

Гладкоскок Максим КС31

Варіант 2

Теоретичні питання

1. Щотакое метод `super`? Як і для чого він використовується?
2. Як працюють `singletons` у Ruby? Чим вони відрізняються від звичайних об'єктів?
3. Щотакое `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. Щотакє метод `super`? Як і для чого він використовується?

Метод `super` в Ruby використовується для виклику методу з однойменною назвою у суперкласі (батьківському класі). Це дозволяє реалізовувати поведінку, яка розширює або модифікує функціонал батьківського класу.

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

`Singleton` — це дизайн-патерн, який гарантує існування лише одного екземпляра класу. У Ruby для створення `Singleton`-об'єктів часто використовується модуль `Singleton`:

```
require 'singleton'
```

```
class MySingleton  
  include Singleton  
end
```

```
instance1 = MySingleton.instance
```

```
instance2 = MySingleton.instance
```

```
puts instance1 == instance2 # true
```

Відмінність від звичайних об'єктів полягає в тому, що `Singleton` дозволяє створювати тільки один екземпляр, який зберігається та використовується упродовж виконання програми.

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

Метод `method_missing` викликається, коли об'єкт отримує запит на виконання методу, який не визначений. Це дозволяє створювати динамічні методи і є важливою частиною метапрограмування

4. Як перевантажити оператори у Ruby? Наведіть приклад.

Оператори у Ruby є методами, які можна перевантажувати шляхом їхньої перезапису. Наприклад, оператор `+` можна перевантажити для роботи з власними об'єктами:

```
class Point

  attr_accessor :x, :y

  def initialize(x, y)
    @x, @y = x, y
  end

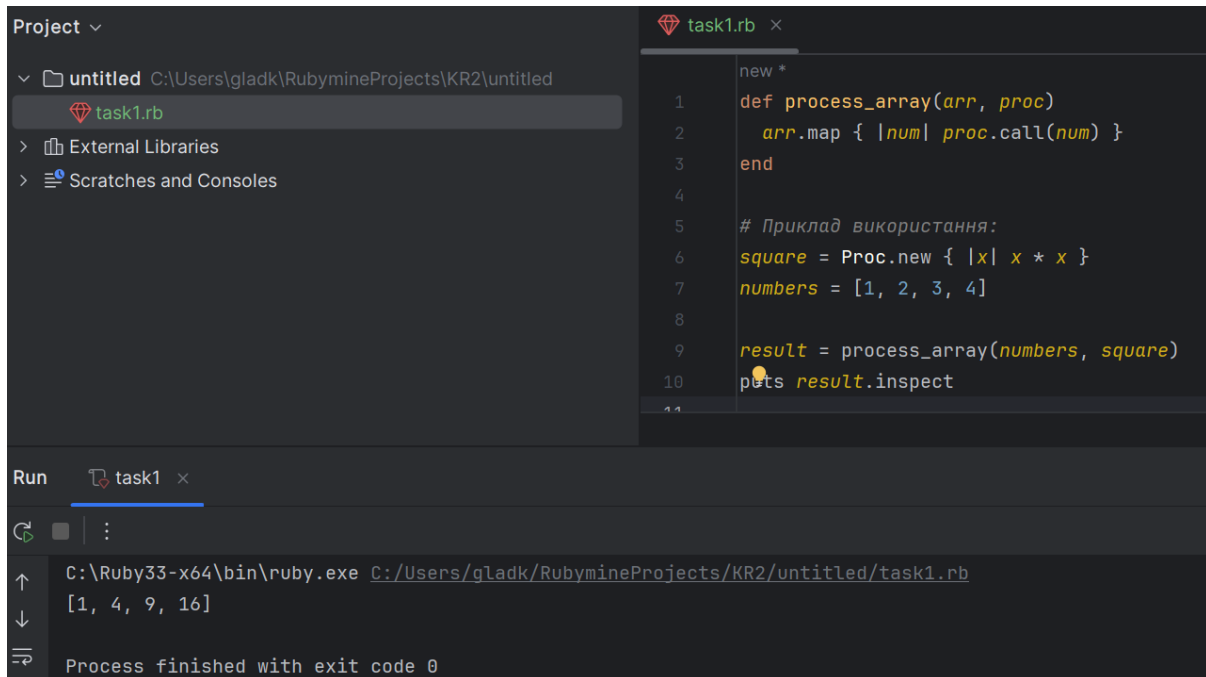
  def +(other)
    Point.new(@x + other.x, @y + other.y)
  end

  def to_s
    "(#{x}, #{y})"
  end
end

p1 = Point.new(1, 2)
p2 = Point.new(3, 4)
puts p1 + p2
# Output: (4, 6)
```

Практичні завдання

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



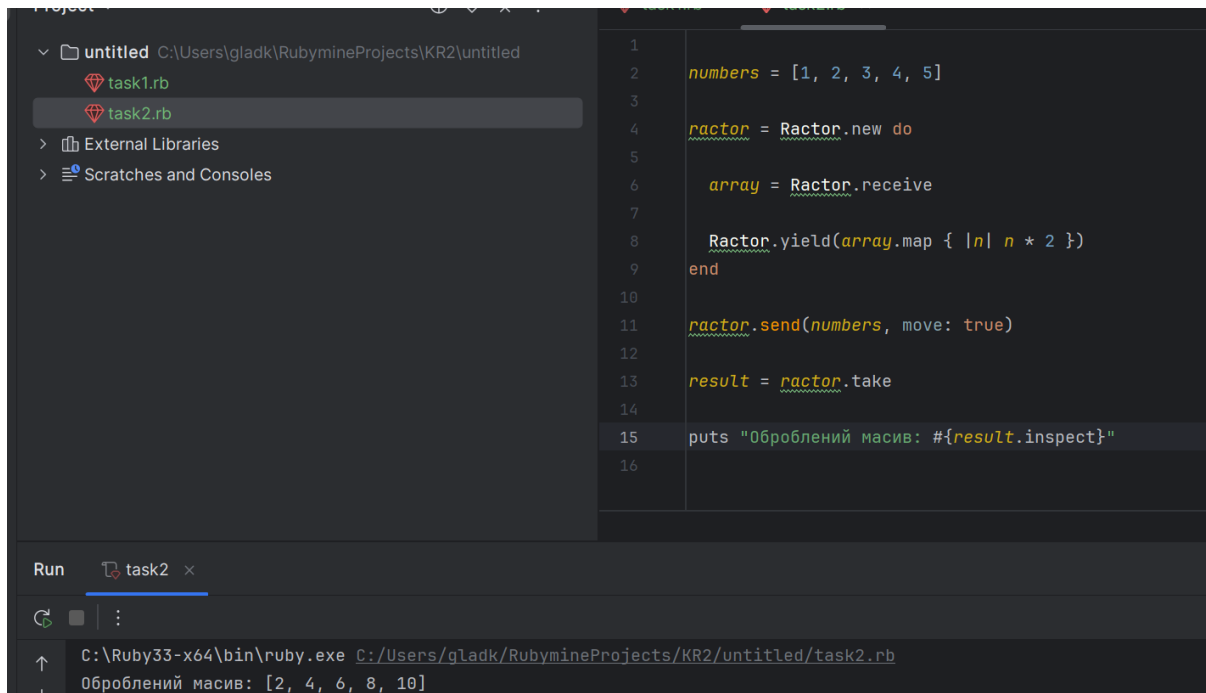
```
Project ▾
  ▾ untitled C:\Users\gladk\RubymineProjects\KR2\untitled
    task1.rb
  > External Libraries
  > Scratches and Consoles

task1.rb x
1 new *
2 def process_array(arr, proc)
3   arr.map { |num| proc.call(num) }
4 end
5
6 # Приклад використання:
7 square = Proc.new { |x| x * x }
8 numbers = [1, 2, 3, 4]
9 result = process_array(numbers, square)
10 puts result.inspect
11

Run task1 x
C:\Ruby33-x64\bin\ruby.exe C:/Users/gladk/RubymineProjects/KR2/untitled/task1.rb
[1, 4, 9, 16]
Process finished with exit code 0
```

Рисунок 1 – Завдання 1

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



```
Project ▾
  ▾ untitled C:\Users\gladk\RubymineProjects\KR2\untitled
    task1.rb
    task2.rb
  > External Libraries
  > Scratches and Consoles

task2.rb x
1
2 numbers = [1, 2, 3, 4, 5]
3
4 ractor = Ractor.new do
5   array = Ractor.receive
6   Ractor.yield(array.map { |n| n * 2 })
7 end
8
9 ractor.send(numbers, move: true)
10
11 result = ractor.take
12
13 puts "Оброблений масив: #{result.inspect}"
14
15
16

Run task2 x
C:\Ruby33-x64\bin\ruby.exe C:/Users/gladk/RubymineProjects/KR2/untitled/task2.rb
Оброблений масив: [2, 4, 6, 8, 10]
```

Рисунок 2 – Завдання 2