

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАХІДНОУКРАЇНСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
Факультет комп'ютерних інформаційних технологій
Кафедра кібербезпеки

МЕТОДИЧНІ ВКАЗІВКИ
до виконання лабораторних робіт
з дисципліни «ПРОГРАМУВАННЯ ДЛЯ КІБЕРБЕЗПЕКИ»
для здобувачів першого (бакалаврського) рівня вищої освіти
спеціальності F5 «Кібербезпека та захист інформації»

Тернопіль – 2026

Методичні вказівки до виконання лабораторних робіт з дисципліни «Програмування для кібербезпеки» для здобувачів освітньо-професійної програми «Кібербезпека» першого (бакалаврського) рівня вищої освіти спеціальності «Кібербезпека та захист інформації» / Укл.: Івасьєв С.В. – Тернопіль: ЗУНУ, 2026. – 55 с.

Відповідальний за випуск: Івасьєв С.В., к.т.н., доцент кафедри кібербезпеки

Рецензенти:

Батько Ю.М. к.т.н., доцент кафедри комп'ютерної інженерії факультету комп'ютерних інформаційних технологій Західноукраїнського національного університету

Методичні вказівки розглянуті та схвалені на засіданні кафедри кібербезпеки, протокол № 1 від 26 серпня 2026 р.

ВСТУП

Методичні вказівки призначені для виконання 12 лабораторних робіт з дисципліни «Програмування для кібербезпеки». Практикум побудовано навколо Python як мови для автоматизації, опрацювання даних, розроблення службових утиліт і реалізації базових захисних механізмів. Послідовність робіт організовано від фундаментальних конструкцій мови до завершеного мініпроєкту командного рядка.

Основний принцип практикуму — поєднання програмування з безпечною обробкою даних. Студент має не лише отримати правильний результат, а й перевірити вхідні значення, передбачити типові помилки, уникати необґрунтованих небезпечних операцій, забезпечувати зрозумілі повідомлення та тестувати граничні випадки. Кожна лабораторна робота містить 30 індивідуальних варіантів. Варіант визначає викладач.

Основні розділи Python, що охоплюються практикумом

- базовий синтаксис, типи даних, оператори, введення і виведення;
- розгалуження, цикли та алгоритмічне опрацювання послідовностей;
- функції, модулі та оброблення винятків;
- списки, кортежі, множини, словники та comprehensions;
- локальні файли, pathlib, CSV, JSON і журнали подій;
- об'єктно-орієнтоване програмування;
- рядки, регулярні вирази, валідація і нормалізація;
- bytes, bytearray, UTF-8, hex і Base64;
- hashlib, hmac, secrets і контроль цілісності;
- ipaddress і локальна клієнт-серверна взаємодія;
- агрегація журналів, часові вікна і прозорі правила виявлення аномалій;
- argparse, logging, модульна структура, тестування та документування.

Рекомендоване програмне середовище

Python 3.x (актуальна стабільна версія, доступна в лабораторії); PyCharm Community Edition, Visual Studio Code або інше середовище за погодженням із викладачем. Більшість робіт виконується засобами стандартної бібліотеки Python. Сторонні пакети допускаються лише тоді, коли це прямо передбачено завданням або погоджено з викладачем.

Загальні вимоги до виконання

- для кожної роботи створювати окремий каталог і зберігати вихідний код разом із тестовими даними;
- надавати файлам і функціям змістовні імена; уникати пробілів у назвах модулів;
- перевіряти зовнішні дані до їх використання; контролювано обробляти очікувані помилки;
- не зберігати у вихідному коді реальні паролі, токени, ключі або персональні дані;
- виконувати тестування щонайменше на звичайних, граничних і некоректних даних;

ЗМІСТ ЛАБОРАТОРНОГО ПРАКТИКУМУ

№	Тема	сторінка
1	Основи Python. Типи даних, оператори та безпечне введення даних	5
2	Розгалуження та цикли. Аналіз умов і повторюваних подій	9
3	Функції, модулі та оброблення винятків	13
4	Колекції Python. Списки, кортежі, множини та словники	17
5	Робота з файлами, pathlib, CSV і JSON. Оброблення журналів	21
6	Об'єктно-орієнтоване програмування в Python	25
7	Рядки та регулярні вирази. Валідація і нормалізація даних	29
8	Байтові послідовності, кодування та представлення даних	32
9	Хешування, HMAC, безпечні випадкові значення та контроль цілісності	36
10	Мережеві дані в Python: IP-адреси та безпечна локальна взаємодія через socket	40
11	Аналіз журналів безпеки та виявлення аномальних подій	44
12	Підсумковий мініпроект: безпечна утиліта командного рядка на Python	48

ЛАБОРАТОРНА РОБОТА №1

Тема: Основи Python. Типи даних, оператори та безпечне введення даних

Мета: Ознайомитися з базовою структурою програм Python, основними типами даних, операторами, введенням і виведенням; навчитися перевіряти коректність простих вхідних даних і виконувати обчислення для навчальних задач кібербезпеки.

Очікувані результати виконання

- створювати та запускати прості програми Python;
- використовувати типи `int`, `float`, `bool` і `str` та виконувати явні перетворення типів;
- застосовувати арифметичні, логічні та порівняльні оператори;
- форматовувати результати за допомогою f-рядків;
- виконувати базову перевірку вхідних значень до їх використання у програмі.

Теоретичні відомості

1.1. Місце Python у задачах кібербезпеки

Python широко застосовується для автоматизації рутинних операцій, опрацювання журналів подій, аналізу текстових і структурованих даних, побудови невеликих службових утиліт та прототипування засобів контролю. Для навчальних задач важливо не лише отримати правильний результат, а й забезпечити передбачуване поведіння програми з помилковими або неочікуваними даними. У цій роботі основна увага приділяється базовому синтаксису мови та коректному введенню даних.

1.2. Структура програми та змінні

Програма Python складається з інструкцій, що виконуються послідовно. Блоки коду визначаються відступами. Імена змінних мають бути змістовними та відповідати призначенню даних. Python використовує динамічну типізацію: тип пов'язаний із поточним значенням об'єкта, а не з окремою декларацією змінної. Це спрощує написання коду, але вимагає уважності під час перетворення зовнішніх даних.

1.3. Базові типи даних

Для цілих значень використовується `int`, для дійсних — `float`, для логічних — `bool`, для тексту — `str`. Значення, отримане через `input()`, завжди є рядком, тому перед числовими обчисленнями його необхідно перетворити. Явне перетворення виконується функціями `int()`, `float()`, `str()` та `bool()`. Не слід припускати, що введене користувачем значення завжди відповідає очікуваному формату.

1.4. Оператори та порядок обчислень

Python підтримує арифметичні оператори `+`, `-`, `*`, `/`, `//`, `%`, `**`; оператори порівняння `==`, `!=`, `<`, `<=`, `>`, `>=`; логічні оператори `and`, `or`, `not`. Під час обчислення показників, відсотків і коефіцієнтів необхідно враховувати нульові знаменники, допустимі межі та одиниці вимірювання. Для зручного подання числових результатів у f-рядках можна задавати точність, наприклад `{value:.2f}`.

1.5. Базова валідація даних

Валідація — це перевірка того, що значення має очікуваний формат і потрапляє у допустимий діапазон. У найпростіших задачах достатньо перевірити, що рядок не порожній, ціле значення не є від'ємним, кількість невдалих подій не перевищує загальної кількості, а відсоток перебуває в межах від 0 до 100. Перевірка виконується до розрахунків. У наступних роботах для складніших випадків будуть використані винятки та функції.

Приклад 1. Форматування результату

```
total = 250
failed = 17
success = total - failed
failed_percent = failed / total * 100
```

```
print(f"Успішні: {success}")
print(f"Невдали: {failed} ({failed_percent:.2f}%)")
```

Приклад 2. Проста перевірка меж

```
value_text = input("Введіть значення від 0 до 100: ").strip()

if value_text.isdigit():
    value = int(value_text)
    if 0 <= value <= 100:
        print(f"Прийнято: {value}")
    else:
        print("Помилка: значення поза допустимими межами")
else:
    print("Помилка: очікується ціле невід'ємне число")
```

Приклад тестування

Вхідні дані: total = 1000, failed = 37

Очікувана структура результату:

- загальна кількість подій;
- кількість успішних і невдалих подій;
- частка невдалих подій у відсотках;
- повідомлення про помилку для некоректного набору даних.

Порядок виконання роботи

1. Ознайомитися з теоретичними відомостями та контрольними питаннями.
2. Створити окремий каталог лабораторної роботи та підготувати вихідні або тестові дані відповідно до свого варіанта.
3. Спроекувати алгоритм і визначити, які конструкції Python необхідні для його реалізації.
4. Реалізувати програму з перевіркою вхідних даних і зрозумілими повідомленнями про помилки.
5. Перевірити програму щонайменше на трьох коректних, одному граничному та одному некоректному наборі даних.
6. Зафіксувати результати тестування у звіті та пояснити отриманий результат.
7. Підготуватися до захисту: уміти пояснити ключові фрагменти коду та змінити параметр або умову за вказівкою викладача.

Індивідуальні завдання

Виконати завдання відповідно до номера варіанта. Якщо для завдання потрібен набір подій, журнал або каталог файлів, студент використовує синтетичні дані, створені самостійно, або матеріали, надані викладачем.

Варіант	Завдання
1	Обчислити частку успішних і невдалих спроб входу за заданою загальною кількістю та кількістю невдалих спроб. Перевірити, що значення не від'ємні і невдалі спроби не перевищують загальну кількість.

Варіант	Завдання
2	За кількістю подій низького, середнього та високого рівнів обчислити загальну кількість подій і частку кожної категорії у відсотках.
3	За розміром файла в байтах вивести його розмір у KiB та MiB з округленням до двох знаків після коми.
4	За довжиною пароля та кількістю використаних груп символів обчислити умовний показник складності як добуток цих двох значень.
5	За кількістю активних облікових записів і кількістю облікових записів із MFA обчислити відсоток покриття MFA.
6	За кількістю знайдених вразливостей і кількістю виправлених вразливостей обчислити відсоток усунення та кількість невиправлених.
7	За кількістю мережевих пакетів і середнім розміром пакета обчислити загальний обсяг переданих даних у байтах та MiB.
8	За часом початку і завершення перевірки в секундах обчислити тривалість виконання та середню кількість перевірених об'єктів за секунду.
9	За кількістю користувачів, адміністраторів і сервісних облікових записів обчислити загальну кількість та частку привілейованих облікових записів.
10	За кількістю резервних копій та середнім розміром однієї копії в GiB обчислити загальний необхідний обсяг сховища з 15% резервом.
11	За кількістю сповіщень SIEM і кількістю підтверджених інцидентів обчислити частку сповіщень, що стали інцидентами.
12	За кількістю перевірених файлів і кількістю файлів із невідповідним хешем обчислити відсоток порушень цілісності.
13	За загальною кількістю портів і кількістю дозволених портів обчислити кількість заборонених та відсоток дозволених.
14	За кількістю оновлених і неоновлених вузлів обчислити загальну кількість вузлів та відсоток актуальних систем.
15	За кількістю повідомлень електронної пошти і кількістю виявлених фішингових повідомлень обчислити частку фішингових повідомлень.
16	За кількістю дозволених і заблокованих мережевих з'єднань обчислити загальну кількість та відсоток заблокованих.
17	За кількістю співробітників та кількістю тих, хто пройшов навчання з кібербезпеки, обчислити відсоток охоплення навчанням.
18	За кількістю подій журналу за годину та тривалістю спостереження в годинах обчислити очікувану кількість записів у журналі.
19	За початковим і кінцевим розміром журналу в MiB обчислити приріст та відсоток збільшення.
20	За кількістю серверів і середньою кількістю служб на сервері обчислити загальну кількість служб, що підлягають інвентаризації.

Варіант	Завдання
21	За числом спроб автентифікації та середнім часом однієї перевірки в мілісекундах обчислити приблизний сумарний час оброблення в секундах.
22	За кількістю користувачів та середньою кількістю ролей на одного користувача обчислити приблизну кількість призначень ролей.
23	За обсягом мережевого трафіку в MiB і тривалістю сеансу в секундах обчислити середню швидкість передавання в MiB/с.
24	За кількістю активів і кількістю активів без відповідального власника обчислити відсоток активів із призначеним власником.
25	За кількістю правил міжмережевого екрана і кількістю правил, що не використовувалися, обчислити відсоток потенційно зайвих правил.
26	За кількістю критичних оновлень і кількістю встановлених оновлень обчислити кількість відсутніх та відсоток встановлення.
27	За кількістю токенів доступу і кількістю прострочених токенів обчислити частку прострочених токенів.
28	За кількістю подій від одного джерела та загальною кількістю подій обчислити частку цього джерела у відсотках.
29	За кількістю знайдених і перевірених індикаторів компрометації обчислити відсоток опрацьованих індикаторів.
30	За кількістю файлів у каталозі та середнім часом обчислення хешу одного файла оцінити загальний час перевірки цілісності.

Контрольні питання

1. Які базові типи даних Python використовуються у цій роботі?
2. Який тип повертає функція `input()`?
3. Чим оператор `/` відрізняється від `//`?
4. Коли може виникнути помилка ділення на нуль?
5. Для чого застосовують f-рядки?
6. Що таке явне перетворення типів?
7. Чому вхідні дані потрібно перевіряти до обчислень?
8. Які перевірки доцільно виконати для значень, поданих у відсотках?

ЛАБОРАТОРНА РОБОТА №2

Тема: Розгалуження та цикли. Аналіз умов і повторюваних подій

Мета: Навчитися керувати ходом виконання програми за допомогою умовних конструкцій і циклів та застосовувати їх для аналізу послідовностей навчальних подій безпеки.

Очікувані результати виконання

- використовувати if, elif та else для класифікації значень;
- будувати цикли for і while;
- застосовувати range(), enumerate(), break і continue;
- обчислювати лічильники, суми, мінімальні та максимальні значення;
- реалізовувати прозорі порогові правила виявлення подій.

Теоретичні відомості

2.1. Умовні конструкції

Умовні конструкції дають змогу виконувати різні гілки алгоритму залежно від значення логічного виразу. У задачах аналізу подій це використовується для класифікації рівнів ризику, перевірки порогів, визначення стану або вибору дії. Умови мають бути взаємно зрозумілими, а межі діапазонів — однозначними, щоб те саме значення не потрапляло одночасно до двох категорій.

2.2. Цикл for

Цикл for застосовується, коли потрібно послідовно опрацювати елементи колекції або відому кількість повторень. Функція range() формує послідовність цілих значень, а enumerate() дозволяє одночасно отримувати номер позиції та сам елемент. Для статистичного аналізу зазвичай використовують змінні-лічильники та накопичувачі.

2.3. Цикл while

Цикл while виконується, поки умова істинна. Він доречний для повторного введення значення, для алгоритмів із невідомою наперед кількістю кроків або для оброблення потоку до появи завершальної умови. Необхідно гарантувати, що умова циклу колись стане хибною; інакше виникне нескінченний цикл.

2.4. break, continue та вкладені конструкції

Оператор break достроково завершує найближчий цикл, а continue переходить до наступної ітерації. Зловживання ними ускладнює читання коду, тому їх слід застосовувати лише тоді, коли вони роблять алгоритм простішим. Вкладені цикли доцільні для двовимірних даних або попарних перевірок, але потребують оцінювання кількості операцій.

2.5. Порогові правила та граничні значення

У навчальних задачах кібербезпеки часто застосовують поріг: наприклад, подія вважається аномальною, якщо значення перевищує задану межу. Слід чітко розрізняти $>$ і $>=$ та тестувати значення безпосередньо на межі. Порогове правило є лише формальним критерієм і саме по собі не доводить наявності інциденту.

Приклад 1. Класифікація оцінки

```
score = 72

if score < 40:
    level = "низький"
elif score < 70:
    level = "середній"
else:
    level = "високий"

print(level)
```

Приклад 2. Пошук перевищень порогу

```
events = [2, 4, 9, 3, 11, 5]
threshold = 8
count = 0

for index, value in enumerate(events, start=1):
    if value > threshold:
        count += 1
        print(f"Інтервал {index}: {value}")

print(f"Кількість перевищень: {count}")
```

Приклад тестування

Для послідовності 2, 4, 9, 3, 11, 5 та порогу 8 програма має позначити 3-й і 5-й інтервали та вивести кількість перевищень 2. Окремо перевіряються значення 8, 0 та порожня послідовність.

Порядок виконання роботи

1. Ознайомитися з теоретичними відомостями та контрольними питаннями.
2. Створити окремий каталог лабораторної роботи та підготувати вихідні або тестові дані відповідно до свого варіанта.
3. Спроекувати алгоритм і визначити, які конструкції Python необхідні для його реалізації.
4. Реалізувати програму з перевіркою вхідних даних і зрозумілими повідомленнями про помилки.
5. Перевірити програму щонайменше на трьох коректних, одному граничному та одному некоректному наборі даних.
6. Зафіксувати результати тестування у звіті та пояснити отриманий результат.
7. Підготуватися до захисту: уміти пояснити ключові фрагменти коду та змінити параметр або умову за вказівкою викладача.

Індивідуальні завдання

Виконати завдання відповідно до номера варіанта. Якщо для завдання потрібен набір подій, журнал або каталог файлів, студент використовує синтетичні дані, створені самостійно, або матеріали, надані викладачем.

Варіант	Завдання
1	Ввести кількість невдалих входів за кожну з 10 хвилин. Визначити хвилини, у яких число невдач перевищило заданий поріг, і загальну кількість перевищень.
2	Обробити 12 умовних оцінок ризику від 0 до 100 та розподілити їх на низькі, середні й високі за заданими порогоми.
3	Ввести 20 кодів стану НТТР. Порахувати кількість відповідей 2xx, 4xx і 5xx та визначити найчисленнішу групу.
4	Обробити 15 тривалостей сеансів. Вивести сеанси, коротші за 1 хвилину або довші за 8 годин, як аномальні.
5	Для 20 значень використання CPU визначити кількість перевищень 80% та найдовшу послідовність таких перевищень.

Варіант	Завдання
6	Ввести 15 значень розміру файлів. Порахувати, скільки файлів є малими, середніми та великими за заданими межами.
7	Обробити 20 значень кількості пакетів за інтервал. Визначити інтервали, де значення перевищує середнє більш ніж у 2 рази.
8	Ввести 15 результатів перевірки цілісності (1 — збіг, 0 — невідповідність). Порахувати невідповідності та визначити, чи перевищено допустимий поріг.
9	Обробити 30 оцінок складності паролів від 0 до 100. Порахувати значення у трьох категоріях та визначити частку слабких.
10	Ввести 20 номерів портів і визначити, скільки належать до діапазонів well-known, registered та dynamic/private.
11	Обробити 25 значень часу відповіді сервера. Порахувати відповіді до 100 мс, 100–500 мс і понад 500 мс.
12	Ввести 14 добових кількостей інцидентів. Знайти день із максимальним значенням і кількість днів вище середнього.
13	Обробити 20 значень кількості запитів до сервісу. Визначити перший інтервал, коли значення перевищить заданий критичний поріг.
14	Ввести 25 числових ідентифікаторів подій. Порахувати парні та непарні значення і суму значень у заданому діапазоні.
15	Обробити 20 рівнів заповнення диска. Порахувати стани normal (<70), warning (70–89) та critical (>=90).
16	Ввести 10 значень часу блокування облікового запису. Визначити мінімум, максимум і кількість блокувань довших за задане значення.
17	Обробити 30 значень результату перевірки URL: 1 — дозволено, 0 — заблоковано. Визначити найдовшу послідовність заблокованих URL.
18	Ввести 20 номерів версій оновлення у вигляді цілих чисел. Порахувати вузли з версією нижче мінімально дозволеної.
19	Обробити 24 погодинні значення кількості подій. Вивести години, у яких кількість перевищує середнє значення дня.
20	Ввести 20 значень довжини токенів. Порахувати токени коротші за задану мінімальну довжину та вивести їх індекси.
21	Обробити 15 чисел від 0 до 255 як байти. Вивести значення, що виходять за допустимий заданий піддіапазон.
22	Ввести 20 оцінок критичності активів від 1 до 5. Побудувати частотний розподіл за категоріями 1–5.
23	Обробити 25 значень затримки мережі. Визначити кількість послідовних інтервалів, у яких затримка зростала.
24	Ввести 20 кодів завершення процесів. Порахувати нульові та ненульові коди й визначити відсоток помилкових завершень.

Варіант	Завдання
25	Обробити 30 значень кількості невдалих входів користувачів. Порахувати користувачів із 0, 1–3 та понад 3 невдачами.
26	Ввести 15 значень температури серверного обладнання. Визначити кількість попереджень і критичних значень за двома порогами.
27	Обробити 20 значень кількості відкритих з'єднань. Вивести всі локальні максимуми, більші за сусідні значення.
28	Ввести 25 значень розміру мережевих пакетів. Порахувати пакети менші 64, від 64 до 1500 і більші 1500 байтів.
29	Обробити 20 балів виявлення аномалії від 0 до 1. За заданим порогом сформувати список індексів аномальних спостережень.
30	Ввести 30 значень тривалості блокування IP. Визначити середнє, кількість значень вище середнього та максимальне значення.

Контрольні питання

1. Як працює конструкція if/elif/else?
2. Коли доцільно використовувати for, а коли while?
3. Що повертає range()?
4. Для чого потрібна enumerate()?
5. Чим відрізняються break і continue?
6. Чому важливо перевіряти граничні значення порогів?
7. Як реалізувати лічильник подій?
8. Чому перевищення порогу не можна автоматично вважати підтвердженням інцидентом?

ЛАБОРАТОРНА РОБОТА №3

Тема: Функції, модулі та оброблення винятків

Мета: Навчитися декомпозувати програму на функції та модулі, передавати параметри, повертати результати й контролювати обробляти очікувані помилки.

Очікувані результати виконання

- створювати функції з параметрами та результатами;
- використовувати позиційні, іменовані та стандартні аргументи;
- розділяти програму на модулі;
- використовувати try/except/else/finally;
- генерувати ValueError для некоректних значень та не приховувати неочікувані помилки.

Теоретичні відомості

3.1. Декомпозиція програми

Функція інкапсулює окрему завершену операцію. Декомпозиція зменшує дублювання, спрощує тестування і локалізує помилки. Хороша функція виконує одну зрозумілу дію, має змістовну назву та чітко визначені вхідні й вихідні дані. У задачах кібербезпеки окремими функціями доцільно оформлювати нормалізацію, перевірку форматів, обчислення показників і формування висновків.

3.2. Параметри та return

Параметри передають функції необхідні дані. Оператор return завершує функцію та повертає результат. Якщо функція має лише друкувати інформацію, її складніше повторно використовувати й тестувати, тому розрахункові функції краще будувати так, щоб вони повертали значення, а виведення виконувалося окремо.

3.3. Модулі

Модуль — це файл Python, який містить функції, класи або константи. Поділ на модулі дозволяє відокремити бізнес-логіку від введення, виведення і тестів. Імпортувати слід лише потрібні модулі. Конструкція `if __name__ == "__main__":` дає змогу відрізнити безпосередній запуск файла від його імпорту як модуля.

3.4. Винятки

Виняток сигналізує про ситуацію, в якій операцію неможливо коректно продовжити. Блок try містить потенційно помилкову операцію, except обробляє конкретні типи винятків, else виконується за відсутності помилки, а finally — незалежно від результату. Не слід використовувати порожній except, оскільки він приховує причину збою.

3.5. Генерування винятків і безпечні повідомлення

Оператор raise дає змогу явно повідомити про некоректне значення, наприклад про номер порту поза діапазоном або від'ємну кількість подій. Текст повідомлення має бути корисним для користувача, але не повинен без потреби розкривати внутрішні шляхи, секрети чи інші чутливі дані.

Приклад 1. Функція з перевіркою

```
def percent(part: int, total: int) -> float:
    if total <= 0:
        raise ValueError("total має бути додатним")
    if not 0 <= part <= total:
        raise ValueError("part має бути в межах 0..total")
    return part / total * 100

print(f"{percent(17, 250):.2f}%")
```

Приклад 2. Контрольоване оброблення помилки

```
try:
    value = int(input("Введіть ціле число: "))
except ValueError:
    print("Помилка: введено не ціле число")
else:
    print(f"Прийнято: {value}")
```

Рекомендована структура

```
project/
main.py      # взаємодія з користувачем
validators.py # перевірки та нормалізація
metrics.py   # обчислення
tests/       # тестові дані або тести
```

Основну логіку слід розміщувати у функціях, а `main.py` використовувати як точку входу.

Порядок виконання роботи

1. Ознайомитися з теоретичними відомостями та контрольними питаннями.
2. Створити окремий каталог лабораторної роботи та підготувати вихідні або тестові дані відповідно до свого варіанта.
3. Спроекувати алгоритм і визначити, які конструкції Python необхідні для його реалізації.
4. Реалізувати програму з перевіркою вхідних даних і зрозумілими повідомленнями про помилки.
5. Перевірити програму щонайменше на трьох коректних, одному граничному та одному некоректному наборі даних.
6. Зафіксувати результати тестування у звіті та пояснити отриманий результат.
7. Підготуватися до захисту: уміти пояснити ключові фрагменти коду та змінити параметр або умову за вказівкою викладача.

Індивідуальні завдання

Виконати завдання відповідно до номера варіанта. Якщо для завдання потрібен набір подій, журнал або каталог файлів, студент використовує синтетичні дані, створені самостійно, або матеріали, надані викладачем.

Варіант	Завдання
1	Створити функції для перевірки довжини пароля, підрахунку груп символів і формування текстового висновку щодо політики паролів.
2	Створити модуль із функціями для обчислення відсотка невдалих входів і перевірки перевищення порогу.
3	Реалізувати функцію нормалізації імені користувача (видалення крайніх пробілів, приведення до нижнього регістру) та функцію перевірки його довжини.
4	Реалізувати функції для перетворення розміру файла між B, KiB, MiB і GiB з обробленням невідомої одиниці.
5	Створити функцію класифікації оцінки ризику за трьома категоріями; порогові передавати аргументами.

Варіант	Завдання
6	Реалізувати модуль для обчислення середнього, мінімального і максимального часу відповіді сервісу.
7	Створити функцію перевірки номера порту та функцію визначення його діапазону. Некоректні значення мають викликати ValueError.
8	Реалізувати функцію для обчислення SHA-подібного навчального контрольного значення як суми кодів символів за модулем 256 (не називати його криптографічним хешем).
9	Створити функції для підрахунку активних, заблокованих та прострочених облікових записів за списком статусів.
10	Реалізувати функцію перевірки відсоткового значення (0–100) і функцію формування рівня попередження.
11	Створити модуль для розрахунку темпу подій: кількість подій за секунду, хвилину та годину.
12	Реалізувати функції для перетворення секунд у години/хвилини/секунди та перевірки від’ємних значень.
13	Створити функцію, що приймає список оцінок уразливостей та повертає кількість оцінок у заданому інтервалі.
14	Реалізувати функцію пошуку найбільшого значення без використання max() і винести її в окремий модуль.
15	Створити функції для перевірки формату простого ідентифікатора: дозволити літери, цифри, дефіс і підкреслення без регулярних виразів.
16	Реалізувати функцію безпечного ділення з обробленням ділення на нуль та некоректного типу даних.
17	Створити функцію підрахунку частоти заданого коду події у списку та функцію формування короткого звіту.
18	Реалізувати модуль для оцінювання заповнення диска: вільний обсяг, зайнятий відсоток, текстовий стан.
19	Створити функцію для перевірки часової мітки як невід’ємного цілого числа та функцію обчислення різниці двох міток.
20	Реалізувати функцію для перевірки MAC-подібного рядка за довжиною і роздільниками без регулярних виразів.
21	Створити модуль із функціями для підрахунку успішних і невдалих результатів перевірок цілісності.
22	Реалізувати функцію для обчислення середнього лише з коректних числових значень; некоректні пропускати з повідомленням.
23	Створити функцію, що приймає список ролей і повертає True, якщо серед них є одна з привілейованих ролей, заданих окремим набором.
24	Реалізувати функції для обчислення умовного показника ризику як добутку ймовірності та впливу з перевіркою діапазонів 1–5.

Варіант	Завдання
25	Створити модуль для перетворення текстового рівня critical/high/medium/low у числовий пріоритет і навпаки.
26	Реалізувати функцію для пошуку першого значення, що перевищує поріг, із поверненням його індексу або None.
27	Створити функцію для підрахунку унікальних значень у списку без використання set(), а потім порівняти результат із set().
28	Реалізувати функцію перевірки, чи входить числове значення до заданого закритого інтервалу; некоректні межі мають спричиняти виняток.
29	Створити модуль для обчислення відсотка покриття MFA та формування стану: недостатнє, прийнятне, повне.
30	Реалізувати функцію safe_int(), яка перетворює рядок у ціле число, а у разі помилки повертає задане значення за замовчуванням.

Контрольні питання

1. Що таке декомпозиція програми?
2. Чим параметр функції відрізняється від аргументу?
3. Навіщо повертати результат через return?
4. Для чого використовують модулі?
5. Що означає конструкція if `__name__ == "__main__"`?
6. Чим `except ValueError` кращий за порожній `except`?
7. Коли доцільно застосовувати `raise`?
8. Яку інформацію не слід включати до повідомлень про помилки?

ЛАБОРАТОРНА РОБОТА №4

Тема: Колекції Python. Списки, кортежі, множини та словники

Мета: Навчитися обирати та використовувати колекції Python для накопичення, пошуку, групування і статистичного аналізу даних.

Очікувані результати виконання

- створювати та змінювати списки;
- використовувати кортежі для незмінних структур;
- застосовувати множини для унікалізації та операцій над наборами;
- будувати словники для відображення ключ-значення;
- використовувати comprehensions і стандартні прийоми агрегації.

Теоретичні відомості

4.1. Вибір структури даних

Колекція має відповідати характеру задачі. Список зберігає впорядковану змінювану послідовність; кортеж — впорядковану незмінну; множина — набір унікальних елементів без індексації; словник — відповідність між ключами та значеннями. Правильний вибір структури часто спрощує алгоритм і зменшує кількість ручних перевірок.

4.2. Списки та зрізи

Списки підтримують індексацію, зрізи, `append()`, `extend()`, `remove()`, `pop()`, `sort()`. Під час оброблення журналів або масивів показників не слід змінювати список у циклі, який одночасно його перебирає, якщо це не контролюється явно. Для формування нового набору значень часто безпечніше створити новий список.

4.3. Множини

Множини зручні для пошуку унікальних значень, перетину, об'єднання і різниці наборів. Наприклад, дозволи двох ролей можна порівнювати операторами `&`, `|` і `-`. Порядок елементів множини не слід використовувати як значущу частину результату.

4.4. Словники та підрахунок частот

Словник дає змогу швидко отримати значення за ключем. Для підрахунку частот застосовують шаблон `d[key] = d.get(key, 0) + 1` або `collections.Counter`. У задачах аналізу подій ключем може бути користувач, адреса, код стану чи тип події.

4.5. Генератори колекцій

List/set/dict comprehensions дозволяють компактно створювати нові колекції. Їх варто використовувати для простих перетворень і фільтрації. Якщо вираз стає складним або містить кілька умовних рівнів, краще перейти до звичайного циклу для збереження читабельності.

Приклад 1. Частоти значень

```
users = ["anna", "ivan", "anna", "oleh", "ivan", "anna"]
frequency = {}

for user in users:
    frequency[user] = frequency.get(user, 0) + 1

for user, count in frequency.items():
    print(user, count)
```

Приклад 2. Операції над множинами

```
role_a = {"read", "write", "export"}
role_b = {"read", "audit"}
```

```
print("Спільні:", role_a & role_b)
print("Лише A:", role_a - role_b)
print("Усі:", role_a | role_b)
```

Приклад тестування

Для списку ["u1", "u2", "u1", "u3", "u2", "u1"] очікується словник частот {"u1": 3, "u2": 2, "u3": 1}, множина унікальних значень {"u1", "u2", "u3"} і можливість відсортувати підсумок за спаданням частоти.

Порядок виконання роботи

1. Ознайомитися з теоретичними відомостями та контрольними питаннями.
2. Створити окремий каталог лабораторної роботи та підготувати вихідні або тестові дані відповідно до свого варіанта.
3. Спроекувати алгоритм і визначити, які конструкції Python необхідні для його реалізації.
4. Реалізувати програму з перевіркою вхідних даних і зрозумілими повідомленнями про помилки.
5. Перевірити програму щонайменше на трьох коректних, одному граничному та одному некоректному наборі даних.
6. Зафіксувати результати тестування у звіті та пояснити отриманий результат.
7. Підготуватися до захисту: уміти пояснити ключові фрагменти коду та змінити параметр або умову за вказівкою викладача.

Індивідуальні завдання

Виконати завдання відповідно до номера варіанта. Якщо для завдання потрібен набір подій, журнал або каталог файлів, студент використовує синтетичні дані, створені самостійно, або матеріали, надані викладачем.

Варіант	Завдання
1	За списком імен користувачів визначити унікальні імена, дублікати та кількість появ кожного імені.
2	За списком IP-адрес як рядків підрахувати частоту кожної адреси та вивести п'ять найчастіших.
3	За словником {користувач: роль} сформувати окремі списки користувачів для кожної ролі.
4	За списком кодів HTTP сформувати словник частот і відсортувати коди за спаданням частоти.
5	За двома множинами дозволених і фактично використаних сервісів визначити дозвалені, але не використані; використані, але не дозвалені; спільні.
6	За списком словників із полями user, status, attempts знайти користувачів зі status="locked" та attempts>3.
7	За словником {файл: хеш} і другим словником контрольних хешів визначити нові, видалені та змінені файли.
8	За списком доменів нормалізувати реєстр, видалити дублікати та відсортувати результат.
9	За списком пар (IP, кількість подій) побудувати словник і знайти адреси зі значенням вище середнього.

Варіант	Завдання
10	За словником {актив: критичність} згрупувати активи за рівнями 1–5.
11	За списком словників подій згрупувати їх за полем severity та підрахувати кількість у кожній групі.
12	За двома множинами користувачів — активні та з MFA — визначити активних користувачів без MFA.
13	За списком розширень файлів побудувати частотний словник та вивести три найпоширеніші.
14	За словником {порт: сервіс} сформувати зворотний словник {сервіс: список портів}.
15	За списком ролей користувача та множиною привілейованих ролей визначити перетин і сформувати висновок.
16	За списком словників з полями host і os побудувати словник {ОС: кількість вузлів}.
17	За списком ідентифікаторів подій вилучити дублікати, зберігши порядок першої появи.
18	За словником {користувач: список груп} знайти всіх користувачів, що входять одночасно до двох заданих груп.
19	За списком URL виділити унікальні доменні частини, вважаючи вхідні URL уже розбитими на домен і шлях.
20	За словником {вузол: список відкритих портів} знайти вузли, де присутній хоча б один порт із заданої множини контролю.
21	За списком словників авторизацій підрахувати успішні й невдалі входи для кожного користувача.
22	За двома словниками інвентаризації активів визначити активи зі зміненою критичністю.
23	За списком хешів файлів визначити дублікати хешів і кількість файлів із кожним дубльованим значенням.
24	За словником {IP: країна} та списком подій побудувати статистику кількості подій за країнами.
25	За списком імен файлів згрупувати їх за розширенням у словник {розширення: список файлів}.
26	За словником {користувач: last_login_days} сформувати множину користувачів, які не входили понад задану кількість днів.
27	За списком словників із полями process і parent сформувати множину унікальних пар процес-батьківський процес.
28	За списком рядків індикаторів компрометації розділити їх на три колекції: IP-подібні, доменні та інші за простими правилами.
29	За словником {правило: hits} знайти правила з нульовим лічильником та сформувати впорядкований список кандидатів на перегляд.

Варіант	Завдання
30	За списком словників {asset, owner, criticality} побудувати словник відповідальних осіб і перелік їхніх активів.

Контрольні питання

1. Які властивості має list?
2. Коли доцільно використовувати tuple?
3. Чим set відрізняється від list?
4. Для чого використовують dict.get()?
5. Що таке ключ словника?
6. Які операції підтримують множини?
7. Коли comprehensions покращують код, а коли ускладнюють?
8. Чому не слід покладатися на порядок елементів множини?

ЛАБОРАТОРНА РОБОТА №5

Тема: Робота з файлами, pathlib, CSV і JSON. Оброблення журналів

Мета: Навчитися безпечно читати та записувати локальні файли, працювати з pathlib, CSV і JSON та формувати структуровані результати опрацювання журналів.

Очікувані результати виконання

- відкривати текстові файли з явним кодуванням;
- використовувати pathlib.Path для локальних шляхів;
- читати та записувати CSV через модуль csv;
- серіалізувати та десеріалізувати JSON;
- контролювати обробляти відсутні, пошкоджені або некоректні файли.

Теоретичні відомості

5.1. Текстові та двійкові файли

Для текстових файлів доцільно явно задавати encoding="utf-8". Контекстний менеджер with автоматично закриває файл навіть у випадку помилки. У цій роботі використовуються лише локальні навчальні файли, створені студентом або надані викладачем.

5.2. pathlib

Клас Path забезпечує незалежніший від операційної системи спосіб роботи зі шляхами. Методи exists(), is_file(), is_dir(), glob() та iter_dir() дають змогу перевіряти об'єкти файлової системи. Перед записом слід переконатися, що цільовий шлях належить до робочого каталогу лабораторної роботи та не перезаписує сторонні дані.

5.3. CSV

CSV використовується для табличних даних. Модуль csv коректно обробляє роздільники та лапки, тому не варто розбирати складний CSV через split(). DictReader і DictWriter дозволяють працювати з назвами полів. Перед використанням значень потрібно перевіряти наявність обов'язкових колонок і коректність типів.

5.4. JSON

JSON зручний для структурованих вкладених даних. json.load() читає дані з файла, json.dump() записує. Параметр ensure_ascii=False дозволяє зберігати український текст без escape-послідовностей, а indent робить файл читабельним. Дані JSON не слід вважати довіреними лише тому, що файл синтаксично коректний.

5.5. Опрацювання журналів

Журнал доцільно розглядати як послідовність записів із визначеною схемою: час, суб'єкт, джерело, статус, повідомлення тощо. Етапи опрацювання: читання, перевірка структури, нормалізація, агрегація, формування звіту. Некоректні рядки слід або відхиляти з поясненням, або накопичувати окремо для аналізу.

Приклад 1. Читання CSV

```
import csv
from pathlib import Path

path = Path("events.csv")
with path.open("r", encoding="utf-8", newline="") as f:
    reader = csv.DictReader(f)
    for row in reader:
        print(row.get("user"), row.get("status"))
```

Приклад 2. Запис JSON

```
import json
from pathlib import Path

report = {"processed": 120, "errors": 3}
Path("report.json").write_text(
    json.dumps(report, ensure_ascii=False, indent=2),
    encoding="utf-8"
)
```

Зразок вхідного CSV

```
timestamp,user,status
2026-09-01T10:00:00,anna,success
2026-09-01T10:01:00,ivan,failed
2026-09-01T10:02:00,anna,failed
```

Очікуваний результат — агрегований CSV/JSON відповідно до варіанта та окреме повідомлення про некоректні рядки.

Порядок виконання роботи

1. Ознайомитися з теоретичними відомостями та контрольними питаннями.
2. Створити окремий каталог лабораторної роботи та підготувати вихідні або тестові дані відповідно до свого варіанта.
3. Спроекувати алгоритм і визначити, які конструкції Python необхідні для його реалізації.
4. Реалізувати програму з перевіркою вхідних даних і зрозумілими повідомленнями про помилки.
5. Перевірити програму щонайменше на трьох коректних, одному граничному та одному некоректному наборі даних.
6. Зафіксувати результати тестування у звіті та пояснити отриманий результат.
7. Підготуватися до захисту: уміти пояснити ключові фрагменти коду та змінити параметр або умову за вказівкою викладача.

Індивідуальні завдання

Виконати завдання відповідно до номера варіанта. Якщо для завдання потрібен набір подій, журнал або каталог файлів, студент використовує синтетичні дані, створені самостійно, або матеріали, надані викладачем.

Варіант	Завдання
1	Прочитати текстовий журнал входів формату "timestamp,user,status" і сформувати CSV зі статистикою успішних та невдалих входів для кожного користувача.
2	Прочитати CSV із полями host,os,status та сформувати JSON зі списком вузлів, у яких status="outdated".
3	Прочитати JSON-масив подій із полями severity і message та записати окремий текстовий файл лише з подіями high і critical.
4	Побудувати звіт про кількість файлів кожного розширення у заданому локальному тестовому каталозі.
5	Для набору текстових файлів обчислити їх розміри та сформувати CSV з полями name,size_bytes,size_kib.

Варіант	Завдання
6	Прочитати CSV із результатами перевірки цілісності file,expected,actual та сформувати JSON лише для невідповідностей.
7	Прочитати журнал "IP;user;result" і сформувати список IP, з яких було щонайменше три невдалі спроби.
8	Прочитати JSON зі списком активів і сформувати CSV з активами, для яких не задано owner.
9	Прочитати CSV із полями user,mfa та сформувати текстовий звіт про користувачів без MFA.
10	Прочитати журнал із рядками "timestamp code message" та підрахувати кількість появ кожного числового коду.
11	Прочитати JSON зі списком правил firewall {id,enabled,hits} та сформувати CSV для увімкнених правил із hits=0.
12	Знайти у тестовому каталозі всі файли з розширенням .log і сформувати JSON із їх іменами та розмірами.
13	Прочитати CSV журналу веб-сервера з полями ip,status,path та сформувати статистику кодів 4xx для кожної IP-адреси.
14	Прочитати текстовий список доменів, нормалізувати регістр, вилучити дублікати та записати відсортований результат у новий файл.
15	Прочитати JSON із подіями {user,action} та сформувати словник частоти дій для кожного користувача; результат зберегти як JSON.
16	Прочитати CSV із полями asset,criticality,owner та сформувати окремий CSV лише для активів критичності 4–5.
17	Прочитати текстовий журнал і підрахувати кількість рядків, порожніх рядків та рядків, що починаються із символу #.
18	Об'єднати два CSV-файли інвентаризації вузлів за спільним полем host та сформувати список вузлів, присутніх лише в одному файлі.
19	Прочитати JSON із полями file,hash та сформувати текстовий manifest у форматі "hash filename".
20	Прочитати CSV із подіями {time,source,severity} та сформувати окремі JSON-файли для кожного рівня severity.
21	Прочитати текстовий список імен файлів та перевірити, які з них існують у заданому тестовому каталозі; результат записати в CSV.
22	Прочитати CSV із полями user,last_login_days та сформувати JSON зі списком облікових записів, неактивних понад задану кількість днів.
23	Прочитати JSON зі списком програм {name,version,approved} та сформувати CSV для програм, у яких approved=false.
24	Прочитати текстовий журнал помилок і сформувати частотний словник перших слів кожного непорожнього рядка; зберегти в JSON.

Варіант	Завдання
25	Прочитати CSV із полями host,port,state та сформувати JSON зі списком відкритих портів для кожного host.
26	Прочитати JSON зі списком користувачів і ролей та сформувати CSV для користувачів, які мають більше двох ролей.
27	Побудувати рекурсивний перелік файлів у тестовому каталозі та зберегти відносний шлях і розмір кожного файла в CSV.
28	Прочитати CSV журналу перевірок {check,result} і сформувати короткий текстовий звіт із кількістю PASS і FAIL.
29	Прочитати JSON з інформацією про резервні копії {name,size_gib,status} і сформувати список невдалих копій та їх сумарний розмір.
30	Прочитати текстовий журнал, вилучити порожні рядки та рядки-коментарі, пронумерувати решту і записати в новий файл.

Контрольні питання

1. Чому бажано явно задавати UTF-8?
2. Для чого використовується with?
3. Які переваги pathlib порівняно з ручним складанням шляхів?
4. Чому CSV не слід розбирати лише split()?
5. Чим json.load() відрізняється від json.loads()?
6. Які поля журналу варто перевіряти до аналізу?
7. Як обробляти пошкоджені рядки?
8. Чому робота має виконуватися лише з локальними навчальними файлами?

ЛАБОРАТОРНА РОБОТА №6

Тема: Об'єктно-орієнтоване програмування в Python

Мета: Навчитися моделювати сутності предметної області за допомогою класів Python, визначати стан і поведінку об'єктів та застосовувати композицію і просте спадкування.

Очікувані результати виконання

- створювати класи та об'єкти;
- визначати `__init__`, атрибути та методи;
- використовувати властивості та інкапсуляцію на рівні інтерфейсу класу;
- будувати зв'язки між об'єктами через композицію;
- застосовувати спадкування там, де воно відображає реальне відношення типів.

Теоретичні відомості

6.1. Клас і об'єкт

Клас описує структуру та поведінку типу об'єктів, а об'єкт є конкретним екземпляром класу. Конструктор `__init__` ініціалізує стан. Параметр `self` посиляється на поточний об'єкт. У навчальних моделях кібербезпеки класами можуть бути `UserAccount`, `SecurityEvent`, `Asset`, `FileRecord` або `Policy`.

6.2. Атрибути та методи

Атрибути зберігають стан, методи реалізують операції над цим станом. Важливо не перетворювати клас на простий контейнер, якщо поведінка пов'язана із самим об'єктом. Наприклад, перевірку критичності події природно реалізувати методом `SecurityEvent.is_critical()`.

6.3. Інкапсуляція в Python

Python покладається переважно на домовленості. Ім'я з одним початковим підкресленням позначає внутрішній атрибут. Властивість `@property` дозволяє надати контрольований інтерфейс читання або перевірити значення під час встановлення. Це не є криптографічним або системним захистом, а засобом проєктування інтерфейсу класу.

6.4. Композиція та агрегація

Композиція означає, що один об'єкт містить або використовує інший. Наприклад, `UserAccount` може містити набір `Role`, а `Asset` — посилання на `Owner`. У більшості прикладних моделей композиція простіша й гнучкіша за глибокі ієрархії спадкування.

6.5. Спадкування та поліморфізм

Спадкування слід застосовувати, якщо дочірній клас справді є спеціалізованим різновидом базового. Перевизначення методу дозволяє змінювати поведінку. Надмірне спадкування ускладнює код, тому в цій роботі достатньо простих ієрархій із одним рівнем спадкування.

Приклад 1. Простий клас

```
class SecurityEvent:
    def __init__(self, source: str, severity: int):
        if not 1 <= severity <= 5:
            raise ValueError("severity має бути 1..5")
        self.source = source
        self.severity = severity

    def is_critical(self) -> bool:
        return self.severity >= 4

event = SecurityEvent("gateway", 4)
print(event.is_critical())
```

Приклад 2. Композиція

```
class Role:
    def __init__(self, name: str):
        self.name = name

class UserAccount:
    def __init__(self, username: str, roles: list[Role]):
        self.username = username
        self.roles = roles

    def role_names(self) -> list[str]:
        return [role.name for role in self.roles]
```

Структура перевірки класу

1. Створити коректний об'єкт.
2. Вивести ключові атрибути.
3. Викликати основні методи.
4. Спробувати створити об'єкт із некоректними значеннями.
5. Перевірити взаємодію двох пов'язаних об'єктів.

Порядок виконання роботи

1. Ознайомитися з теоретичними відомостями та контрольними питаннями.
2. Створити окремий каталог лабораторної роботи та підготувати вихідні або тестові дані відповідно до свого варіанта.
3. Спроекувати алгоритм і визначити, які конструкції Python необхідні для його реалізації.
4. Реалізувати програму з перевіркою вхідних даних і зрозумілими повідомленнями про помилки.
5. Перевірити програму щонайменше на трьох коректних, одному граничному та одному некоректному наборі даних.
6. Зафіксувати результати тестування у звіті та пояснити отриманий результат.
7. Підготуватися до захисту: уміти пояснити ключові фрагменти коду та змінити параметр або умову за вказівкою викладача.

Індивідуальні завдання

Виконати завдання відповідно до номера варіанта. Якщо для завдання потрібен набір подій, журнал або каталог файлів, студент використовує синтетичні дані, створені самостійно, або матеріали, надані викладачем.

Варіант	Завдання
1	Створити класи UserAccount і Role. UserAccount містить список ролей та метод has_privileged_role().
2	Створити класи SecurityEvent і EventSource. Подія має рівень критичності, джерело та метод is_critical().
3	Створити класи Asset і Owner. Актив має критичність 1–5 та посилання на відповідального власника.
4	Створити класи FileRecord і HashRecord для опису файла та його контрольного значення; реалізувати метод compare_hash().

Варіант	Завдання
5	Створити базовий клас Alert і похідні LoginAlert та IntegrityAlert зі спільним методом summary().
6	Створити класи FirewallRule і FirewallPolicy. Політика містить набір правил і метод для підрахунку увімкнених правил.
7	Створити класи Host і Service. Host містить список сервісів та метод get_open_ports().
8	Створити класи Incident і Evidence. Інцидент містить колекцію доказів і метод add_evidence().
9	Створити базовий клас Credential та похідні PasswordCredential і TokenCredential з методом is_valid().
10	Створити класи LogEntry і LogFile. LogFile містить список записів та метод filter_by_severity().
11	Створити класи Vulnerability і Asset. Актив містить список уразливостей та метод max_severity().
12	Створити класи BackupJob і BackupResult. Завдання резервного копіювання містить останній результат і метод is_successful().
13	Створити базовий клас ScannerResult і похідні FileScanResult та ConfigScanResult зі спільним методом status_text().
14	Створити класи NetworkEndpoint і Connection. З'єднання містить два endpoint та метод is_internal() за заданою ознакою endpoint.
15	Створити класи PolicyRule і PasswordPolicy. PasswordPolicy містить правила мінімальної довжини та необхідних груп символів.
16	Створити класи User і AuthenticationAttempt. User містить список спроб та метод failed_attempts_count().
17	Створити класи ProcessInfo і ProcessTreeNode. Вузол дерева містить дочірні процеси та метод add_child().
18	Створити класи IOC і IOCCollection. Колекція має метод add_unique() та фільтрацію за типом IOC.
19	Створити базовий клас Report та похідні TextReport і JsonReport зі спільним методом render().
20	Створити класи CertificateInfo і CertificateStore. Сховище має метод expired_items() за умовним числом днів до завершення.
21	Створити класи Permission і AccessProfile. Профіль містить набір дозволів та метод can(action).
22	Створити класи SoftwareItem і Inventory. Inventory має метод unapproved_items().
23	Створити класи DetectionRule і RuleSet. Набір правил має метод enabled_rules() та підрахунок правил за критичністю.
24	Створити базовий клас Message і похідні EmailMessage та ChatMessage; реалізувати поліморфний метод sender_id().

Варіант	Завдання
25	Створити класи Session і User. User містить список сеансів та метод longest_session().
26	Створити класи KeyMetadata і KeyStore. Сховище містить метадані ключів і метод items_near_expiry().
27	Створити класи Finding і AuditReport. Звіт містить результати перевірок та метод findings_by_severity().
28	Створити класи RuleMatch і Event. Подія містить список спрацювань правил та метод matched_rule_ids().
29	Створити класи Device і DeviceGroup. Група містить пристрої та метод outdated_devices().
30	Створити базовий клас AuthFactor та похідні PasswordFactor і MfaFactor; користувач має список факторів і метод mfa_enabled().

Контрольні питання

1. Чим клас відрізняється від об'єкта?
2. Для чого використовується __init__?
3. Що означає self?
4. Що позначає початкове підкреслення в імені атрибута?
5. Для чого використовують @property?
6. Чим композиція відрізняється від спадкування?
7. Коли спадкування є виправданим?
8. Чому глибокі ієрархії класів можуть ускладнювати супровід?

ЛАБОРАТОРНА РОБОТА №7

Тема: Рядки та регулярні вирази. Валідація і нормалізація даних

Мета: Навчитися обробляти текстові дані, виконувати нормалізацію та використовувати регулярні вирази для перевірки й вилучення даних із чітко визначеним форматом.

Очікувані результати виконання

- використовувати методи str для очищення і нормалізації;
- застосовувати split(), join(), replace(), startswith() та endswith();
- створювати базові регулярні вирази;
- використовувати re.fullmatch(), re.search(), re.finditer();
- розрізняти синтаксичну перевірку формату й змістову валідацію.

Теоретичні відомості

7.1. Нормалізація тексту

Перед порівнянням текстові дані часто потребують нормалізації: видалення крайніх пробілів, приведення до єдиного регістру, уніфікації роздільників або видалення зайвих пробілів. Нормалізація має бути контрольованою: не можна бездумно змінювати дані, для яких регістр або пробіли мають значення.

7.2. Методи рядків

Методи strip(), lower(), upper(), split(), join(), replace(), startswith(), endswith() розв'язують значну частину задач без регулярних виразів. Простий метод рядка зазвичай кращий за складний regex, якщо він однозначно описує вимогу.

7.3. Основи регулярних виразів

Регулярний вираз задає шаблон символів. Класи символів, квантифікатори та групи дозволяють описувати структуровані фрагменти. Для перевірки всього рядка доцільно використовувати re.fullmatch(), а для пошуку фрагментів — re.search() або re.finditer(). Сирі рядки r"..." спрощують запис зворотних слешів.

7.4. Валідація формату

Regex підтверджує лише відповідність шаблону, але не гарантує змістову коректність. Наприклад, рядок може бути схожим на IP-адресу, але містити октет 999. Для IP-адрес у наступній роботі буде використано спеціалізований модуль ipaddress.

7.5. Безпечне застосування regex

Надмірно складні шаблони можуть бути важкими для підтримки та повільними на спеціально підібраних рядках. У навчальних роботах слід використовувати прості обмежені шаблони й тестувати їх на довгих та некоректних рядках. Вхідний текст не повинен підставлятися у regex як готовий шаблон без потреби.

Приклад 1. Нормалізація

```
raw = "  ADMIN.User  "
normalized = raw.strip().lower()
print(normalized)
```

Приклад 2. Перевірка імені

```
import re

pattern = re.compile(r"[A-Za-z][A-Za-z0-9_-]{3,19}")
username = "student_01"

if pattern.fullmatch(username):
    print("Формат коректний")
```

```
else:  
    print("Формат некоректний")
```

Приклад набору тестів

Коректні: student_01, user.name, A-123

Некоректні: 1user, ab, user space, дуже_довге_імя_поза_умовою

Для кожного тесту у звіті слід пояснити, чому результат очікується саме таким.

Порядок виконання роботи

1. Ознайомитися з теоретичними відомостями та контрольними питаннями.
2. Створити окремий каталог лабораторної роботи та підготувати вихідні або тестові дані відповідно до свого варіанта.
3. Спроекувати алгоритм і визначити, які конструкції Python необхідні для його реалізації.
4. Реалізувати програму з перевіркою вхідних даних і зрозумілими повідомленнями про помилки.
5. Перевірити програму щонайменше на трьох коректних, одному граничному та одному некоректному наборі даних.
6. Зафіксувати результати тестування у звіті та пояснити отриманий результат.
7. Підготуватися до захисту: уміти пояснити ключові фрагменти коду та змінити параметр або умову за вказівкою викладача.

Індивідуальні завдання

Виконати завдання відповідно до номера варіанта. Якщо для завдання потрібен набір подій, журнал або каталог файлів, студент використовує синтетичні дані, створені самостійно, або матеріали, надані викладачем.

Варіант	Завдання
1	Перевірити імена користувачів: 4–20 символів, латинські літери, цифри, крапка, дефіс і підкреслення; перший символ — літера.
2	Вилучити з тексту всі SHA-256-подібні значення як 64 шістнадцяткові символи.
3	Перевірити MAC-адреси у форматі XX:XX:XX:XX:XX:XX без перевірки реального мережевого інтерфейсу.
4	Вилучити з журналу пари key=value, де key складається з латинських літер і підкреслення.
5	Перевірити умовний ідентифікатор події формату EVT-YYYY-NNNN.
6	Вилучити з тексту всі дати у форматі YYYY-MM-DD та окремо роки, місяці й дні.
7	Перевірити часові мітки формату HH:MM:SS, дозволяючи лише 00–23, 00–59, 00–59.
8	Вилучити коди HTTP із рядків виду "status=404" або "status=500".
9	Перевірити доменні імена спрощеного формату: мітки з літер, цифр і дефіса, розділені крапками.
10	Вилучити з журналу значення user="..." у подвійних лапках.
11	Перевірити ідентифікатор токена формату tok_ і 32 латинські літери або цифри.
12	Знайти у тексті всі послідовності з 8 і більше шістнадцяткових символів.

Варіант	Завдання
13	Перевірити рядок версії формату vMAJOR.MINOR.PATCH, де всі компоненти числові.
14	Вилучити з тексту адреси електронної пошти за спрощеним навчальним шаблоном та пояснити обмеження такого шаблону.
15	Перевірити імена файлів журналів формату app-YYYYMMDD.log.
16	Вилучити з рядка всі поля у квадратних дужках, наприклад [ERROR], [auth], [user42].
17	Перевірити номер правила формату RULE-AAA-9999.
18	Замінити у тексті всі послідовності пробілів і табуляцій одним пробілом.
19	Вилучити з журналу рядки, що починаються з одного з рівнів INFO, WARN, ERROR або CRITICAL.
20	Перевірити умовний UUID-подібний ідентифікатор 8-4-4-4-12 шістнадцяткових символів.
21	Вилучити пари src=<значення> та dst=<значення> з рядків журналу без перевірки коректності самих адрес.
22	Перевірити назви змінних Python за спрощеним правилом: літера або _, далі літери, цифри або _.
23	Вилучити з тексту числові коди в дужках, наприклад (1001), (2003).
24	Перевірити умовний номер інциденту INC-YYYYMM-NNN.
25	Знайти всі рядки, що містять слово failed незалежно від регістру, але не як частину довшого слова.
26	Вилучити з тексту URL-схеми http:// та https:// і порахувати їх частоту без повного парсингу URL.
27	Перевірити рядки, що містять рівно 32 шістнадцяткові символи, як MD5-подібний формат без оцінювання криптографічної безпеки.
28	Вилучити з журналу PID із фрагментів pid=1234.
29	Перевірити назву хоста за спрощеним правилом: 1–63 символи, літери, цифри, дефіс; без дефіса на початку або в кінці.
30	Нормалізувати рядки журналу: прибрати зайві пробіли навколо символу = за допомогою re.sub().

Контрольні питання

1. Що таке нормалізація тексту?
2. Коли достатньо методів str без regex?
3. Чим fullmatch відрізняється від search?
4. Для чого використовують сирі рядки r"..."?
5. Що означає квантифікатор {m,n}?
6. Чому regex не гарантує змістову коректність IP-адреси?
7. Які ризики мають надмірно складні регулярні вирази?
8. Чому слід тестувати некоректні та довгі рядки?

ЛАБОРАТОРНА РОБОТА №8

Тема: Байтові послідовності, кодування та представлення даних

Мета: Сформувані розуміння різниці між текстом і байтами та навчитися коректно виконувати кодування, декодування, перетворення у hex і Base64.

Очікувані результати виконання

- розрізняти str, bytes і bytearray;
- кодувати текст у UTF-8 і декодувати назад;
- перетворювати bytes у hex та відновлювати дані;
- використовувати Base64 як спосіб представлення, а не шифрування;
- контролювати обробляти помилки декодування.

Теоретичні відомості

8.1. Текст і байти

str у Python є послідовністю символів Unicode, а bytes — послідовністю чисел 0–255. Файли, мережеві протоколи та криптографічні функції часто працюють саме з байтами. Перехід між str і bytes завжди має відбуватися з явним кодуванням.

8.2. UTF-8

UTF-8 є поширеним кодуванням Unicode зі змінною довжиною символів. Кількість символів у рядку та кількість байтів після encode("utf-8") можуть відрізнятися, особливо для кирилиці та інших не-ASCII символів. Помилка UnicodeDecodeError означає, що послідовність байтів не може бути декодована згідно з обраними правилами.

8.3. bytes і bytearray

bytes є незмінним типом, а bytearray — змінним. Обидва типи підтримують індексацію, де окремий елемент є цілим числом 0–255. bytearray зручний для навчального редагування байтових буферів, але при роботі з даними потрібно контролювати індекси та не змінювати вихідні дані без явної потреби.

8.4. Hex

Hex-представлення відображає кожен байт двома шістнадцятковими цифрами. Метод bytes.hex() створює текстове представлення, а bytes.fromhex() виконує зворотнє перетворення. Hex не забезпечує конфіденційність і не є шифруванням.

8.5. Base64

Base64 перетворює двійкові дані у текстовий алфавіт, зручний для передавання через текстові канали. Результат зазвичай довший за вихідні дані. Base64 легко декодується і не може використовуватися для приховування секретів.

Приклад 1. UTF-8 та hex

```
text = "Кібербезпека"
data = text.encode("utf-8")

print(len(text), "символів")
print(len(data), "байтів")
print(data.hex())
print(data.decode("utf-8"))
```

Приклад 2. Base64

```
import base64

data = b"example-data"
encoded = base64.b64encode(data)
```

```

decoded = base64.b64decode(encoded, validate=True)

print(encoded.decode("ascii"))
print(decoded)

```

Приклад аналізу

Для тексту "ABC" довжина у символах і байтах UTF-8 однакова. Для слова "дані" кількість байтів більша за кількість символів. У звіті потрібно показати обидва значення та пояснити причину.

Порядок виконання роботи

1. Ознайомитися з теоретичними відомостями та контрольними питаннями.
2. Створити окремий каталог лабораторної роботи та підготувати вихідні або тестові дані відповідно до свого варіанта.
3. Спроекувати алгоритм і визначити, які конструкції Python необхідні для його реалізації.
4. Реалізувати програму з перевіркою вхідних даних і зрозумілими повідомленнями про помилки.
5. Перевірити програму щонайменше на трьох коректних, одному граничному та одному некоректному наборі даних.
6. Зафіксувати результати тестування у звіті та пояснити отриманий результат.
7. Підготуватися до захисту: уміти пояснити ключові фрагменти коду та змінити параметр або умову за вказівкою викладача.

Індивідуальні завдання

Виконати завдання відповідно до номера варіанта. Якщо для завдання потрібен набір подій, журнал або каталог файлів, студент використовує синтетичні дані, створені самостійно, або матеріали, надані викладачем.

Варіант	Завдання
1	Перетворити заданий текст у UTF-8 bytes, вивести кількість байтів, hex-представлення та відновити початковий текст.
2	Закодувати заданий текст у Base64 і декодувати назад; порівняти довжину вихідних і закодованих даних.
3	Прийняти hex-рядок, перетворити його у bytes та спробувати декодувати як UTF-8 з обробленням помилки.
4	Для списку цілих чисел 0–255 створити bytes, вивести hex та суму значень байтів.
5	Прочитати невеликий локальний файл у двійковому режимі та вивести перші 16 байтів у hex.
6	Порівняти UTF-8 подання латинського та українського тексту: кількість символів і байтів.
7	Створити bytearray із заданого bytes-об'єкта та змінити один байт за індексом із перевіркою меж.
8	Закодувати текст спочатку в UTF-8, потім у Base64; виконати зворотні операції у правильному порядку.
9	Прийняти два hex-рядки однакової довжини і виконати побайтову операцію XOR; результат вивести лише як навчальне перетворення, не як шифрування.

Варіант	Завдання
10	Визначити частоту значень байтів у невеликому двійковому файлі та вивести п'ять найчастіших.
11	Перевірити, чи починається файл із заданої послідовності magic bytes, наданої у варіанті викладачем.
12	Перетворити bytes у список цілих значень і відновити bytes із цього списку.
13	Прийняти рядок Base64, перевірити можливість декодування і вивести hex отриманих байтів.
14	Для тексту обчислити просту суму байтів UTF-8 за модулем 256 і пояснити, чому це не криптографічний хеш.
15	Прочитати файл у блоках по 64 байти та порахувати кількість блоків без завантаження всього файла в пам'ять.
16	Знайти у bytes-послідовності всі позиції заданого байта.
17	Створити bytes із ASCII-тексту та перевірити, чи всі байти лежать у діапазоні 32–126.
18	Порівняти два bytes-об'єкти та вивести індекси позицій, де вони відрізняються.
19	Перетворити ціле число 0–65535 у два байти у big-endian і little-endian порядку та відновити значення.
20	Прочитати перші 32 байти файла та вивести одночасно hex і друковані ASCII-символи, замінюючи інші крапкою.
21	Закодувати список коротких рядків у Base64 та зберегти результат у текстовий файл по одному значенню на рядок.
22	Прочитати файл із Base64-рядками та декодувати лише коректні, підрахувавши помилки.
23	Для bytearray виконати заміну всіх нульових байтів на 0x20 та порівняти до/після.
24	Створити функцію hexdump(), яка виводить дані блоками по 16 байтів без ASCII-колонки.
25	Прийняти hex-рядок із пробілами, нормалізувати його та перетворити у bytes.
26	Порахувати кількість нульових, друкованих ASCII та інших байтів у невеликому тестовому файлі.
27	З'єднати три bytes-об'єкти, вивести їх загальну довжину і перевірити наявність заданої підпослідовності.
28	Розділити bytes-послідовність на блоки фіксованої довжини та вивести hex кожного блока.
29	Перевірити коректність парної кількості символів у hex-рядку до виклику bytes.fromhex().
30	Порівняти Base64- та hex-представлення одного набору байтів за довжиною рядка.

Контрольні питання

1. Чим str відрізняється від bytes?
2. Які значення може містити один елемент bytes?

3. Для чого потрібні `encode()` і `decode()`?
4. Чому довжина тексту та UTF-8-послідовності може відрізнятися?
5. Чим `bytes` відрізняється від `bytearray`?
6. Чи є `hex` шифруванням?
7. Чи є `Base64` шифруванням?
8. Коли виникає `UnicodeDecodeError`?

ЛАБОРАТОРНА РОБОТА №9

Тема: Хешування, HMAC, безпечні випадкові значення та контроль цілісності

Мета: Навчитися застосовувати стандартні засоби Python для обчислення криптографічних хешів, перевірки цілісності, HMAC та генерації безпечних випадкових значень у навчальних локальних задачах.

Очікувані результати виконання

- обчислювати SHA-256 і SHA-512 для рядків та файлів;
- читати великі файли блоками;
- створювати та перевіряти manifest контрольних сум;
- обчислювати HMAC і порівнювати значення через `compare_digest()`;
- використовувати `secrets` для токенів та розрізняти `secrets` і `random`.

Теоретичні відомості

9.1. Криптографічна хеш-функція

Хеш-функція перетворює дані довільної довжини у значення фіксованої довжини. Навіть невелика зміна вхідних даних зазвичай істотно змінює результат. Хеш не є шифруванням і не передбачає відновлення вихідних даних. У роботі використовуються SHA-256 та SHA-512 із `hashlib`.

9.2. Хешування файлів

Великі файли не слід повністю завантажувати у пам'ять. Їх читають блоками, послідовно передаючи дані у `update()`. Manifest — це таблиця шляхів і очікуваних хешів. Під час перевірки потрібно розрізняти щонайменше стани OK, CHANGED та MISSING.

9.3. HMAC

HMAC поєднує криптографічну хеш-функцію із секретним ключем і забезпечує перевірку цілісності та автентичності повідомлення між сторонами, які знають спільний ключ. Для порівняння HMAC слід використовувати `hmac.compare_digest()`, а не звичайний `==`, коли це стосується секретних перевірочних значень.

9.4. Безпечні випадкові значення

Модуль `random` призначений для моделювання та загальних випадкових задач і не є джерелом криптографічно стійкої випадковості. Для токенів, одноразових значень і секретів слід використовувати `secrets.token_bytes()`, `token_hex()` або `token_urlsafe()`.

9.5. Паролі та похідні ключі

Паролі не слід зберігати як звичайні хеші без додаткових заходів. Для навчального ознайомлення Python має `hashlib.pbkdf2_hmac()`, що використовує сіль і багато ітерацій. У реальних системах політику зберігання паролів визначають актуальні стандарти та спеціалізовані бібліотеки. У цій лабораторній не потрібно створювати власну схему автентифікації.

Приклад 1. SHA-256 файла блоками

```
from hashlib import sha256
from pathlib import Path

def file_sha256(path: Path) -> str:
    digest = sha256()
    with path.open("rb") as f:
        for block in iter(lambda: f.read(64 * 1024), b''):
            digest.update(block)
    return digest.hexdigest()
```

Приклад 2. HMAC

```
import hmac
from hashlib import sha256

key = b"training-key"
message = b"important-data"
tag = hmac.new(key, message, sha256).hexdigest()

received = tag
print(hmac.compare_digest(tag, received))
```

Приклад 3. Генерація токена

```
import secrets

token = secrets.token_urlsafe(24)
print(token)
```

Зразок manifest

```
sha256 path
1f2d... data/report.csv
9a03... data/events.json
```

Під час перевірки результат повинен містити стан кожного файла та підсумкову кількість OK/CHANGED/MISSING.

Порядок виконання роботи

1. Ознайомитися з теоретичними відомостями та контрольними питаннями.
2. Створити окремий каталог лабораторної роботи та підготувати вихідні або тестові дані відповідно до свого варіанта.
3. Спроекувати алгоритм і визначити, які конструкції Python необхідні для його реалізації.
4. Реалізувати програму з перевіркою вхідних даних і зрозумілими повідомленнями про помилки.
5. Перевірити програму щонайменше на трьох коректних, одному граничному та одному некоректному наборі даних.
6. Зафіксувати результати тестування у звіті та пояснити отриманий результат.
7. Підготуватися до захисту: уміти пояснити ключові фрагменти коду та змінити параметр або умову за вказівкою викладача.

Індивідуальні завдання

Виконати завдання відповідно до номера варіанта. Якщо для завдання потрібен набір подій, журнал або каталог файлів, студент використовує синтетичні дані, створені самостійно, або матеріали, надані викладачем.

Варіант	Завдання
1	Обчислити SHA-256 для заданого рядка і перевірити, як змінюється результат після зміни одного символу.
2	Обчислити SHA-512 для локального файла блоками по 64 KiB.
3	Створити manifest із SHA-256 для всіх файлів верхнього рівня тестового каталогу.

Варіант	Завдання
4	Перевірити файли за раніше збереженим manifest і вивести OK/CHANGED/MISSING.
5	Обчислити HMAC-SHA256 для повідомлення із заданим секретним ключем та перевірити його через compare_digest().
6	Показати, що зміна одного байта повідомлення робить HMAC-перевірку невдалою.
7	Згенерувати 20 випадкових токенів secrets.token_hex(16) та перевірити їх унікальність.
8	Згенерувати URL-безпечний токен через secrets.token_urlsafe() і вивести його довжину.
9	Створити функцію для порівняння SHA-256 двох файлів та перевірити її на однакових і різних файлах.
10	Для набору рядків обчислити SHA-256 та зберегти відповідності у JSON.
11	Створити перевірку цілісності одного конфігураційного файла: зберегти еталонний SHA-256 і порівняти після зміни файла.
12	Обчислити HMAC для кожного рядка текстового файла та сформувати новий файл "рядок;mac".
13	Перевірити файл із парами "повідомлення;mac" за заданим ключем та порахувати коректні/некоректні записи.
14	Порівняти час хешування того самого файла SHA-256 і SHA-512 через time.perf_counter(), не роблячи висновків про криптостійкість лише за швидкістю.
15	Згенерувати випадковий 6-значний числовий код через secrets.randbelow() без використання random.
16	Згенерувати набір випадкових байтів secrets.token_bytes(32) і вивести Base64-представлення.
17	Реалізувати PBKDF2-HMAC-SHA256 для навчального збереження пароля з випадковою сіллю; зберегти лише сіль та похідне значення.
18	Реалізувати перевірку введенного пароля проти PBKDF2-представлення з використанням compare_digest().
19	Для 10 файлів побудувати словник filename→SHA-256 та знайти файли з однаковим хешем у тестовому наборі.
20	Створити функцію hash_stream(file_obj), що обчислює SHA-256 без знання розміру файла наперед.
21	Створити JSON-manifest з полями path,size,sha256 для тестового каталогу.
22	Перевірити, чи відповідає введений SHA-256 формату 64 hex-символів, а потім порівняти його з фактичним хешем файла.
23	Згенерувати випадковий секретний ключ для HMAC, зберегти його в окремому локальному тестовому файлі та використати для підпису повідомлення.

Варіант	Завдання
24	Обчислити SHA-256 для одного й того самого тексту у двох різних кодуваннях (UTF-8 та UTF-16) і пояснити різницю.
25	Створити функцію <code>verify_manifest()</code> , яка повертає списки <code>changed</code> , <code>missing</code> і <code>ok</code> .
26	Згенерувати 100 випадкових токенів заданої довжини та перевірити відсутність дублікатів у тесті.
27	Обчислити HMAC-SHA512 для JSON-рядка після стабільної серіалізації <code>json.dumps(..., sort_keys=True)</code> .
28	Показати, що різний порядок ключів JSON може змінити HMAC без стабільної серіалізації, і виправити це через <code>sort_keys=True</code> .
29	Створити функцію <code>secure_filename_tag(name)</code> , що повертає перші 12 hex-символів SHA-256 від імені як неавтентифікаційний короткий ідентифікатор.
30	Побудувати простий локальний журнал контролю цілісності: дата перевірки, ім'я файлу, SHA-256, результат порівняння.

Контрольні питання

1. Які властивості має криптографічна хеш-функція?
2. Чому хеш не є шифруванням?
3. Навіщо читати великі файли блоками?
4. Що таке `manifest` контрольних сум?
5. Чим HMAC відрізняється від звичайного хешу?
6. Для чого застосовують `compare_digest()`?
7. Чому `random` не підходить для секретних токенів?
8. Чому паролі не слід зберігати як простий SHA-256?

ЛАБОРАТОРНА РОБОТА №10

Тема: Мережеві дані в Python: IP-адреси та безпечна локальна взаємодія через socket

Мета: Навчитися коректно працювати з IP-адресами та мережами через `ipaddress` і реалізувати простий клієнт-серверний обмін виключно в локальному навчальному середовищі.

Очікувані результати виконання

- перевіряти IPv4/IPv6-адреси через `ipaddress`;
- отримувати параметри мережі та перевіряти належність адреси;
- визначати перекриття мереж;
- створювати простий TCP-сервер і клієнт на `localhost`;
- задавати обмеження розміру, тайм-аут і простий текстовий протокол.

Теоретичні відомості

10.1. Модуль `ipaddress`

`ipaddress` виконує синтаксичний і числовий аналіз IP-адрес без мережевого доступу. `ip_address()` створює об'єкт адреси, `ip_network()` — об'єкт мережі. Властивості `is_private`, `is_global`, `version` та інші дають змогу класифікувати адресу. Для мереж можна отримати `network_address`, `broadcast_address`, `netmask`, `prefixlen` та перевірити `membership` через оператор `in`.

10.2. Мережі та підмережі

CIDR-нотація поєднує адресу мережі з довжиною префікса. Методи `subnets()`, `supernet()`, `overlaps()` і `address_exclude()` допомагають виконувати локальні обчислення над адресними просторами. Ці операції не встановлюють з'єднань і не виконують сканування.

10.3. Сокети

Сокет є програмним кінцем мережевої взаємодії. У TCP сервер створює `socket`, прив'язує його до адреси через `bind()`, переходить у режим `listen()` і приймає з'єднання через `accept()`. Клієнт викликає `connect()`. Для лабораторної всі приклади обмежуються `127.0.0.1` або `::1`.

10.4. Протокол прикладного рівня

Навіть простий навчальний сервіс потребує визначеного формату повідомлень: кодування UTF-8, максимального розміру, правила завершення, набору допустимих команд і формату відповіді. Сервер не повинен виконувати введений текст як команду операційної системи або Python-код.

10.5. Тайм-аути та обмеження

Для мережевих операцій доцільно задавати `timeout`, обмежувати кількість прийнятих байтів і коректно закривати сокети. Мета роботи — зрозуміти обмін даними між двома локальними програмами, а не перевіряти сторонні вузли чи порти.

Приклад 1. Аналіз IP-адреси

```
from ipaddress import ip_address, ip_network

ip = ip_address("192.168.10.25")
network = ip_network("192.168.10.0/24")

print("private:", ip.is_private)
print("in network:", ip in network)
print("network:", network.network_address)
print("broadcast:", network.broadcast_address)
```

Приклад 2. Мінімальний локальний TCP-сервер

```
import socket

with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as server:
    server.settimeout(20)
    server.bind(("127.0.0.1", 50000))
    server.listen(1)
    conn, _ = server.accept()
    with conn:
        data = conn.recv(1024)
        text = data.decode("utf-8", errors="strict")
        conn.sendall(f"LEN={len(text)}.encode("utf-8"))
```

Приклад 3. Локальний клієнт

```
import socket

with socket.create_connection(("127.0.0.1", 50000), timeout=5) as client:
    client.sendall("training".encode("utf-8"))
    print(client.recv(1024).decode("utf-8"))
```

Правила безпечного виконання

Дозволено: 127.0.0.1, ::1, локальні тестові файли, адреси як дані для `ipaddress`.
Не виконується: сканування мереж, перевірка сторонніх сервісів, автоматичний перебір портів, надсилання пакетів на зовнішні адреси.

Порядок виконання роботи

1. Ознайомитися з теоретичними відомостями та контрольними питаннями.
2. Створити окремий каталог лабораторної роботи та підготувати вихідні або тестові дані відповідно до свого варіанта.
3. Спроекувати алгоритм і визначити, які конструкції Python необхідні для його реалізації.
4. Реалізувати програму з перевіркою вхідних даних і зрозумілими повідомленнями про помилки.
5. Перевірити програму щонайменше на трьох коректних, одному граничному та одному некоректному наборі даних.
6. Зафіксувати результати тестування у звіті та пояснити отриманий результат.
7. Підготуватися до захисту: уміти пояснити ключові фрагменти коду та змінити параметр або умову за вказівкою викладача.

Індивідуальні завдання

Виконати завдання відповідно до номера варіанта. Якщо для завдання потрібен набір подій, журнал або каталог файлів, студент використовує синтетичні дані, створені самостійно, або матеріали, надані викладачем.

Варіант	Завдання
1	Перевірити список IPv4-адрес і розділити їх на private та non-private через <code>ipaddress</code> .
2	Для заданої IPv4-мережі визначити network address, broadcast address, netmask і кількість адрес.
3	Перевірити, які із заданих IP-адрес належать до заданої мережі.

Варіант	Завдання
4	Для двох мереж визначити, чи перекриваються вони через overlaps().
5	Нормалізувати список мереж у CIDR-нотації та відсортувати їх за адресою мережі.
6	Визначити для кожної адреси властивості is_private, is_loopback, is_multicast та сформувану таблицю.
7	Для заданої мережі вивести перші та останні п'ять адрес без повного переліку великої мережі.
8	Перевірити коректність списку IP-адрес, окремо зберігши некоректні рядки.
9	Для списку адрес визначити найменшу задану викладачем мережу зі списку кандидатів, до якої належить кожна адреса.
10	За двома списками CIDR-мереж визначити точні збіги та мережі, що присутні лише в одному списку.
11	Реалізувати TCP echo-сервер на localhost та клієнт, який надсилає один текстовий рядок.
12	Реалізувати локальний сервер, що повертає клієнту довжину отриманого повідомлення.
13	Реалізувати локальний сервер, що приймає JSON із полями user і action та повертає status="received".
14	Реалізувати локальний сервер, що приймає рядок і повертає його SHA-256; використовувати лише localhost.
15	Реалізувати локальний клієнт-сервер для перевірки HMAC повідомлення: клієнт надсилає message і mac, сервер повертає true/false.
16	Реалізувати локальний сервер, що приймає ціле число 0–65535 і повертає класифікацію діапазону порту без перевірки реального стану порту.
17	Реалізувати локальний сервер, який приймає список IP-рядків у JSON і повертає лише коректні адреси.
18	Реалізувати локальний клієнт, що надсилає три послідовні повідомлення одному серверу в межах одного з'єднання.
19	Реалізувати локальний сервер із тайм-аутом, який коректно завершує очікування клієнта після заданого часу.
20	Реалізувати локальний сервер, що повертає поточний час і довжину повідомлення клієнта.
21	Реалізувати локальний сервер, що приймає Base64-рядок, декодує його і повертає довжину байтів або повідомлення про помилку.
22	Реалізувати локальний сервер, що приймає ім'я користувача і перевіряє його за простим шаблоном; повертає valid/invalid.
23	Реалізувати локальний сервер, що приймає числову оцінку ризику 0–100 і повертає low/medium/high.

Варіант	Завдання
24	Реалізувати локальний сервер, що приймає JSON-масив чисел та повертає min, max і average.
25	Реалізувати локальний сервер, який відхиляє повідомлення довші за задану кількість байтів і повертає контрольовану помилку.
26	Реалізувати клієнт із обробленням ConnectionRefusedError та TimeoutError для localhost.
27	Реалізувати простий текстовий протокол localhost: клієнт надсилає "PING", сервер повертає "PONG"; інші команди — "ERROR".
28	Реалізувати localhost-сервер, що приймає hex-рядок, перетворює його у bytes і повертає Base64 або повідомлення про помилку.
29	Реалізувати localhost-сервер, що приймає дату YYYY-MM-DD і повертає valid/invalid після перевірки стандартним модулем datetime.
30	Реалізувати локальний сервер, який журналює у файл час, адресу клієнта (loopback), довжину повідомлення та результат оброблення.

Контрольні питання

1. Чим ip_address() відрізняється від ip_network()?
2. Що означає CIDR-префікс?
3. Як перевірити належність адреси до мережі?
4. Що показує overlaps()?
5. Яка послідовність дій TCP-сервера?
6. Для чого потрібен timeout?
7. Чому потрібно обмежувати розмір повідомлення?
8. Чому лабораторний socket-сервіс обмежується localhost?

ЛАБОРАТОРНА РОБОТА №11

Тема: Аналіз журналів безпеки та виявлення аномальних подій

Мета: Навчитися будувати відтворюваний конвеєр аналізу синтетичних журналів: перевіряти записи, нормалізувати дані, агрегувати події та застосовувати прозорі правила для виявлення аномалій.

Очікувані результати виконання

- парсити часові мітки через `datetime`;
- групувати події за користувачем, адресою, типом або часовим інтервалом;
- будувати прості часові вікна;
- реалізовувати детерміновані правила виявлення аномалій;
- формувати пояснюваний звіт з причиною спрацювання правила.

Теоретичні відомості

11.1. Конвеєр аналізу журналів

Типовий навчальний конвеєр складається з п'яти етапів: завантаження, валідація, нормалізація, агрегація, застосування правил і звітування. Розділення етапів дає змогу окремо перевіряти якість даних і логіку правила.

11.2. Часові мітки

Для ISO-подібних часових міток можна використовувати `datetime.fromisoformat()`. Усі події в одному наборі мають бути приведені до узгодженого формату та часової зони. Порівнювати час як довільні рядки небезпечно, якщо формати різняться.

11.3. Агрегація

Події групують за ключем: користувачем, IP, джерелом, типом або комбінацією полів. Для кожної групи обчислюють кількість, частку, часовий діапазон, набір унікальних значень або інші показники. Структури `dict`, `set` і `collections.defaultdict` спрощують агрегацію.

11.4. Часові вікна

Багато правил залежать не від загальної кількості подій, а від їх концентрації за певний час. У простому випадку події сортують за часом і для кожної позиції підтримують початок вікна. Лабораторні правила мають бути детермінованими, а розмір вікна і поріг — явно зазначеними у звіті.

11.5. Аномалія не дорівнює інциденту

Спрацювання правила означає, що запис відповідає формальному критерію. Воно може бути хибнопозитивним. У звіті слід вказувати, яке правило спрацювало, які події стали підставою, який поріг застосовано та яких даних бракує для остаточної інтерпретації.

Приклад 1. Парсинг часу

```
from datetime import datetime

raw = "2026-09-01T10:15:30"
ts = datetime.fromisoformat(raw)
print(ts.hour, ts.minute)
```

Приклад 2. Підрахунок невдалих входів

```
from collections import defaultdict

events = [
    {"user": "anna", "status": "failed"},
    {"user": "anna", "status": "success"},
    {"user": "ivan", "status": "failed"},
]
```

```

]

failed = defaultdict(int)
for event in events:
    if event["status"] == "failed":
        failed[event["user"]] += 1

print(dict(failed))

```

Зразок структури звіту про спрацювання

rule_id: LOGIN_FAIL_THRESHOLD
subject: anna
period: 2026-09-01T10:00:00 .. 2026-09-01T10:05:00
observed: 7
threshold: >5
reason: кількість невдалих входів перевищила поріг

Такий запис пояснює, чому правило спрацювало, але не робить висновку про причину подій.

Порядок виконання роботи

1. Ознайомитися з теоретичними відомостями та контрольними питаннями.
2. Створити окремий каталог лабораторної роботи та підготувати вихідні або тестові дані відповідно до свого варіанта.
3. Спроекувати алгоритм і визначити, які конструкції Python необхідні для його реалізації.
4. Реалізувати програму з перевіркою вхідних даних і зрозумілими повідомленнями про помилки.
5. Перевірити програму щонайменше на трьох коректних, одному граничному та одному некоректному наборі даних.
6. Зафіксувати результати тестування у звіті та пояснити отриманий результат.
7. Підготуватися до захисту: уміти пояснити ключові фрагменти коду та змінити параметр або умову за вказівкою викладача.

Індивідуальні завдання

Виконати завдання відповідно до номера варіанта. Якщо для завдання потрібен набір подій, журнал або каталог файлів, студент використовує синтетичні дані, створені самостійно, або матеріали, надані викладачем.

Варіант	Завдання
1	Виявити користувачів із понад 5 невдалими входами у межах одного заданого часового інтервалу.
2	Виявити IP-адреси з невдалими входами до трьох і більше різних користувачів.
3	Виявити користувачів, у яких після серії з 3+ невдалих входів відбувся успішний вхід.
4	Виявити події входу поза дозволеним часовим діапазоном, заданим викладачем.
5	Виявити адреси, для яких кількість подій більш ніж удвічі перевищує середнє по адресах.

Варіант	Завдання
6	Виявити користувачів із входами з більш ніж двох різних IP у межах одного тестового дня.
7	Виявити вузли з понад 10 подіями рівня ERROR або CRITICAL.
8	Виявити файли, для яких зафіксовано більше однієї невідповідності контрольного хешу.
9	Виявити активи критичності 4–5, для яких у журналі є хоча б одна подія high або critical.
10	Виявити повторні події з однаковими полями user,action,timestamp як потенційні дублікати журналу.
11	Виявити події з некоректним або відсутнім полем severity і винести їх в окремий звіт як проблеми якості даних.
12	Виявити користувачів із понад 20 подіями за короткий фіксований інтервал часу.
13	Виявити IP-адреси, що одночасно мають успішні й невдалі входи для багатьох різних користувачів.
14	Виявити вузли з різким збільшенням кількості подій: поточний інтервал $> 3 \times$ попереднього ненульового.
15	Виявити процеси, для яких однакове ім'я має незвичну кількість різних батьківських процесів у синтетичних даних.
16	Виявити користувачів, у яких кількість невдалих входів перевищує 50% усіх їхніх спроб.
17	Виявити адреси з подіями на більш ніж 5 різних портах у синтетичному журналі; не виконувати реальне сканування.
18	Виявити записи, де тривалість сеансу перевищує заданий поріг або дорівнює нулю.
19	Виявити користувачів, що виконували дії з двох взаємовиключних тестових ролей, заданих у довіднику.
20	Виявити активи без owner, для яких зафіксовано події high або critical.
21	Виявити події із часовою міткою, меншою за попередню подію в тому самому джерелі, як потенційну проблему синхронізації часу.
22	Виявити джерела, що генерують понад 40% усіх подій журналу.
23	Виявити спроби доступу до ресурсів, яких немає у наданому дозволеному переліку.
24	Виявити облікові записи, які позначені disabled, але мають події успішного входу у синтетичному журналі.
25	Виявити вузли з версією ПЗ нижче мінімально дозволеної та одночасно з подіями high/critical.
26	Виявити правила firewall із hits=0 протягом усього наданого тестового періоду.
27	Виявити події, що мають однаковий ідентифікатор, але різний зміст, як проблему цілісності журналювання.

Варіант	Завдання
28	Виявити користувачів із послідовністю login-success → privilege-change у межах заданого короткого інтервалу; лише позначити для аналізу без висновку про атаку.
29	Виявити URL-шляхи з найбільшою часткою кодів 4xx/5xx у тестовому веб-журналі.
30	Побудувати комбіноване правило: severity high/critical І критичність активу 4–5 І подія поза робочими годинами; сформулювати пояснення кожного спрацювання.

Контрольні питання

1. Які етапи містить конвеєр аналізу журналів?
2. Навіщо нормалізувати часові мітки?
3. Чому події потрібно сортувати перед аналізом часових вікон?
4. Для чого використовується defaultdict?
5. Що таке порогове правило?
6. Що таке хибнопозитивне спрацювання?
7. Які дані потрібно включити до пояснення спрацювання?
8. Чому аномалію не можна автоматично порівнювати до інциденту?

ЛАБОРАТОРНА РОБОТА №12

Тема: Підсумковий мініпроект: безпечна утиліта командного рядка на Python

Мета: Інтегрувати в одному завершеному мініпроекті роботу з аргументами командного рядка, файлами, журналюванням, модульною структурою, перевіркою даних і тестуванням.

Очікувані результати виконання

- створювати CLI через argparse;
- організовувати код у кілька модулів;
- використовувати logging без витоку секретних даних;
- перевіряти аргументи, шляхи й структуру вхідних даних;
- створювати базові автоматизовані тести та README з прикладами запуску.

Теоретичні відомості

12.1. Архітектура невеликої утиліти

Підсумкова робота має бути не одним великим файлом, а невеликим проектом. Мінімально варто відокремити точку входу, прикладну логіку, роботу з файлами та тести. Така структура полегшує перевірку окремих функцій і повторне використання коду.

12.2. argparse

Модуль argparse створює інтерфейс командного рядка, довідку та базову перевірку аргументів. Підкоманди зручні, якщо утиліта має кілька режимів, наприклад create-manifest і verify-manifest. Аргументи слід перевіряти до виконання основної операції.

12.3. logging

Журналювання допомагає фіксувати перебіг роботи утиліти. Рівні DEBUG, INFO, WARNING, ERROR і CRITICAL мають різне призначення. Паролі, секретні ключі, токени та інші секрети не повинні потрапляти у журнал. Для навчальної утиліти достатньо консольного або локального файлового логування.

12.4. Безпечна робота з шляхами та файлами

Перед читанням або записом слід перевіряти тип і наявність шляху, передбачати конфлікти і не перезаписувати дані без явної логіки. Якщо завдання працює з каталогом, область дії має бути обмежена каталогом, вказаним користувачем для лабораторної роботи.

12.5. Тестування

Автоматизований тест перевіряє відому властивість функції на контрольованих даних. Для невеликого проекту достатньо unittest або pytest, якщо він доступний у лабораторії. Необхідно тестувати нормальні, граничні та некоректні входи. Тимчасові файли доцільно створювати у спеціальному тестовому каталозі.

12.6. Документація

README повинен містити призначення, вимоги, структуру запуску, приклади команд, формат вхідних і вихідних даних, обмеження та відомі помилки. Користувач утиліти має розуміти, що станеться до її запуску.

Приклад 1. Каркас CLI

```
import argparse

def build_parser() -> argparse.ArgumentParser:
    parser = argparse.ArgumentParser(description="Навчальна утиліта")
    sub = parser.add_subparsers(dest="command", required=True)

    check = sub.add_parser("check")
    check.add_argument("path")
    return parser
```

```
if __name__ == "__main__":
    args = build_parser().parse_args()
    print(args.command, args.path)
```

Приклад 2. Журналювання

```
import logging

logging.basicConfig(
    level=logging.INFO,
    format="%(asctime)s %(levelname)s %(message)s"
)
logging.info("Початок оброблення")
```

Приклад 3. Простий unit-тест

```
import unittest

def normalize_user(value: str) -> str:
    return value.strip().lower()

class NormalizeTests(unittest.TestCase):
    def test_spaces_and_case(self):
        self.assertEqual(normalize_user(" Admin "), "admin")

if __name__ == "__main__":
    unittest.main()
```

Рекомендована структура проєкту

```
project/
  main.py
  core.py
  io_utils.py
  validators.py
  tests/
    test_core.py
  data/
  README.md
  requirements.txt # лише якщо є сторонні залежності
```

Перевага надається стандартній бібліотеці Python, якщо завдання не потребує іншого.

Порядок виконання роботи

1. Ознайомитися з теоретичними відомостями та контрольними питаннями.
2. Створити окремий каталог лабораторної роботи та підготувати вихідні або тестові дані відповідно до свого варіанта.
3. Спроекувати алгоритм і визначити, які конструкції Python необхідні для його реалізації.
4. Реалізувати програму з перевіркою вхідних даних і зрозумілими повідомленнями про помилки.

5. Перевірити програму щонайменше на трьох коректних, одному граничному та одному некоректному наборі даних.
6. Зафіксувати результати тестування у звіті та пояснити отриманий результат.
7. Підготуватися до захисту: уміти пояснити ключові фрагменти коду та змінити параметр або умову за вказівкою викладача.

Індивідуальні завдання

Виконати завдання відповідно до номера варіанта. Якщо для завдання потрібен набір подій, журнал або каталог файлів, студент використовує синтетичні дані, створені самостійно, або матеріали, надані викладачем.

Варіант	Завдання
1	Утиліта контролю цілісності каталогу: create-manifest і verify-manifest на основі SHA-256.
2	Утиліта аналізу журналу входів: статистика за користувачами, IP та виявлення перевищення порогу невдалих входів.
3	Утиліта перевірки політики паролів за заданими правилами без збереження введених паролів у журналі.
4	Утиліта інвентаризації локального тестового каталогу: шлях, розмір, розширення, SHA-256 у CSV/JSON.
5	Утиліта нормалізації списку IOC із локального файлу: видалення дублікатів, приведення реєстру доменів, класифікація типів за простими правилами.
6	Утиліта перевірки JSON-конфігурації: обов'язкові поля, типи значень, діапазони та зрозумілий звіт про помилки.
7	Утиліта аналізу веб-журналу: частоти HTTP-кодів, топ IP, топ шляхів і частка 4xx/5xx.
8	Утиліта аналізу CSV-інвентаризації ПЗ: перелік неузгодженого, застарілого та невідомого ПЗ за локальним довідником.
9	Утиліта перевірки списку IP-адрес і CIDR: коректність, private/loopback/multicast, належність до дозволених мереж.
10	Утиліта створення і перевірки HMAC для локальних текстових повідомлень із ключем, що передається через безпечний тестовий спосіб, визначений викладачем.
11	Утиліта пошуку дублікатів файлів у тестовому каталозі за SHA-256.
12	Утиліта перевірки контрольних сум із manifest-файла формату "sha256 filename".
13	Утиліта аналізу подій за severity з формуванням JSON-звіту і окремого списку critical-подій.
14	Утиліта аналізу облікових записів із CSV: MFA, статус, кількість ролей, дата останньої активності як умовне число днів.
15	Утиліта перевірки структури лог-файла за регулярним виразом і формування списку рядків, що не відповідають формату.
16	Утиліта перетворення між text, hex і Base64 з явним попередженням, що Base64 не є шифруванням.

Варіант	Завдання
17	Утиліта аналізу локального JSON-журналу процесів: частоти process, parent і незвичні пари за заданим довідником.
18	Утиліта аналізу політики firewall з CSV: вимкнені правила, rules with hits=0, дублікати ідентифікаторів.
19	Утиліта перевірки структури локального набору резервних копій: наявність очікуваних файлів, розмір, SHA-256.
20	Утиліта формування короткого звіту про критичні активи та пов'язані high/critical події з двох локальних файлів.
21	Утиліта порівняння двох JSON-інвентаризацій активів і виявлення added/removed/changed.
22	Утиліта аналізу журналу автентифікації з правилом "3+ невдачі перед успіхом" та пояснюваним звітом.
23	Утиліта перевірки списку доменів за спрощеним шаблоном, нормалізації та формування чистого списку без дублікатів.
24	Утиліта аналізу статусів резервного копіювання: success/failed, сумарні розміри, невдалі завдання, статистика за вузлами.
25	Утиліта безпечного генератора токенів на основі secrets з вибором hex або URL-safe формату та перевіркою параметрів.
26	Утиліта локального клієнта/сервера на localhost для перевірки HMAC повідомлення; сервер не приймає з'єднання з інших інтерфейсів.
27	Утиліта аналізу якості журналу: порожні поля, дублікати event_id, некоректні timestamps та невідомі severity.
28	Утиліта сортування та фільтрації списку вразливостей із CSV за severity, asset і status без мережевого отримання даних.
29	Утиліта перевірки дозволених розширень файлів у тестовому каталозі та формування звіту про невідповідності.
30	Утиліта локального аудиту прав доступу до тестових файлів на рівні метаданих, доступних через pathlib/os, без зміни системних дозволів.

Контрольні питання

1. Для чого використовується argparse?
2. Що таке підкоманда CLI?
3. Навіщо розділяти проєкт на модулі?
4. Які рівні logging існують?
5. Які дані не можна записувати у журнал?
6. Які групи тестів повинні бути у мініпроєкті?
7. Що має містити README?
8. Чому утиліта повинна мати передбачувані повідомлення про помилки та коди завершення?

РЕКОМЕНДОВАНІ ЕЛЕКТРОННІ РЕСУРСИ

1. Python Software Foundation. Python 3 Documentation. URL: <https://docs.python.org/3/> (дата звернення: 25.09.2026).
2. Python Software Foundation. The Python Standard Library. URL: <https://docs.python.org/3/library/> (дата звернення: 25.09.2026).
3. Python Software Foundation. hashlib — Secure hashes and message digests. URL: <https://docs.python.org/3/library/hashlib.html> (дата звернення: 25.09.2026).
4. Python Software Foundation. ipaddress — IPv4/IPv6 manipulation library. URL: <https://docs.python.org/3/library/ipaddress.html> (дата звернення: 25.09.2026).
5. Python Software Foundation. socket — Low-level networking interface. URL: <https://docs.python.org/3/library/socket.html> (дата звернення: 25.09.2026).