Міністерство освіти і науки України
Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

М. О. Маркін, М. В. Філіппова, О. М. Маркіна

ОБЧИСЛЮВАЛЬНА ТЕХНІКА ТА ПРОГРАМУВАННЯ

ЧАСТИНА 3. ТЕХНОЛОГІЯ СИСТЕМНОГО ПРОГРАМУВАННЯ

Рекомендації до виконання розрахунково-графічної роботи

Рекомендовано Методичною радою КПІ ім. Ігоря Сікорського
як навчальний посібник для здобувачів ступеня бакалавра
за освітньою програмою «Інформаційні вимірювальні технології»
спеціальності 175 Інформаційно-вимірювальні технології

Електронне мережеве навчальне видання

Київ
КПІ ім. Ігоря Сікорського
2023

УДК 004.65:175
О-13

Укладачі: *Маркін Максим Олександрович,* канд. техн. наук, доцент
Філіппова Марина В'ячеславівна, канд. техн. наук, доцент
Маркіна Ольга Миколаївна, канд. техн. наук, доцент

Рецензент: *Батрак Є. О.*, канд. техн. наук, доцент кафедри інформаційних систем та технологій факультету інформатики та обчислювальної техніки КПІ ім. Ігоря Сікорського

Відповідальний редактор: *Защепкіна Н. М.,* д-р. техн. наук, професор

Гриф надано Методичною радою КПІ ім. Ігоря Сікорського
(протокол № 3 від 07.12.2023 р.)
за поданням Вченої ради приладобудівного факультету
(протокол № 9/23 від 30.10..2023 р.)

О-13 **Обчислювальна техніка та програмування. Частина 3. Технологія системного програмування** [Електронний ресурс] : рек. до виконання розрахунк.-граф. роботи : навч. посіб. для здобувачів ступеня бакалавра за освіт. програмою «Інформаційні вимірювальні технології» спец. 175 Інформаційно-вимірювальні технології / КПІ ім. Ігоря Сікорського ; уклад.: М. О. Маркін, М. В. Філіппова, О. М. Маркіна. – Електрон. текст. дані (1 файл). – Київ : КПІ ім. Ігоря Сікорського, 2023. – 50 с.

Посібник містить роз'яснення щодо виконання індивідуального семестрового завдання у вигляді розрахунково-графічної роботи з освітнього компонента «Обчислювальна техніка та програмування. Частина 3. Технологія системного програмування».

В навчальному посібнику представлені основні концепції об'єктно-орієнтованого програмування, розробки графічних інтерфейсів та подієво-орієнтованого програмування. Наведено загальні вимоги щодо оформлення та структури роботи, необхідні теоретичні відомості, приклад програмної реалізації типового завдання мовою програмування Python, критерії оцінювання роботи та список рекомендованої літератури. Запропоновано індивідуальні завдання для групи з 50 студентів.

Для здобувачів освіти за спеціальностями 152 «Метрологія та інформаційно-вимірювальна техніка» та 175 «Інформаційно-вимірювальні технології» всіх форм навчання.

УДК 004.65:175

Реєстр. № НП Обсяг 3,58 авт. арк.

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
проспект Берестейський, 37, м. Київ, 03056
https://kpi.ua
Свідоцтво про внесення до Державного реєстру видавців, виготовлювачів
і розповсюджувачів видавничої продукції ДК № 5354 від 25.05.2017 р.

ЗМІСТ

ВСТУП

Індивідуальне семестрове завдання – форма організації навчання, яка має на меті поглиблення, узагальнення та закріплення знань, які студенти отримують у процесі навчання, а також розвиток самостійного мислення та творчих здібностей студентів, підвищення їх пізнавальної активності, виховання наукового світогляду.

Основними видами індивідуальних семестрових завдань студентів є: реферат, розрахункова і розрахунково-графічна робота, домашня контрольна робота, курсова робота, курсовий проєкт.

Індивідуальне семестрове завдання є видом позааудиторної індивідуальної роботи студента навчального чи навчально-дослідного характеру. Це – завершена робота в межах навчальної програми дисципліни, яка виконується на основі знань, отриманих під час лекцій, семінарських, практичних та лабораторних занять, охоплює декілька тем або зміст навчальної дисципліни в цілому і завершується підготовкою відповідного письмового звіту.

Індивідуальне семестрове завдання виконується студентами самостійно із забезпеченням необхідних консультацій з окремих питань з боку викладача. Як правило, індивідуальні завдання виконуються окремо кожним студентом. У тих випадках, коли завдання мають комплексний характер, до їх виконання можуть залучатися декілька студентів.

Наявність позитивної оцінки, отриманої студентом за індивідуальне завдання, є необхідною умовою допуску до семестрового контролю з дисципліни.

Посібник призначений для якісної організації позааудиторної індивідуальної роботи студентів навчального та навчально-дослідного характеру.

ЗАГАЛЬНІ ПОЛОЖЕННЯ

Розрахунково-графічна робота – індивідуальне семестрове завдання, що передбачає вирішення конкретного практичного навчального завдання з використанням відомого та (або) самостійно вивченого теоретичного матеріалу. Значну частину такої роботи складає графічний матеріал, що виконується вручну, або з використанням засобів комп'ютерної графіки з дотриманням вимог відповідних нормативно-технічних документів.

Виконання розрахунково-графічної роботи сприяє поглибленому вивченню студентом теоретичного матеріалу, систематизації отриманих теоретичних знань, формуванню вмінь використання теоретичних положень дисципліни для розв'язання конкретних практичних завдань.

Виконання роботи здійснюється мовою Python у довільному програмному середовищі, яке дозволяє розробляти консольні додатки.

Мета роботи – опанування парадигми об'єктно-орієнтовного програмування; стандартних засобів мови Python для роботи з динамічною пам'яттю; вдосконалення навичок системного програмування мовою Python для вирішення задач створення прикладних програм.

1 ВИМОГИ ДО ОФОРМЛЕННЯ ТА СТРУКТУРИ РОБОТИ

1.1 Загальні вимоги

Роботу готують у вигляді електронного документу або друкують за допомогою принтера на одному боці аркуша білого паперу формату А4 (210х297 мм) через півтора міжрядкових інтервали до тридцяти рядків на сторінці, 14-м кеглем. Таблиці та ілюстрації можна подавати на аркушах формату А3.

До загального обсягу роботи не входять додатки, список використаних джерел, таблиці та рисунки, які повністю займають площу сторінки. Але всі сторінки зазначених структурних одиниць підлягають суцільній нумерації.

Текст роботи необхідно друкувати, залишаючи поля таких розмірів: ліве – не менше 20 мм, праве – не менше 10 мм, верхнє – не менше 20 мм, нижнє – не менше 20 мм.

Допускається наявність не більше двох виправлень на одній сторінці.

Текст основної частини поділяють на розділи, підрозділи, пункти та підпункти.

Заголовки структурних частин "ЗМІСТ", "ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ", "ВСТУП", "ВИСНОВКИ", "ДОДАТКИ", "СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ", НАЗВИ РОЗДІЛІВ друкують великими літерами по центру рядка. Заголовки підрозділів друкують маленькими літерами (крім першої великої) з абзацного відступу. Крапку є кінці заголовка не ставлять. Якщо заголовок складається з двох або більше речень, їх розділяють крапкою. Заголовки пунктів друкують маленькими літерами (крім першої великої) з абзацного відступу в розрядці у підбір до тексту. В кінці заголовка, надрукованого в підбір до тексту, ставиться крапка. Відстань між заголовком (за винятком заголовка пункту) та текстом повинна дорівнювати 2 інтервали.

Кожну структурну частину роботи треба починати з нової сторінки.

1.2. Нумерація

Нумерацію сторінок, розділів, підрозділів, пунктів, підпунктів, рисунків (малюнків), таблиць, формул подають арабськими цифрами без знака №. Першою сторінкою роботи є титульний аркуш, який включають до загальної нумерації сторінок. На титульному аркуші номер сторінки не ставлять, на наступних сторінках номер проставляють у правому верхньому куті сторінки без крапки в кінці.

Такі структурні частини, як зміст, вступ, висновки, список використаної літератури, джерела, додатки не мають порядкового номера. Всі аркуші, на яких розміщені згадані структурні частини роботи, нумерують звичайним чином. Не нумерують лише їх заголовки, тобто не можна друкувати: "1. ВСТУП" або "Розділ 6. ВИСНОВКИ". Розділи нумерують у межах викладення суті роботи і позначають арабськими цифрами без крапки, починаючи з цифри «1».

Підрозділи нумерують у межах кожного розділу. Номер підрозділу складається з номера розділу і порядкового номера підрозділу, між якими ставлять крапку. Після номера підрозділу крапку не ставлять, наприклад: "2.3" (третій підрозділ другого розділу). Потім у тому ж рядку наводять заголовок підрозділу.

Пункти нумерують у межах кожного підрозділу. Номер пункту складається з порядкових номерів розділу, підрозділу, пункту, між якими ставлять крапку. В кінці номера повинна стояти крапка, наприклад: "1.3.2." (другий пункт третього підрозділу першого розділу). Потім у тому ж рядку наводять заголовок пункту. Пункт може не мати заголовка.

Підпункти нумерують у межах кожного пункту за такими ж правилами, як пункти.

Ілюстрації (фотографії, креслення, схеми, графіки, карти) і таблиці необхідно наводити безпосередньо після тексту, де вони згадані вперше, або на наступній сторінці. Ілюстрації і таблиці, розміщені на окремих сторінках,

включають до загальної нумерації сторінок. Таблицю, рисунок або креслення, розміри якого більше формату А4, враховують як одну сторінку і розміщують у відповідних місцях після згадування у тексті або в додатках. Ілюстрації позначають словом «Рисунок» і нумерують послідовно в межах розділу, за винятком ілюстрацій, поданих у додатках. Номер ілюстрації повинен складатися з номера розділу і порядкового номера ілюстрації, між якими ставиться крапка. Наприклад: Рисунок 1.2 – Назва рисунку (другий рисунок першого розділу). Номер ілюстрації, її назва і пояснювальні підписи розміщують послідовно під ілюстрацією. Якщо в розділі подано одну ілюстрацію, то її нумерують за загальними правилами. Ілюструвати роботу слід, виходячи із певного загального задуму, за ретельно продуманим тематичним планом, що допомагає уникнути ілюстрацій випадкових, пов’язаних із другорядними деталями тексту і запобігти невиправданим пропускам ілюстрацій до найважливіших тем. Кожна ілюстрація має відповідати тексту, а текст – ілюстрації.

Не варто оформлювати посилання на ілюстрації як самостійні фрази, в яких лише повторюється те, що міститься у підписі. У тому місці, де викладається тема, пов’язана з ілюстрацією, і де читачеві треба вказати на неї, розміщують посилання у вигляді виразу в круглих дужках "(рис.3.1)" або зворот типу: "...як це видно з рис. 3.1" або "... як це показано на рис. 3.1".

Таблиці нумерують послідовно (за винятком таблиць, поданих у додатках) в межах розділу. В правому верхньому куті над відповідним заголовком таблиці розміщують напис " Таблиця" із зазначеннями номера. Номер таблиці повинен складатися з номера розділу і порядкового номера таблиці, між якими ставиться крапка, наприклад: "Таблиця 1.2" (друга таблиця першого розділу). Кожна таблиця повинна мати назву, яку розміщують над таблицею і друкують симетрично до тексту. Назву наводять жирним шрифтом. Текст у таблиці варто друкувати кеглем 12 з одинарним інтервалом. Якщо в розділі є лише одна таблиця, її нумерують за загальними правилами. При перенесенні частини таблиці на інший аркуш (сторінку) слово "Таблиця" і номер її вказують один раз

справа над першою частиною таблиці, над іншими частинами пишуть слова "Продовж, табл." і вказують номер таблиці, наприклад: "Продовж. табл.1.2". Якщо цифрові або інші дані в якому-небудь рядку таблиці не подають, то в ньому ставлять прочерк.

При використанні формул необхідно дотримуватися певних правил. Формули (якщо їх більше однієї) нумерують у межах розділу. Номер формули складається з номера розділу і порядкового номера формули в розділі, між якими ставлять крапку. Номери формул пишуть біля правого поля аркуша на рівні відповідної формули в круглих дужках, наприклад: (3.1) (перша формула третього розділу).

Найбільші, а також довгі і громіздкі формули, котрі мають у складі знаки суми, добутку, диференціювання, інтегрування, розміщують на окремих рядках. Це стосується також і всіх нумерованих формул. Для економії місця кілька коротких однотипних формул, відокремлених від тексту, можна подати в одному рядку, а не одну під одною. Невеликі і нескладні формули, що не мають самостійного значення, вписують всередині рядків тексту. Пояснення значень символів і числових коефіцієнтів треба подавати безпосередньо під формулою в тій послідовності, в якій вони наведені у формулі. Значення кожного символу і числового коефіцієнта треба подавати з нового рядка. Перший рядок пояснення починають зі слова "де" без двокрапки.

Двокрапку перед формулою ставлять лише у випадках, передбачених правилами пунктуації, а) у тексті перед формулою є узагальнююче слово; б) цього вимагає побудова тексту, що передує формулі.

Рівняння і формули треба виділяти з тексту вільними рядками. Вище і нижче кожної формули потрібно залишити не менше одного вільного рядка. Якщо рівняння не вміщується в один рядок, його слід перенести після знаку рівності (=), або після знаків плюс (+), мінус (–), множення. Нумерувати слід лише ті формули, на які є посилання в наступному тексті. Інші нумерувати не рекомендується. Порядкові номери позначають арабськими цифрами в круглих

дужках біля правого поля сторінки без крапок від формули до її номера. Номер, який не вміщується у рядку з формулою, переносять у наступний нижче формули. Номер формули при її перенесенні вміщують на рівні останнього рядка. Якщо формулу взято в рамку, то номер такої формули записують зовні рамки з правого боку навпроти основного рядка формули. Номер формули-дробу подають на рівні основної горизонтальної риски формули.

Загальне правило пунктуації в тексті з формулами таке: формула входить до речення як його рівноправний елемент. Тому в кінці формул і в тексті перед ними розділові знаки ставлять відповідно до правил пунктуації.

Розділовими знаками між формулами, котрі йдуть одна під одною і не відокремлені текстом, можуть бути кома або крапка з комою безпосередньо за формулою до її номера.

1.3 Цитування та посилання на використані джерела

При написанні роботи студент повинен посилатися на цитовану літературу, або на ту літературу, звідки взято ідеї, висновки, задачі, питання, вивченню яких присвячена робота. Посилатися слід на останні видання публікацій.

Якщо використовують відомості, матеріали з монографій, оглядових статей, інших джерел з великою кількістю сторінок, тоді в посиланні необхідно точно вказати номери сторінок, ілюстрацій, таблиць, формул з джерела, на яке є посилання.

Посилання в тексті на літературні джерела слід зазначати порядковим номером за переліком посилань, виділеним двома квадратними дужками, наприклад, "... у працях [1-5].

Для підтвердження власних аргументів посиланням на авторитетне джерело або для критичного аналізу того чи іншого друкованого твору слід наводити цитати. Науковий етикет потребує точно відтворювати цитований

текст, бо найменше скорочення наведеного витягу може спотворити зміст, закладений автором.

Загальні вимоги до цитування такі:

а) текст цитати починається і закінчується лапками і наводиться в тій граматичній формі, в якій він поданий у джерелі, із збереженням особливостей авторського написання. Наукові терміни, запропоновані іншими авторами, не виділяються лапками, за винятком тих, що викликали загальну полеміку. У цих випадках використовується вираз "так званий";

б) цитування повинно бути повним, без довільного скорочення авторського тексту та без перекручень думок автора. Пропуск слів, речень, абзаців при цитуванні допускається без перекручення авторського тексту і позначається трьома крапками, Вони ставляться у будь-якому місці цитати (на початку, всередині, наприкінці). Якщо перед випущеним текстом або за ним стояв розділовий знак, то він не зберігається;

в) кожна цитата обов’язково супроводжується посиланням на джерело;

г) при непрямому цитуванні (переказі, викладі думок інших авторів своїми словами), що дає значну економію тексту, слід бути гранично точним у викладенні думок автора, коректним щодо оцінювання його результатів і давати відповідні посилання на джерело;

д) якщо необхідно виявити ставлення до окремих слів або думок з цитованого тексту, то після них у круглих дужках ставлять знак оклику або знак питання.

1.4 Додатки

Додатки оформлюють як продовження роботи на наступних її сторінках або у вигляді окремої частини (книги), розміщуючи їх у порядку появи посилань у тексті. Якщо додатки оформлюють на наступних сторінках, кожний такий додаток повинен починатися з нової сторінки. Додаток повинен мати заголовок, надрукований угорі малими літерами з першої великої симетрично відносно

тексту сторінки. Посередині рядка над заголовком малими літерами з першої великої друкується слово "Додаток Б" і велика літера, що позначає додаток.

Додатки слід позначати послідовно великими літерами української абетки, за винятком літер Ґ, Є, І, Ї, Й, О, Ч, Ь, наприклад, додаток А, додаток Б. Один додаток позначається як додаток.

При оформленні додатків окремою частиною (книгою) на титульному аркуші під назвою роботи друкують великими літерами слово "ДОДАТКИ".

Текст кожного додатка за необхідності може бути поділений на розділи й підрозділи, які нумерують у межах кожного додатка. У цьому разі перед кожним номером ставлять позначення додатка (літеру) і крапку, наприклад, А.2 - другий розділ додатка А; В.3.1 – перший підрозділ третього розділу додатка В.

Ілюстрації, таблиці та формули, розміщені в додатках, нумерують у межах кожного додатка, наприклад: рис. Д.1.2-другий рисунок першого розділу додатка Д); формула (А.1)- перша формула додатка А.

2 ІНДИВІДУАЛЬНІ ЗАВДАННЯ

1. Створіть клас "Користувач", який має атрибути name, email та метод для виведення інформації про користувача на екран.
2. Розробіть програму, яка обробляє подію натискання на кнопку і виводить повідомлення на екран.
3. Створіть графічний інтерфейс для калькулятора з кнопками для чисел та операцій (+, -, *, /).
4. Створіть клас "Товар", який має атрибути name, price та метод для виведення інформації про товар.
5. Розробіть інтерфейс для створення та відображення списку товарів у вікні програми.
6. Створіть програму для вибору кольору і відображення вибраного кольору на екрані.
7. Реалізуйте можливість вибору дати та часу за допомогою графічного інтерфейсу.
8. Створіть клас "Електронний годинник", який може відображати поточний час у вікні програми.
9. Розробіть головне меню з різними командами і підменю для окремих команд.
10. Створіть вкладки у вікні програми з різними панелями для кожної вкладки.
11. Розробіть діалогове вікно для введення текстового повідомлення користувачем.
12. Зробіть можливість перетягувати та розміщувати елементи на вікні програми.
13. Створіть інтерфейс для вибору файлу на комп'ютері та відображення шляху до вибраного файлу.
14. Розробіть інтерфейс для роботи з базою даних, включаючи додавання, видалення та редагування записів.

15. Створіть календарний інтерфейс для планування подій та завдань.
16. Реалізуйте можливість відображення графіків та діаграм на графічному інтерфейсі.
17. Створіть інтерфейс для керування відеопрогравачем (відтворення, пауза, перемотка).
18. Зробіть можливість вставки та відображення зображень на інтерфейсі.
19. Розробіть інтерфейс для роботи з геоданими та відображенням географічних об'єктів на карті.
20. Створіть вікно для введення інформації про користувача (ім'я, адреса, телефон тощо).
21. Розробіть програму для створення та редагування векторних графічних файлів (SVG).
22. Розробіть інтерфейс для роботи з текстовими та числовими таблицями (таблицями даних).
23. Створіть інтерфейс для управління роботом або дроном за допомогою графічних команд.
24. Реалізуйте можливість збереження та відновлення стану програми з використанням файлів конфігурації.
25. Створіть інтерфейс для роботи з мережевими пристроями (перевірка доступності, конфігурація).
26. Розробіть інтерфейс для роботи з мультимедійними файлами (аудіо та відео).
27. Створіть програму для створення та редагування 3D-моделей та їх відображення в інтерфейсі.
28. Розробіть інтерфейс для роботи з базами даних та виконання SQL-запитів.
29. Створіть інтерфейс для використання камери та захоплення фотографій або відео.
30. Створіть графічний інтерфейс для відображення статистики та графіків в реальному часі.

31. Створіть інтерфейс для роботи з сенсорними даними (температура, вологість, тощо).
32. Створіть графічний інтерфейс для відображення та аналізу фінансових даних (біржовий термінал).
33. Розробіть графічний інтерфейс для відображення та аналізу медичних даних та зображень (медичний інформаційний система).
34. Створіть графічний інтерфейс для генерації випадкових чисел та відображення їх на екрані.
35. Розробіть програму для створення та редагування текстових документів з можливістю збереження змін.
36. Створіть інтерфейс для відображення та аналізу метеорологічних даних, включаючи температуру та вологість.
37. Реалізуйте можливість динамічного створення та видалення графічних об'єктів на полотні програми.
38. Створіть інтерфейс для відстеження та аналізу акційних ринків та фінансових індексів.
39. Розробіть графічний інтерфейс для взаємодії з роботом-повітряним дроном.
40. Створіть програму для аналізу та візуалізації даних з датчиків руху.
41. Реалізуйте можливість створення ігор на основі текстових інтерфейсів та їх графічного відображення.
42. Створіть інтерфейс для аналізу та відображення даних з датчиків серцевого ритму.
43. Розробіть програму для автоматизованої генерації звітів на основі вхідних даних.
44. Створіть графічний інтерфейс для керування світлодіодними світильниками в приміщенні.
45. Реалізуйте можливість створення та редагування анімацій та мультфільмів.

46. Створіть інтерфейс для відображення та аналізу даних з супутникових систем позиціювання.
47. Розробіть програму для створення та аналізу графіків функцій та математичних моделей.
48. Створіть інтерфейс для керування роботом-маніпулятором та його рухом.
49. Реалізуйте можливість роботи з великими обсягами даних та їх обробки через графічний інтерфейс.
50. Створіть графічний інтерфейс для навчання та тестування алгоритмів штучного інтелекту та машинного навчання.

Структура звіту

Звіт про виконання РГР (розрахунково-графічної роботи) має бути структурованим і докладним документом, який відображає вашу роботу та досягнуті результати. Звіт повинен містити наступні розділи та інформацію:

1. Титульний аркуш:
   - Назва виконаної роботи (РГР).
   - ПІБ студента.
   - Назва навчального закладу.
   - Спеціальність і група студента.
   - ПІБ викладача або наукового керівника.
   - Дата виконання РГР.
2. Анотація:
   - Короткий опис мети та завдань РГР.
   - Основні результати та висновки.
3. Зміст:
   - Перелік основних розділів та підрозділів звіту з відповідними сторінками.
4. Вступ:
   - Визначення та пояснення мети РГР.

- Огляд теми та основних завдань.
- Обґрунтування актуальності роботи.

5. Теоретична частина:
   - Огляд літературних джерел та теоретичних основ.
   - Пояснення використаних методів та підходів.
6. Практична частина:
   - Опис методів та алгоритмів, використованих у роботі.
   - Розгляд процесу розробки програмного коду (якщо РГР включає програмування).
   - Відображення виконаних розрахунків, графічних діаграм, зображень тощо.
   - Результати та висновки:
   - Представлення отриманих результатів у числовій, графічній або іншій формі.
   - Висновки на основі результатів роботи.
   - Висловлення особистої оцінки і рекомендацій для подальших досліджень.
7. Список використаних джерел:
   - Перелік літературних джерел, статей, інтернет-ресурсів та іншої інформації, які ви використали під час підготовки РГР.
8. Додатки (за потреби):
   - Додаткові матеріали, такі як код програми, скріншоти, графіки, таблиці, тощо.

Звіт має бути оформлений чітко та логічно, з правильними посиланнями та форматуванням.

До звіту ставляться такі вимоги:

- структура звіту повинна бути строго дотримана;
- звіт повинен бути відформатований;

- кожна структурна частина повинна починатися з нової сторінки;
- всі сторінки в звіті, крім титульної, повинні бути пронумеровані;
- звіт повинен містити опис всіх особливостей реалізації коду.

3 ТЕОРЕТИЧНІ ВІДОМОСТІ

Базові поняття ООП

Python – це об'єктно-орієнтована мова програмування. Основною ідеєю об'єктно-орієнтованого програмування (ООП) є об'єднання в єдине ціле даних і функціоналу для їх обробки в певний єдиний тип даних. Такий комбінований тип даних називається класом.

ООП базується на чотирьох основних поняттях: абстрагування, інкапсуляція, успадкування і поліморфізм.

Абстрагування – це виділення самих істотних характеристик об'єкта, які відрізняють його від всіх інших об'єктів і чітко визначають його з точки зору подальшого розгляду та аналізу.

Інкапсуляція – це об’єднання в одному класі даних і функцій для їх обробки.

Успадкуванням називається властивість класів створювати нові (породжені) класи на основі вже існуючих (базових). Породжені класи успадковують властивості базових класів. При цьому успадковані властивості можуть бути модифіковані. Крім того у породжені класи можуть бути добавлені нові властивості.

Поліморфізм – це властивість споріднених класів, породжених від загального базового класу, вирішувати схожі за змістом задачі різними способами.

В програмі можна створювати змінні, які є екземплярами того чи іншого класу. Такі екземпляри прийнято називати об’єктами класу. Кожен об’єкт повторює структуру свого класу, тобто має такі ж змінні (поля) і такі ж функції (методи). Змінні касу називають полями, а функції класу – методами. Поля і методи разом називають атрибутами класу. Поля можуть бути двох типів: вони можуть локальними для кожного конкретного екземпляру класу, а можуть бути загальними для всіх екземплярів даного класу. Їх називають змінними екземпляру і змінними класу відповідно.

Створення власного класу

Створення власного класу без успадкування

При програмуванні можна використовувати стандартні класи, які є складовою частиною мови програмування Python, а можна створювати свої власні класи. Створити клас можна за допомогою ключового слова *class*:

```
#визначаємо клас з іменем point
class point:
    #поля класу – координати крапки у просторі за замовчуванням
(0,0,0)
    x=0
    y=0
    z=0
    #методи класу:
    #get_coord – дозволяє прочитати координати
    def get_coord(self):
        return [self.x, self.y, self.z]
    #set_coord – дозволяє встановити координати
    def set_coord(self,x,y,z):
        self.x=x
        self.y=y
        self.z=z
```

Створимо екземпляр класу point і попрацюємо з ним:

```
a=point()
#переконаємось, що за замовчуванням координати крапки (0,0,0)
a.get_coord()
```

На екрані з’явиться повідомлення:

```
[0, 0, 0]
```

А тепер змінимо координати і виведемо їх значення на екран:

```
a.set_coord(1,2,3)
a.get_coord()
```

На екрані з’явиться повідомлення:

```
[1, 2, 3]
```

Зверніть увагу на те, що кожен метод має аргумент self, який є покажчиком на конкретний екземпляр класу. Він є обов’язковим для всіх методів класу. При визові методу його вказувати не треба, транслятор підставить його автоматично. Також зверніть увагу на те, що дякуючи

покажчику self не виникає ніякого конфлікту між внутрішніми змінними об'єкту (x, y, z) і параметрами методу set_coord з такими самими іменами.

Створення власного класу на базі вже існуючого. Композиція та успадкування

Створення власних класів на базі вже існуючих може значно прискорити процес розробки програми. У мові програмування Python для реалізації власних класів на базі вже існуючих є два механізми: композиція і успадкування.

Композиція – це коли до складу створюваного класу входять екземпляри інших класів. Використовувані класи повинні бути визначені в програмі до їх першого використання. Наприклад можна створити клас section (відрізок) на основі композиції двох екземплярів класу point:

```
class section():
    p1=point()
    p2=point()

    def get_coords(self):
        return [self.p1.get_coord(), self.p2.get_coord()]

    def set_coords(self,x1,y1,z1,x2,y2,z2):
        self.p1.set_coord(x1,y1,z1)
        self.p2.set_coord(x2,y2,z2)

    def length(self):
        tmp=self.get_coords()
        len=(tmp[0][0]-tmp[1][0])**2+(tmp[0][1]-
tmp[1][1])**2+(tmp[0][2]- tmp[1][2])**2
        return len**(0.5)
```

При успадкуванні клас будується на основі базового класу наслідуючи його властивості. При цьому є можливість модифікувати успадковані властивості відповідно до потреб створюваного класу.

Наведемо приклад. Побудуємо на базі класу point, який моделює точку у тривимірному просторі, клас four_dimentional_point, моделює точку у чотиривимірному просторі, де четверта координата, це час. При розробці класу використаємо механізм успадкування.

```
#задаємо ім'я класу - four_dimentional_point,
```

```
#а в дужках вказуємо базовий клас point
#новий клас успадкував структуру базового класу
class four_dimentional_point(point):
    #додамо ще одну координату до структури нового класу
    t=0

    #оскільки змінилась кількість координат,
    #нам необхідно модифікувати методи,
    def get_coord(self):
        return [self.x, self.y, self.z, self.t]

    def set_coord(self,x,y,z,t):
        self.x=x
        self.y=y
        self.z=z
        self.t=t
```

Класс four_dimentional_point можна реалізувати дещо по іншому. У попередньому прикладі методи даного класу було повністю переписано. Якщо методи дуже великі і складні це може бути недоцільно. Тоді викликати аналогічний метод базового класу, а потім виконати додаткові дії.

```
class four_dimentional_point(point):
    t=0
    #даний метод реалізуємо так само, як і в попередньому
прикладі
    def get_coord(self):
         return [self.x, self.y, self.z, self.t]

    #а цей метод реалізуємо через виклик
    #аналогічного методу базового класу
    def set_coord(self,x,y,z,t):
        point.set_coord(self,x, y, z)
        self.t=t
```

У мові програмування Python можливе множинне наслідування від кількох класів одночасно. В цьому випадку їх імена перераховуються в дужках через кому:

```
class person:
    name='unknown'

class adress:
    adress='unknown'

class full_inform(person, adress):
    pass
```

Тепер виконаємо кілька команд :

```
s=full_inform()
s.name # на екрані зявиться напис – 'unknown'
s.adress # на екрані зявиться напис – 'unknown'
```

Ініціалізація об’єктів. Метод init

При використанні ООП часто виникає ситуація, коли необхідно виконати деякі обов’язкові при створенні об’єкту. Для реалізації такої можливості у мові програмування Python передбачений метод __init__. Цей метод виконується першим при створенні об’єкту. Щоб скористатися цією можливістю достатньо при описі класу визначити метод __init__.

Модифікуємо наш клас point таким чином, щоб можна було створювати екземпляри не тільки із значенням координат за замовчуванням, але і з наперед заданими значеннями координат. Для цього введемо до класу метод __init__ з аргументами за замовчуванням.

```
class point:
    def  init (self,x=0,y=0,z=0):
        self.x=x
        self.y=y
        self.z=z

def get_coord(self):
    return [self.x, self.y, self.z]

def set_coord(self,x,y,z):
    self.x=x
    self.y=y
    self.z=z
```

Створимо екземпляр класу і виконаємо кілка команд:

```
#створюємо екземпляр класу із значеннями за замовчуванням
n=point()
#перевіримо значення координат n.get_coord() #на екрані зявиться – [0, 0, 0]
```

```
#а тепер створимо екземпляр з координатами (1, 2, 3)
d=point(1, 2, 3)
# перевіримо значення координат d.get_coord()#на екрані зявиться – [1, 2, 3]
```

А ось так можна, використовуючи метод __init__, модифікувати приклад з наслідування від двох класів із попереднього підрозділу:

```
class person:
    #name='unknown'
    def __init__(self, name='unknown'):
        self.name=name

class adress:
    #adress='unknown'
    def __init__(self, adress='unknown'):
        self.adress=adress

class full_inform(person, adress):
    def __init__(self,name='unknown', adress='unknown'):
        self.name=name
        self.adress=adress
```

Тепер можна створювати екземпляри класу full_inform із заданими значеннями полів:

```
s = full_inform('Василь Пупкін', 'Україна, Київ, вул. Хрещатик 26, кв. 7')
s.name #на екрані зявиться – 'Василь Пупкін'
s.adress #на екрані зявиться – 'Україна, Київ, вул. Хрещатик 26, кв. 7'
```

Поліморфізм

Фактично поліморфізм – це можливість обробки різних типів даних (які належать до різних класів) за допомогою однієї функції, або методу. У попередньому підрозділі ми вже спостерігали поліморфізм між класами, пов'язаними успадкуванням. У кожного класу може бути, наприклад, своя функція ініціалізації. Крім того ми змінювали методи виведення на екран інформації про об'єкт. Все це приклади поліморфізму. Однак класи не обов'язково повинні бути пов'язані успадкуванням. Поліморфізм як один з ключових елементів ООП існує незалежно від успадкування. Класи можуть

бути не родинними, але мати однакові методи, як в наступному прикладі:

```
#визначимо два різних класи, які мають метод з однаковим імя'м – total
class T1:
    n=10

    def total(self, N):
        self.total = int(self.n) + int(N)

class T2:
    def total(self,s):
        self.total = len(str(s))

#створимо по екземпляру кожного з цих класів
t1 = T1()
t2 = T2()
#звернемось до методу total для кожного з об’єктів і порівняємо результат
t1.total(45)
t2.total(45)
print(t1.total) # на екрані зявиться – 55
print(t2.total) # на екрані зявиться – 2
```

Як бачимо результат різний для різних типів, що власне і є проявом поліморфізму. Поліморфізм дає можливість реалізовувати одноманітні інтерфейси для об'єктів різних класів. Наприклад, різні класи можуть передбачати різний спосіб виведення на екран інформації об'єктів. При цьому вони можуть мати однакову назву методу виведення на екран, що спрощує аналіз програми, робить код більш зрозумілим.

Перенавантаження операторів

Одним з прикладів поліморфізму можуть слугувати математичні оператори, які на основі однакового інтерфейсу по різному обробляють цілі числа, числа з плаваючою крапкою та комплексні числа, що досягається за рахунок перенавантаження цих операторів.

Перенавантаження операторів і функцій – це ефективний засіб реалізації поліморфізму. Мова програмування Python має ряд засобів для спрощення процесу перенавантаження операторів і функцій.

Для прикладу спробуємо перенавантажити функцію print для відображення на екрані повідомлення про об’єкт створеного програмістом (не вбудованого в мову) класу.

```
#опишемо клас class A:
def __init__(self, v1, v2):
    self.field1 = v1
    self.field2 = v2
#створимо екземпляр класу a = A(3, 4)
#спробуємо вивести його на екран за допомогою функції print
print(a)
```

На екрані з’явиться, щось на кшталт наведеного нижче напису:

```
<main.A object at 0x7f840c8acfd0>
```

Це нас не влаштовує ми хочемо отримати більш інформативне повідомлення. Тому перенавантажимо функцію print. Для цього треба додати до класу спеціальний метод __str__(). Цей метод буде формувати рядок, який буде виводити функція print():

```
class A:
    def __init__(self, v1, v2):
        self.field1 = v1
        self.field2 = v2

    def __str__(self):
        return str(self.field1) + " " + str(self.field2)
```

А тепер створимо екземпляр класу і виведемо його на екран за допомогою функції print():

```
a = A(3, 4)
print(a) # на екрані зявиться напис – (3 4)
```

Як бачимо, механізм перенавантаження у мові Python реалізовано інакше, ніж у C++. Оскільки у мові Python всі дані – це об’єкти, а всі функції

– це методи, то для перенавантаження методів і операторів використовують спеціальні вбудовані у мову функції, які також є методами. Розглянемо деякі з них.

__add__(self, other) – метод перенавантаження оператора складання *x + y;* викликається *x.__add__(y)*;

__and__(self, other) – викликається при виконанні операції побітове І (*x & y*);

__bool__(self) – викликається при перевірці істинності. Якщо цей метод не визначений, викликається метод __len__ (об'єкти, що мають ненульову довжину, вважаються дійсними);

__bytes__(self) – викликається функцією *bytes* при перетворенні до типу байт;

__call__(self [, args ...]) – здійснює виклик екземпляра класу як функції;

__contains__(self, item) – здійснює перевірку на приналежність елементу до контейнера (*item in self*);

__del__() – деструктор об'єктів класу, викликається при видаленні об'єктів;

__delattr__(self, name) – викликається при видаленні атрибута (*del obj.name*);

__delitem__(self, key) – викликається при видаленні елемента за індексом;

__divmod__(self, other) – здійснює обчислення частки і залишку від цілочислового ділення (*divmod(x, y)*);

__eq__(self, other) – викликається при виконанні операції *x == y* викликає метод *x.__eq__(y);*

__floordiv__(self, other) – викликається, коли має місце операція цілочисельного ділення (*x // y*);

__format__(self, format_spec) – використовується функцією *format* (а також методом *format* у рядків);

__ge__(self, other) – викликається при виконанні операції *x ≥ y* викликає метод *x.__ge(y)*;

__getattr__(self, name) – викликається, коли атрибут реалізації класу не знайдений в звичайних місцях (наприклад, у примірника немає методу з такою назвою);

__getitem__(self, key) – викликається, коли має місце доступ за індексом (або за ключем);

__gt__(self, other) – викликається при виконанні операції *x > y* викликає метод *x.__gt__(y)*;

__hash__(self) – викликається при отриманні хеш-суми об'єкту, наприклад, для додавання в словник;

__init__() – конструктор об'єктів класу, викликається при створенні об'єктів;

__iter__(self) – повертає ітератор для контейнера;

__le__(self, other) – викликається при виконанні операції *x ≤ y* викликає метод *x.__le__(y)*;

__len__(self) – викликається при визначенні довжини об'єкта;

__lshift__(self, other) – викликається при виконанні операції бітовий зсув вліво (*x << y*);

__lt__(self, other) – викликається при виконанні операції *x <y* викликає метод *x.__lt__(y)*;

__mul__(self, other) – викликається при виконанні операції множення *(x * y)*;

__mod__(self, other) – викликається при виконанні операції залишок від ділення (*x% y*);

__ne__(self, other) – викликається при виконанні операції *x != y* викликає метод *x.__ne__(y);*

__new__(cls [...]) – керує створенням екземпляру класу. В якості обов'язкового аргументу приймає клас. Повертає екземпляр класу для його подальшої його передачі методу *__init__*;

__or__(self, other) – викликається, коли має місце операція побітове АБО (*x | y*);

__pow__(self, other [, modulo]) – викликається, коли має місце операція піднесення до степеня (*x ** y, pow(x, y [, modulo])*);

__reversed__(self) – змінює порядок елементів ітератора на зворотній;

__rshift__(self, other) – викликається при виконанні операції бітовий зсув вправо (*x* >> *y*);

__setattr__(self, name, value) – викликається, коли атрибуту об'єкта виконується присвоювання;

__setitem__(self, key, value) – викликається, коли має місце призначення елемента за індексом;

__str__() – перетворення об'єкта до рядкового представлення, викликається, коли об'єкт передається функцій *print()* і *str()*;

__sub__(self, other) – викликається, коли має місце операція віднімання *(x - y)*;

__truediv__(self, other) – викликається, коли має місце операція ділення (*x / y*);

__xor__(self, other) – викликається, коли має місце операція побітове виключає Або (*x* ^ *y*);

Декоратори

В Python все є об'єктом. У цьому сенсі Python повністю відповідає ідеям ООП. Функції також є об’єктами. Це дає додаткові можливості для роботи з ними. Так, наприклад функції можна передавати в якості аргументів, присвоювати функції змінним. Так, наприклад, можна привласнити ім’я функції змінній і викликати функцію, звертаючись до даної змінної:

```
#нехай маємо функцію
def fun_hello(a='Світ'):
    print('Доброго дня, '+a+'!')

#запустимо функцію шляхом звернення до неї
fun_hello('студент') #на екрані зявиться напис: Доброго дня, студент!
#привласнимо ім’я функції змінній
hello=fun_hello
#запустимо функцію звернувшись до змінної
hello('студент') #на екрані зявиться напис: Доброго дня, студент!
```

Як бачимо результат однаковий в обох випадках. Можна також передавати функцію як аргумент в іншу функцію:

```
#спочатку визначимо функцію, яка буде використовуватись як аргумент за
замовчуванням
def pass_fun():
    pass    #ця функція не виконує ніяких дій

#а тепер визначимо функцію, яка приймає іншу функцію як аргумент
def my_fun(a=pass_fun()):
    return a

#а тепер запустимо функцію my_fun без аргументу і з аргументом:
my_fun()    #за замовчуванням функція не виконує ніяких дій

my_fun(fun_hello())     #на екрані зявиться напис: Доброго дня, Світ!
my_fun(fun_hello('студент'))    #на екрані зявиться напис: Доброго дня, студент!
```

В даному прикладі функція *my_fun()* виконує роль обгортки для функції *fun_hello()*. Такі функції називають декораторами, оскільки вони маскують функцію, яку обгортають. В якості функції, яку обгортають може бути використана зовнішня функція (написана користувачем або бібліотечна), а може бути функція визначена всередині функції обгортки:

```
#визначимо функцію my_fun1
def my_fun1(a):
    #визначимо функцію inner_fun
    def inner_fun(b):
        return b**2
        return inner_fun(a)

#виконаємо функцію my_fun1 з аргументом 3
my_fun1(3)      # на екрані з’явиться число 9
```

Основне призначення функцій декораторів полягає в тому, що вони дають нам можливість змінити поведінку функції, не змінюючи її код. Така можливість може бути корисною, наприклад, коли модифікації потребує бібліотечна функція, яку не бажано змінювати, оскільки вона використовується в багатьох інших програмах. Наведемо приклад:

```
#функція, поведінку якої треба змінити не міняючи код функції
```

```
def lib_fun():
    print('Я - бібліотечна функція! Мій код не бажано міняти!')

#створюємо функцію декоратор
def decor_fun(fun_for_decoration):
    def inner_fun():
        print('Доповнюємо додатковими діями поведінку функції lib_fun!')
        fun_for_decoration()
        print('А ми Ваш код не змінюємо, а декоруємо!')
    return inner_fun

#підмінюємо вихідну функцію декорованою
lib_fun = decor_fun(lib_fun)
#запускаємо модифіковану функцію
lib_fun()
```

Представлений приклад можна реалізувати і в інший спосіб:

```
#створюємо функцію декоратор
def decor_fun(fun_for_decoration):
    def inner_fun():
        print('Доповнюємо додатковими діями поведінку функції lib_fun!')
        fun_for_decoration()
        print('А ми Ваш код не змінюємо, а декоруємо!')
    return inner_fun

#декоруємо функцію
@decor_fun
def lib_fun():
    print('Я - бібліотечна функція! Мій код не бажано міняти!')

#запускаємо модифіковану функцію
lib_fun()
```

Можна використовувати кілька декораторів для однієї функції. Можна також декорувати не тільки самостійні функції, а й методи об'єктів. При цьому потрібно тільки не забувати про обов'язковий для методів аргумент self.

Графічні можливості мови python

Майже всі сучасні програми використовують GUI (graphical user interface) – графічний інтерфейс користувача. Існує багато різних бібліотек для роботи з GUI. У мові програмування Python за замовчуванням

використовується бібліотека Tkinter, яка є складовою частиною мови Python. Тому будемо розглядати саме цю бібліотеку. Оскільки бібліотека Tkinter має дуже велику кількість функцій, то за браком часу ми розглянемо лише невелику їх частину, необхідну для розуміння основних принципів роботи з цією бібліотекою.

Основи графічного інтерфейсу користувача

Графічний інтерфейс складається з елементів – вікон, кнопок, меню, тощо. Елементи графічного інтерфейсу називають віджетами. Для розуміння суті роботи з GUI необхідно запам'ятати, що додатки з графічним інтерфейсом користувача подієво-орієнтовані. Тобто кожен віджет пов'язаний з певною подією, на яку він реагує виконуючи запрограмовані дії. Події можуть бути різними: натиснута клавіша клавіатури або кнопка миші, переривання від таймера, тощо.

Подієво-орієнтоване програмування базується на об'єктно-орієнтованому і структурному. Навіть якщо ми не створюємо власних класів та об'єктів, то все-одно ними користуємося. Всі віджети – це об'єкти, породжені вбудованими класами.

Щоб написати GUI-програму, треба виконати наступні дії:

- створити головне вікно програми і конфігурувати його властивості;
- створити і конфігурувати потрібні віджети (задати їх властивості, задати спосіб їх розміщення в головному вікні і т.д.);
- визначити події, на які повинна реагувати програма;
- визначити обробники подій (набір дій виконуваних програмою при виникненні тієї чи іншої події);
- запустити цикл обробки подій.

Створення і конфігурація віджетів виконується за допомогою спеціальних функцій із використовуваної бібліотеки (у нашому випадку *Tkinter*). Розглянемо деякі з цих функцій.

tkinter.Tk() – створює основне вікно програми з конфігурацією за замовчуванням повертає посилання на екземпляр класу. Екземпляр класу можна конфігурувати за допомогою вбудованих методів класу: *geometry(string)* – задає розмір вікна у пікселях. Розмір вказують у змінній *string* у вигляді рядка. Наприклад – '400x250';

title(string) – задає заголовок вікна. Заголовок вказують у змінній *string* у вигляді рядка;

mainloop() – метод створеного вікна, який відображає головне вікно програми і запускає його обробник подій.

Button(window) – створює кнопку у заданому вікні. Повертає покажчик на екземпляр класу *Button*.

Таблиця 1 – Властивості віджета *Button*

Властивість	**Значення**
width	Ширина кнопки у пікселях. За замовчуванням дорівнює такій кількості пікселів, щоб текст умістився в кнопці впритул до її меж.
height	Висота кнопки. За замовчуванням дорівнює такій кількості пікселів, щоб текст містився в кнопці впритул до її меж.
text	Текст на кнопці. За замовчуванням текст буде відображатися по центру кнопки. Можливо зробити багаторядковий текст, використовуючи \ n.
bg	Фон кнопки, який вона матиме 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 межі кнопки, яка буде до її натискання.
activebackground	Колір фону (коли кнопка натиснена).
activeforeground	Колір тексу (коли кнопка натиснена).
disabledbackground	Колір фону (коли властивість *state = DISABLED*).

Властивість	Значення
disabledforeground	Колір тексту (коли властивість *state* = *DISABLED*).
state	Стан кнопки (*NORMAL, DISABLED*). *NORMAL* – звичайний стан кнопки, при якому вона може взаємодіяти з користувачем. *DISABLED* – такий стан кнопки, при якому вона не може взаємодіяти з користувачем.
compound	Розташування картинки на кнопці (*CENTER, BOTTOM, LEFT, RIGHT, TOP*). За замовчуванням картинка на кнопці буде відображатися замість тексту. *BOTTOM* – картинка буде відображатися під текстом. *LEFT* – картинка буде відображатися ліворуч від тексту. *RIGHT* – картинка відображатиметься праворуч від тексту. *TOP* – картинка буде відображатися над текстом.
relief	Рельєф кнопки (*FLAT, GROOVE, RIDGE, SUNKEN, RAISED*).
overrelief	Рельєф кнопки коли над нею знаходиться курсор (*FLAT, GROOVE, RIDGE, SUNKEN, RAISED*).
justify	Вирівнювання тексту (*CENTER, RIGHT, LEFT*). За замовчуванням текст буде відображатися 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.
image	Ім'я картинки, відображеної на кнопці.
font	Вид шрифту на кнопці. Властивість має мати вигляд: "ім’я_шрифта=розмір".

Radiobutton(window) – створює радіокнопку у заданому вікні. Повертає покажчик на екземпляр класу *Radiobutton.*

Таблиця 2 – Властивості віджета Radiobutton

Властивість	Значення
width	Ширина радіо-кнопки.
height	Висота радіо-кнопки.
text	Текст на радіо-кнопці.

Властивість	Значення
bg	Фон радіо-кнопки.
fg	Колір центу.
bd	Ширина межі радіо-кнопки.
activebackground	Колір фону (коли радіо-кнопка натиснена).
activeforeground	Колір тексту (коли радіо-кнопка натиснена).
disabledbackground	Колір фону (коли властивість *state* = *DISABLED*).
disabledforeground	Колір тексту (коли властивість *state* = *DISABLED*).
state	Стан радіо-кнопки (*NORMAL*, *DISABLED*).
compound	Розташування картинки на радіо-кнопці (*CENTER*, *BOTTOM*, *LEFT*, *RIGHT*, *TOP*).
justify	Вирівнювання тексту.
relief	Рельєф радіо-кнопки (*FLAT*, *GROOVE*, *RIDGE*, *SUNKEN*, *RAISE*).
overrelief	Рельєф радіо-кнопки коли над нею знаходиться курсор (*FLAT*, *GROOVE*, *RIDGE*, *SUNKEN*, *RAISE*).
image	Ім'я, яке буде відображено на радіо-кнопці.
selectimage	Ім'я, яке буде відображено на радіо-кнопці (коли вона обрана).
font	Вид шрифту на радіо-кнопці.
indicatoron	Стиль відображення радіо-кнопки (якщо *true*, то буде показуватися гурток поруч з радіо-кнопкою, інакше немає).
value	Значення, яке буде присвоюватися змінної, зазначеної в параметрі *variable* при виборі радіо-кнопки.
variable	Ім'я змінної, у якій буде зміняться значення на зазначене у властивості *value* при виборі радіо-кнопки.

Checkbutton(window) – створює кнопку-прапорець у заданому вікні. Повертає покажчик на екземпляр класу *Checkbutton.*

Таблиця 3 – Властивості віджета Checkbutton

Властивість	Значення
width	Ширина прапорця.

Властивість	Значення
height	Висота прапорця.
text	Текст поряд з прапорцем.
bg	Фон прапорця.
fg	Колір прапорця.
bd	Ширина кордону прапорця.
activebackground	Колір фону (коли прапорець обрано).
activeforeground	Колір тексту (коли прапорець обрано).
disabledbackground	Колір фону (коли властивість *state* = *DISABLED*).
disabledforeground	Колір тексту (коли властивість *state* = *DISABLED*).
state	Стан прапорця (*NORMAL*, *DISABLED*).
compound	Розташування картинки на прапорці (*CENTER*, *BOTTOM*, *LEFT*, *RIGHT*, *TOP*).
justify	Вирівнювання тексту.
relief	Рельєф прапорця (*FLAT*, *GROOVE*, *RIDGE*, *SUNKEN*, *RAISE*).
overrelief	Рельєф прапорця коли над ним знаходиться курсор (*FLAT*, *GROOVE*, *RIDGE*, *SUNKEN*, *RAISE*).
image	Ім'я, яке буде відображено на кнопці-прапорці.
selectimage	Ім'я, яке буде відображено на кнопці-прапорці (коли він обраний).
font	Вид шрифту на кнопці-прапорці.
indicatoron	Стиль відображення прапорця (якщо *true*, то буде показуватися гурток поруч з ф, інакше немає).
onvalue	Значення, яке буде присвоюватися змінній, зазначеній у параметрі *variable* при виборі кнопки-прапорця.
offvalue	Значення, яке буде присвоюватися змінній, зазначеній в параметрі *variable* при виборі другої (не даної) кнопки- прапорця.
variable	Ім'я змінної, у якій буде змінюватись значення на зазначене у властивості *value* при виборі кнопки-прапорця.

Entry(window) – створює однорядкове текстове поле. Повертає посилання на екземпляр класу *Entry*.

Таблиця 4 – Властивості віджета *Entry*

Властивість	**Значення**
width	Ширина поля.
bg	Фон поля.
fg	Колір 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.
highlightcolor	Колір другої межі (коли поле має фокус).
highlightbackground	Колір другої межі (коли поле не має фокус).
highlightthickness	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 в однорядковому текстовому полі.
textvariable	Ім'я змінної, в якій зберігається весь текст, що знаходиться в полі.
selectbackground	Колір фону виділеного фрагмента тексту.
selectforeground	Колір виділеного фрагмента тексту.
insertontime	Час, який курсор видно.
insertofftime	Час, який курсор не видно.

Таблиця 5 – Методи віджета *Entry*

Метод	**Виконувана дія**
a.get(початок, кінець)	Отримує фрагмент тексту від символу, позиція якого визначається першим параметром, до символу, позиція якого визначена другим параметром.

a.insert(позиція,текст)	Вставляє текст у поле перед символом, індекс якого вказаний як параметр позиції. (Нумерація йде з 0.)
a.delete(початок, кінець)	Видаляє текст від символу, позиція якого визначається першим параметром, до символу, позиція якого визначена другим параметром.

Text(window) – створює багаторядкове текстове поле. Повертає покажчик на екземпляр класу *Text*.

Таблиця 6 – Властивості віджета *Text*

Властивість	**Значення**
width	Ширина поля.
height	Висота поля.
bg	Фон поля.
fg	Колір поля.
bd	Ширина межі поля.
activebackground	Колір фону (коли у полі набирають текст).
activeforeground	Колір тексту (коли у полі набирають текст).
disabledbackground	Колір фону (коли властивість *state = DISABLED*).
disabledforeground	Колір тексту (коли властивість *state = DISABLED*).
state	Стан поля (*NORMAL, DISABLED*).
justify	Вирівнювання тексту.
highlightcolor	Колір другої межі (коли поле має фокус).
highlightbackground	Колір другої межі (коли поле не має фокусу).
highlightthickness	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 в багаторядковому текстовому полі.
selectbackground	Колір фону виділеного фрагмента тексту.
selectforeground	Колір тексту виділеного фрагмента тексту.
insertontime	Час, який курсор видно.

Властивість	Значення
insertofftime	Час, який курсор не видно.

Таблиця 7 – Методи віджета *Text*

Метод	Виконувана дія
a.get(start, end)	Отримує фрагмент тексту від символу, позиція якого визначається першим параметром, до символу, позиція якого визначена другим параметром. (Нумерація рядків йде з 1, а символів в рядку з 0.). Позиція повинна мати значення виду "%d.%d " % (x,y), або вона може описуватися одним і слів: *CURRENT* (поточна позиція курсору), *END* (позиція останнього символу в полі).
a.insert(position, text)	Вставляє текст в поле перед символом, індекс якого вказаний як параметр позиції. (Нумерація рядків йде з 1, а символів у рядку з 0.). Позиція при вказівці повинна мати значення виду "% d.% d " % (x,y) або вона може описуватися одним і слів: *CURRENT* (поточна позиція курсору), *END* (позиція останнього символу в полі).
a.delete(start, end)	Видаляє текст від символу, позиція якого визначається першим параметром, до символу, позиція якого визначена другим параметром. (Нумерація рядків йде з 1, а символів у рядку з 0.). Позиція повинна мати значення виду "% d.% d " % (x,y) або вона може описуватися одним і слів: *CURRENT* (поточна позиція курсору), *END* (позиція останнього символу в полі).
a.tag_configure(name, parameters)	Змінює властивості тегу. Текст в діапазоні, який був зазначений при створенні тегу буде змінюватися при зміні властивостей тегу. *background* – фон тексту *foreground* – колір тексту
a.tag_add(name, start, end)	Додає тег (об'єкт, що дозволяє форматувати текст) до поля *a*. (Нумерація рядків йде з 1, а символів в рядку з 0.). Початок або кінець повинні мати значення виду "% d.% d "% (x,y) або вони можуть описуватися одним із слів: *CURRENT* (поточна позиція курсору), *END*

Метод	Виконувана дія
	(позиція останнього символу в полі).

Label(window) – створює мітку повертає покажчик на екземпляр віджета *Label*.

Таблиця 8 – Властивості віджета *Label*

Властивість	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 у фокусі).
activeforeground	Колір тексту (коли мітка у фокусі).
disabledbackground	Колір фону (коли властивість *state = DISABLED*).
disabledforeground	Колір тексту (коли властивість *state = DISABLED*).
state	Стан мітки (*NORMAL, DISABLED*).
compound	Розташування картинки на мітці (*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, яке буде відображено на мітці.
font	Вид шрифту на мітці.
textvariable	Ім'я змінної, у якій буде зберігатися текст на мітці.

Scale(window) – створює у вікні повзунок. Повертає покажчик на екземпляр віджету *Scale*.

Таблиця 9 – Властивості віджета *Scale*

Властивість	**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.
highlightcolor	Колір другої межі (коли повзунок має фокус).
highlightbackground	Колір другий кордону (коли повзунок не має фокус).
highlightthickness	Ширина другої межі.
relief	Рельєф повзунка (*FLAT, GROOVE, RIDGE, SUNKEN, RAISE*).
overrelief	Рельєф повзунка коли над ним знаходиться курсор (*FLAT, GROOVE, RIDGE, SUNKEN, RAISE*).
font	Вид шрифту у розмітки повзунка.
*from_*	Перше число, яке відзначається рядом з повзунком.
to	останнє число, яке відзначається рядом з повзунком.
tickinterval	Відстань між діленнями (у пікселях).
resolution	Мінімальна кількість пікселів, на яку користувач може пересунути повзунок.
orient	Орієнтація смуги з повзунком (*HORIZONTAL, VERTICAL*).
length	Довжина лінії, якою рухається повзунок.

Scrollbar(window) – створює полосу прокруттки вікна. Повертає покажчик на екземпляр віджета *Scrollbar*.

Таблиця 10 – Властивості віджета *Scrollbar*

Властивість	**Значення**
bg	Фон смуги прокрутки.
activebackground	Колір фону (коли смугу прокрутки рухають).
disabledbackground	Колір фону (коли властивість *state = DISABLED*).
state	Стан поля (*NORMAL, DISABLED*).
relief	Рельєф смуги прокрутки (*FLAT, GROOVE, RIDGE, SUNKEN, RAISE*).
overrelief	Рельєф смуги прокрутки коли над ним знаходиться курсор (*FLAT, GROOVE, RIDGE, SUNKEN, RAISE*).
command	Ім'я віджета, зміст якого буде прокручуватися. Доступні лише значення віджет.xview (віджет буде прокручуватися горизонтально) та віджет.yview (віджет буде прокручуватися вертикально).

Frame(window) – створює фрейм вікна. Повертає екземпляр віджета *Frame*.

Таблиця 11 – Властивості віджета *Frame*

Властивість	**Значення**
width	Ширина фрейму.
height	Висота фрейму.
bg	Фон фрейму.
activebackground	Колір фону (коли властивість *state = ENABLED*).
disabledbackground	Колір фону (коли властивість *state = DISABLED*).
state	Стан фрейму (*NORMAL, DISABLED*).
relief	Рельєф межі фрейму (*FLAT, GROOVE, RIDGE, SUNKEN, RAISE*).
highlightcolor	Колір другої межі (коли фрейм має фокус).
highlightbackground	Колір другої межі (коли фрейм не має фокусу).
highlightthickness	Ширина другої межі.

LabelFrame(window) – створює фрейм з міткою. Повертає покажчик на екземпляр віджета *LabelFrame.*

Таблиця 12 – Властивості віджета *LabelFrame*

Властивість	**Значення**
width	Ширина фрейму.
height	Висота фрейму.
bg	Фон фрейму.
activebackground	Колір фону (коли *state* = *ENSABLED*).
disabledbackground	Колір фону (коли *state* = *DISABLED*).
state	Стан фрейма (*NORMAL*, *DISABLED*).
relief	Рельєф межі фрейму (*FLAT*, *GROOVE*, *RIDGE*, *SUNKEN*, *RAISE*).
highlightcolor	Колір другої межі (коли фрейм має фокус).
highlightbackground	Колір другої межі (коли фрейм не має фокус).
highlightthickness	Ширина другої межі.
text	Заголовок фрейму.

Listbox(window) – створює список. Повертає екземпляр віджета *Listbox.*

Таблиця 13 – Властивості віджета *Listbox*

Властивість	**Значення**
width	Ширина списку.
height	Висота списку.
bg	Фон списку.
disabledforeground	Колір фону (коли властивість state = DISABLED).
state	Стан списку (*NORMAL*, *DISABLED*).
relief	Рельєф межі списку (*FLAT*, *GROOVE*, *RIDGE*, *SUNKEN*, *RAISE*).
highlightcolor	Колір другої межі (коли список має фокус).
highlightbackground	Колір другої межі (коли список не має фокусу).
highlightthickness	Ширина другої межі.

selectbackground	Колір фону виділених елементів списку.
selectforeground	Колір тексту виділених елементів списку.
text	Заголовок списку.

Таблиця 14 – Методи віджета *Listbox*

Метод	**Виконувана дія**
a.get(початок, кінець)	Отримує фрагмент списку з назв елементів списку. У фрагмент потраплять елементи, індекси яких, знаходяться в діапазоні, міх першим і останнім його параметрами.
a.insert(позиція, назва_елементу)	Вставляє елемент у список після даного індексу. (Нумерація йде з 0.)
a.delete(початок, кінець)	Видаляє зі списку всі елементи, індекси яких, потрапляють в діапазон індексів початок…кінець,

Menu(window) – створює меню. Повертає екземпляр віджету *Menu*. *add_command(label='Назва пункту',command=виконувана функція)* - додає пункт у меню.

add_cascade(label='Назва', menu=pop_menu) – додає підменю у меню.

window.config(menu=name_menu) – відображає вказане меню.

Розглянемо приклад:

```
#імпортуємо бібліотеку tkinter
import tkinter

#створюємо обробник події вибору пункту меню exit_item (вихід з програми)
def exit_fun():
    window.destroy()

#створюємо обробник події натискання кнопки exit_about
#(закрити вікно about_win )
def about_command():
    about_win=tkinter.Tk()
    about_win.title('Про програму')
    about_win.geometry('200x100')
```

```
    #створюємо фрейм
    about_frame = tkinter.Frame(about_win)
    #about_frame.grig(columnspan=1, rowspan=3)
    about_frame.pack()

    #створюємо і конфігуруємо текст у вікні; колір шрифта чорний
    about_message=tkinter.Label(about_frame, fg='black')
    about_message.justify='CENTER'
    about_message.config(text='Тестова програма. Версія 1.0')
    about_message.pack(side='top')

    #створюємо і конфігуруємо кнопку закриття вікна 'Про програму'
    exit_about=tkinter.Button(about_frame, text='Вихід',
command=about_win.destroy)
    exit_about.pack(side='right')

    #запускаємо цикл обробки подій вікна about_win
    about_win.mainloop()

#створюємо і конфігуруємо головне вікно програми
window = tkinter.Tk()
window.geometry('800x400')
window.title('Перша GUI програма!')

#створюємо головне меню
main_menu = tkinter.Menu(window)
#додаємо у головне меню підменю
pop_menu = tkinter.Menu(main_menu)
#додаємо у підменю пункт і зв'язуємо його з командою
exit_item = pop_menu.add_command(label='Вихід', command = exit_fun)
#додаємо у головне меню пункт з яким зв'язане підменю
main_menu.add_cascade(label='Основне', menu = pop_menu)
#додаємо у головне меню пункт
main_menu.add_command(label = 'Про програму', command = about_command)
#відображаємо головне меню у вікні
window.config(menu = main_menu)

#створюємо фрейм
frame = tkinter.Frame(window)
frame.pack()
#запускаємо цикл обробки подій головного вікна
window.mainloop()
```

Рисунок 1 – Вікно простої графічної програми на мові Python

Після запуску цієї програми з’явиться вікно, зображене на рис. 1. Якщо обрати пункт меню *'Про програму'* то відкриється вікно з коротким описом програми (рис. 2), яке можна закрити натиснувши за допомогою маніпулятора мишки кнопку *'Вихід'*. Вийти з програми можна обравши з меню *'Основне'* пункт *'Вихід'*.

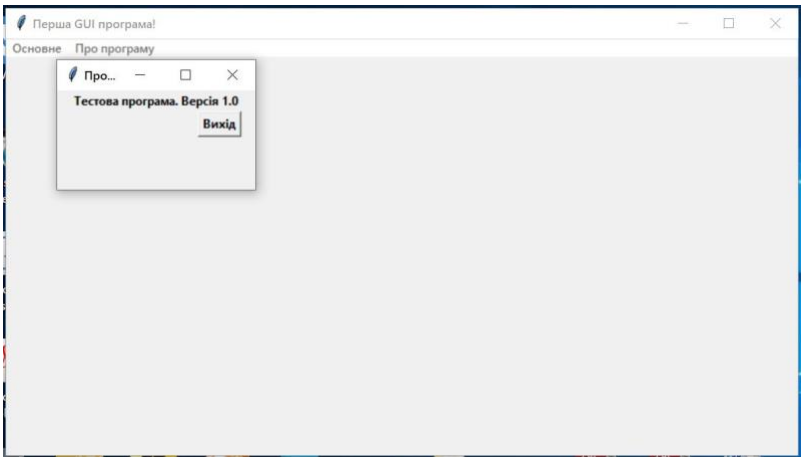

Рисунок 2 – Вікно програми після вибору пункту меню «Про програму»

4 КРИТЕРІЇ ОЦІНЮВАННЯ

Оцінювання розрахунково-графічної роботи проводиться на основі аналізу наступних факторів:

- Правильність та повнота виконання розрахункової частини програми (коду):
  - Забезпечення коректності алгоритмів та їхнього відображення в програмному коді.
  - Відсутність помилок та ефективність в роботі алгоритмів.
  - Чітке коментування коду для полегшення розуміння логіки програми і забезпечення його читабельності.
- Правильність виконання графічної частини програми (графічний інтерфейс):
  - Забезпечення естетичного та функціонального дизайну графічного інтерфейсу.
  - Відповідність графічного інтерфейсу вимогам користувачів та задумам програми.
  - Визначення та обробка коректності введених користувачем даних через графічний інтерфейс.
- Дотримання вимог до виконання роботи з програмування:**
  - Використання сучасних та ефективних підходів при написанні коду.
  - Впровадження засобів тестування для перевірки функціональності та стабільності програми.
  - Документування коду та виготовлення необхідної технічної документації.
  - Забезпечення можливості розширення та підтримки програмного продукту у майбутньому.
  - Відповідність стандартам та кращим практикам програмування.

- «Відмінно», повне правильне виконання роботи відповідно до зазначених вимог – 28-30 балів;
- «Добре», повне виконання роботи, незначні недоліки програмної реалізації; деяка невідповідність вимогам чи структурі роботи – 25-27 балів;
- «Задовільно», неповне виконання роботи, помилки програмної реалізації; невідповідність вимогам чи структурі роботи – 22-24 балів;
- «Достатньо», неповне виконання роботи, помилки програмної реалізації; невідповідність вимогам чи структурі роботи – 18-21 балів;
- «Незадовільно», неповне виконання роботи, помилки програмної реалізації; повна невідповідність вимогам чи структурі роботи – 1-17 балів;
- «Роботу не зараховано», робота не відповідає варіанту або роботу не виконано – 0 балів.

РЕКОМЕНДОВАНА ЛІТЕРАТУРА

1. ДСТУ 3008:2015. Інформація та документація. Звіти у сфері науки і техніки. Структура та правила оформлювання / Нац. стандарт України. – Вид. офіц. – [На зміну 3008-95 ; чинний від 2017-07-01]. – Київ : ДП «УкрНДНЦ», 2015. – 26 с.

2. ДСТУ 8302:2015. Інформація та документація. Бібліографічне посилання. Загальні положення та правила складання / Нац. стандарт України. – Вид. офіц. – [Уведено вперше ; чинний від 2016-07-01]. – Київ : ДП «УкрНДНЦ», 2016. – 17 с.

3. Добролюбова М.В. Обчислювальна техніка та програмування. Процедурна складова мови C# [Електронний ресурс]: навч. посіб. для студ. спеціальності 152 «Метрологія та інформаційно-вимірювальна техніка» / М. В. Добролюбова, М. О. Маркін, М. В. Філіппова; КПІ ім. Ігоря Сікорського. – Електронні текстові дані (1 файл: 2,68 Мбайт). – Київ : КПІ ім. Ігоря Сікорського, 2021. – 154 с.

4. Об'єктно-орієнтоване програмування мовою PYTHON : Конспект лекцій [Електронний ресурс] : навч. посіб. для студ. спеціальності 153 «Мікро- та наносистемна техніка» освітньої програми «Мікро- та наноелектроніка» / КПІ ім. Ігоря Сікорського ; уклад.: Д. Д. Татарчук, Ю. В. Діденко. – Електронні текстові данні (1 файл: 1,3 Мбайт). – Київ : КПІ ім. Ігоря Сікорського, 2021. – 129 с.

5. Програмування мовою Python /О.М. Васильєв. – Тернопіль :Видавництво "Навчальна книга-Богдан",2021. – 503 с.

6. PYTHON : Алгоритмізація та програмування :навчальний посібник /В.А. Висоцька, О.В. Оборська ; Міністерство освіти і науки України, Національний університет "Львівська політехніка".– Львів :Видавництво "Новий Світ-2000",2021. – 514 с.

7. Чиста архітектура :мистецтво розроблення програмного забезпечення / Роберт Мартін; переклад з англійської І. Бондар-Терещенко. – Харків :Фабула,2019. – 367 с.

8. Проектування програмних систем :конспект лекцій для студентів спеціальності 122 "Комп'ютерні науки та інформаційні технології" /Д.В. Бреславський, Ю.М. Коритко ; Міністерство освіти і науки України, Національний технічний університет "Харківський політехнічний інститут".– Харьків :[Панов А. М.], 2017. – 130 с.

9. Обчислювальна техніка та програмування :навчальний посібник для студентів вищих навчальних закладів, які навчаються за освітньо-професійною програмою бакалавра за напрямом підготовки "Метрологія та інформаційно-вимірювальні технології" /Л. О. Борковський, О. В. Кочеткова, О. В. Борковський; Міністерство освіти і науки України, Національний авіаційний університет. – Київ :НАУ, 2016. – 268 с.