

Міністерство освіти і науки України
ННІ Комп'ютерних Наук та Штучного Інтелекту
Кафедра Інтелектуальних програмних систем і технологій



ВАРІАНТ 2

Виконала : студентка групи КС-31
Михайловська Олена

З дисципліни “Стек
технологій програмування”

1. Як працює пошук констант і простори імен у модулях? `Module.nesting`.
2. `nil`, `false`, `truthy`-правила в Ruby; оператори `&.` та `||=`.
3. Як працюють `yield` та `block_given??` Коли передавати `&block` параметром?
4. Для чого потрібні модулі у Ruby? Mixin проти простору імен.
5. Наслідування vs композиція в Ruby: коли що обирати і чому?


Практична робота

1. Реалізувати зовнішній ітератор для читання великого файлу батчами по N рядків (`FileBatchEnumerator`).
2. Написати сервіс `Notifier`, який приймає будь-який об'єкт із методом `#deliver(message)`. Додати два адаптери (`Email/Slack mock`).

ПОШУК КОНСТАНТ І ПРОСТОРИ ІМЕН У RUBY. MODULE.NESTING

У Ruby константи (імена, які починаються з великої літери, наприклад User, PI, Math) шукаються згідно з ієрархією вкладених модулів та класів.



- **Ruby** дивиться, де саме ти зараз знаходишся — у якому модулі або класі.
- Потім шукає константу у поточному просторі імен, і якщо не знаходить —  піднімається вище по ланцюжку вкладеності.

Приклад:


```
module A
  X = 10
  module B
    module C
      puts Module.nesting # => [A::B::C, A::B, A]
      puts X              # => 10 (знайшов у A)
    end
  end
end
```



Приклад:

- `Module.nesting` показує масив усіх поточних модулів/класів від внутрішнього до зовнішнього.

- Ruby шукає `X` у `A::B::C`, потім у `A::B`, потім у `A`.

 Якщо не знайде константу — Ruby шукає у `Object` (глобальний простір).

**NIL, FALSE, TRUTHY-
ПРАВИЛА + ОПЕРАТОРИ
& | || =**

У Ruby є лише два "хибні" значення:

- false
- nil

Усе інше — вважається ~~"true"~~
(**truthy**). Навіть 0, "" (порожній рядок)
і [] (порожній масив) — це true.

Приклад:

```
if 0  
  puts "0 - це truthy"  
end
```

```
if nil  
  puts "не виконається"  
end
```



Оператор &. — "safe navigation" (безпечний виклик)

Використовується, щоб не отримати помилку,
якщо об'єкт nil.

```
user = nil
```

```
puts user&.name # => nil (а не помилка)
```



Тобто Ruby викликає .name, тільки якщо user не nil.

Оператор ||= — "признач, якщо nil або false"

```
name = nil  
name ||= "Guest"  
puts name # => "Guest"
```



👉 Якщо name було nil або false, Ruby присвоїть нове значення.

Якщо там уже щось є (навіть 0 чи пустий рядок) — нічого не змінює.

**YIELD, BLOCK_GIVEN?, I
КОЛИ ПЕРЕДАВАТИ
&BLOCK**

У Ruby можна передавати
блок коду у метод.



`yield` — викликає блок

```
def hello  
  yield  
end
```



```
hello { puts "Привіт!" } # => "Привіт!"
```

block_given? — перевіряє, чи передано блок

```
def maybe
  if block_given?
    yield
  else
    puts "Без блоку"
  end
end
```

```
maybe { puts "З блоком!" }
maybe
```

Якщо блоку нема — `yield` викличе помилку, тому спершу перевіряють `block_given?`.

Коли передавати **&block**

Іноді потрібно передати блок далі
або викликати його вручну.



Тоді ми пишемо **&block** у
параметрах — Ruby перетворює
блок у об'єкт типу Proc.

```
def outer(&block)
  puts "У зовнішньому методі"
  inner(&block)
end
```

```
def inner(&block)
  puts "У внутрішньому"
  block.call
end
```

```
outer { puts "Сам блок" }
```

- `yield` — просто викликає поточний блок.
- `&block` — дозволяє передати або зберегти блок як об'єкт.



**ДЛЯ ЧОГО ПОТРІБНІ
МОДУЛІ. MIXIN VS
ПРОСТІР ІМЕН**

Модуль = контейнер для коду
(методів, констант тощо).



Використовується для двох
головних цілей:

1. Простір імен (namespace)

Щоб уникнути конфліктів
імен:

```
module Geometry
  PI = 3.14
  def self.area(radius)
    PI * radius**2
  end
end
```

```
puts Geometry.area(5)
```

Без модуля може бути
кілька PI в різних
частинах програми, і
Ruby не зрозуміє, яке
саме.

2. Mixin (домішування поведінки)

```
module Flyable  
  def fly  
    puts "Я лечу!"  
  end  
end
```

```
class Bird  
  include Flyable  
end
```

```
Bird.new.fly # => "Я лечу!"
```

додати методи до класу,
але не через
наслідування.



Використання	Що робить
Namespace	просто групує код, щоб не було плутанини
Mixin	додає методи модуля в клас (ніби вставляє всередину)

НАСЛІДУВАННЯ VS КОМПОЗИЦІЯ

Наслідування (<)

Коли клас є конкретним
видом іншого класу:

```
class Dog < Animal  
end
```

● Використовуємо, коли зв'язок — "є" (is-a):



Собака — це тварина.

◆ Мінус: можна наслідувати тільки один клас. І якщо базовий клас змінюється, усе може полапатись.

Композиція (через модулі або ІНШІ об'єкти)

Коли клас має іншу поведінку або функцію:

```
module Swimmable  
  def swim  
    puts "Пливу!"  
  end  
end
```

```
class Dog  
  include Swimmable  
end
```



● Використовуємо, коли зв'язок — "має" (has-a): Собака вміє плавати, але не є плавцем.

Простими словами:

Ситуація	Що краще
Клас — це вид іншого класу	Наслідування
Хочеш базовий клас для спільної логіки	Наслідування
Хочеш уникнути зайвих зв'язків	Композиція
Клас просто має певну поведінку	Композиція (через модуль)

