K-RND LAB · MASTER PREP

🎓 Посібник, який має навчити: ЄВІ / ЄФВВ

Не шпаргалка і не сухий чекліст. Це навчальний маршрут: теорія людською мовою, науковий сенс, аналогії з життя, приклади коду, схеми пам'яті й пастки реальних тестів.

Головна ідея: не зазубрити відповідь, а впізнати механізм питання.

НАВІГАЦІЯ

Що тут є

перестановки, комбінації, пропуски, обмеження

reading

Блок 11B
GB English toolkit: усі базові часи, conditionals, reading markers

Як читати: один блок за раз. Після блоку одразу пройти 10-15 питань. Якщо помилка — записати не літеру, а пару “я переплутала X із Y”.

БЛОК 00

🧭 Як вчитися, щоб не плутатися

Ціль: Перетворити незнайомі слова на впізнавані ситуації. На іспиті виграє не той, хто зазубрив більше, а той, хто швидко розпізнає тип питання.

🧭 Карта пам'яті

ключові слова ↔ **поняття** ↔ **аналогія** ↔ **приклад** ↔ **пастка** ↔ **відповідь**

🔬 **Науково, але зрозуміло:**
Більшість тестових питань перевіряє категоризацію: чи можна за описом віднести ситуацію до правильного поняття. Тому треба вчити не лише визначення, а й межі поняття: чим воно НЕ є.

🏠 **Аналогія:**
Це як набір інструментів. Молоток, викрутка й ключ усі корисні, але якщо в умові “закрутити гвинт”, молоток відпадає, навіть якщо він знайомий.

🧪 Пояснення через приклади

Як читати питання

1. Знайти дієслово: обрати, визначити, НЕ є, найточніше. 2. Підкреслити ознаку. 3. Відкинути сусідні поняття. 4. Обрати найвужчу відповідь.

```
Питання: яка програма збирає
об'єктні модулі в один виконуваний
модуль?
Ознака: збирає об'єктні модулі
Відповідь: компонувальник / linker
```

Як запам'ятовувати

Для кожного терміна треба мати три гачки: коротке визначення, життєву аналогію і тестову пастку.

```
Hash = відбиток пальця файлу.
Encryption = сейф із ключем.
Signature = підпис і печатка.
Пастка: hash не розшифровують.
```

⚠️ **Типові пастки:**

- Не відповідати на перше знайоме слово.
- У питаннях з “не є” шукати зайвий варіант.
- У “найточніше” не брати ширше поняття, якщо є точніше.

🔗 **Пов'язано з:** ТЗНК · English reading · усі IT-визначення

🎯 Екзаменаційне спорядження

Професійна прив'язка: Будь-яка IT/аналітична роль: junior developer, QA, data analyst, support engineer.

🎯 Що реально питають

- Екзамен часто перевіряє не “чи знаєш слово”, а чи бачиш ключову умову.
- Перед відповіддю знайди в питанні маркер: “найточніше”, “не є”, “після”, “будь-який”, “обов'язково”.
- Якщо варіант звучить знайомо, але не відповідає маркеру, це пастка, а не відповідь.

Пастки: де ловлять

- Не обирати перше знайоме слово.
- Не додумувати умову, якої нема в тексті.
- Не рахувати складну задачу, поки не визначено тип: сума, добуток, перестановка, комбінація.

Міні-дріл: Після кожної помилки записати: 1) що було ключовим словом, 2) чому правильний варіант саме він, 3) чим заманив неправильний.

БЛОК 01

🔢 Бінарний рахунок і представлення даних

Ціль: Навчитися рахувати у двійковій системі, розуміти біти/байти і не лякатися питань про пам'ять, кодування, IP, маски, права доступу.

🧭 Карта пам'яті

0/1 ↔ **bit** ↔ **byte** ↔ **powers of two** ↔ **memory** ↔ **files** ↔ **networks**

🔬 Науково, але зрозуміло:
Комп'ютер зберігає інформацію як послідовність бітів. Біт має два стани: 0 або 1. Двійкове число працює так само, як десяткове, але замість розрядів 1, 10, 100 використовує 1, 2, 4, 8, 16, 32...

🏠 Аналогія:
Уяви ряд вимикачів. Кожен вимикач або вимкнений (0), або ввімкнений (1). Комбінація вимикачів кодує число, літеру, колір, звук, дозвіл доступу.

🧪 Пояснення через приклади

Як перевести binary → decimal

Підписуємо розряди справа наліво: 1, 2, 4, 8, 16. Додаємо тільки ті, де стоїть 1.

```
10110₂
розряди: 16 8 4 2 1
біти:     1 0 1 1 0
сума: 16 + 4 + 2 = 22₁₀
```

Як перевести decimal → binary

Беремо найбільший степінь двійки, який влізає, ставимо 1, віднімаємо, повторюємо.

```
25₁₀
25 = 16 + 8 + 1
розряди 16 8 4 2 1
біти     1  1 0 0 1
25₁₀ = 11001₂
```

Байт і пам'ять

1 byte = 8 bits. Один байт може мати 256 комбінацій, бо $2^8 = 256$.

```
00000000 = 0
11111111 = 255
Тому unsigned byte часто має діапазон 0..255
```

Бінарне в житті

Колір у RGB часто має канали 0..255. IP-адреса IPv4 складається з 4 байтів. Права файлів у Linux теж зручно думати як біти.

```
read=4, write=2, execute=1
7 = 4+2+1 = rwx
5 = 4+1 = r-x
```

⚠️ **Типові пастки:**

- **101_2 — це не сто один, а 5.**
  Маленьке $_2$ означає, що число записане у двійковій системі. Там розряди не 100/10/1, а 4/2/1.

  Приклад: $101_2 = 1*4 + 0*2 + 1*1 = 5_{10}$.

- **1 byte = 8 bits, не 10.**
  Byte — це стандартна група з 8 бітів. Десяткова звичка “10” тут не працює, бо комп'ютерна пам'ять побудована на степенях двійки.

  Приклад: 8 bits дають $2^8 = 256$ різних комбінацій.

- **$2^8 = 256$ комбінацій, але unsigned byte має значення 0..255.**
  Комбінацій 256, бо рахуємо всі варіанти від 00000000 до 11111111. Якщо починаємо з нуля, останнє значення буде 255.

  Приклад: $00000000_2 = 0$, $11111111_2 = 255$.

- **Бінарний пошук і бінарна система — різні речі.**
  Бінарна система — спосіб запису чисел через 0 і 1. Бінарний пошук — алгоритм пошуку у відсортованому списку через поділ навпіл.

  Приклад: $33_{10} = 100001_2$ — це система числення; пошук 33 у sorted array — це алгоритм.

🔗 **Пов'язано з:** операційні системи · мережі · криптографія · алгоритми

🎯 Екзаменаційне спорядження

Професійна прив'язка: Computer systems, embedded/basic hardware, networks, cybersecurity, data engineering.

🎯 Що реально питають

- Біти, байти, кодування, двійкова/десяткова системи та діапазони значень.
- 2^n означає кількість комбінацій; якщо нумерація з нуля, останнє значення буде 2^n - 1.
- Binary у “binary search” і binary як система числення - різні теми, але обидві люблять у тестах.

Пастки: де ловлять

- 101_2 читається як 5_{10}, а не як сто один.
- 1 byte = 8 bits, тому 1 KB у технічних контекстах може бути 1024 bytes, але в маркетингу іноді 1000.
- Signed/unsigned плутають: unsigned byte 0..255, signed byte часто -128..127.

Міні-дріл: Навчитися швидко розкладати числа по степенях 2: 1,2,4,8,16,32,64,128,256. Для 33: 32+1 = 100001_2.

БЛОК 02

💻 Процедурне програмування і виконання коду

Ціль: Навчитися читати код як послідовність змін стану: змінні, присвоєння, цикли, масиви, функції, рекурсія.

🧭 Карта пам'яті

state ↔ assignment ↔ condition ↔ loop ↔ function ↔ result

🔬 Науково, але зрозуміло:
Процедурна програма виконує інструкції послідовно. Пам'ять містить поточні значення змінних. Кожна команда може змінити стан, а результат залежить від порядку виконання.

🏠 Аналогія:
Кнопка гучності — чудова аналогія. Вона не має одного сталого результату. Якщо гучність 3, натискання + дає 4. Якщо вже 8, те саме натискання дає 9. Команда діє на поточний стан.

🧪 Пояснення через приклади

Присвоєння

Праворуч рахуємо зі старим значенням, потім записуємо результат у змінну.

```
let x = 2;
x = x + 3;
// старе x було 2
// нове x стало 5
```

Цикл

Цикл повторює дію. Щоб не помилитися, треба вести таблицю: і, значення змінної, умова.

```
let volume = 3;
for (let i = 0; i < 2; i++) {
  volume = volume + 1;
}
// i=0: volume 3→4
// i=1: volume 4→5
// відповідь: 5
```

Масив та індекс

У багатьох мовах перший елемент має індекс 0. Це одна з найчастіших пасток.

```
const a = [10, 20, 30];
a[0] = 10
a[1] = 20
a[2] = 30
```

Рекурсія

Рекурсія має базовий випадок і крок, який наближає до базового випадку.

```
function fact(n) {
  if (n === 1) return 1;
  return n * fact(n - 1);
}
fact(4) = 4*3*2*1 = 24
```

Компілятор, інтерпретатор, компонувальник

Компілятор перекладає код у нижчий рівень. Інтерпретатор виконує поступово. Компонувальник/linker збирає об'єктні модулі в виконуваний файл.

```
main.c → compiler → main.o
math.c → compiler → math.o
main.o + math.o → linker → app.exe
```

⚠ **Типові пастки:**

- **Компілятор ≠ компонувальник.**
  Компілятор перекладає окремий файл/код у проміжний або машинний формат. Компонувальник бере вже готові об'єктні модулі й зшиває їх в один виконуваний файл.

  `Приклад: main.c → main.o робить compiler; main.o + math.o → app.exe робить linker.`

- **a[2] при індексації з 0 — третій елемент.**
  Перший елемент має індекс 0, другий — 1, третій — 2. Людська лічба і комп'ютерний індекс зсунуті на 1.

  `Приклад: a = [4, 7, 1]; a[2] = 1.`

- **У циклі важливі старт, умова, крок і тіло циклу.**
  Помилка часто виникає не в арифметиці, а в тому, скільки разів тіло циклу реально виконалося.

Приклад: for (i=0; i<3; i++) виконається для i=0,1,2 — тобто 3 рази.

- **Рекурсія без базового випадку може бути нескінченною.** Функція має колись зупинитися. Базовий випадок — це умова, де вона вже не викликає саму себе.

  Приклад: fact(1) return 1 — база; fact(n) викликає fact(n-1) — крок.

🔗 **Пов'язано з:** алгоритми · операційні системи · ООП

🎯 Екзаменаційне спорядження

Професійна прив'язка: Software developer, QA automation, algorithmic problem solving.

🎯 Що реально питають

- Трасування коду: значення змінних після присвоєнь, циклів, умов і рекурсії.
- Компілятор, інтерпретатор, лінкер/компонувальник, лексичний аналізатор - це різні етапи.
- Масиви часто мають індексацію з нуля; a[2] - третій елемент.

Пастки: де ловлять

- У присвоєнні права частина рахується першою: x = x + 1 не рівність, а оновлення.
- У циклі помилка часто в межі: < або <=, старт з 0 чи з 1.
- Рекурсія має базовий випадок; без нього стек викликів росте до помилки.

Міні-дріл: Для кожного фрагмента коду зробити таблицю: крок, значення змінних, умова, результат.

БЛОК 03

📦 ООП: клас, об'єкт, інкапсуляція, наслідування, поліморфізм

Ціль: Не зубрити принципи ООП, а бачити їх у пристроях, кнопках, програмах і API.

🧭 Карта пам'яті

class ↔ object ↔ state ↔ method ↔ interface ↔ behavior

🔬 Науково, але зрозуміло:

ООП моделює систему як об'єкти зі станом і поведінкою. Клас описує шаблон. Об'єкт є конкретним екземпляром. Інкапсуляція ховає внутрішню реалізацію, наслідування повторно використовує структуру, поліморфізм дозволяє одному інтерфейсу мати різні реалізації.

🏠 Аналогія:

Одна кнопка Play у різних програмах: у Spotify запускає звук, у YouTube відео, у грі сцену. Кнопка однакова для користувача, але реалізація різна — це поліморфізм.

🧪 Пояснення через приклади

Клас і об'єкт

Клас — креслення, об'єкт — конкретна річ за кресленням.

```
class Button {
  constructor(label) { this.label
= label; }
}
const save = new Button('Save');
// Button = клас
// save = об'єкт
```

Інкапсуляція

Користувач тисне кнопку гучності, але не бачить аудіодрайвер, системний мікшер і внутрішнє поле level.

```
class Volume {
  #level = 5;
  up() { if (this.#level < 10)
this.#level++; }
  value() { return this.#level; }
}
```

Поліморфізм

Один виклик, різна поведінка. Це корисно, коли код хоче працювати з різними об'єктами однаково.

```
apps.forEach(app => app.play());
// MusicApp.play() → звук
// VideoApp.play() → відео
// GameApp.play() → сцена
```

API

API — контракт між програмами. Клієнт знає, що попросити і який формат відповіді чекати.

```
GET /users/42
→ { id: 42, name: 'Anna' }
```

⚠️ **Типові пастки:**

- **Клас ≠ об'єкт.**
  Клас — опис або шаблон. Об'єкт — конкретний екземпляр, який уже має власні значення полів.

  `Приклад: class Phone — шаблон; myPhone з гучністю 7 і темною темою — об'єкт.`

- **Інкапсуляція — не просто private.**
  Сенс не в слові private, а в тому, що об'єкт сам контролює, як змінюється його внутрішній стан.

  `Приклад: Volume.up() не дозволяє зробити level=999, бо метод перевіряє межу.`

- **Поліморфізм — не “багато класів”.**
  Поліморфізм виникає, коли одна команда або інтерфейс запускає різну поведінку в різних об'єктах.

  `Приклад: play() у MusicApp дає звук, у VideoApp — відео, у GameApp — сцену.`

- **UML class diagram ≠ sequence diagram.**
  Class diagram показує статичну структуру: класи, поля, зв'язки. Sequence diagram показує часовий порядок повідомлень.

  `Приклад: “User має Orders” — class diagram; “User натиснув Pay → API → Bank” — sequence.`

🔗 **Пов'язано з:** процедурне програмування · API · UML · software engineering

🎯 Екзаменаційне спорядження

Професійна прив'язка: Backend/frontend developer, QA, software architect.

🎯 Що реально питають

- Клас - шаблон; об'єкт - конкретний екземпляр.
- Інкапсуляція - контроль доступу й приховування внутрішньої реалізації за стабільним інтерфейсом.
- Наслідування - is-a; композиція/агрегація - has-a; поліморфізм - один інтерфейс, різні реалізації.

Пастки: де ловлять

- Не кожен зв'язок “має” є наслідуванням.
- Private поле саме по собі не пояснює інкапсуляцію, якщо немає ідеї контрольованого доступу.
- Поліморфізм - не просто багато методів, а можливість викликати однакову операцію для різних типів.

Міні-дріл: Питання “Cat extends Animal” → inheritance. “Car has Engine” → composition. “Button може бути SaveButton або CancelButton” → polymorphism.

БЛОК 03В

📦 ООП поглиблено: наслідування, композиція, агрегація, тестування

Ціль: Закрити прогалину: не просто назвати принципи ООП, а розуміти, як відрізняти наслідування від композиції/агрегації та як це питають у тестах.

🧭 Карта пам'яті

is-a ↔ has-a ↔ part-of ↔ interface ↔ test ↔ bug

🔬 Науково, але зрозуміло:
Наслідування моделює відношення “є різновидом” (is-a). Композиція й агрегація моделюють “має” або “складається з” (has-a/part-of). У software engineering також важливо розуміти тестування: unit, integration, system, acceptance.

🏠 Аналогія:
Електросамокат є транспортом — це наслідування. Самокат має батарею — це композиція. Університет має студентів, але студенти можуть існувати без конкретного університету — це агрегація.

🧪 Пояснення через приклади

Наслідування: is-a

Використовувати, коли дочірній клас справді є різновидом базового.

```
class Vehicle {}
class Scooter extends Vehicle {}
class Car extends Vehicle {}
// Scooter is a Vehicle
```

Композиція: сильне part-of

Частина належить цілому й зазвичай не має сенсу без нього.

```
class Phone {
  constructor() {
    this.battery = new Battery();
  }
}
// battery є частиною phone
```

Агрегація: слабше has-a

Об'єкт містить посилання на інший об'єкт, але той може існувати окремо.

```
class University {
  constructor(students) {
    this.students = students;
  }
}
// student може перейти в інший university
```

Тестування

Unit test перевіряє маленьку функцію. Integration test перевіряє взаємодію частин. System test перевіряє всю систему.

```
unit: add(2,3) === 5
integration: API + database
system: user login flow
```

⚠️ **Типові пастки:**

- **Не кожен has-a — наслідування.**
  Якщо об'єкт має частину, це не означає, що він є різновидом цієї частини.
  `Приклад: Car має Engine, але Car не є Engine.`
- **Composition ≠ Aggregation.**
  У композиції частина сильно належить цілому. В агрегації частина може існувати незалежніше.
  `Приклад: House-Room ближче до composition; Team-Player ближче до aggregation.`
- **Unit test ≠ Integration test.**
  Unit ізолює маленьку одиницю коду. Integration перевіряє, чи частини працюють разом.
  `Приклад: Перевірити формулу знижки — unit; перевірити checkout + payment API — integration.`

🔗 **Пов'язано з:** ООП · software engineering · API · UML

🎯 Екзаменаційне спорядження

Професійна прив'язка: Software engineer, backend developer, QA engineer, API designer.

🎯 Що реально питають

- Composition означає сильну залежність частини від цілого; aggregation - слабший зв'язок.
- Unit test перевіряє малу одиницю, integration test - взаємодію частин.
- API contract важливий: клієнт очікує стабільні endpoint, формат даних і коди відповіді.

Пастки: де ловлять

- Team has Players частіше aggregation, бо player може існувати без team.
- House has Rooms частіше composition, бо кімната як частина моделі не живе окремо від будинку.
- Unit test не має залежати від реальної бази даних або зовнішньої мережі.

Міні-дріл: Для кожного has-a запитай: якщо видалити ціле, частина логічно зникає? Так - composition, ні - aggregation.

БЛОК 04

📈 Алгоритми і структури даних

Ціль: Розуміти, як дані організовані і як алгоритм рухається по них: пошук, сортування, stack, queue, BFS, DFS, Big-O.

🧭 Карта пам'яті

data ↔ operation ↔ steps ↔ complexity ↔ memory

🔬 Науково, але зрозуміло:
Алгоритм — формальна послідовність кроків. Структура даних визначає, як інформація лежить у пам'яті і які операції зручні. Big-O описує, як ростуть витрати алгоритму зі збільшенням розміру входу.

🏠 Аналогія:
Шукати слово в словнику з першої сторінки — лінійний пошук. Відкрити посередині й щоразу відкидати половину — бінарний пошук.

🧪 Пояснення через приклади

Big-O

O(1) — приблизно однаково швидко. O(n) — пройти всі n елементів. O(log n) — щоразу зменшувати задачу в кілька разів. $O(n^2)$ — часто два вкладені цикли.

```
for item in items:
  print(item)
# один прохід → O(n)

for a in items:
  for b in items:
    compare(a, b)
# вкладені цикли → O(n²)
```

Бінарний пошук

Працює тільки на відсортованих даних. Якщо дані не відсортовані, середина не дає інформації, яку половину відкидати.

```
[1,3,5,7,9], шукаємо 7
middle = 5 → 7 більше → права половина
middle = 7 → знайдено
```

Stack і Queue

Stack: останній зайшов, перший вийшов. Queue: перший зайшов, перший вийшов.

```
stack: push A, push B, pop → B
queue: add A, add B, take → A
```

BFS і DFS

BFS обходить граф шарами і часто використовує queue. DFS іде вглиб і часто використовує stack або рекурсію.

```
BFS: спершу всі сусіди 1-го рівня
DFS: один шлях до кінця, потім назад
```

⚠️ **Типові пастки:**

- **Binary search потребує sorted data.**
  Середина має сенс тільки тоді, коли ліворуч точно менші значення, а праворуч більші. Інакше неможливо знати, яку половину відкидати.

  `Приклад: [1,3,5,7,9] — можна; [7,1,9,3,5] — спершу треба сортувати.`

- **BFS ≠ DFS.**
  BFS іде шарами через чергу, DFS іде вглиб через стек або рекурсію. Вони можуть відвідати ті самі вершини, але в іншому порядку.

  `Приклад: Для найкоротшого шляху в неваговому графі зазвичай беруть BFS.`

- **Stack ≠ Queue.**
  У stack дістаємо останнє додане, у queue — найстаріше. Це різна поведінка при однаковій ідеї "контейнера".

  `Приклад: Stack: тарілки; Queue: черга до каси.`

- **O(n log n) часто асоціюється з ефективними сортуваннями.**
  Алгоритм ділить задачу на частини log n разів і на кожному рівні обробляє n елементів.

  `Приклад: Merge Sort: поділ навпіл + злиття всіх елементів.`

🔗 **Пов'язано з:** бінарний рахунок · програмування · ML · ТЗНК

🎯 Екзаменаційне спорядження

Професійна прив'язка: Developer, data engineer, ML engineer, technical interviewer.

🎯 Що реально питають

- Big-O описує ріст часу/пам'яті при збільшенні n.
- Stack: LIFO; Queue: FIFO; Hash table: швидкий доступ за ключем; Tree/Graph: ієрархії та зв'язки.
- BFS і DFS питають через “найкоротший шлях у невагомому графі”, “черга”, “стек/рекурсія”.

Пастки: де ловлять

- Binary search працює тільки на відсортованих даних.
- O(1) не означає “миттєво”, а означає незалежність від n.
- Greedy не завжди оптимальний; dynamic programming потрібен, коли є повторювані підзадачі й оптимальна підструктура.

Міні-дріл: Знати асоціації: BFS=queue, DFS=stack/recursion, merge sort=divide and conquer, DP=таблиця підзадач.

БЛОК 04В

🖥️ Архітектура комп'ютера: CPU, RAM, cache, I/O

Ціль: Додати те, що часто є в програмі IT: як комп'ютер виконує інструкції, де лежать дані, чому cache швидший за RAM, що таке I/O.

🧭 Карта пам'яті

CPU ↔ registers ↔ cache ↔ RAM ↔ disk ↔ I/O

🔬 **Науково, але зрозуміло:**
CPU виконує інструкції. Регістри — найшвидша маленька пам'ять усередині CPU. Cache зберігає часто потрібні дані ближче до процесора. RAM швидша за диск, але втрачає дані без живлення. Disk/SSD зберігає дані довго.

🏠 **Аналогія:**
Робочий стіл: у руках — регістри, на столі — cache, у шафі поруч — RAM, в архіві на іншому поверсі — disk. Чим ближче, тим швидше, але менше місця.

🧪 Пояснення через приклади

Ієрархія пам'яті

Швидкість і місткість обмінюються: ближча пам'ять швидша, але менша й дорожча.

```
registers: дуже швидко, дуже мало
cache: швидко, мало
RAM: середньо, більше
SSD/HDD: повільніше, дуже багато
```

Fetch-decode-execute

CPU бере інструкцію, декодує її і виконує.

```
fetch: взяти інструкцію з пам'яті
decode: зрозуміти команду
execute: виконати операцію
```

I/O

Input/Output — взаємодія з зовнішніми пристроями або файлами.

```
keyboard input
screen output
file read/write
network request
```

⚠️ **Типові пастки:**

- **RAM ≠ disk.**
  RAM тимчасова й швидка. Disk/SSD постійний і повільніший.
  `Приклад: Відкритий документ у RAM; збережений файл на SSD.`
- **Cache ≠ browser cache у вузькому сенсі.**
  У архітектурі CPU cache — швидка пам'ять біля процесора. Browser cache — інша прикладна ідея кешування.
  `Приклад: L1/L2/L3 cache прискорює доступ CPU до даних.`
- **I/O часто повільніше за обчислення.**
  Читання з диска або мережі потребує чекання зовнішнього пристрою.
  `Приклад: Скласти два числа швидше, ніж завантажити файл із мережі.`

🔗 **Пов'язано з:** бінарний рахунок · ОС · алгоритми · мережі

🎯 Екзаменаційне спорядження

Професійна прив'язка: System programmer, DevOps/SRE, backend engineer, performance analyst.

🎯 Що реально питають

- CPU виконує інструкції, RAM тримає активні дані, disk зберігає довго, cache прискорює доступ.
- Interrupt - сигнал процесору обробити подію; driver - програмний посередник між ОС і пристроєм.
- I/O часто стає bottleneck, бо доступ до диска/мережі повільніший за обчислення в CPU.

Пастки: де ловлять

- RAM не є постійним сховищем.
- Cache у CPU - не те саме, що browser cache, хоча ідея “швидке тимчасове зберігання” спільна.
- Більше ядер не завжди дає прискорення, якщо задача не паралелізується.

Міні-дріл: Уяви ресторан: CPU - кухар, RAM - робочий стіл, disk - склад, cache - продукти під рукою.

БЛОК 05

Бази даних і SQL

Ціль: Навчитися бачити таблиці, ключі, зв'язки, нормалізацію і порядок SQL-запиту.

Карта пам'яті

entity ↔ table ↔ key ↔ relationship ↔ query ↔ result

Науково, але зрозуміло:
Реляційна база даних зберігає дані в таблицях. Рядок — один запис. Колонка — атрибут. Primary key унікально ідентифікує рядок. Foreign key зв'язує таблиці. SQL описує, який результат потрібен, а СУБД вирішує, як його отримати.

Аналогія:
База даних — це не просто Excel. Це архів із правилами: у кожного документа є номер, посилання не можуть вести в нікуди, а доступ видається ключами.

Пояснення через приклади

Моделі даних

Концептуальна модель описує світ словами. Логічна — таблицями і зв'язками. Фізична — як це зберігається в конкретній СУБД.

```
Концептуально: студент записується на курс.
Логічно: Student, Course, Enrollment.
Фізично: індекси, типи колонок, storage.
```

Primary / Foreign key

Primary key — паспорт рядка. Foreign key — посилання на паспорт в іншій таблиці.

```
users(id PRIMARY KEY, name)
orders(id PRIMARY KEY, user_id REFERENCES users(id))
```

WHERE / GROUP BY / HAVING

WHERE фільтрує рядки до групування. GROUP BY збирає групи. HAVING фільтрує групи після агрегатів.

```
SELECT city, SUM(total)
FROM orders
WHERE status = 'paid'
GROUP BY city
HAVING SUM(total) > 10000;
```

GRANT

GRANT видає права доступу до об'єктів БД.

```
GRANT SELECT ON students TO
analyst;
```

⚠️ **Типові пастки:**

- **GRANT, не ALLOW/PERMISSION.**
  У SQL ключове слово для надання прав доступу — саме GRANT. Інші слова можуть звучати логічно англійською, але не є оператором стандарту SQL.

  `Приклад: GRANT SELECT ON students TO analyst;`

- **WHERE до GROUP BY, HAVING після GROUP BY.**
  WHERE працює з окремими рядками. HAVING працює з групами, які вже отримали SUM, COUNT, AVG.

  `Приклад: WHERE status='paid' фільтрує чеки; HAVING SUM(total)>10000 фільтрує міста після підрахунку.`

- **Нормалізація зменшує дублювання й аномалії.**
  Якщо email клієнта записаний у 100 замовленнях, його треба міняти у 100 місцях. Нормалізація виносить клієнта в окрему таблицю.

  `Приклад: users(id,email) + orders(user_id,total) краще, ніж дублювати email у кожному order.`

- **ER-model — сутності, атрибути, зв'язки.**
  ER ще не про фізичні індекси або конкретне зберігання. Це схема предметної області.

  `Приклад: Student має name; Course має title; Student enrolls in Course.`

🔗 **Пов'язано з:** OLTP/OLAP · кібербезпека · data science

🎯 Екзаменаційне спорядження

Професійна прив'язка: Data engineer, backend developer, database administrator, analyst.

🎯 Що реально питають

- Концептуальна модель описує сутності бізнесу; логічна - таблиці й зв'язки; фізична - реалізацію в СУБД.
- Primary key унікально визначає рядок; foreign key посилається на ключ іншої таблиці.
- WHERE фільтрує рядки до групування, HAVING фільтрує групи після GROUP BY.

Пастки: де ловлять

- GRANT - команда надання прав, REVOKE - відкликання.
- ER-model не є нормалізацією; ER показує сутності/зв'язки, нормалізація прибирає дублювання.
- JOIN без умови може створити декартів добуток і різко збільшити кількість рядків.

Міні-дріл: Тримай схему: SELECT columns → FROM table → JOIN → WHERE → GROUP BY → HAVING → ORDER BY.

БЛОК 06

🏗️ OLTP, OLAP, Data Warehouse

Ціль: Зрозуміти різницю між системою щоденних операцій і системою аналітики.

🧭 Карта пам'яті

OLTP ↔ ETL/ELT ↔ Warehouse ↔ Mart ↔ OLAP ↔ Report

🔬 **Науково, але зрозуміло:**
OLTP оптимізований для транзакцій: швидких, точних змін невеликих порцій даних. OLAP оптимізований для аналізу великих історичних даних, агрегатів і багатовимірних зрізів.

🏠 **Аналогія:**
OLTP — каса супермаркету, яка пробиває кожен чек. OLAP — кабінет аналітика, де дивляться продажі за містами, роками, категоріями.

🧪 Пояснення через приклади

OLTP

Багато INSERT/UPDATE, важлива цілісність кожної операції.

```
INSERT INTO orders(user_id, total)
VALUES (42, 250);
```

OLAP

Багато SELECT з GROUP BY, SUM, AVG, зрізами за вимірами.

```
SELECT region, year, SUM(total)
FROM sales
GROUP BY region, year;
```

Dimensions / Measures

Dimensions — за чим дивимось: дата, місто, товар. Measures — що рахуємо: сума, кількість, середнє.

```
dimensions: date, region, product
measures: revenue, quantity
```

⚠️ **Типові пастки:**

- **OLTP не для великих аналітичних зрізів.**
  OLTP оптимізований для швидких змін маленьких порцій даних, а не для важких історичних агрегатів.

  `Приклад: Записати оплату зараз → OLTP; продажі за 3 роки по регіонах → OLAP.`

- **OLAP не для щосекундних транзакцій каси.**
  OLAP читає й агрегує багато даних. Для постійних INSERT/UPDATE потрібна транзакційна система.

  `Приклад: Каса супермаркету не має чекати, поки аналітичний cube перерахує звіт.`

- **Data Mart — тематичний шматок Data Warehouse.**
  Warehouse збирає широку картину, а mart робиться для конкретної команди або предметної області.

  `Приклад: Загальне сховище університету → mart для вступної кампанії.`

- **Cube має dimensions і measures.**
  Dimensions відповідають на “за чим розрізаємо?”, measures — “що рахуємо?”.

  `Приклад: region/year/product — dimensions; revenue/count — measures.`

🔗 **Пов'язано з:** SQL · data science · звітність

🎯 Екзаменаційне спорядження

Професійна прив'язка: Data engineer, BI analyst, analytics engineer, warehouse developer.

🎯 Що реально питають

- OLTP - транзакції: швидко записати/оновити конкретні записи.
- OLAP - аналітика: швидко читати агрегати, зрізи, trends.
- Data Warehouse збирає очищені історичні дані з різних джерел; Data Mart - тематична частина.

Пастки: де ловлять

- Каса магазину - OLTP, звіт продажів за рік - OLAP.
- ETL/ELT не є самим сховищем; це процес переміщення й трансформації даних.
- Dimension відповідає “за чим розрізати”, measure - “що міряти”.

Міні-дріл: Питання “багато користувачів одночасно змінюють записи” → OLTP. “аналітичний куб/звіт” → OLAP.

БЛОК 06B

📐 Математика в IT: функції, графіки, логарифми, експонента

Ціль: Додати графіки й математичні форми, які допомагають у Big-O, ML, ТЗНК і задачах з функціями.

🧭 Карта пам'яті

linear ↔ quadratic ↔ hyperbola ↔ exponential ↔ logarithm ↔ parabola

🔬 **Науково, але зрозуміло:**
Функція описує залежність y від x. Лінійна росте рівномірно. Квадратична дає параболу. Обернена пропорційність має форму гіперболи. Експонента росте дуже швидко. Логарифм росте повільно і є оберненою ідеєю до експоненти.

🏠 **Аналогія:**
Лінійна — щодня додавати по 10 грн. Експонента — щодня подвоювати гроші. Логарифм — скільки разів треба подвоїти, щоб дійти до числа.

🧪 Пояснення через приклади

Лінійна функція

y = kx + b. Графік — пряма. У Big-O це схоже на O(n).

```
x: 1 2 3 4
y: 2 4 6 8
кожен крок додає однаково
```

Парабола

$y = x^2$. Росте швидше за лінійну. У задачах це часто асоціація з вкладеними циклами $O(n^2)$.

```
x: 1 2 3 4
y: 1 4 9 16
форма: U-подібна парабола
```

Гіпербола / обернена пропорційність

y = 1/x. Коли x росте, y зменшується. Часто питають як форму графіка або залежність швидкості/часу.

```
x: 1 2 4 10
y: 1 0.5 0.25 0.1
```

Експонента і логарифм

2^x росте дуже швидко. $\log_2(n)$ відповідає на питання: скільки разів поділити/подвоїти.

```
2^5 = 32
log₂(32) = 5
binary search ≈ log₂(n) кроків
```

⚠️ **Типові пастки:**

- **Експонента ≠ парабола.**
  x^2 росте квадратично, а 2^x множиться на 2 при кожному кроці. Експонента швидше вивирається вгору.

  `Приклад: при x=10: x²=100, 2^x=1024.`
- **log n росте повільно.**
  Логарифм рахує кількість поділів/подвоєнь, тому навіть для великого n число кроків невелике.

  `Приклад: log₂(1024)=10.`
- **1/x зменшується, коли x росте.**
  Ділимо 1 на все більше число, тому результат стає меншим.

  `Приклад: 1/2=0.5, 1/10=0.1.`

🔗 **Пов'язано з:** Big-O · binary search · ML · ТЗНК

🎯 Екзаменаційне спорядження

> **Професійна прив'язка:** Data scientist, ML engineer, data analyst, algorithm engineer.

🎯 Що реально питають

- Лінійна функція росте рівномірно, квадратична утворює параболу, експонента росте дуже швидко.
- log n росте повільно; тому binary search O(log n) значно кращий за лінійний пошук на великих n.
- 1/x - гіпербола; при зростанні x значення зменшується і наближається до нуля.

Пастки: де ловлять

- Експонента 2^n і квадрат n^2 - не одне й те саме.
- Парабола має форму U або перевернуту U, а не S-curve.
- Відсоткова зміна від нової бази не скасовує попередню зміну автоматично.

> **Міні-дріл:** Запам'ятати порядок росту: log n < n < n log n < n^2 < 2^n < n!.

БЛОК 07

Кібербезпека і криптографія

Ціль: Розвести поняття, які в тестах виглядають схоже: hash, encryption, signature, authentication, authorization, TLS, IPsec.

Карта пам'яті

CIA ↔ hash ↔ encryption ↔ signature ↔ TLS ↔ access

Науково, але зрозуміло:
Криптографія дає різні механізми для різних задач. Hash перевіряє цілісність. Encryption приховує зміст. Digital signature підтверджує автора і незмінність. Authentication встановлює особу, authorization визначає права.

Аналогія:
Hash — відбиток пальця файлу. Encryption — сейф із ключем. Digital signature — підпис і печатка. Authorization — список кімнат, куди дозволено заходити.

Пояснення через приклади

Hash

Hash односторонній. Його не розшифровують назад. Його використовують для перевірки, чи змінилися дані.

```
hash('hello') → 2cf24d...
hash('Hello') → 185f8d...
маленька зміна → інший відбиток
```

Encryption

Шифрування має зворотну операцію з ключем: encrypt/decrypt.

```
cipher = encrypt(message, key)
plain = decrypt(cipher, key)
```

Українські алгоритми

Купина — hash-функція. Калина, Струмок, AES/Rijndael — шифри.

```
Питання: який алгоритм є криптографічною hash-функцією?
Відповідь: Купина.
```

⚠ **Типові пастки:**

- **Hash не розшифровують.**
  Hash — односторонній відбиток. Його задача не приховати повідомлення для подальшого відкриття, а перевірити збіг/цілісність.
  `Приклад: Пароль часто порівнюють через hash: вводиш пароль → hash → порівняння збереженого hash.`
- **Купина — hash.**
  У питанні про українські криптоалгоритми варіанти можуть бути всі знайомі, але ролі різні: Купина — hash, Калина/Струмок — шифри.
  `Приклад: “Який є криптографічною hash-функцією?” → Купина.`
- **AES/Rijndael — шифр.**
  AES приховує зміст і має операцію decrypt із ключем. Це відрізняє його від hash.
  `Приклад: encrypt(message,key) → cipher; decrypt(cipher,key) → message.`
- **Authentication ≠ Authorization.**
  Authentication встановлює особу. Authorization перевіряє дозволи цієї особи.
  `Приклад: Паспорт на вході — authentication; чи можна в архів — authorization.`
- **HTTPS = HTTP поверх TLS.**
  HTTP описує веб-запит/відповідь. TLS додає захищений канал, сертифікати й шифрування транспорту.
  `Приклад: https://bank.com означає, що HTTP-обмін іде через TLS.`

🔗 **Пов'язано з:** мережі · SQL GRANT · операційні системи

🎯 Екзаменаційне спорядження

> **Професійна прив'язка:** Cybersecurity analyst, backend developer, DevOps/SRE, DBA.

🎯 Що реально питають

- Hash є одностороннім відбитком; encryption можна розшифрувати ключем.
- Authentication перевіряє “хто ти”, authorization - “що тобі дозволено”.
- TLS/HTTPS захищає канал, але не робить сам сайт автоматично безпечним від усіх атак.

Пастки: де ловлять

- Купина - hash function; Kalyna і AES/Rijndael - block ciphers.
- Encoding не є encryption: Base64 легко декодується без секрету.
- Password hashing має використовувати salt і повільні алгоритми, а не просто швидкий hash.

> **Міні-дріл:** Пара: login=password → authentication. Role admin/user → authorization. SHA/Kupyna → hash. AES/Kalyna → encryption.

БЛОК 08

🌐 Мережі і операційні системи

Ціль: Знати ролі протоколів і компонентів ОС: DNS, DHCP, TCP/UDP, HTTP/HTTPS, процес, потік, файлова система, driver.

🧭 Карта пам'яті

device ↔ **OS** ↔ **network** ↔ **protocol** ↔ **server** ↔ **response**

🔬 Науково, але зрозуміло:
Мережа працює шарами: адресація, транспорт, прикладні протоколи, захист. Операційна система керує ресурсами: процесорним часом, пам'яттю, файлами, пристроями.

🏠 Аналогія:
Відкрити сайт — це як відправити лист: DNS знаходить адресу, TCP домовляється про доставку, TLS запечатує конверт, HTTP формулює прохання.

🧪 Пояснення через приклади

DNS / DHCP

DNS перетворює домен на IP. DHCP видає IP-адресу пристрою в мережі.

```
DNS: example.com → 93.184.216.34
DHCP: laptop отримав 192.168.1.20
```

TCP / UDP

TCP надійний і стежить за доставкою. UDP швидший, але без гарантій.

```
TCP: web, files, email
UDP: video call, games, DNS query
```

Process / Thread

Process — запущена програма. Thread — лінія виконання всередині процесу.

```
Browser process:
  UI thread
  network thread
  rendering thread
```

Driver

Driver — програмний посередник між ОС і пристроєм.

```
app → OS API → driver → printer
```

⚠ **Типові пастки:**

- **DNS ≠ DHCP.**
  DNS відповідає на питання “який IP у цього доменного імені?”. DHCP відповідає на питання “яку IP-адресу видати цьому пристрою?”.

  `Приклад: example.com → 93.184.216.34 це DNS; laptop отримав 192.168.1.20 це DHCP.`

- **HTTP ≠ HTTPS.**
  HTTP — протокол запиту сторінок. HTTPS — той самий HTTP, але переданий через захищений TLS-канал.

  `Приклад: HTTP як листівка; HTTPS як лист у запечатаному конверті.`

- **TCP надійніший, UDP швидший без гарантій.**
  TCP контролює доставку, порядок і повторну передачу. UDP просто надсилає датаграми без такого контролю.

  `Приклад: Файл краще TCP; відеодзвінок часто UDP, бо важливіша швидкість.`

- **Process ≠ Thread.**
  Process має власний простір ресурсів. Thread — потік виконання всередині процесу, який може ділити пам'ять з іншими потоками процесу.

  `Приклад: Browser process може мати UI thread, network thread, render thread.`

- **Driver ≠ пристрій.**
  Пристрій — hardware. Driver — програма-посередник, через яку ОС розмовляє з hardware.

```
Приклад: Printer — пристрій; printer driver — код, що
пояснює ОС, як друкувати.
```

🔗 **Пов'язано з:** бінарний рахунок · кібербезпека · файли

🎯 Екзаменаційне спорядження

Професійна прив'язка: Sysadmin, DevOps, SRE, backend engineer, network engineer.

🎯 Що реально питають

- DNS перетворює домен на IP; DHCP видає IP-конфігурацію; ARP шукає MAC за IP у LAN.
- TCP дає надійність і порядок, UDP дає меншу затримку без гарантій доставки.
- Process має власну пам'ять, thread ділить пам'ять процесу; deadlock - взаємне очікування ресурсів.

Пастки: де ловлять

- HTTP 404 - ресурс не знайдено, 403 - заборонено, 500 - помилка сервера.
- Port не дорівнює IP: IP знаходить хост, port знаходить сервіс на хості.
- Firewall фільтрує трафік, але не замінює authentication.

Міні-дріл: Схема: browser → DNS → IP → TCP/TLS → HTTP request → server → HTTP response.

БЛОК 08B

🌐 Мережі поглиблено: packets, switch/router, sysadmin/devops база

Ціль: Закрити мережеві й sysadmin/devops пастки: пакет/кадр, switch/router, ARP/NAT, ports/status codes, SSH, firewall, CI/CD, container, load balancer.

🧭 Карта пам'яті

frame ↔ packet ↔ switch ↔ router ↔ port ↔ firewall ↔ deploy

🔬 Науково, але зрозуміло:
Дані в мережі передаються шарами. На канальному рівні Ethernet-кадр працює з MAC-адресами. На мережевому рівні IP-пакет працює з IP-адресами. Switch зазвичай пересилає кадри в LAN, router пересилає пакети між мережами. DevOps-база додає автоматизацію доставки: build, test, deploy, monitoring.

🏠 Аналогія:
Switch — диспетчер усередині офісу: знає, до якого столу нести лист за MAC. Router — поштове відділення між містами: вирішує, в яку мережу відправити пакет за IP. Load balancer — адміністратор черги, що розподіляє людей між віконцями.

🧪 Пояснення через приклади

Frame vs Packet

Frame живе на L2 і має MAC-адреси. Packet живе на L3 і має IP-адреси. У тесті це часто плутають.

```
Ethernet frame: src MAC, dst MAC, payload
IP packet: src IP, dst IP, payload
Switch дивиться MAC; router дивиться IP.
```

Switch vs Router

Switch працює всередині локальної мережі, router з'єднує різні мережі.

```
Laptop A -> switch -> Laptop B у тій самій LAN
LAN 192.168.1.0/24 -> router -> Internet
```

ARP, NAT, ports

ARP знаходить MAC за IP у LAN. NAT перетворює приватні адреси на публічну. Port показує конкретну службу на хості.

```
ARP: 192.168.1.5 -> AA:BB:CC...
NAT: 192.168.1.5 -> 203.0.113.10
HTTPS port: 443
```

Sysadmin/DevOps мінімум

SSH — захищений доступ до сервера. Firewall фільтрує трафік. CI/CD автоматично збирає, тестує і доставляє зміни. Container пакує застосунок із залежностями.

```
git push -> CI build -> tests -> container image -> deploy
load balancer -> server A / server B
```

⚠️ **Типові пастки:**

- **Switch ≠ Router.**
  Switch зазвичай працює з MAC у межах LAN. Router працює з IP і маршрутами між мережами.

  `Приклад: Питання про MAC table → switch; питання про default gateway → router.`

- **Packet ≠ Frame.**
  Frame — нижчий рівень передачі в локальній мережі. Packet — IP-рівень між мережами.

  `Приклад: Ethernet frame містить MAC; IP packet містить IP.`

- **Port ≠ IP.**
  IP знаходить хост, port знаходить службу всередині хоста.

  `Приклад: 192.168.1.10:443 — хост 192.168.1.10, служба HTTPS.`

- **CI/CD ≠ container.**
  CI/CD — процес автоматизації. Container — артефакт/середовище, яке

може бути результатом build.

`Приклад: Pipeline будує Docker image і деплоїть його.`

🔗 **Пов'язано з:** мережі · операційні системи · кібербезпека · deployment

---

🎯 Екзаменаційне спорядження

Професійна прив'язка: Network engineer, sysadmin, DevOps/SRE, platform engineer.

🎯 Що реально питають

- Frame - канальний рівень і MAC; packet - мережевий рівень і IP.
- Switch з'єднує пристрої в LAN за MAC; router з'єднує мережі за IP.
- CI/CD автоматизує build-test-deploy; container пакує застосунок із залежностями; load balancer розподіляє трафік.

Пастки: де ловлять

- Docker/container - не віртуальна машина в повному сенсі.
- Load balancer не виправляє помилки коду, він розподіляє запити.
- SSH - безпечний remote shell, не протокол для вебсторінок.

Міні-дріл: Питання про локальну доставку → switch/MAC/frame. Про міжмережеву доставку → router/IP/packet.

БЛОК 08C

📦 OSI/TCP-IP рівні: де frame, packet, port і HTTP

Ціль: Навчитися не плутати рівні мережі: фізичний сигнал, кадр, IP-пакет, TCP/UDP-сегмент, порт і прикладний протокол.

🧭 Карта пам'яті

physical ↔ data link ↔ network ↔ transport ↔ application

🔬 Науково, але зрозуміло:

OSI-модель розкладає мережеву передачу на шари. На нижчих рівнях іде фізична передача сигналу й кадри Ethernet, на мережевому рівні працює IP-пакет, на транспортному - TCP або UDP з портами, на прикладному - HTTP, DNS, SMTP, SSH. У тестах часто питають не всю модель, а конкретне місце терміна в цій драбині.

🏠 Аналогія:

Лист іде шарами: папір і конверт - фізичний/канальний рівень, адреса міста - IP, номер кабінету - port, зміст листа - application protocol. Якщо переплутати місто й кабінет, лист не дійде.

🧪 Пояснення через приклади

Frame vs packet vs segment

Frame живе на канальному рівні й містить MAC-адреси. Packet живе на мережевому рівні й містить IP-адреси. TCP/UDP segment/datagram живе на транспортному рівні й працює з портами.

```
Ethernet frame: src MAC -> dst MAC
IP packet: src IP -> dst IP
TCP segment: src port -> dst port
HTTP: GET /page
```

Де який пристрій

Hub працює дуже низько й просто ретранслює сигнал. Switch дивиться на MAC. Router дивиться на IP. Firewall може фільтрувати за IP, port, protocol, rules.

```
MAC? -> switch
IP route? -> router
port 443 allowed? -> firewall
GET /api? -> application/server
```

TCP/IP практично

У реальному інтернеті частіше говорять TCP/IP stack: link, internet, transport, application. Це коротша практична модель, але ідея шарів та сама.

```
Link: Ethernet/Wi-Fi
Internet: IP
Transport: TCP/UDP
Application: HTTP/DNS/SSH
```

Питання-пастка

Якщо питають про URL, HTTP, DNS name - це application. Якщо питають про IP - network. Якщо питають про port - transport. Якщо питають про MAC - data link.

```
URL -> application
IP -> network
port -> transport
MAC -> data link
```

⚠ **Типові пастки:**

- **Port не є адресою комп'ютера.**
  IP знаходить машину або мережевий інтерфейс, port знаходить процес/сервіс на цій машині.
  `Приклад: 192.168.1.10:5432 = IP хоста + порт PostgreSQL.`
- **HTTP не працює “замість TCP”.**
  HTTP сидить вище, а TCP зазвичай переносить HTTP-запити на транспортному рівні.
  `Приклад: HTTPS = HTTP поверх TLS поверх TCP.`
- **Switch і router можуть бути в одному фізичному пристрої, але ролі різні.**
  Домашній Wi-Fi роутер часто має switch-порти, NAT, firewall і access point в одному корпусі.

```
Приклад: У тесті дивись не на коробку, а на функцію: MAC чи IP?
```

🔗 **Пов'язано з:** мережі · кібербезпека · sysadmin/devops · HTTP/DNS/SSH

🎯 Екзаменаційне спорядження

Професійна прив'язка: Network engineer, DevOps/SRE, backend developer, cybersecurity analyst.

🎯 Що реально питають

- Знати, де живуть MAC, IP, port, HTTP, DNS, SSH.
- Розуміти різницю між OSI-моделлю і практичним TCP/IP stack.
- Уміти пояснити, чому frame, packet і segment/datagram не синоніми.

Пастки: де ловлять

- MAC не маршрутизується через інтернет як IP.
- Port 443 не означає HTTPS завжди, але типово асоціюється з HTTPS.
- Firewall може працювати на різних рівнях, тому дивись, за чим фільтрує: IP, port, application rule.

Міні-дріл: Намалюй драбину: MAC -> IP -> port -> HTTP. Кожне мережеве питання став на цю драбину.

БЛОК 09

🤖 Data Science / ML

Ціль: Зрозуміти словник ML на іспитовому рівні: classification, regression, features, target, train/test, overfitting, SVM, PCA.

🧭 Карта пам'яті

data ↔ features ↔ target ↔ model ↔ prediction ↔ evaluation

🔬 **Науково, але зрозуміло:**
Модель машинного навчання шукає закономірність у даних. Classification прогнозує категорію, regression прогнозує число. Overfitting означає, що модель завчила тренувальні дані, але погано узагальнює.

🏠 **Аналогія:**
Модель як студент. Якщо студент вивчив лише конкретні відповіді одного пробника, але не зрозумів тему, на новому варіанті буде провал — це overfitting.

🧪 Пояснення через приклади

Classification vs Regression

Classification дає клас. Regression дає число.

```
email → spam/not spam  // classification
flat → predicted price  // regression
```

Features / Target

Features — вхідні ознаки. Target — те, що модель має передбачити.

```
X = [hours_studied, mistakes]
y = exam_score
```

SVM

SVM шукає гіперплощину, яка розділяє класи з максимальним відступом до найближчих точок.

```
class A   | margin |   class B
найближчі точки = support vectors
```

PCA

PCA зменшує розмірність: багато ознак стискає до кількох головних компонент.

```
100 features → PCA → 2 components
```

⚠️ **Типові пастки:**

- **PCA не класифікує, а зменшує розмірність.**
  PCA не вирішує, до якого класу належить об'єкт. Він перетворює багато ознак на меншу кількість головних компонент.

  `Приклад: 100 features → 2 components, але клас spam/not spam ще треба прогнозувати іншим методом.`

- **SVM не мінімізує support vectors; він максимізує margin.**
  Support vectors — це найближчі до межі точки, які визначають положення гіперплощини. Мета — провести межу з найбільшим відступом.

  `Приклад: class A | широкий коридор | class B — це інтуїція SVM.`

- **Classification ≠ Regression.**
  Classification повертає категорію, regression повертає числове значення.

  `Приклад: “складе/не складе” — classification; “який бал набере” — regression.`

- **Overfitting видно, коли train добре, test погано.**
  Модель запам'ятала навчальні приклади, але не вивчила загальне правило для нових даних.

  `Приклад: train accuracy 99%, test accuracy 55% — типова ознака overfitting.`

🔗 **Пов'язано з:** алгоритми · статистика · data warehouse

🎯 Екзаменаційне спорядження

Професійна прив'язка: Data scientist, ML engineer, analyst, AI engineer.

🎯 Що реально питають

- Classification прогнозує клас; regression прогнозує число; clustering групує без міток.
- Training set навчає, validation налаштовує, test оцінює фінальну якість.
- SVM шукає гіперплощину з максимальним margin; PCA зменшує розмірність.

Пастки: де ловлять

- Overfitting: train score високий, test score низький.
- Correlation не доводить causation.
- Accuracy не завжди добра метрика при дисбалансі класів.

Міні-дріл: Email spam/not spam → classification. Ціна квартири → regression. Схожі клієнти без labels → clustering.

БЛОК 10

🧠 ТЗНК: комбінаторика, логіка, відсотки

Ціль: Навчитися розпізнавати: додавати чи множити, порядок важливий чи ні, що точно випливає з умови.

🧭 Карта пам'яті

read ↔ or/and ↔ order ↔ sets ↔ percent ↔ conclusion

🔬 **Науково, але зрозуміло:**
ТЗНК перевіряє операції мислення: формалізацію умови, комбінування варіантів, роботу з множинами, відсоткові зміни, логічні висновки.

🏠 **Аналогія:**
Комбінаторика — це меню. Якщо обираємо каву або чай, додаємо. Якщо обираємо напій і десерт, множимо.

🧪 Пояснення через приклади

Правило суми

Якщо треба обрати один варіант із кількох груп: або A, або B — додаємо.

```
кава: 3 види
чай: 2 види
обрати один напій → 3 + 2 = 5
```

Правило добутку

Якщо треба виконати кілька кроків: A і B — множимо.

```
3 напої і 4 десерти
набір напій+десерт → 3 * 4 = 12
```

Перестановка vs Комбінація

Якщо ролі різні або порядок важливий — перестановка. Якщо просто група без порядку — комбінація.

```
голова і секретар з 5 людей:
5*4=20
команда з 2 людей: 5*4/2=10
```

Відсотки

Кожен відсоток рахується від поточної бази.

```
100 +20% = 120
120 −20% = 96
не 100
```

⚠️ **Типові пастки:**

- **АБО часто означає додавання.**
  Якщо вибір взаємовиключний — береться один варіант із групи A або один із групи B. Такі кількості додаються.

  `Приклад: 3 види кави або 2 види чаю → 3+2=5 напоїв.`

- **І / пара / послідовність часто означає множення.**
  Якщо треба зробити кілька незалежних кроків, кожен варіант першого кроку поєднується з кожним варіантом другого.

  `Приклад: 3 напої і 4 десерти → 3*4=12 наборів.`

- **Must be true — не додумувати.**
  У таких задачах правильне тільки те, що обов'язково випливає з умови. Життєві припущення можуть бути правдоподібні, але не доведені.

  `Приклад: Якщо A>B і B>C, то A>C must be true. Але “A найкращий” — уже домисел.`

- **Відсотки рахуються від нової бази.**
  Після першої зміни початкове число вже інше, тому друга зміна застосовується до нового значення.

  `Приклад: 100 +20% = 120; 120 -20% = 96.`

🔗 **Пов'язано з:** алгоритми · binary · reading logic

🎯 Екзаменаційне спорядження

Професійна прив'язка: Будь-яка роль, де треба швидко мислити: analyst, developer, manager.

🎯 Що реально питають

- Правило суми: або один сценарій, або інший, сценарії не перетинаються.
- Правило добутку: послідовні незалежні вибори або кілька кроків.
- Відсотки треба рахувати від актуальної бази, а не від початкової, якщо база змінилася.

Пастки: де ловлять

- “Будь-який з цих” часто означає суму, а “пара/послідовність” - добуток.
- Якщо порядок важливий - перестановки/розміщення; якщо ні - комбінації.
- Must be true означає тільки те, що гарантовано випливає з умови.

Міні-дріл: Став питання: вибір один чи кілька кроків? порядок важливий? повторення дозволені? є обмеження?

БЛОК 10В

🧩 ТЗНК поглиблено: перестановки, комбінації, пропуски, обмеження

Ціль: Навчитися розрізняти хитрі комбінаторні формулювання: порядок важливий/неважливий, є пропуски, є заборони, є повторення.

🧭 Карта пам'яті

або ↔ і ↔ порядок ↔ повторення ↔ обмеження ↔ залишок

🔬 **Науково, але зрозуміло:**
Комбінаторика рахує кількість можливих варіантів. Ключові рішення: чи важливий порядок, чи можна повторювати, чи всі місця різні, чи є заборонені варіанти.

🏠 **Аналогія:**
Це як розсадити людей за столом. Якщо місця підписані — порядок важливий. Якщо просто обрати групу людей для команди — порядок неважливий.

🧪 Пояснення через приклади

Вибір без порядку

Команда з 2 людей із 5: Анна+Богдан те саме, що Богдан+Анна.

```
5*4 / 2 = 10
ділимо на 2, бо кожна пара
порахована двічі
```

Ролі різні

Голова і секретар з 5 людей: Анна голова, Богдан секретар — не те саме, що навпаки.

```
5 варіантів на голову
4 залишилось на секретаря
5*4 = 20
```

Є пропуски / недоставлене місце

Якщо частина позицій уже зайнята або деякі варіанти заборонені, спочатку фіксуємо обмеження.

```
3 місця, А вже на першому.
Залишилось 2 людини на 2 місця:
2*1 = 2
```

З повторенням

Якщо цифри/літери можна повторювати, кількість варіантів не зменшується після вибору.

```
PIN з 4 цифр, повтори дозволені:
10*10*10*10 = 10000
```

Без повторення

Якщо повтори заборонені, після кожного вибору варіантів менше.

```
4-значний код з різних цифр:
10*9*8*7
```

⚠ **Типові пастки:**

- **Не завжди треба формула C(n,k).**
  На тесті часто швидше зрозуміти логіку місць і ролей, ніж згадувати формулу.

  `Приклад: Голова+секретар: 5*4, бо ролі різні.`
- **Якщо порядок неважливий — треба прибрати дублікати.**
  Пара A,B і B,A — одна команда, але множення рахує її двічі.

  `Приклад: 5*4/2 для команди з двох.`
- **Обмеження рахуються першими.**
  Якщо хтось уже стоїть на місці або щось заборонено, простір варіантів змінюється.

  `Приклад: Якщо перша цифра не може бути 0, то для першої позиції 9 варіантів, не 10.`

🔗 **Пов'язано з:** ТЗНК · ймовірність · множини · логіка

🎯 Екзаменаційне спорядження

Професійна прив'язка: Analyst, algorithmic thinker, product/data roles, exam reasoning.

🎯 Що реально питають

- Перестановка - порядок усіх елементів; розміщення - порядок частини; комбінація - вибір без порядку.
- Обмеження краще враховувати першими: зафіксувати місце, виключити заборонені пари, розбити на випадки.
- Euler/Venn diagrams допомагають із множинами, перетинами й “хоча б один”.

Пастки: де ловлять

- Подвійний підрахунок у перетинах множин.
- Фраза “не поруч” часто рахується через complement: усі варіанти мінус поруч.
- Пропущені місця в таблиці/послідовності часто мають два правила: по рядках і по стовпцях.

Міні-дріл: Якщо “хоча б один” - часто легше рахувати 1 - “жодного”. Якщо “не поруч” - усі - “поруч”.

БЛОК 10C / ДУЖЕ ПОВІЛЬНО І БЕЗ ПЛУТАНИНИ

🧠 Комбінаторика: як не плутати додавання, множення, ролі й команди

Ціль: навчитися не вгадувати формулу, а бачити сценарій. На тесті спершу читаємо умову як маленьку історію: кого вже взяли, кого треба добрати, чи є ролі, чи є заборонені пари.

🔬 **Науково, але людською мовою:**
Комбінаторика рахує кількість різних результатів. Два результати різні лише тоді, коли для умови тесту вони справді відрізняються. Якщо Петро-Іван і Іван-Петро виконують одну й ту саму роль "два члени команди", це один результат. Якщо Петро — голова, а Іван — заступник, то обмін ролями дає інший результат.

🧭 Перше дерево рішень

1. АБО → додавання

Обираємо один шлях із кількох альтернатив. Результат не складається з двох частин.

```
3 перші страви або 6 других
беремо одну страву
3 + 6 = 9
```

2. І → множення

Один результат складається з кількох частин. Кожен варіант першої частини поєднується з кожним варіантом другої.

```
5 сиропів і 4 присипки
кава = сироп + присипка
5 * 4 = 20
```

3. Є ролі → не ділимо

Голова і заступник — це різні бейджі. Якщо люди помінялися бейджами, результат змінився.

```
8 людей
голова: 8 варіантів
заступник: 7 варіантів
8 * 7 = 56
```

4. Просто група → ділимо

Якщо всі рівноправні, порядок усередині групи не створює новий результат.

```
2 гравці з 10 на перевірку
10 * 9 = 90 рахує А–Б і Б–А
90 / 2 = 45
```

🧩 Капітан: два різні типи задач

Капітан уже відомий і має входити

Капітана не обираємо. Він уже сидить у команді. Треба лише добрати решту.

```
Команда з 3 із 7.
Капітан уже входить.

1 місце зайняте капітаном.
Залишилось 2 місця.
Кандидатів без капітана: 6.

C(6,2) = 6 * 5 / 2 = 15
```

Капітана треба призначити

Тут "капітан" — роль, яку ще треба видати комусь. Роль важлива, тому порядок/призначення важливе.

```
Із 7 людей обрати:
капітан, спікер, секретар.

капітан: 7
спікер: 6
секретар: 5

7 * 6 * 5 = 210
```

⚠️ **Пастка зі словом "капітан":**

- **"Капітан клубу має входити"** — капітан уже є конкретною людиною. Його не обираємо, просто фіксуємо всередині.

  `Команда з 3 із 7 → добрати 2 із 6.`

- **"Обрати капітана команди"** — капітан ще не визначений. Це роль, отже вибір ролі рахується множенням.

  `Капітан і заступник із 8 → 8*7.`

🚫 Двоє не можуть бути разом

Швидкий спосіб: усі мінус погані

Коли заборонено “разом”, часто найпростіше порахувати всі команди й відняти ті, де заборона порушена.

```
Команда з 3 із 7.
А і Б не можуть бути разом.

Усі команди:
C(7,3) = 35

Погані команди:
А і Б уже всередині,
треба добрати 1 із решти 5:
C(5,1) = 5

Добрі:
35 − 5 = 30
```

Повільний спосіб: розбити на випадки

Цей спосіб довший, але добре навчає логіки.

```
Випадок 1: немає ні А, ні Б
C(5,3) = 10

Випадок 2: є А, немає Б
C(5,2) = 10

Випадок 3: є Б, немає А
C(5,2) = 10

Разом:
10 + 10 + 10 = 30
```

🎯 Рівно один vs хоча б один

Рівно один із двох

Входить А або Б, але не обидва. Спершу обираємо, хто саме з двох, потім добираємо решту.

```
Команда з 3 із 7.
Рівно один із А/Б.

хто з двох: 2 способи
добрати ще 2 із решти 5:
C(5,2)=10

2 * 10 = 20
```

Хоча б один із двох

Може входити один, а можуть входити обидва. Найлегше: усі мінус жодного.

```
Усі команди:
C(7,3)=35

Погані: немає ні А, ні Б.
Тоді обираємо 3 із решти 5:
C(5,3)=10

35 − 10 = 25
```

🏠 **Аналогія, яку варто тримати:**
Команда без ролей — це “люди в одній кімнаті”: неважливо, хто зайшов першим. Ролі — це “бейджі на грудях”: якщо бейджі поміняли, результат інший. Заборонена пара — це “ці двоє не сидять за одним столом”: або рахуємо всі столи й викидаємо погані, або чесно розписуємо випадки.

🧪 **Міні-дрил перед тестом:**

- **Якщо бачиш “або”** — перевір, чи це альтернативи. Якщо так, додаємо.

  `суп або салат → 3+6`

- **Якщо бачиш “і”** — перевір, чи один результат складається з кількох частин. Якщо так, множимо.

  `сироп і присипка → 5*4`

- **Якщо бачиш посади** — ролі різні, не ділимо.

  `голова+заступник → 8*7`

- **Якщо бачиш команду/комісію/пару** — ролей нема, ділимо на перестановки всередині групи.

  `3 з 7 → 7*6*5/(3*2*1)`

- **Якщо є “не разом”** — часто найшвидше: усі мінус погані.

  `C(7,3)-5`

БЛОК 11

GB English: grammar patterns and reading

Ціль: Не перекладати все дослівно, а впізнавати граматичні шаблони й знаходити доказ у тексті.

🧭 Карта пам'яті

sentence role ↔ grammar pattern ↔ collocation ↔ context ↔ evidence

🔬 Науково, але зрозуміло:
Use of English перевіряє граматичну роль слова і сталі мовні зв'язки. Reading перевіряє, чи можна знайти доказ у тексті і зробити обмежений висновок.

🏠 Аналогія:
Англійська в тесті — пазл. Слово має підходити не лише за перекладом, а й за формою, прийменником, часом і роллю в реченні.

🧪 Пояснення через приклади

Passive voice

Фокус на об'єкті, який отримує дію.

```
Active: The team wrote the report.
Passive: The report was written by the team.
pattern: be + V3
```

Conditionals

Third conditional описує нереальну минулу умову.

```
If I had studied, I would have passed.
if + past perfect, would have + V3
```

Collocations

Деякі слова треба запам'ятовувати парами.

```
depend on
responsible for
pay attention to
interested in
```

Reading inference

Inference — висновок із тексту, а не фантазія.

```
Text: She left without taking her umbrella.
Careful inference: it probably was not raining at that moment.
```

⚠ **Типові пастки:**

- **depend on, не depend from.**
  Це стала колокація. Її не завжди можна вивести логікою з українського перекладу, треба впізнавати як готову пару.
  `Приклад: The result depends on data quality.`
- **evidence — uncountable.**
  Uncountable nouns не вживаються з a/an і часто йдуть з little/much/some, а не many/few.
  `Приклад: There is little evidence, не a little evidences.`
- **whose показує possession.**
  Whose відповідає на “чий/чия/чиє”, а не просто замінює who.
  `Приклад: The student whose essay won the prize...`
- **Inference має мати доказ у passage.**
  Inference — це логічний висновок з тексту, а не загальна думка або життєвий досвід.
  `Приклад: Якщо в тексті “she left without umbrella”, можна обережно припустити, що дощу в той момент не було, але не можна вигадувати погоду на весь день.`

📎 **Пов'язано з:** логіка ТЗНК · читання умов · словник понять

🎯 Екзаменаційне спорядження

Професійна прив'язка: International IT communication, documentation, reading technical specs.

🎯 Що реально питають

- Reading перевіряє main idea, detail, inference, reference, vocabulary in context.
- Use of English перевіряє сталі прийменники, word forms, linkers, conditionals, tenses.
- Правильна відповідь у reading має доказ у тексті, а не просто звучить логічно.

Пастки: де ловлять

- Extreme words типу always/never часто занадто сильні.
- Synonym trap: у відповіді не ті самі слова, але той самий зміст.
- False friend: actual = фактичний, not актуальний; data is plural/uncountable depending context.

Міні-дріл: У passage підкреслити sentence-evidence. Якщо не можеш показати речення-доказ, відповідь ризикована.

БЛОК 11В

GB English toolkit: усі базові часи, conditionals, reading markers

Ціль: Дати компактний, але повний набір граматичних шаблонів, які найчастіше потрібні для Use of English і reading.

🧭 Карта пам'яті

tenses ↔ passive ↔ conditionals ↔ modals ↔ collocations ↔ reading evidence

🔬 **Науково, але зрозуміло:**
Граматика в тесті — це розпізнавання ролі слова в реченні. Часи показують часову рамку, conditionals — тип умови, modals — силу твердження, collocations — сталі поєднання.

🏠 **Аналогія:**
Речення як механізм: час — це коли працює механізм, modal — наскільки дія обов'язкова/можлива, preposition — маленька деталь, без якої механізм не замикається.

🧪 Пояснення через приклади

12 часів як сітка

Є 3 часові зони: past, present, future. Є 4 аспекти: simple, continuous, perfect, perfect continuous.

```
Present Simple: I study.
Present Continuous: I am studying.
Present Perfect: I have studied.
Present Perfect Continuous: I have been studying.
```

Past / Future patterns

Past Simple — завершена дія в минулому. Future Simple — will + V. Going to — план або очевидний намір.

```
I studied yesterday.
I will study tomorrow.
I am going to revise SQL tonight.
```

Conditionals 0/1/2/3

Zero — загальна істина. First — реальна майбутня умова. Second — нереальна теперішня/майбутня. Third — нереальна минула.

```
0: If water boils, it evaporates.
1: If I study, I will pass.
2: If I studied more, I would pass.
3: If I had studied, I would have passed.
```

Reading markers

У reading відповідь часто ховається в маркерах контрасту, причини, наслідку, прикладу.

```
contrast: however, although, whereas
cause: because, since, due to
result: therefore, as a result
example: for instance, such as
```

Корисні фразові зв'язки

Ці пари часто трапляються у cloze/use of English.

```
depend on
responsible for
interested in
capable of
similar to
different from
pay attention to
take part in
```

⚠ **Типові пастки:**

- **Present Perfect ≠ Past Simple.**
  Past Simple має завершений час у минулому. Present Perfect важливий для результату/досвіду до теперішнього моменту.

  `Приклад: I lost my key yesterday. / I have lost my key, so I cannot enter.`

- **Second і Third Conditional не плутати.**
  Second говорить про нереальне зараз/у майбутньому, Third — про минуле, яке вже не змінити.

  `Приклад: If I were ready, I would pass. / If I had been ready, I would have passed.`

- **Reading inference не має бути фантазією.**
  Правильний висновок має спиратися на конкретний рядок або ідею в тексті.

```
Приклад: however сигналізує контраст: відповідь часто після нього змінює напрям думки.
```

🔗 **Пов'язано з:** English · ТЗНК verbal · reading logic

🎯 Екзаменаційне спорядження

Професійна прив'язка: English for IT, academic reading, documentation, team communication.

🎯 Що реально питають

- Present Simple - регулярність/факт; Present Continuous - зараз/тимчасово; Present Perfect - результат до тепер.
- Past Simple - завершена дія в минулому; Past Perfect - дія до іншої минулої дії; Future Perfect - буде завершено до моменту.
- Conditionals: zero fact, first real future, second unreal present/future, third unreal past.

Пастки: де ловлять

- If I had known, I would have acted - third conditional, не second.
- Despite + noun/gerund; although + clause.
- Responsible for, interested in, depend on, similar to, different from.

Міні-дріл: Для conditionals дивись форму після if: present → first/zero; past → second; had + V3 → third.

ДОДАТОК А

🧭 Карта IT-професій: куди лягають блоки

💻 Software Developer

Процедурне програмування, ООП, алгоритми, структури даних, API, тестування.

```
must know: loops, arrays,
recursion, OOP, Big-O,
unit/integration tests
```

🗄️ Data Engineer / DBA

SQL, ключі, нормалізація, OLTP/OLAP, Data Warehouse, права доступу, індекси.

```
must know: SELECT/JOIN/GROUP
BY/HAVING, GRANT, PK/FK, ETL/ELT
```

🌐 Sysadmin / DevOps / SRE

ОС, процеси, мережі, DNS/DHCP, SSH, firewall, CI/CD, containers, load balancing.

```
must know: process/thread, ports,
TCP/UDP, SSH, Docker, deploy
pipeline
```

🔐 Cybersecurity

Hash vs encryption, authentication/authorization, TLS/HTTPS, permissions, least privilege.

```
must know: hash != encrypt,
Kupyna=hash, AES/Kalyna=cipher,
authn/authz
```

🤖 Data Science / ML

Classification/regression/clustering, overfitting, train/test split, PCA, SVM, metrics.

```
must know: class vs number,
margin, dimensionality reduction,
imbalance
```

🧪 QA / Test Engineer

Трасування коду, граничні випадки, unit/integration/e2e, логіка умов, читання вимог.

```
must know: edge cases, expected vs
actual, test levels, requirements
traps
```

Як це читати: професія не означає, що на іспиті буде питання “хто такий DevOps”. Вона показує, навіщо тема існує і які слова в тестах можуть бути сусідніми.

ДОДАТОК В

🎯 Фінальний чеклист перед Variant 1 / 2 / 3

IT: 15 хв швидкого повторення

Binary, loops, OOP, Big-O, SQL, networks, OS, security, ML.

```
1) 33 -> 100001
2) WHERE before GROUP BY, HAVING after
3) switch=MAC, router=IP
4) hash one-way, encryption reversible
5) classification=class, regression=number
```

ТЗНК: 10 хв схем

Сума/добуток, порядок, повторення, обмеження, відсотки, множини, must be true.

```
OR -> add
AND/sequence -> multiply
order matters -> arrangements/permutations
order not matters -> combinations
at least one -> complement
```

English: 10 хв маркерів

Tenses, conditionals, linkers, prepositions, reading evidence.

```
depend on
responsible for
similar to
although + clause
despite + noun/gerund
had + V3 -> would have + V3
```

Після симуляції

Не просто дивитися score. Виписати тип пастки: термін, порядок, формула, граматика, reading evidence.

```
Wrong because:
[ ] confused terms
[ ] missed keyword
[ ] counted wrong base
[ ] guessed grammar
[ ] no evidence in text
```

⚠️ **Найнебезпечніша помилка:** вчити все як окремі факти. На реальних тестах питають межі між поняттями: де схоже, але не те саме.

ОНОВЛЕННЯ V18

🧭 Карта підготовки: що саме треба вміти на тесті

Мета: не просто впізнавати правильну відповідь, а розуміти, за якою ознакою вона правильна. На реальному тесті часто перевіряють не термін, а межу між двома схожими термінами.

🧠 ТЗНК

Логіка висловлювань, комбінаторика, відсотки, таблиці, графіки, висновки з тексту. Головне питання: що саме дозволено умовою, а що ми додумали самі?

GB English

Grammar in context, Use of English, reading for evidence, linkers, collocations. Головне питання: яка граматична роль слова в реченні і де доказ у тексті?

💻 IT

Програмування, ООП, алгоритми, архітектура, ОС, мережі, БД, безпека, ML/Data Science. Головне питання: яку роль виконує об'єкт, команда або технологія?

Науково: екзамен будується на класифікації. Якщо поняття мають схожі слова, тест перевіряє ознаку розрізнення: порядок важливий чи ні, дія одна чи послідовна, права доступу чи запит даних, switch чи router, inheritance чи composition.

Приклад у житті: якщо в меню написано "кава або чай", це правило суми. Якщо написано "напій і десерт", це правило добутку. Якщо "обери 3 людей у комісію", порядок не важливий. Якщо "президент, секретар, бухгалтер", порядок важливий, бо ролі різні.

БЛОК 10 / ПОГЛИБЛЕНО

🧠 ТЗНК: комбінаторика без паніки

> **Фокус:** навчитися бачити дію в умові: "або", "і", "порядок важливий", "порядок не важливий", "заборонено", "принаймні", "рівно".

Піддисципліни всередині блоку

🧩 Правило суми

Використовуємо, коли треба обрати один варіант з кількох неперетинних груп: А або В. Варіанти не виконуються разом.

```
a можна вибрати 6 способами, b – 4 способами.
Обрати один: a або b.
Разом: 6 + 4 = 10
```

🧩 Правило добутку

Використовуємо, коли вибір складається з кількох кроків: спочатку А, потім В. Кожен варіант А може поєднатися з кожним варіантом В.

```
3 напої і 4 десерти.
Обрати напій і десерт.
Разом: 3 * 4 = 12
```

🧮 Перестановки

Порядок важливий, і використовуються всі об'єкти. Якщо 4 людини стають у чергу, А-Б-В-Г і Б-А-В-Г - різні варіанти.

```
4 людини в чергу:
4! = 4 * 3 * 2 * 1 = 24
```

🧮 Розміщення

Порядок важливий, але використовуються не всі об'єкти. Наприклад, з 8 людей треба обрати президента і секретаря.

```
A(8,2) = 8 * 7 = 56
бо перша роль має 8 варіантів,
друга – 7 після першого вибору
```

🧮 Комбінації

Порядок не важливий. Комісія {А, Б, В} - це та сама комісія, що {В, Б, А}. Тому ділимо на кількість перестановок усередині групи.

```
C(8,3) = 8 * 7 * 6 / (3 * 2 * 1) = 56
```

🚧 Заборонені варіанти

Спочатку рахуємо всі варіанти, потім віднімаємо ті, які порушують умову. Це часто найпростіший шлях.

```
Комісія з 3 людей із 8, А і Б не можуть бути разом.
Усі: C(8,3)=56
Погані: А і Б уже взяті, третій з решти 6
Правильні: 56 − 6 = 50
```

Як розв'язувати задачу "a - 6 способів, b - 4 способи, обрати один"

Крок 1: знайти ключове слово

Умова каже: обрати один: a або b. Слово 'або' означає, що ми не будуємо пару (a,b), а відкриваємо дві окремі доріжки.

```
доріжка А: 6 варіантів
доріжка В: 4 варіанти
```

Крок 2: перевірити, чи дії одночасні

Якщо треба було б обрати a і b, тоді було б 6*4. Але тут треба обрати будь-який один об'єкт з двох типів.

```
не пара: (a,b)
а один вибір: або a, або b
```

Крок 3: скласти доріжки

Коли доріжки альтернативні, додаємо. Коли кроки послідовні, множимо.

```
6 + 4 = 10
```

Пастка

Тест часто дає варіант 24, бо мозок бачить два числа і хоче множити. Але множення - тільки коли обидва вибори потрібні одночасно.

```
обрати a або b -> +
обрати a і b -> *
```

- **Слова 'будь-який з цих об'єктів' часто означають суму.**Бо треба один об'єкт з об'єднання груп, а не набір з двох об'єктів.

  `m способів для a, n способів для b -> m+n`

- **Слова 'пара', 'послідовність', 'код', 'маршрут' часто означають добуток.**Бо результат складається з кількох позицій, і кожна позиція має свої варіанти.

  `2 літери і 3 цифри -> 26*26*10*10*10`

- **'Не разом', 'без', 'крім' часто зручно рахувати через віднімання.**Усі варіанти мінус заборонені майже завжди простіше, ніж будувати правильні з нуля.

- **'Принаймні один' часто зручно рахувати через доповнення.**Принаймні один = усі варіанти мінус жодного.

  `хоча б одна дівчина = усі команди - команди без дівчат`

БЛОК 10 / ЛОГІКА

🧠 ТЗНК: імплікація, висновки, таблиці, графіки

✅ Імплікація P → Q

Фраза 'якщо P, то Q' хибна тільки в одному випадку: P істинне, а Q хибне. Обіцянка порушена лише тоді, коли умову виконали, а результат не дали.

```
Якщо прибереш кімнату (P), отримаєш цукерку (Q).
P=true, Q=false -> брехня.
Інші випадки -> правило не порушено.
```

✅ Must be true

Потрібно вибрати те, що неминуче випливає з умови. Не те, що ймовірно, знайомо або звучить розумно.

```
Якщо A>B і B>C,
то A>C обов'язково.
А от 'А найбільше у світі' не випливає.
```

📊 Таблиці

Спершу читаємо назви рядків і колонок, потім одиниці виміру. Пастка: порівняти відсотки з різних баз.

```
20% від 100 = 20
20% від 50 = 10
Однаковий відсоток не означає однакову кількість.
```

📈 Графіки функцій

Лінійна функція дає пряму. Квадратична - параболу. Обернена пропорційність y=1/x має дві гілки. Експонента швидко росте або спадає.

```
y=x -> пряма
y=x^2 -> парабола
y=1/x -> гіпербола
y=2^x -> експонента
```

🧮 Логарифм

Логарифм питає: до якого степеня треба піднести основу, щоб отримати число. Це обернена дія до експоненти.

```
2^3 = 8
log2(8) = 3
```

🧮 Відсотки

Завжди питай: від якої бази рахуємо? Збільшення і зменшення на той самий відсоток не повертає до початкового числа.

```
100 + 20% = 120
120 − 20% = 96
```

> **Науково:** логічні задачі перевіряють не пам'ять, а валідність переходу від умови до висновку. Комбінаторика перевіряє структуру простору варіантів: альтернативи додаються, незалежні кроки множаться, порядок або зберігається, або стирається діленням.

> **Приклад у житті:** якщо доставка каже "якщо замовлення до 12:00, привеземо сьогодні", вона збрехала тільки тоді, коли замовлення було до 12:00, а доставка не приїхала сьогодні. Якщо замовлення було після 12:00, правило нічого не обіцяло.

БЛОК 11 / ENGLISH

GB English: що реально ловлять

⏱️ Tenses

Час обираємо не за перекладом, а за маркером і логікою події. Present Perfect - результат до тепер. Past Simple - завершений час у минулому. Past Continuous - дія в процесі в минулому.

```
I have already finished.
I finished yesterday.
I was reading when he called.
```

🔁 Passive voice

Якщо важливий об'єкт дії, а не виконавець, часто потрібен passive: be + V3.

```
The report was prepared yesterday.
The data are stored in a database.
```

🔀 Conditionals

Умовні речення перевіряють реальність ситуації: реальна, малоймовірна, минула нереальна.

```
If it rains, we will stay home.
If I knew, I would tell you.
If I had known, I would have told you.
```

🩹 Modals

must - внутрішня необхідність або сильний висновок. have to - зовнішнє правило. should - порада. may/might - можливість.

```
You must be tired. -> сильний висновок
You have to submit the form. -> правило
```

🧱 Articles

a/an - один із багатьох, the - конкретний або вже відомий. Нульовий артикль часто з абстрактними/загальними поняттями.

```
a database -> якась база
the database -> конкретна база
```

🧲 Collocations

На тесті часто треба знати не переклад, а природну пару слів: make a decision, do research, depend on, responsible for.

```
make a mistake
do homework
depend on
interested in
```

🔗 Linkers

however - контраст, therefore - наслідок, although - контраст усередині речення, because - причина, despite - після нього noun/gerund.

```
Although it was late, we continued.
Despite being tired, we continued.
```

🔎 Reading

Не шукай 'красиву' відповідь. Шукай рядок-доказ. Якщо відповідь ширша за текст або додає припущення, вона підозріла.

```
Question asks: why?
Find because / reason / caused by.
Then compare wording, not emotion.
```

- **Переклад прийменника дослівно**Українське 'залежить від' не допомагає, англійською буде depend on.
- **Плутанина despite / although**Despite + noun/gerund. Although + повне речення з підметом і присудком.

  `despite the rain / although it rained`
- **Present Perfect vs Past Simple**Already/just/yet часто тягнуть Perfect, yesterday/last year - Past Simple.
- **Reading: відповідь звучить логічно, але її нема в тексті**На іспиті правильна відповідь має мати доказ, а не просто бути правдоподібною.

БЛОКИ IT / ПОВНА РАМКА

💻 IT: піддисципліни, які треба впізнавати

🧱 Процедурне програмування

Програма організована як послідовність команд і процедур. Дані й функції часто існують окремо.

```
function volumeUp(device) {
  device.volume += 1;
}
```

🧱 ООП: клас і об'єкт

Клас - шаблон. Об'єкт - конкретний екземпляр. Інкапсуляція ховає внутрішній стан і дає керований доступ.

```
class Speaker {
  volume = 10;
  up() { this.volume++; }
}
const jbl = new Speaker();
```

🧱 Наслідування

Новий клас отримує властивості й методи базового класу. Це 'is-a': ноутбук є пристроєм. Пастка: не плутати з композицією, де об'єкт має інший об'єкт.

```
class Device {}
class Laptop extends Device {}
// Laptop is a Device
```

🧱 Поліморфізм

Одна команда викликає різну поведінку залежно від об'єкта. Як кнопка гучності: на телефоні змінює звук дзвінка, у плеєрі - медіа, у навушниках - локальний рівень.

```
devices.forEach(d => d.volumeUp());
// method name same,
// behavior can differ
```

📈 Алгоритми і Big-O

Big-O описує, як росте час/пам'ять при збільшенні входу. O(1) - стало, O(log n) - дуже повільний ріст, O(n) - лінійно, $O(n^2)$ - вкладені порівняння.

```
binary search -> O(log n)
linear scan -> O(n)
nested loops -> O(n^2)
```

📦 Структури даних

Array швидко бере за індексом. Stack - останній зайшов, перший вийшов. Queue - перший зайшов, перший вийшов. Hash table шукає за ключем.

```
stack.push(x); stack.pop();
queue.enqueue(x); queue.dequeue();
```

🗄️ SQL і БД

SELECT читає, JOIN з'єднує таблиці, WHERE фільтрує рядки до групування, HAVING фільтрує групи після GROUP BY, GRANT дає права.

```
SELECT user_id, COUNT(*)
FROM orders
WHERE status='paid'
GROUP BY user_id
HAVING COUNT(*) > 3;
```

🗄️ Моделі даних

Концептуальна модель описує сутності бізнесу. Логічна - таблиці, ключі, зв'язки. Фізична - як це зберігає конкретна СУБД.

```
student - course - enrollment
PK: student_id
FK: enrollment.student_id
```

IT / NETWORKS, OS, SECURITY, ML

🌐 Мережі, sysadmin/devops, безпека, Data Science

🌐 OSI

OSI - модель із 7 рівнів. Для тесту важливо не вивчити все як вірш, а прив'язати приклади: фізика, кадр, пакет/IP, TCP/UDP, HTTP/DNS.

```
L2: Ethernet, MAC, switch
L3: IP, router
L4: TCP/UDP, ports
L7: HTTP, DNS, SMTP
```

🌐 Packet / frame / segment

На L2 часто говорять frame і MAC. На L3 - packet і IP. На L4 - segment/datagram і ports. Пастка: switch не маршрутизує за IP як router.

```
switch -> MAC table
router -> routing table / IP
DNS -> domain to IP
```

💻 Sysadmin minimum

Користувачі, права, процеси, журнали, служби, резервні копії, оновлення. На тесті питають роль: хто керує ресурсами і доступом.

```
Linux:
ps aux
systemctl status nginx
chmod 640 file
journalctl -u service
```

🛠️ DevOps minimum

DevOps - не тільки 'деплой'. Це CI/CD, автоматизація, контейнери, моніторинг, інфраструктура як код.

```
git push -> CI tests -> build -> deploy
Dockerfile -> image
Kubernetes -> orchestration
```

💾 Операційні системи

ОС керує процесами, пам'яттю, файловою системою, пристроями та доступом. Kernel - ядро, shell - інтерфейс команд.

```
process -> scheduler
RAM -> memory manager
file -> filesystem
USB -> driver
```

🔐 Криптографія

Hash - односторонній відбиток. Encryption - можна розшифрувати ключем. Digital signature - підтверджує автора й цілісність.

```
Kupyna -> hash
AES/Rijndael -> symmetric encryption
RSA/ECDSA -> signatures / asymmetric
```

🤖 ML / Data Science

Classification передбачає клас, regression - число, clustering - групи без міток. SVM шукає гіперплощину з максимальним margin.

```
classification: spam / not spam
regression: price
clustering: customer groups
```

📊 Data Engineering зв'язок

Для екзамену не треба йти вглиб професії, але корисно бачити роль: SQL, сховища, ETL/ELT, якість даних, права доступу, моніторинг пайплайнів.

```
source -> extract -> transform -> warehouse
quality checks -> dashboard
```

- **Switch vs router**Switch працює переважно на L2 і дивиться на MAC. Router працює на L3 і маршрутизує IP-пакети між мережами.
- **Hash vs encryption**Hash не 'розшифровують'. Якщо можна повернути оригінал ключем - це шифрування, не хеш.
- **Inheritance vs composition**Inheritance: Laptop is a Device. Composition: Laptop has a Battery.
- **WHERE vs HAVING**WHERE до групування, HAVING після групування. Якщо є агрегат COUNT/SUM у фільтрі - часто HAVING.
- **Compiler / interpreter / linker**Compiler перекладає код, interpreter виконує поступово, linker збирає об'єктні модулі в виконуваний

модуль.