ЖИТОМИРСЬКИЙ ВІЙСЬКОВИЙ ІНСТИТУТ ІМЕНІ С. П. КОРОЛЬОВА

Кафедра інформаційних технологій та кібербезпеки
(повна назва кафедри)

ЗАТВЕРДЖУЮ
Начальник кафедри
полковник
Руслан ЖОВНОВАТЮК
“_____” _____ 202__ р.

МЕТОДИЧНІ ВКАЗІВКИ ДЛЯ САМОСТІЙНОЇ РОБОТИ ТА ВИКОНАННЯ ІНДИВІДУАЛЬНИХ ЗАВДАНЬ

з навчальної дисципліни

ТЕХНОЛОГІЇ ПРОГРАМУВАННЯ

Обговорена та затверджена на засіданні кафедри комп’ютерно-інтегрованих технологій та кібербезпеки
“___” _____ 2025 р.
Протокол № ____

Житомир
2025

1. ЗАГАЛЬНІ ПОЛОЖЕННЯ

1.1 Самостійна робота здобувачів освіти - це форма організації освітнього процесу, за якої здобувачі освіти опановують освітній компонент у час, вільний від навчальних занять. Вона спрямована на глибоке засвоєння навчального матеріалу, розвиток аналітичного мислення, формування навичок самонавчання та відповідальності за власний освітній результат.

1.2 Навчальний час, відведений на самостійну роботу, регламентується робочою програмою навчальної дисципліни і повинен складати не менше, ніж 1/3, і не більше, ніж 2/3 загального обсягу навчального часу, відведеного на вивчення навчальної дисципліни. Робочою програмою навчальної дисципліни “Технології програмування” передбачено виконання самостійної роботи курсантами загальним обсягом 48 год.

1.3 Самостійна робота здобувачів освіти здійснюється з метою:

відпрацювання та засвоєння навчального матеріалу, закріплення та поглиблення знань, умінь та навичок;

виконання індивідуальних завдань з освітнього компонента (курсові роботи (проєкти), контрольні роботи, розрахунково-графічні роботи, реферати тощо), воєнно-наукових і кваліфікаційних робіт;

підготовки до майбутніх занять та контрольних заходів;

формування у здобувачів освіти самостійності та ініціативи в пошуку та набутті знань.

1.4 Самостійна робота здобувачів освіти забезпечується інформаційно-методичними засобами (програми, методичні вказівки, завдання, підручники, навчальні посібники тощо) та матеріально-технічними засобами (макети, тренажери, елементи озброєння та військової техніки тощо), передбаченими програмою освітнього компонента. Для самостійної роботи здобувачів освіти рекомендується відповідна наукова та професійна навчальна література.

1.5 Створення належних умов для самостійної роботи здобувачів освіти покладається на начальників структурних підрозділів. Безпосередньо її організовують командири підрозділів курсантів. Облік самостійної роботи

курсантів ведуть командири навчальних підрозділів у журналі обліку навчальних занять (розділ обліку самостійної роботи).

1.6 Індивідуальні завдання з освітнього компонента є невід'ємною складовою самостійної роботи здобувачів освіти. Індивідуальні завдання сприяють більш поглибленому вивченню здобувачами освіти теоретичного матеріалу, закріпленню та узагальненню отриманих знань, формуванню вміння використовувати знання для комплексного вирішення відповідних професійних завдань.

1.7 До індивідуальних завдань відносяться реферати, есе, розрахункові, графічні, аналітичні, розрахунково-графічні завдання, контрольні, курсові, кваліфікаційні роботи.

Основними видами самостійної роботи здобувачів освіти є:

№ з/п	Назва видів самостійної роботи	Кількість годин
1.	Опрацювання лекційного матеріалу	20
2.	Підготовка до групових та практичних занять	16
3.	Виконання індивідуальних завдань (есе)	6
4.	Підготовка до контрольних робіт	3
5.	Підготовка до підсумкового контролю знань	3
	Разом	**48**

2. ОПРАЦЮВАННЯ ЛЕКЦІЙНОГО МАТЕРІАЛУ

Відпрацювання лекційного матеріалу є невід'ємною складовою самостійної роботи курсанта, спрямованої на закріплення отриманих теоретичних знань та підготовку до виконання практичних завдань. Метою даного етапу є формування та закріплення у курсантів системних теоретичних знань з основ програмування мовою *Python*, функціонального та об'єктно-орієнтованого підходів, методів роботи з даними та файловими структурами, а також забезпечення усвідомленого розуміння принципів побудови програмних рішень, необхідних для подальшого виконання практичних завдань і професійних функцій.

Після кожної лекції курсанту необхідно самостійно опрацювати та осмислити поданий матеріал. Для цього рекомендовано:

уважно перечитати записи, уточнити незрозумілі моменти та за потреби доповнити конспект інформацією з підручників або онлайн-ресурсів;

вивчити ключові терміни, визначення та основні положення теми - курсант повинен розуміти базові поняття, важливі визначення та принципи роботи відповідних технологій програмування;

сформувати короткий структурований конспект теоретичного матеріалу, який повинен містити перелік основних понять, схеми, алгоритми, приклади, а також висновки та узагальнення;

розібрати та зрозуміти всі приклади коду, наведені під час лекції, переписати та запустити їх у середовищі (наприклад, *Visual Studio Code*) мовою *Python*; зробити нотатки щодо роботи кожного прикладу;

самостійно створити декілька власних прикладів коду за темою лекції - це закріплює розуміння принципів програмування і розвиває практичні навички;

виконати завдання для самоконтролю за темою лекції: відповіді на контрольні запитання; створення невеликих фрагментів коду; розв’язання типових задач; виконання тестових питань;

перевірити розуміння матеріалу через короткий самоаналіз: курсант повинен оцінити що зрозумів повністю; які питання потребують повторного опрацювання; з якими труднощами зіткнувся;

підготувати питання для викладача: якщо виникли прогалини, незрозумілі теми або складні механізми — записати питання для уточнення на наступному занятті або консультації;

після кожної лекції курсант повинен бути готовим застосувати вивчену теорію у програмному коді; використовувати відповідні конструкції *Python*; розв’язувати задачі, що базуються на темі лекції.

Курсант повинен забезпечити систематичність засвоєння матеріалу: впорядковувати матеріали; оновлювати власний глосарій термінів; підтримувати структуру теоретичних записів для подальшої роботи.

У разі пропуску лекції курсант зобов’язаний самостійно вивчити матеріал за допомогою наданих ресурсів (лекційні презентації, онлайн-ресурс, підручники, методичні вказівки тощо), а також, за потреби, звернутися до викладача у визначені графіком консультацій дні для отримання консультації.

Таким чином, систематичне та відповідальне опрацювання лекційного матеріалу сприяє формуванню повноцінної компетентності, забезпечує підготовку до успішного виконання практичних завдань і підвищує загальний рівень володіння мовою програмування.

3. ПІДГОТОВКА ДО ГРУПОВИХ ТА ПРАТКИЧНИХ ЗАНЯТЬ

Метою групових та практичних занять з дисципліни “Технології програмування” є формування у курсантів стійких умінь та навичок застосування теоретичних знань для розв’язання конкретних програмних задач, розвиток алгоритмічного мислення та здатності створювати працездатні програмні модулі мовою *Python*, а також розвиток у курсантів навичок колективної роботи, взаємонавчання та презентації програмних рішень, формування вміння узгоджено застосовувати теоретичні знання та практичні навички в умовах спільного виконання навчальних завдань.

Практичні заняття спрямовані на те, щоб курсанти:

навчились коректно застосовувати конструкції мови програмування для вирішення задач різного рівня складності;

здобули досвід написання, аналізу, тестування та налагодження програмних фрагментів;

оволоділи прийомами роботи зі структурами даних, файлами, модулями, бібліотеками та технологіями, що розглядаються в курсі;

сформували навички самостійного пошуку рішень і обґрунтованого вибору алгоритмів;

підготувалися до виконання комплексних практичних завдань, модульних робіт та подальших навчальних дисциплін, пов’язаних із професійною діяльністю.

Групові заняття забезпечують:

розвиток комунікативних компетентностей у сфері програмування та технічного обговорення рішень;

здатність пояснювати власні проєктні рішення та аргументувати обрані алгоритми;

вміння працювати в малій групі над спільним програмним завданням;

підвищення рівня усвідомлення та закріплення матеріалу через навчання одне одного;

формування здатності до аналізу, порівняння та оцінювання різних підходів у програмуванні;

відпрацювання навичок презентації матеріалу, демонстрації коду та відповіді на запитання аудиторії.

До кожного практичного заняття курсантам рекомендовано попередньо ознайомитися з темою та вивчити відповідний теоретичний матеріал за власним конспектом, підручником або електронними ресурсами. Необхідно переглянути та проаналізувати всі приклади коду, наведені на лекції: запустити їх у середовищі програмування, розібрати принцип роботи кожного фрагмента та зробити примітки щодо використаних конструкцій. Також слід відпрацювати ключові операції (цикли, функції, масиви, робота з файлами тощо) та підготуватися до виконання більш складних практичних завдань.

Під час підготовки до практичних і групових занять курсанту необхідно перевірити наявність та працездатність програмного забезпечення: переконатися, що встановлено *Python*, доступні необхідні бібліотеки (*NumPy, pathlib, os* тощо), а робоче середовище (*VS Code, PyCharm, IDLE*) налаштовано та готове до виконання програмного коду. Також важливо ознайомитися зі

змістом майбутніх практичних завдань, визначити потрібні інструменти та продумати можливі підходи до їх розв’язання.

Окрім того, підготовка до занять передбачає самооцінку рівня засвоєння матеріалу та формування переліку питань щодо складних або незрозумілих аспектів теми.

Підготовка до групового заняття має ширший, комунікативний та презентаційний характер. Вона передбачає ознайомлення курсантів зі структурою та цілями заняття, розуміння власної ролі (доповідач чи виконавець), змісту навчального блоку та вимог до подання матеріалу. Курсанти-доповідачі повинні опрацювати тему, підготувати презентацію, приклади коду, продумати структуру свого виступу та відпрацювати пояснення й відповіді на можливі запитання. Усі учасники групового заняття мають повторити матеріал теми, засвоїти основні поняття, правила застосування інструментів програмування, а також типові алгоритми та шаблони рішень.

Підготовка також включає готовність курсантів до роботи в малій групі, що передбачає співпрацю, обговорення рішень та здатність аналізувати й оцінювати код інших учасників. Крім цього, необхідно попередньо ознайомитися з практичними завданнями групового заняття, проаналізувати їх складність, визначити необхідні засоби та спланувати можливі способи розв’язання. Для ефективної участі курсант повинен підготувати матеріали, необхідні для роботи, зокрема конспект, нотатки, фрагменти коду, блок-схеми, таблиці або інші самостійно підготовлені приклади. Важливо також усвідомлювати критерії оцінювання: розуміти, за що нараховуються бали, які вимоги ставляться до доповідача та які критерії використовуються для оцінки роботи під час заняття.

Під час практичних та групових занять оцінюється не лише правильність виконання завдань, а й рівень володіння теоретичним матеріалом, уміння застосовувати його на практиці та здатність аргументувати обрані способи розв’язання. З метою підвищення ефективності навчання рекомендується поєднувати теоретичну підготовку з активною практичною діяльністю:

регулярно виконувати програмні завдання, аналізувати приклади коду, працювати в малих групах, брати участь у дискусіях і презентаціях, а також застосовувати набуті знання у самостійних мініпроєктах. Під час роботи на заняттях доцільно активно використовувати гарячі клавіші (*Ctrl+C*, *Ctrl+V*, *Ctrl+Z* тощо), зберігати проміжні результати, не боятися експериментувати з інструментами, звертатися за допомогою до викладача або користуватися довідковими системами, що сприяє формуванню впевнених практичних навичок і більш глибокому розумінню матеріалу.

4. ЗАВДАННЯ ДЛЯ САМОСТІЙНОЇ РОБОТИ

№ з/п	Тема заняття. Навчальні питання	Кількість годин
1.	*Введення у програмування мовою Python*. 1. Порядок вивчення навчальної дисципліни. 2. Базові поняття у програмуванні. 3. Синтаксис. Типи даних. 4. Лінійна програма.	1
2.	*Редактори початкового коду для розробки програм мовою Python.* 1. Редактор початкового коду *VS Code*. 2. Встановлення розширень. 3. Техніка розробки програми.	1
3.	*Розробка прогам з циклами.* 1. Цикл з умовою продовження. 2. Цикл по колекції. 3. Переривання та продовження циклів. 4. Основи алгебри висловлювань. 5. Умови та умовні оператори. 6. Розгалуження.	1
4.	*Типи колекцій для збереження даних: списки, кортежі, словники, множини. Операції з колекціями.* 1. Списки. Кортежі. 2. Словники. 3. Множини.	1
5.	*Інструменти для налагодження коду програми.* 1. Синтаксичні та логічні помилки програми, виключення. 2. Інструменти для налагодження коду програми.	1

№ з/п	Тема заняття. Навчальні питання	Кількість годин
6.	*Символи та рядки. Форматування рядків.* 1. Символи та рядки. 2. Форматування рядків. 3. Операції з рядками.	1
7.	*Створення функцій, їх види та застосування.* 1. Функції та їх аргументи. 2. Функціональний тип. 3. Анонімні та рекурсивні функції. 4. Локальні та глобальні змінні.	1
8.	*Функції-генератори.* 1. Призначення та особливості запису коду функції-генератору. 2. Розробка та застосування функції-генератора.	1
9.	*Декоратори для функцій.* 1. Призначення та правила розробки декораторів. 2. Розробка та застосування декораторів для функцій.	1
10.	*Обробка помилок.* 1. Типи помилок. 2. Перехоплення та обробка помилок.	1
11.	*Поняття та типи файлів, доступ та робота з файловою системою засобами мови програмування Python.* 1. Поняття та типи файлів. 2. Базові операції з файлами. 3. Модулі для роботи з файлами.	1
12.	*Поняття про структуровані текстові файли.* 1. Файли гіпертекстової розмітки HTML. 2. Файли для серіалізації даних JSON. 3. Структуровані текстові файли CSV.	1
13.	*Модулі. Використання модулів у програмі. Пакети.* 1. Поняття модуля. 2. Робота з модулями: підключення, доступ до елементів модуля. 3. Створення модуля. 4. Пакети.	1
14.	*Робота з додатковими пакетами для обробки даних. NumPy.* 1. Встановлення та підключення NumPy. 2. Створення масивів та доступ до його елементів. 3. Математичні операції з елементами масивів. 4. Вибір та маніпуляція з елементами масиву.	1

№ з/п	Тема заняття. Навчальні питання	Кількість годин
15.	*Об'єктно-орієнтоване програмування. Класи. Методи класу.* 1. Поняття Об'єктно-орієнтованого програмування. 2. Класи. Атрибути та методи класу. 3. Наслідування.	1
16.	*Створення класу. Вбудовані методи класу.* 1. Документування класу. 2. Методи порівняння екземплярів класу. 3. Операції з екземплярами класу.	1
17.	*Створення та використання віртуального оточення.* 1. Суть та передумови для створення віртуального оточення. 2. Створення та робота у віртуальному оточенні.	1
18.	*Системи контролю версій коду програми.* 1. Поняття та принципи роботи систем контролю версій. 2. Встановлення та налаштування. 3. Базові операції.	1
19.	*Генератор-вирази для роботи з колекціями.* 1. Абстракція списків. 2. Абстракція словників.	1
20.	*Візуалізація даних за допомогою пакету Matplotlib.* 1. Встановлення та робота з пакетом *Matplotlib*. 2. Підготовка вихідних даних. 3. Візуалізація даних.	1

5. ІНДИВІДУАЛЬНЕ ЗАВДАННЯ

5.1 Робочою програмою навчальної дисципліни передбачено виконання індивідуального завдання щодо оформлення есе, презентації та доповіді за визначеною темою. Доповідь здійснюється у вигляді методичної практики.

Видача тем та порядку оформлення есе, презентації та доповіді здійснюється викладачем не пізніше практичного заняття 2/1 (для першої підгрупи) та практичного 3/1 (для другої підгрупи). Результати виконання завдання здаються у визначеній викладачем формі (з обов'язковою реєстрацією есе та внесенням інформації у журнал обліку контрольних робіт кафедри) не пізніше групового 2/4 (МП) (для першої підгрупи) та групового 3/5 (МП) (для другої підгрупи).

Виконання індивідуального завдання та отримання позитивної оцінки є обов'язковою умовою допуску курсанта до контрольних заходів семестрового контролю.

5.2 Вимоги до написання есе

Есе – самостійна письмова творча робота невеликого обсягу з вільною аргументованою композицією.

Мета написання есе полягає в описі курсантом власного бачення сутності конкретного питання в межах проблематики навчальної дисципліни.

Написання есе дає змогу автору навчитися чітко і грамотно формулювати власні думки, структурувати інформацію, виділяти причинно-наслідкові зв'язки, находити доречні приклади і робити висновки.

Обсяг есе – до 10 сторінок комп'ютерного набору тексту.

Оформлення есе має відповідати вимогам до наукової публікації: містити титульний аркуш, зміст, вступ, основну частину, висновки, перелік використаних джерел.

Письмова робота виконується державною мовою та друкується з однієї сторони аркушу.

Під час виконання роботи обов'язково посилаються на джерела інформації, з яких позичено інформацію із дотриманням всіх правил цитування.

Правила комп'ютерного набору тексту:

письмову роботу відпрацьовують у форматі *А*4 (210х297 мм);

поля: ліве – 30 мм, праве 15 мм, верхнє – 20 мм, нижнє – 20 мм;

текстовий редактор – *Word*;

гарнітура шрифту – *Times New Roman*, вирівнювання – по ширині;

кегель (розмір) шрифту – 14;

абзац – 1,25 см;

міжрядковий інтервал – 1,5 інтервалу;

назви структурних частин роботи "ЗМІСТ", "ВСТУП", "РОЗДІЛ", "ВИСНОВКИ", "ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ" друкують великими

літерами симетрично до тексту (шрифт – 14 напівжирний), вирівнюючи по центру;

усі названі структурні частини роботи починають з нової сторінки;

відступ між пунктами – два полуторних інтервали;

заголовки пунктів друкують маленькими літерами, крім першої великої, з абзацного відступу, шрифт – 14 напівжирний. Крапку в кінці заголовка не ставлять. Відстань між заголовком розділу та заголовком підрозділу – два полуторних інтервали;

у роботі допускається виділення важливого тексту курсивом, недопустимо виділяти частину тексту за допомогою інших шрифтів, крім *Times New Roman*, іншого розміру, крім 14, з іншим інтервалом тощо;

недопустимо розміщувати заголовок пункту внизу сторінки, якщо під ним не вміщується щонайменше два рядки тексту – у такому разі назву пункту і текст переносять на наступну сторінку.

Розділи, пункти і підпункти роботи слід нумерувати арабськими цифрами. Розділи нумерують у межах роботи і позначають арабськими цифрами (1; 2;…). Номер пункту складається з номеру розділу і порядкового номера підрозділу, відокремленого крапкою (1.1; 1.2; 1.3;…). У такому ж порядку нумерують і підпункти. Номер підпункту складається із трьох цифр, відокремлених крапками (1.1.1; 1.2.2; 1.3.1;…).

Усі сторінки в письмовій роботі нумеруються арабськими цифрами без крапки у правому верхньому куті сторінки. Титульний аркуш включають до загальної нумерації сторінок роботи, але номер сторінки на ньому не ставлять.

Робота оцінюється максимально у 5 балів (доповідь – до 5 додаткових балів).

Роботи, що оформлені із порушенням вимог, оцінюватися не будуть і відправлятимуться на доопрацювання. Кожне доопрацювання тягне за собою зниження максимальної оцінки на 1 бал.

Оцінюючи роботу, викладач зважає на те, наскільки повно розкрито тему, наскільки чітко та послідовно викладено зміст завдань, наскільки адекватно зроблено висновки, чи оформлено роботу у відповідності до вимог.

Позитивна оцінка за есе (мінімум 3 бали) є допуском до здачі заліку.

5.3 Орієнтовний перелік тем есе

№ з/п	Теми есе
1.	Історія створення та розвиток бібліотеки *Matplotlib*.
2.	Порівняння *Matplotlib* з іншими засобами візуалізації даних у *Python*.
3.	Основні типи графіків у *Matplotlib* та їх практичне застосування.
4.	Роль осей та масштабів у побудові графіків у *Matplotlib*.
5.	Налаштування стилю та дизайну графіків: кольори, шрифти, маркери.
6.	Створення гістограм та їх використання в аналізі даних.
7.	Побудова лінійних графіків та їх застосування у військово-технічному аналізі.
8.	Використання стовпчикових діаграм для порівняння даних.
9.	Побудова кругових діаграм та особливості їх інтерпретації.
10.	Двовимірні діаграми розсіювання та їх роль у статистичному аналізі.
11.	Створення багатосерійних графіків: робота з кількома наборами даних.
12.	Додавання підписів, легенд та сітки для покращення читабельності графіка.
13.	Візуалізація часових рядів у *Matplotlib*.
14.	Використання *Matplotlib* для навчальних задач у програмуванні.
15.	Експорт графіків у зображення та інтеграція у документи.
16.	Реалізація простих анімацій у *Matplotlib*.
17.	Створення підграфіків (*subplots*): організація складних макетів.
18.	Інструменти *Matplotlib* для роботи з великими наборами даних.
19.	Використання *Matplotlib* у поєднанні з *NumPy* та *Pandas*.
20.	*Mamplotlib* у наукових та інженерних дослідженнях: приклади застосування.
21.	Налаштування тривимірних графіків у *Matplotlib*.
22.	Застосування *Matplotlib* у моделюванні військових процесів і задач.

23.	Порівняльна характеристика інтерфейсів *pyplot* та *object-oriented API* у *Matplotlib*.
24.	Побудова теплових карт (*heatmaps*) і їх значення в аналізі даних.
25.	Створення професійних презентацій та звітів за допомогою *Matplotlib*.

5.4 Есе та доповіді (презентації) не повинні містити ІзОД. Есе у паперовому та електронному вигляді здати викладачу за три доби до проведення групового заняття. Бути готовим доповісти під час групового заняття, тривалість доповіді до 10 хвилин. Оцінюється вміння курсантів готувати доповідь та презентувати її аудиторії. Після кожної доповіді проводиться обговорення із визначенням слабких та сильних сторін. Слово для обговорення надається курсантам.

6. ПИТАННЯ ДЛЯ САМОКОНТРОЛЮ

Змістовний модуль 1. Основи програмування мовою Python

1. Що таке інтерпретатор *Python* і для чого він потрібен?
2. Які існують режими запуску *Python*-коду?
3. Які редактори коду рекомендується використовувати для програмування на *Python*?
4. Що таке змінна та як відбувається присвоєння значення?
5. Які основні типи даних підтримує *Python*?
6. У чому різниця між числовими типами *int* та *float*?
7. Що таке булевий тип даних?
8. Що таке вираз і оператор у *Python*?
9. Як виконується введення даних користувачем?
10. Як працює оператор *if*?
11. У чому різниця між *if–elif–else* та вкладеними умовами?
12. Як працює цикл *for* у *Python*?
13. Як працює цикл *while* та в яких випадках його доцільно застосовувати?

14. Для чого потрібні оператори *break* і *continue*?
15. Що таке ітерабельний об’єкт?
16. Як створити список у *Python*?
17. Як додати та видалити елемент із списку?
18. Чим кортеж відрізняється від списку?
19. Як працюють словники та що є ключем і значенням?
20. У яких випадках використовують множини (*set*)?
21. Які основні операції підтримують множини?
22. Які методи списків потрібно знати для програмування?
23. Що таке індексація та зрізи в колекціях?
24. Які основні методи роботи з рядками?
25. Як працює форматування рядків у *Python* (*f-string*)?
26. Що таке *escape*-послідовності?
27. Як перевірити довжину рядка чи колекції?
28. Як сортувати список?
29. Які інструменти використовуються для налагодження коду?
30. Що таке помилка інтерпретації та помилка виконання?

Змістовний модуль 2. Функціональне програмування

31. Що таке функція та навіщо її використовувати?
32. Як визначається функція в *Python*?
33. Що таке параметри та аргументи функції?
34. Чим відрізняються позиційні та іменовані аргументи?
35. Що таке аргументи за замовчуванням?
36. Як працюють змінні позиційні аргументи **args*?
37. Як працюють змінні іменовані аргументи ***kwargs*?
38. Що таке область видимості змінних?
39. Яка різниця між локальними та глобальними змінними?
40. Що таке рекурсія?
41. Що таке анонімна функція (λ-функція)?
42. Як використовуються λ-функції у *map*(), *filter*(), *reduce*()?

43. Для чого потрібна функція-генератор?

44. Як працює оператор *yield*?

45. У чому різниця між *return* та *yield*?

46. Що таке обробка помилок?

47. Як працюють конструкції *try–except*?

48. Які види помилок найчастіше виникають у *Python*?

49. Як коректно створити власний виняток?

50. Що таке *finally* та в яких випадках він виконується?

51. Що таке файлова система?

52. Що таке шлях до файлу? Різниця між абсолютним та відносним шляхом?

53. Як відкрити файл у *Python*?

54. Які режими відкриття файлів підтримує функція *open*()?

55. Як прочитати файл построково?

56. Як записати текст у файл?

57. Чому важливо закривати файл після роботи з ним?

58. Для чого використовується формат *JSON*?

59. Як прочитати дані з *JSON*-файлу?

60. Що таке *CSV*-файл і як він структурується?

61. Як записувати дані у форматі *CSV*?

62. Що таке *HTML*-документ і які основні теги він містить?

63. Як створити найпростіший *HTML*-файл засобами *Python*?

64. Які основні функції модуля *os* застосовуються при роботі з каталогами?

65. Що таке модуль у *Python*?

66. Як імпортувати модуль у програму?

67. У чому різниця між *import module*, *from module import name*?

68. Що таке пакет (*package*)?

69. Для чого використовується файл *init.py*?

70. Як встановлювати сторонні пакети *Python*?

71. Чому важливо документувати код?

72. Що таке віртуальне середовище (*virtual environment*) у *Python*?

73. Що таке *NumPy* та для чого він використовується?

74. Чим *NumPy*-масив відрізняється від списку *Python*?

75. Як створити масив *NumPy*?

76. Як отримати доступ до елементів масиву?

77. Як виконуються арифметичні операції над масивами?

78. Як працює індексація та зрізи в *NumPy*?

79. Що таке бібліотека *Matplotlib* і для чого вона використовується?

80. Як створити найпростіший графік за допомогою *pyplot*?

81. Як додати заголовок, підписи осей та легенду на графік?

Змістовний модуль 3. Об'єктно-орієнтовне програмування

82. Що таке об'єктно-орієнтоване програмування та у чому його переваги?

83. Що таке клас і об'єкт у *Python*?

84. Для чого використовується метод *__init__*()?

85. Яку роль відіграє ключове слово *self* у методах класу?

86. Чим відрізняються атрибути класу від атрибутів екземпляра?

87. Що таке інкапсуляція та як вона реалізується у *Python*?

88. Як позначаються приватні та захищені атрибути?

89. Що таке наслідування та яку функцію воно виконує?

90. Як створити клас-нащадок на основі класу-батька?

91. Для чого використовується функція *super*()?

92. Що означає перевизначення методів у класах-нащадках?

93. Що таке поліморфізм і як він проявляється у *Python*?

94. Що таке множинне наслідування та які ризики воно має?

95. Що таке *MRO* (порядок вирішення методів)?

96. Що таке магічні (*dunder*) методи в *Python*?

97. Для чого застосовуються методи *__str__*() та *__repr__*()?

98. Як у *Python* можна реалізувати композицію?

99. Чому ООП полегшує розробку великих програмних систем?

100. У яких випадках недоцільно застосовувати ООП?

7. РЕКОМЕНДОВАНА ЛІТЕРАТУРА ДО НАВЧАЛЬНОЇ ДИСЦИПЛІНИ

Основна література

1. Технології програмування : навчальний посібник / [В.А. Романько, Р.М. Жовноватюк, О.В. Манько, О.М. Срібний]. – Житомир : ЖВІ, 2025. – 204 с.
2. The Python Tutorial. [Електронний ресурс] – Режим доступу до ресурсу: https://docs.python.org/3/tutorial/index.html.
3. Путівник мовою програмування Python. [Електронний ресурс] – Режим доступу до ресурсу: https://pythonguide.rozh2sch.org.ua/.

Допоміжна література

1. UDAYAN DAS, AUBREY LAWSON, CHRIS MAYFIELD, NARGES NOROUZI. Introduction to Python Programming. OpenStax, Houston, Texas 77005. 2024. 415 p.
2. Костюченко А.О. Основи програмування мовою Python: навчальний посібник. Ч.: ФОП Баликіна С.М., 2020. 180 с
3. Креневич А.П. Python у прикладах і задачах. Частина 1. Структурне програмування. Навчальний посібник із дисципліни "Інформатика та програмування" – К.: ВПЦ "Київський Університет", 2017. – 206 с.

Інформаційні ресурси в мережі Інтернет

1. Технології програмування [Електронний ресурс] – Режим доступу до ресурсу: https://olegmov.github.io/programming_technology_public/

The Python Tutorial [Електронний ресурс] – Режим доступу до ресурсу: https://docs.python.org/3/tutorial/index.html.

2. Путівник мовою програмування Python. [Електронний ресурс] – Режим доступу до ресурсу: https://pythonguide.rozh2sch.org.ua/.

3. Основи Git [Електронний ресурс] – Режим доступу до ресурсу: https://git-scm.com/book/uk/v2.

4. NumPy quickstart [Електронний ресурс] – Режим доступу до ресурсу: https://numpy.org/devdocs/user/quickstart.html.

5. Інструкція користувача matplotlib [Електронний ресурс] – Режим доступу до ресурсу: https://matplotlib.org/stable/tutorials/index.html.