.center(width, [fill])	Повертає вирівняний по-центру рядок, по краях якого стоїт символ fill (пробіл за замовчуванням)
S.expandtabs([tabsize])	Повертає копію рядка, в якому всі символи табуляції заміняються одним пробілом. Якщо TabSize не вказувати то розмір табуляції вважається 8 пробілів
S.lstrip([chars])	Видалення пробілів на початку рядка
S.rstrip([chars])	Видалення пробілів в кінці рядка

Функція чи метод	Призначення
S.strip([chars])	Видалення пробілів на початку і у кінці рядка
S.swapcase()	Переводить символи нижнього регістра в верхній, а верхнього – в нижній
S.title()	Перші літеру кожного слова переводить в верхній регістр, а всі інші в нижній
S.zfill(width)	Робить довжину рядка не менше від значення width, за необхідності заповнює перші символи нулями
S.ljust(width)	Робить довжину рядка не менше від значення width, за необхідності заповнює останні символи нулями
S.rjust(width, fillchar=" ")	Робить довжину рядка не менше від значення width, за необхідності заповнює перші символи символом fillchar

Завдання для самостійної роботи.

1. Написати код програми, яка запитує ім’я та прізвище. А потім виводить повідомлення «Привіт Прізвише Ім’я. Ми з України.».

2. Написати код програми, яка розмішує речення у декілька рядків: (двома способами). Текст:

«Досить суттєвою перевагою Python є наявність стандартної бібліотеки, яка інсталюється разом з Python і містить готові інструменти для роботи з операційною системою, веб-сторінками, базами даних, різними форматами даних, для побудови графічного інтерфейсу програм тощо.»

3. Склади програму, яка виводить на екран словосполучення «Мене звуть Прізвище Ім’я» 5 разів.

4. Використовуючи функцію split, розбити рядок «Python це багатоцільова мова програмування». В результаті повинен вийти список слів.

5. Із списку рядків попередньої задачі скласти речення.

Запитання для самоконтролю

1. Який тип даних називається рядковою величиною?

2. Які символи можуть входити до рядка?

3. Яка максимальна довжина рядка?
4. З якою метою використовуються керуючи ситволи?
5. Операції над рядками, їх синтаксис.
6. Звернення до символів рядкової величини.
7. Основні функції та методи для роботи з рядковими величинами.

7.2. Списки.

Список – це впорядкований набір даних *не обов'язково* одного типу. Аналогом списку в інших мовах програмування є масив, але головною відмінністю є те, що елементами списку в мові Python можуть бути елементи *різних* типів.

Масив – фіксований набір однотипних елементів, що розміщені в строгому порядку один за одним, а доступ до них здійснюється за номером даного елементу в масиві.

Список є змінюваним типом даних, в якому може змінюватися кількість елементів, їх порядок і тип. Список записуються як перелік елементів, розділених комою та взятих у квадратні дужки. Отже масив є списком, а список не обов'язково буде масивом.

Для створення списків існує декілька способів:

Перший спосіб: можна на список перетворити будь-який об'єкт за допомогою функції **list().** Наприклад: перетворити рядкову величину 'привіт' в список

print list('привіт')

Результат:

['п', 'р', 'и', 'в', 'і', 'т']

Другий спосіб: присвоїти значення списку будь-якій змінній. Цим способом можна створювати списки, які можуть мати будь-яку кількість будь-яких об'єктів в тому числі і пусті списки (s = []).

p = ['п', 'р', ['ивіт'], 13]

print (p)

Результат:

['n', 'p', ['ивіт'], 13]

Третій спосіб: створити список за допомогою **генератора списку**. Генератор списків – спосіб створення нового списку, при якому кожен елемент списку є результатом певної операції, яка задовольняє деяку конкретну умову при генерації кожного елемента. Для генерації списків використовують цикл *for*.

Наприклад: Змінити список p з попереднього прикладу таким чином, щоб в ньому елементи списку p записувалися 2 рази.

```
p = ['n', 'p', ['ивіт'], 13]   # список
for i in range (1):          # організація циклу
    p = p*2                  # подвоєння списку
print (p)                    # виведення списку
```

Результат:

['n', 'p', ['ивіт'], 13, 'n', 'p', ['ивіт'], 13]

Важливо зазначити, що методи списків, на відміну від методів рядкових величин, змінюють сам список, і тому результат виконання не потрібує нової змінної.

Елементи числового списку можна згенерувати випадковим чином, використовуючи функції модуля ***random***.

```
from random import*       # виклик функцій модуля random
n=int(input('Введіть кількість елементів списку ='))
a=[]                      # створення пустого списку
for i in range(n):        # організація циклу
   x=randint(1,100)       # генерація випадкового числа x
   a.append(x)            # додавання випадкового числа x в кінець списку a
print (a)                 # виведення списку
```

Результат:

Введіть кількість елементів списку =3

[53, 5, 16]

Функції та методи списків

Доступ до елементів списку відбувається за індексами (по аналогії з рядковими величинами нумерація елементів починається з нуля). Щоб звернутися до елемента списку, потрібно зазначити ім'я списку і в квадратних дужках індекс елемента. Індекс може бути від'ємним, якщо нумерація проводиться з кінця списку.

Якщо потрібно отримати не один елемент списку за індексом, а деяку вибірку елементів зі списку за певним правилом, то потрібно зробити зріз застосувавши оператор *item[start: stop: step]*. Кожен із параметрів **start, stop, step** може бути пропущений, по аналогії з рядковими величинами. Зріз списку теж список.

Оператор *in* дозволяє здійснити перевірку приналежності елемента до списку. Результатом перевірки буде одне із логічних значень True або False. *Синтаксис оператора: елемент in ім'я_списку.*

>>> L=[5,10,1,22]

>>> 5 in L

True

>>> 11 in L

False

Для перевірки, того, що елемент не належить списку, використовують оператор *not in*.

Для списків доступні інші вбудовані функції та методи списків, які представлені в таблиці.

Таблиця методів та функцій списків

Метод, функція	Призначення
L.append(x)	Додає один елемент в кінець списку
L.extend(M)	Додає всі елементи списку M в кінець списку L
L.insert(i, x)	Вставляє на i-тий індекс елемент зі значенням x
L.del(i)	Видаляє i-тий елемент

Метод, функція	Призначення
L.remove(x)	Видаляє перший знайдений елемент зі списку, який має значенння x.
L.pop([i])	Видаляє і-тий елемент и повертає його. Якщо не вказати індекс, видаляється останній елемент
L.index(x, [start [, end]])	Повертає індекс першого елемента зі значенням x (пошук виконується від start до end)
L.count(x)	Визначає кількість елементів зі значенням x
L.sort([key=функция])	Сортує список на основі функції
L.reverse()	Створює реверсивний список
L.copy()	Створює опію списку
L.clear()	Очищає список
len(L)	Визначає кількість елементів у списку
max(L)	Визначає максимальний елемент
min(L)	Визначає мінімальний елемент
sum(L)	Визначає суму елементів списку

Завдання для самостійної роботи.

1. Слово введене з клавіатури перетворити на список і визначити кількість літер в цьому слові.

2. Слово введене з клавіатури перетворити на список з якого видалено усі голосні літери.

3. Слово введене з клавіатури перетворити на список і визначити чи є в цьому списку елемент введений з клавіатури.

4. Слово введене з клавіатури перетворити на список і видалити зі списку 3-й елемент.

5. Створити список, який містить парні цифри в порядку зростання, а за ними непарні в порядку спадання.

6. Створити список із 5 довільних чисел від 1 до 100 і перевірити наявність в ньому числа введеного з клавіатури.

Запитання для самоконтролю

1. Який тип даних називається списком?
2. Чим список відрізняється від масиву?
3. Як можна створити список?
4. Яким чином організовується доступ до елементів списку?
5. Основні функції та методи стисків.

7.3. Кортежі.

Кортеж – це незмінний впорядкований набір даних не обов'язково одного типу. Елементи у кортежі беруть у круглі дужки та розділяються комами. В кортежі можуть бути елементів будь-якого типу.

('n', 'p', ('цвіт'), 13)

Кортеж відрізняється від списку тим, що його елементи, як і в рядку, не можна змінювати. Але кортежі мають ряд переваг:

1. Захист від випадкових змін.
2. Кортежі займають менше місця.
3. Кортежі можна використовувати як ключі словника.
4. Аргументи функції можна передавати в вигляді кортежу.
5. Кортежі доцільно використовувати для обміну значеннями між змінними.

Для створення кортежу існує кілька способів:

Перший спосіб: функція ***tuple()*** використовується для створення кортежів з інших об'єктів інших типів;

Другий спосіб: оператор ***( )*** використовується для створення порожнього кортежу.

Третій спосіб: генерувати новий кортеж на основі вихідного.

Для кортежів можна застосовувати усі методи і функції, що використовуються для списків, які не змінюють списку.

Завдання для самостійної роботи.

1. Слово введене з клавіатури перетворити на кортеж і визначити кількість літер в цьому слові.
2. Слово введене з клавіатури перетворити на кортеж, з якого видалено усі голосні літери.
3. Слово введене з клавіатури перетворити на кортеж і визначити чи є в цьому кортежі елемент введений з клавіатури.
4. Слово введене з клавіатури перетворити на кортеж і видалити зі списку 3-й елемент.
5. Створити кортеж, який містить парні цифри в порядку зростання, а за ними непарні в порядку спадання.
6. Створити кортеж із 5 довільних чисел від 1 до 100 і перевірити наявність в ньому числа введеного з клавіатури.

Запитання для самоконтролю

1. Який тип даних називається кортежем?
2. Чим кортеж відрізняється від списку?
3. Які переваги має кортеж?
4. Як можна створити кортеж?
5. Яким чином організовується доступ до елементів кортежу?
6. Основні функції та методи кортежів.

7.4. Словники.

Словник – невпорядкований змінюваний набір об’єктів *не обов’язково* одного типу, у якому доступ до елементів здійснюється за допомогою спеціальних ключів. Ключі в одному словнику повинні бути унікальними, тобто двох однакових ключів в одному словнику не може бути. Ключі повинні бути незмінюваного типу даних (число, рядок, кортеж). У словнику порядок елементів не важливий, тому для їх виклику вказується прив’язаний до нього унікальний ключ. Словник записується в фігурних дужках, як перелік пар

ключ – значення, розділених комою: {'a':2, 'b':4}. Пара ключ – значення розділяється двокрапкою.

У словник можна додавати нові елементи, вилучати та змінювати існуючі елементи.

Створити словник можна кількома способами.

Перший спосіб. Надати словнику ім’я (за загальноприйнятими правилами для об’єктів) і присвоїти цьому імені взяті в фігурні дужки зв’язані пари об’єктів *ключ:значення.*

Наприклад:

```
D = {'a': 1, 'b': 2}
print(D)
```

Результат

```
{'a': 1, 'b': 2}
```

Другий спосіб: Функція **dict()** дозволяє створити пустий словник.

Наприклад:

```
D = dict()
print(D)
```

Результат

```
{}
```

Третій спосіб: За допомогою генератора словників. Цей спосіб аналогічний до способу генерування списків.

Четвертий спосіб: З використання методу *fromkeys()*. Якщо задавати тільки ключі, то значення словника повертає None.

Приклад.

```
D = dict.fromkeys(['a', 'b', 'c'])
print(D)
```

Результат

```
{'a': None, 'b': None, 'c': None}
```

Можна задати будь-яке значення. В цьому випадку задане значення буде відповідати усім ключам.

Приклад.

D = dict.fromkeys(['a', 'b', 'c'], 100)

print(D)

Результат

{'a': 100, 'b': 100, 'c': 100}

Методи та функції словників

Метод, функція	Призначення
D.clear()	Видаляє усі ключі і значення зі словника
D.copy()	Робить копію словника
D.get(key[, default])	Повертає значення ключа, а якщо його немає то повертає default (за замовчуванням None)
D.items()	Повертає пари (ключ:значення)
D.keys()	Повертає усі ключі словника
D.pop()	Видаляє ключ і його значення зі словника
D.update()	Додає нові пари ключ:значення в існуючий словник
D.values()	Повертає усі значення словника

Щоб перевірити наявність ключа в словнику, використовують оператор *in*. Результатом перевірки буде одна із логічних величин *True* або *False.*

Синтаксис оператора:

ключ in ім'я_словника

Завдання для самостійної роботи.

1. Створити словник, у якому ключем є номер групи, а елементом прізвища старост груп, що навчаються на освітньому рівні бакалавра. Виконайте над словником наступні дії:

- виберіть із словника прізвища старост перших курсів;
- додайте в словник прізвища старост груп магістратури;
- виведіть прізвища усіх старост.
- перевірте чи є в словнику елемент з ключем '000' або '413'.

2. Створити словник, де ключ – прізвище студента, а значення ключа – список його семестрових оцінок з усіх дисциплін.

- Вивести значення словника за заданим з клавіатури ключем.

- Вивести тільки прізвища студентів.
- Визначити середній бал успішності кожного студента.
- Створити новий словник з прізвищами студентів та їх середніми балами успішності.

Запитання для самоконтролю

1. Який тип даних називається словником?
2. Які особливості мають словники?
3. Як можна створити словник?
4. Яким чином організовується доступ до елементів словника?
5. Основні функції та методи словників.

7.5. Множини.

Множина – це такий тип даних, що містить неупорядкований набір унікальних елементів. Множину в Python іноді називають «контейнером». Множину застосовують в тому випадку, якщо наявність елемента в наборі важливіша за порядок розміщення елементів та кількість їх повторень. До множини можуть входити елементи різних типів, але вони можуть бути лише незмінюваних типів даних: числа, рядкові величини, кортежі.

Множина в Python - «контейнер», що містить елементи, які не повторюються у випадковому порядку.

Множина задається, як перелік елементів, розділених комою та взятих в фігурні дужки: {1, 2, 3, 'Ура'}.

Створити множину можна кількома способами.

Перший спосіб. Порожня множина задається функцією set().

a = set()

Другий спосіб. Задати множину, перерахувавши її елементи, взяті в фігурні дужки «{}»:

>>> s1 = {1, 2, 3}

>>> s2 = {'У', 'р', 'а', 4}

Третій спосіб. Створити множину можна з елементів об’єкта, що може ітеруватися (діапазон, список, рядок, словник, кортеж, файл і т.д.), За допомогою функції *set([iterable]).* Але слід зауважити, що до множини можуть буди включені тільки унікальні елементи, причому додаються елементи у випадковому порядку. Тому таким способом множину доцільно створювати для видалення елементів, що повторюються

Наприклад:

```
a = set('Привіт')
print (a)
b = set(range(5))
print (b)
c = set('hello')
print (c)
```

Результат:

```
{'р', 'т', 'в', 'і', 'и', 'П'}
{0, 1, 2, 3, 4}
{'o', 'h', 'e', 'l'}  # літера «l» включена в множину один раз.
```

Операції з множинами

З множинами в Python можна виконувати операції такі ж, як і з іншими даними, і такі ж, як з множинами в математиці. Множина є змінюваним типом даних, тому до неї можна додавати елементи та видаляти їх. Разом з тим у мові Python над множинами передбачено виконання операцій: порівняння, об'єднання, перетину, різниці, симетричної різниці.

Методи та функції множин

Функції, методи	Призначення
len(a)	число елементів у множині (розмір множини).
s.isdisjoint(a)	істина, якщо множини **s** і **a** не мають спільних елементів і хиба, якщо мають
s == a	Перевірка, чи всі елементи **s** належать **a**, всі елементи **a** належать **s**.
s.issubset(a) або **s** <= **a**	Перевірка, чи всі елементи s належать a
s.issuperset(a) або **s** >= **a**	Перевірка, чи всі елементи a належать **s**
s.union(a, ...)	Об'єднання кількох множин.
s.intersection(a, ...) або **s & a & ...**	Переріз кількох множин.
s.difference(a)	Різниця множин s і a. Результатом буде множина, яка містити елементи множини s, які не належать множині a
s.symmetric_ difference(a)	Симетрична різниця множин s і a. Результат буде множина, яка містить елементи, що належать множинам s та a, але не належать обом множинам.
s.update(a)	Об’єднання множин s і a
s.intersection_ update(a)	Видаляє з множини s всі елементи, які не входять до множини a.
s.difference_update(a)	Видаляє з множини s всі елементи, які входять до множини a
set.symmetric_ difference_update(other)	Множина s буде містити симетричну різницю множин s і a.
s.add(x)	Додає елемент x до множини s.
set.remove(x)	Видаляє елемент x із множини s.
set.pop().	Видаляє перший елемент з множини s та повертає його значення як результат виконання функції. Оскільки множина – це невпорядкований набір елементів, то невідомо який саме елемент буде взятий за перший.
set.clear()	Очищує множину set
set.copy()	Копія множини

Завдання для самостійної роботи.

1. Створити множину з довільної кількості випадкових чисел. Виконайте над множиною наступні дії:

- Визначте кількість елементів у множині;
- Перевірте наявність числа введеного з клавіатури.

Запитання для самоконтролю

1. Який тип даних називається множиною?
2. Які особливості мають множини?
3. Як можна задати множину?
4. Основні операції над множинами.

8. РОБОТА З ФАЙЛАМИ

Мова програмування Python стандартними засобами може працювати з двома типами файлів:

- *текстовими***,** що мають формат txt або rtf. Відкрити їх можна будь-яким текстовим редактором.
- *бінарні,* що мають формат bin.

З файлами в Python можливі наступні дії:

- створення;
- відкриття;
- читання з файлу
- запис інформації до файлу;
- зміна імені;
- закриття.

8.1. Створення та відкриття файлів

Для того, щоб відкрити файл, передбачено вбудовану функцію open(). За її допомогою можна відкрити будь-який документ формату txt або rtf. Функція open() може мати наступні аргументи:

- file – повне ім’я файлу. Коротке ім’я можна використовувати в тому випадку, якщо він знаходиться у тій же директорії, що і Python. Ім’я файлу береться в лапки;
- mode – режим відкриття. За замовчуванням встановлюється режим «тільки для читання». Значення режиму береться в лапки.

Синтаксис команди:

Ідентифікатор = open(file, mode)

Наприклад:

file=open('text', 'r')

Цей рядок відкриває раніше створений файл для читання.

Якщо в ході відкривання файлу використати ім’я файлу, якого не існує, і режим ‘***w’,*** то буде створено новий файл.

Наприклад:

f=open ('new_text', 'w')

Перелік режимів наводиться в таблиці. Ці режими можуть бути з комбіновані.

Таблиця режимів відкриття файлів

Режим	Можливості
r	Тільки для читання, «віртуальний курсор» стоїть на початку файлу
rb	Файл для читання у двійковому форматі, «віртуальний курсор» стоїть на початку файлу
r+	Файл для читання та запису, «віртуальний курсор» стоїть на початку файлу
w	Запис інформації до файлу, «віртуальний курсор» стоїть на початку файлу. Якщо файл не знайдено, то створюється новий.
wb	Файл для запису у двійковому форматі, «віртуальний курсор» стоїть на початку файлу. Якщо файл не знайдено, то створюється новий.
w+	Файл для читання та запису, «віртуальний курсор» стоїть на початку файлу. Якщо файл не знайдено, то створюється новий.
wb+	Файл для читання і запису у двійковому форматі, «віртуальний курсор» стоїть на початку файлу. Якщо файл не знайдено, то створюється новий.
a	Файл для додавання інформації, «віртуальний курсор» стоїть в кінці файлу. Якщо файл не знайдено, то створюється новий.
ab	Файл для додавання у двійковому форматі, «віртуальний курсор» стоїть в кінці файлу. Якщо файл не знайдено, то створюється новий.
a+	Файл для додавання і читання інформації, «віртуальний курсор» стоїть в кінці файлу. Якщо файл не знайдено, то створюється новий.
ab+	Файл для додавання і читання у двійковому форматі, «віртуальний курсор» стоїть в кінці файлу. Якщо файл не знайдено, то створюється новий

Після того, як файл було відкрито, згенерується особливий файловий об’єкт і про нього можна отримати відомості. Для цього потрібно скористатися спеціальними атрибутами:

- .name – повертає ім’я відкритого файлу;
- .closed – показує, чи закритий документ;
- .mode – повертає режим доступу.

У функції open() є один недолік: після роботи не потрібно забувати закривати файл. Це потрібно для того, щоб зберегти дані непошкодженими.

Роботу з текстовим файлом можна здійснювати з використанням конструкцій ***with .. as ...*** Представлений вище приклад, можна переписати за допомогою такої конструкції. Він виглядатиме так:

with open('new_text', 'r') as file:

Перевага цього способу полягає в тому, що python самостійно закриває файл, і не потрібно про це завжди пам’ятати.

8.2. Режим читання з файлу

У Python є можливість читати інформацію з файлів. Важливо, щоб файл було відкрито у тому форматі, який дозволяє це зробити.

Метод *read()* призначений для читання файлу, відкритого за допомогою параметра «лише для читання». Якщо команда задається без аргументу в дужках то результатом буде увесь вміст файлу, поданого одним рядком.

Наприклад:

file=open('new_text', 'r')

file.read()

'Ще не вмерли України,\n ні слава, ні воля!\nЩе нам, браття-молодії,\nУсміхнеться доля!\nЗгинуть наші вороженьки,\nЯк роса на сонці;\nЗапанує́м, і ми браття, й ми\nУ своїй сторонці.'

Також цей метод може приймати в якості аргументу число символів, які будуть прочитані. Тобто, якщо аргументом буде число 10, то відобразиться перші 10 символів. При повторному використанні цієї команди, відобразяться наступні 10 символів і т.д.

Наприклад:

file=open('new_text', 'r')

file.read(15)

'Ще не вмерла Ук'

file.read(15)

'раїни,\n ні слава,'

Можна собі уявити, що по тексту рухається «віртуальний курсор» і починаючи з його розташування зчитується задана кількість символів і він переміщається на цю кількість символів. Його можна перемістити на задану позицію, скориставшись методом seek().

Наприклад:

file=open('new_text', 'r')

file.seek(22)

22

file.read(15)

'І слава, і воля'

Це не завжди зручний метод, особливо при роботі з текстами. Розглянемо метод readline(). Цей метод працює з рядками документу і особливо добре підходить для документів великих обсягів. Застосування цієї функції дозволяє не зчитувати весь файл повністю, а звертатися лише до потрібних рядків.

file=open('new_text', 'r')

file.readline()

'Ще не вмерли України,\n'

file.readline()

'ні слава, ні воля!\n'

Прочитати вміст усього файлу по рядках можна, виводячи рядки почергово в циклі.

Наприклад

```
with open('new_text', 'r') as file:
        for line in file.readlines():
           print(line)
```

Ще не вмерла України,

ні слава, ні воля!

Ще нам, браття-молодії,

Усміхнеться доля!

Згинуть наші вороженьки,

Як роса на сонці;

Запанує́м, і ми, браття,

У своїй сторонці.

8.3. Запис до файлу

Для запису файлів в Python використовується метод *write()*. При цьому важливо, щоб відкриття файлу здійснювалося для запису. Якщо файл не існував, його буде створено. В файл можна записати тільки рядкові величини. Усі інші типи даних, потрібно заздалегідь перетворити в рядки. Цим методом можна записувати і великі обсяги інформації. Для цього потрібно подати їх у вигляді списку рядків, що взяті в потрійні лапка.

```
file=open('new_text', 'w')
file.write('''Ще не вмерла України,
ні слава, ні воля!
Ще нам, браття-молодії,
Усміхнеться доля!
Згинуть наші вороженьки,
Як роса на сонці;
Запануєм і ми, браття,
У своїй сторонці.''')
```

У Python є можливість визначити позицію знаходження «віртуального курсору» у файлі. Для точного визначення його позиції передбачений метод **tell()**. Цей метод повідомляє на скільки байт від початку файлу перемістився «віртуальний курсор».

Синтаксис:

ім'я файлу.tell()

Потім можна перемістити «віртуальний курсор» на потрібну позицію скориставшись методом seek(). У якості аргументу вказати одне із трьох значень:

0 перейти на початок файлу,

1 – залишитися на поточній позиції,

2 – перейти в кінець файлу.

8.4. Зміна імені файлу

У мові Python є можливість змінювати назви файлів. Для цього використовується функція *rename()*. Використання цієї функції можливе тільки після імпорту спеціального модуля *os*.

Синтаксис:

*from os import**

rename('old_name', 'new_name')

У цьому записі *old_name* – поточне ім'я файлу, а *new_name* – нове ім'я файлу. Імена файлів обов'язково беруться у одинарні або подвійні лапки.

8.5. Закриття файлу

Після закінчення всіх маніпуляцій з відкритим файлом, його необхідно буде закрити. Це потрібно зробити для того, щоб вивільнити деякі ресурси та запобігти випадкових трат даних у незакритому файлі. Python може й самостійно закрити файл, після того як буде відкрито інший файл. Є кілька варіантів коректного закриття файлу:

1. Якщо файл відкрито з використанням конструкцій ***with .. as ...***, то його Python закриє коректно самостійно.
2. Файл відкритий іншими способами потрібно закрити з допомогою методу close().

Синтаксис:

Ім'я файлу.close()

Завдання для самостійної роботи.

1. Створити файл «python_text»:
   a. Ввести до нього куплет української пісні;
   b. Закрити файл;
   c. Змінити ім'я файлу на «Python_new_text»;
   d. Додати приспів;
   e. Вивести текст пісні;

f. Вивести тільки приспів.

Запитання для самоконтролю

1. Текстові файли яких форматів можна опрацьовувати в Python?
2. Які дії можна виконувати з файлами в Python?
3. Як створити файл в Python?
4. Що означає відкрити файл в Python?
5. Основні режими відкриття файлів?
6. Як організувати процес читання з файлу?
7. Як організувати процес запису в файл?
8. Що таке «віртуальний курсор»? Як ним керувати?
9. Як змінити ім’я файлу?
10. Як закрити файл?

ДЖЕРЕЛА ІНФОРМАЦІЇ.

1. Python 3.10.4 documentation. URL: https://docs.python.org/release/3.10.4/
2. Swaroop Chitlur. A Byte of Python URL: https://github.com/swaroopch/byte-of-python
3. Pythonicway – освітній портал URL: http://pythonicway.com/
4. Вчимося програмувати разом! Python – просто! URL: https://sites.google.com/comp-sc.if.ua/python-easy/%D0%B4%D0%BE%D0%BC%D0%B0%D1%88%D0%BD%D1%8F-%D1%81%D1%82%D0%BE%D1%80%D1%96%D0%BD%D0%BA%D0%B0?authuser=0
5. Козуб Г.О. Програмування : метод. рек. до лаб. робіт для студ. спец. 121 – „Інженерія програмного забезпечення” / Г. О. Козуб, Н. А. Семенов; Держ. закл. „Луган. нац. ун-т імені Тараса Шевченка”. – Старобільськ : ДЗ „ЛНУ імені Тараса Шевченка”, 2020. – 108 с.
6. Костюченко А.О. Основи програмування мовою Python: навчальний посібник. Ч.: ФОП Баликіна С.М., 2020. 180 с.
7. Рамський Ю.С., Цибко Г.Ю. Основи програмування (мова Паскаль): навчальний посібник. Київ, 2003. 140 с.
8. Створення GUI додатку Python tkinter. Метод grid URL: https://www.youtube.com/watch?v=_7F6FsbJepo
9. Юрченко І.В., Сікора В.С. Програмування мовою Python. Навчальний посібник. Чернівці. Чернівецький національний університет. 2022. 104 с.
10. Яковенко А. В. Основи програмування. Python. Частина 1: підручник для студ. спеціальності 122 "Комп’ютерні науки", спеціалізації "Інформаційні технології в біології та медицині" / А. В. Яковенко. Київ : КПІ ім. Ігоря Сікорського, 2018. – 195 с. URL: https://docplayer.net/134178913-Osnovi-programuvannya-python-chastina-1.html