

Бабенко Андрій кс32
Варіант 2

Теоретичні питання

1. Що таке метод `super`? Як і для чого він використовується?
2. Як працюють `singletons` у Ruby? Чим вони відрізняються від звичайних об'єктів?
3. Що таке `method_missing`? Як це пов'язано з метапрограмуванням?
4. Як перевантажити оператори у Ruby? Наведіть приклад.

Практичні завдання

1. Реалізуйте метод, що приймає масив чисел і проку, яка виконує задану математичну операцію для кожного числа.
2. Напишіть програму, яка передає ізольований масив між факторами для обробки його елементів

1. Що таке метод `super`? Як і для чого він використовується?

Метод `super` у `ruby` використовується для виклику однойменного методу з класу-предка (суперкласу) в межах перевизначеного методу в дочірньому класі.

Працює:

- Якщо `super` викликається без дужок, він передає всі аргументи, отримані методом в дочірньому класі, до відповідного методу суперкласу.
- Якщо `super` викликається з дужками і без аргументів, він викликає метод суперкласу без передачі жодних аргументів.
- Можна також передавати до `super` конкретні аргументи.

Використовується:

- Щоб зберегти або доповнити поведінку суперкласу, перевизначаючи метод в дочірньому класі.

```
class Parent
  def greet(name)
    "Hello, #{name}!"
  end
end

class Child < Parent
  def greet(name)
    "#{super(name)} Welcome to Ruby!"
  end
end

child = Child.new
puts child.greet("Alice") # => "Hello, Alice! Welcome to Ruby!"
```

Приклад коду

2. Як працюють Singletons у Ruby? Чим вони відрізняються від звичайних об'єктів?

Singleton у Ruby — це патерн, що гарантує створення лише одного екземпляра класу.

Працюють:

- Ruby надає Singleton модуль для створення класів із поведінкою singleton.
- Включаючи Singleton, Ruby обмежує можливість створення об'єктів: нові об'єкти можна отримати тільки через метод `instance`.

Відмінності від звичайних об'єктів:

- Звичайні класи дозволяють створювати багато об'єктів через `Class.new` або методи ініціалізації (`initialize`).
- У singleton-класах існує лише один об'єкт.

```
require 'singleton'

class Logger

  include Singleton

  def log(message)

    puts "Log: #{message}"

  end

end
```

```
logger1 = Logger.instance  
  
logger2 = Logger.instance  
  
logger1.log("First message") # => "Log: First message"  
  
puts logger1 == logger2    # => true
```

Приклад коду

3. Що таке `method_missing`? Як це пов'язано з метапрограмуванням?

Метод `method_missing` викликається, якщо об'єкт отримує виклик методу, який у нього не визначено.

Працює:

- Клас може перевизначити `method_missing` для обробки викликів неіснуючих методів.
- Це дозволяє створювати динамічну поведінку.

Зв'язок із метапрограмуванням:

- `method_missing` дозволяє програмісту визначати поведінку на основі викликів методів, які не існують, що є ключовим аспектом метапрограмування.

```
class DynamicObject  
  
  def method_missing(method_name, *args, &block)  
    puts "Method '#{method_name}' is not defined. Args: #{args.inspect}"  
  end  
  
end  
  
obj = DynamicObject.new
```

```
obj.unknown_method(1, 2, 3) # => Method 'unknown_method' is not defined.  
Args: [1, 2, 3]
```

Приклад коду

4. Як перевантажити оператори у Ruby? Наведіть приклад.

Ruby дозволяє перевизначати оператори для класів, надаючи їм нову поведінку, завдяки методам, що відповідають операторам.

Працює:

- Оператори в Ruby є методами (+, -, *, / тощо), які можна перевизначити.
- Метод, що відповідає оператору, визначається у класі, і Ruby використовує його при застосуванні оператора до об'єктів цього класу.

```
class Point  
  
  attr_accessor :x, :y  
  
  def initialize(x, y)  
    @x, @y = x, y  
  end  
  
  # Перевантаження оператора +  
  def +(other)  
    Point.new(@x + other.x, @y + other.y)  
  end  
  
  def to_s
```

```
“#{x}, #{y}”  
end  
end  
  
p1 = Point.new(1, 2)  
p2 = Point.new(3, 4)  
result = p1 + p2  
puts result # => (4, 6)
```

Приклад коду

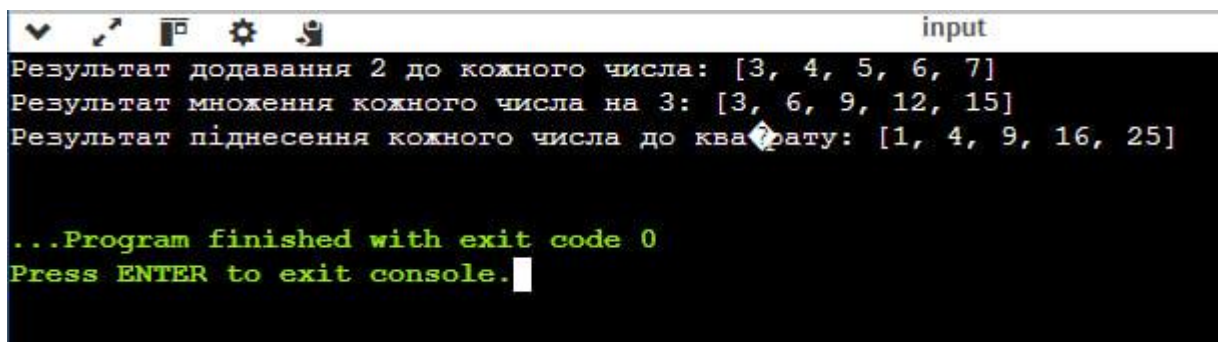
1. Реалізуйте метод, що приймає масив чисел і проку, яка виконує задану математичну операцію для кожного числа.

```
# Метод, що приймає масив чисел і блок для виконання операцій  
def apply_operation(numbers)  
  numbers.map { |num| yield(num) }  
end  
  
# Приклад використання  
numbers = [1, 2, 3, 4, 5]  
  
# Використання блоку для збільшення чисел на 2  
result_add = apply_operation(numbers) { |num| num + 2 }  
puts “Результат додавання 2 до кожного числа: #{result_add}”
```

```
# Використання блоку для множення чисел на 3
result_multiply = apply_operation(numbers) { |num| num * 3 }
puts "Результат множення кожного числа на 3: #{result_multiply}"

# Використання блоку для піднесення чисел до квадрату
result_square = apply_operation(numbers) { |num| num**2 }
puts "Результат піднесення кожного числа до квадрату: #{result_square}"
```

Лістинг 1 – вхідний код програми



```
input
Результат додавання 2 до кожного числа: [3, 4, 5, 6, 7]
Результат множення кожного числа на 3: [3, 6, 9, 12, 15]
Результат піднесення кожного числа до квадрату: [1, 4, 9, 16, 25]

...Program finished with exit code 0
Press ENTER to exit console.
```

Малюнок 1 – результати завдання

2. Напишіть програму, яка передає ізольований масив між факторами для обробки його елементів

```
# Масив чисел для обробки
numbers = [10, 20, 30, 40, 50]

# Ractor для обробки елементів масиву (додавання 5 до кожного елемента)
```

```

ractor = Ractor.new(numbers) do |nums|
  nums.map { |num| num + 5 }
end

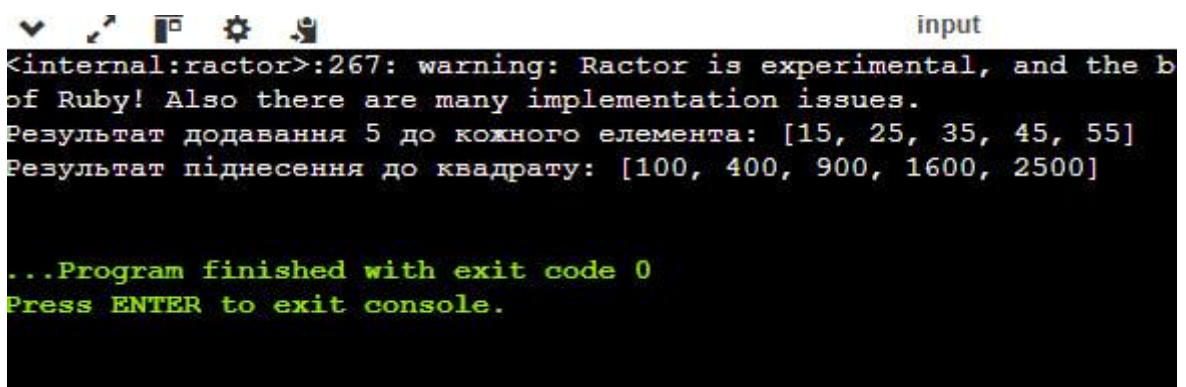
# Отримання результату з Ractor
result = ractor.take
puts "Результат додавання 5 до кожного елемента: #{result}"

# Ractor для піднесення до квадрату
ractor_square = Ractor.new(numbers) do |nums|
  nums.map { |num| num**2 }
end

# Отримання результату з другого Ractor
result_square = ractor_square.take
puts "Результат піднесення до квадрату: #{result_square}"

```

Лістинг 2 – вхідний код програми



```

input
<internal:ractor>:267: warning: Ractor is experimental, and the b
of Ruby! Also there are many implementation issues.
Результат додавання 5 до кожного елемента: [15, 25, 35, 45, 55]
Результат піднесення до квадрату: [100, 400, 900, 1600, 2500]

...Program finished with exit code 0
Press ENTER to exit console.

```

Малюнок 2 – вхідний код програми