ЕКЗАМЕНАЦІЙНИЙ БІЛЕТ № 8

1. Як працює метод respond_to?? Для чого його можна використовувати? (5 балів)

2. Що таке композиція у Ruby? Як вона відрізняється від наслідування? (5 балів)

3. Як працює метод to_proc? У яких випадках він корисний? (5 балів)

4. Що таке параметри за замовчуванням у методах? Як їх використовувати? (5 балів)

5. **(Практичне завдання)** Створіть клас Timer, який вимірює час виконання блока коду. (20 балів)

1. Як працює метод `respond_to?`? Для чого його можна використовувати?

Метод `respond_to?` визначає, чи об'єкт підтримує певний метод. Він повертає `true`, якщо метод доступний для об'єкта, інакше `false`.

Синтаксис:

```
object.respond_to?(:method_name)
```

Приклад:

```
str = "Hello"
puts str.respond_to?(:upcase) # true
puts str.respond_to?(:nonexistent_method) # false
```

Для чого використовується:

- Перевірка, чи об'єкт підтримує метод перед викликом.
- Написання коду, який адаптується до різних типів об'єктів (динамічна перевірка методів).

2. Що таке композиція у Ruby? Як вона відрізняється від наслідування?

Композиція — це принцип, коли один клас включає інший через агрегацію чи делегування, а не через наслідування.

Приклад композиції:

```
class Engine
  def start
    "Engine started"
  end
end

class Car
  def initialize
    @engine = Engine.new
  end

  def start
    @engine.start
  end
end

car = Car.new
puts car.start # "Engine started"
```

Відмінності від наслідування:

Композиція	Наслідування
Дозволяє використовувати функціональність іншого класу без його розширення.	Базується на принципі "є типом".
Гнучкіша структура, менше зв'язування.	Жорсткіша ієрархія класів.
Використовує об'єкти інших класів.	Успадковує методи і властивості.

3. Як працює метод `to_proc`? У яких випадках він корисний?

Метод `to_proc` перетворює об'єкт на прок (`Proc`), який можна використовувати у функціональному стилі, наприклад, у методах `map`, `select`.

Синтаксис:

```
symbol.to_proc
```

Приклад:

```
array = %w[apple banana cherry]
result = array.map(&:upcase) # Викликає :upcase для кожного елемента
puts result # ["APPLE", "BANANA", "CHERRY"]
```

Корисність:

- Спрощення синтаксису.
- Використовується у функціональному стилі програмування.

4. Що таке параметри за замовчуванням у методах? Як їх використовувати?

Параметри за замовчуванням дозволяють методам мати значення, якщо виклик відбувається без передачі аргументу.

Синтаксис:

```
def greet(name = "Guest")
  "Hello, #{name}!"
end
```

Приклад:

```
puts greet("Alice") # "Hello, Alice!"
puts greet          # "Hello, Guest!"
```

Можна використовувати для запобігання порожніх значень якщо користувач нічого не вказав

5. (Практичне завдання) Створіть клас `Timer`, який вимірює час виконання блока коду.

Код:

```
class Timer
  def self.measure
    start_time = Time.now
    yield
    end_time = Time.now
    elapsed_time = end_time - start_time
    puts "Elapsed time: #{elapsed_time.round(5)} seconds"
    elapsed_time
  end
end

# Використання:
Timer.measure do
  sum = 0
  (1..10000000).each { |i| sum += i }
```

```
  puts "Sum: #{sum}"
end
```

Результат виконання:

```
Sum: 50000005000000
Elapsed time: 2.68778 seconds


Process finished with exit code 0
```