СПБПУ ПЕТРА ВЕЛИКОГО

RUBY

Выполнили: Бабахина Софья Александровна
Басалаев Даниил Александрович
Липс Екатерина Константиновна

Группа: 5030102/10201

Преподаватель: Иванов Денис Юрьевич

Санкт-Петербург
2024

ПЛАН

1. Преобразования объектов
2. Ключевое слово self
3. Наследование
4. Области видимости

ПРЕОБРАЗОВАНИЯ ОБЪЕКТОВ

Метод	Описание
`to_s`	Преобразование к строке (String)
`to_i`	Преобразование к целому числу (Integer)
`to_f`	Преобразование к числу с плавающей точкой (Float)
`to_sym`	Преобразование к символу (Symbol)
`to_a`	Преобразование к массиву (Array)
`to_h`	Пребразование к хэшу (Hash)

ПРЕОБРАЗОВАНИЯ ОБЪЕКТОВ

RUBY ПРЕДЛАГАЕТ МЕТОДЫ ДЛЯ ПРЕОБРАЗОВАНИЯ ТИПОВ, ТАКИЕ КАК TO_I, TO_F, TO_S И TO_A, КОТОРЫЕ ПРЕОБРАЗУЮТ ОБЪЕКТ В ЦЕЛОЕ ЧИСЛО, ДРОБНОЕ ЧИСЛО, СТРОКУ ИЛИ МАССИВ СООТВЕТСТВЕННО.

```ruby
p 3 + '2'.to_i  # 5

p 'tv' + 360.to_s  # "tv360"

p '36.6'.to_f  # 36.6
p 2.to_f  # 2.0

array = { fst: :hello, snd: :world }.to_a
p array  # [[:fst, :hello], [:snd, :world]]

t = Time.mktime(2019, 5, 11, 10, 20)
p t  # 2019-05-11 10:20:00 +0000
p t.to_a  # [0, 20, 10, 11, 5, 2019, 6, 131, false, "UTC"]

p [[:fst, :hello], [:snd, :world]].to_h  # {:fst=>:hello, :snd=>:world}
```

ПРЕОБРАЗОВАНИЯ ОБЪЕКТОВ

НЕКОТОРЫЙ ПРИМЕРЫ ИСПОЛЬЗОВАНИЯ JOIN И SPLIT. JOIN - ПРЕОБРАЗУЕТ МАССИВ В СТРОКУ, А SPLIT - НАОБОРОТ

```ruby
p [1, 2, 3, 4, 5].join  # "12345"
p [1, 2, 3, 4, 5].join('-')  # "1-2-3-4-5"
p ['Сергей', 'Петрович', 'Иванов'].join(' ')  # "Сергей Петрович Иванов"
p [1, 2, [3, 4, 5], ['a', 'b']].join('-')  # "1-2-3-4-5-a-b"

p 'Сергей Петрович Иванов'.split  # ["Сергей", "Петрович", "Иванов"]
p '1-2-3-4-5'.split('-')  # ["1", "2", "3", "4", "5"]

p '1-2-3-4-5'.split('-').map(&:to_i)  # [1, 2, 3, 4, 5]

result = '1-2-3-4-5'.split('-') { |x| p x.to_i }
p result

p '1-2-3-4-5'.split('-', 1)  # ["1-2-3-4-5"]
p '1-2-3-4-5'.split('-', 3)  # ["1", "2", "3-4-5"]

p '1--2--3--4-5---'.split('-')   # ["1", "", "2", "", "3", "", "4", "5"]
p '1-2-3-4-5---'.split('-')      # ["1", "2", "3", "4", "5"]
p '1-2-3-4-5---'.split('-', -1)  # ["1", "2", "3", "4", "5", "", "", ""]
```

ПРЕОБРАЗОВАНИЯ ОБЪЕКТОВ

RUBY ПОЗВОЛЯЕТ ПЕРЕГРУЖАТЬ ОПЕРАТОРЫ, ВКЛЮЧАЯ ОПЕРАТОР СЛОЖЕНИЯ. МОЖНО ОПРЕДЕЛИТЬ МЕТОД +, ЧТОБЫ ЗАДАВАТЬ ПОВЕДЕНИЕ СЛОЖЕНИЯ ДЛЯ ПОЛЬЗОВАТЕЛЬСКИХ КЛАССОВ.

```ruby
class Ticket
  attr_accessor :price
  def initialize(price)
    @price = price
  end
  def +(ticket)
    price + ticket.price
  end
end

p Ticket.new(500) + Ticket.new(600)  # 1100
```

ПРЕОБРАЗОВАНИЯ ОБЪЕКТОВ

МОЖНО НАСТРОИТЬ ПОЛЬЗОВАТЕЛЬСКИЙ КЛАСС ТАК, ЧТОБЫ ОН ПОДДЕРЖИВАЛ СЛОЖЕНИЕ С ЧИСЛОМ, ОПРЕДЕЛИВ МЕТОД +.

ВАЖНО ПОНИМАТЬ, ЧТО МЕТОД + НЕ ИЗМЕНЯЕТ ТЕКУЩИЙ ОБЪЕКТ, А СОЗДАЁТ НОВЫЙ, ПОЭТОМУ МОЖНО ДОБАВИТЬ МЕТОД ADD С УКАЗАНИЕМ !, ЧТО СИГНАЛИЗИРУЕТ, ЧТО ЭТОТ МЕТОД МЕНЯЕТ СОСТОЯНИЕ КЛАССА.

```ruby
class Ticket
  attr_accessor :price
  def initialize(price)
    @price = price
  end
  def +(value)
    case value
    when Ticket
      price + value.price
    when Numeric
      ticket = self.dup
      ticket.price += value
      ticket
    end
  end
  def add(value)
    ticket = self.dup
    ticket.price += value
    ticket
  end
  def add!(value)
    @price += value
    self
  end
end

p Ticket.new(500) + Ticket.new(600)  # 1100

t = Ticket.new(500)
p t.add!(10)  # #<Ticket:0x00007fb704066568 @price=510>
p t + 100.0  # #<Ticket:0x00007f2bca7a5830 @price=600.0>
p t + 200  # #<Ticket:0x00007fafd28155e0 @price=700>
p t  # #<Ticket:0x00007f0fb7d15808 @price=500>
```

СПБПУ ПЕТРА ВЕЛИКОГО

ПРЕОБРАЗОВАНИЯ ОБЪЕКТОВ

ТАБЛИЦА ДОПУСТИМЫХ ОПЕРАТОРОВ ДЛЯ ПЕРЕГРУЗКИ (СОЗДАНИЕ СВОЕГО ВЫВЕДЕТ ОШИБКУ)

Оператор	Описание	Пример
+	Арифметический «плюс»	5 + 2
-	Арифметический «минус»	5 - 2
*	Умножение	2 * 3
**	Возведение в степень	2 ** 3
/	Деление	8 / 2
%	Остаток от деления	27 % 5
&	Поразрядное И	5 & 4
|	Поразрядное ИЛИ	5 | 4
^	Поразрядное исключающее ИЛИ	5 ^ 4
>>	Правый поразрядный сдвиг	5 >> 2
<<	Левый поразрядный сдвиг	5 << 2
==	Логическое равенство	2 == '2'.to_i
===	Оператор сравнения в case	Integer === 3
=~	Соответствие регулярному выражению	// =~ ''
!~	Несоответствие регулярному выражению	// !~ ''
<=>	Оператор сравнения	5 <=> 2
<	Оператор «меньше»	5 < 2
<=	Оператор «меньше равно»	5 <= 2
>	Оператор «больше»	5 > 2
>=	Оператор «больше равно»	5 >= 2
+	Унарный «плюс»	+5
-	Унарный «минус»	-5
!	Логическое отрицание	!5.nil?
[]	Квадратные скобки	hello[1]
[]=	Квадратные скобки с присваиванием	hello[1] = 'a'

КЛЮЧЕВОЕ СЛОВО SELF

```ruby
class Ticket
MAX_COUNT = 300
end
def Ticket.max_count
Ticket::MAX_COUNT
end
puts Ticket.max_count # 300
```

Output:

```
300
```

```ruby
class Ticket
  MAX_COUNT = 300
  MAX_PRICE = 1200
  def self.max_count
    MAX_COUNT
  end
  def self.max_price
    MAX_PRICE
  end
end

puts Ticket.max_count # 300
puts Ticket.max_price # 1200
```

Output:

```
300
1200
```

```ruby
class Ticket
MAX_COUNT = 300
def self.max_count # Ticket.max_count
MAX_COUNT
end
end
puts Ticket.max_count # 300
```

КЛЮЧЕВОЕ СЛОВО SELF

```ruby
class Ticket
  attr_accessor :price, :status
  def initialize(price:)
    @price = price
    @status = true
  end
  def buy
    @status = false
    self
  end
end
```

```ruby
ticket = Ticket.new price: 600
puts ticket.buy.price # 600
```

```
600
```

```
 > 'hello'.capitalize
=> "Hello"
 > 'hello'.capitalize.reverse
=> "olleH"
```

КЛЮЧЕВОЕ СЛОВО SELF

ПЕРЕГРУЗКА ОПЕРАТОРОВ

```ruby
class Ticket
  attr_accessor :price
  def initialize(price:)
    @price = price
  end
  def +(number)
    @price += number
    self
  end
end

ticket = Ticket.new(price: 500)
ticket = ticket + 100
puts ticket.price # 600
```

Output:

600

ИНИЦИАЛИЗАЦИЯ ОБЪЕКТА БЛОКОМ

```ruby
class Ticket
attr_accessor :date, :price
def initialize
yield self
end
end
ticket = Ticket.new do |t|
t.price = 600
t.date = Time.mktime(2019, 5, 11, 10, 20)
end
p ticket.price # 600
p ticket.date # 2019-05-11 10:20:00 +0300
```

Output:

600
2019-05-11 10:20:00 +0000

КЛЮЧЕВОЕ СЛОВО SELF

```ruby
class String
def hello
"Hello, #{self}!"
end
end
p 'Ruby'.hello # Hello, Ruby!
```

Output:

```
"Hello, Ruby!"
```

```ruby
class Integer
  SEC_PER_MINUTE = 60
  SEC_PER_HOUR = 3_600
  SEC_PER_DAY = 86_400

  def minutes
    self * SEC_PER_MINUTE
  end

  def hours
    self * SEC_PER_HOUR
  end

  def days
    self * SEC_PER_DAY
  end

end

puts 10.minutes # 600
puts 5.hours # 18000
puts 2.days # 172800
```

Output:

```
600
18000
172800
```

СПБПУ ПЕТРА ВЕЛИКОГО

НАСЛЕДОВАНИЕ

В RUBY НАСЛЕДОВАНИЕ ЗАДАЁТСЯ ОПЕРАТОРОМ <, КОТОРЫЙ УКАЗЫВАЕТ, ЧТО ОДИН КЛАСС НАСЛЕДУЕТСЯ ОТ ДРУГОГО

RUBY НЕ ПОДДЕРЖИВАЕТ МНОЖЕСТВЕННОЕ НАСЛЕДОВАНИЕ, НО МОЖНО ИСПОЛЬЗОВАТЬ МОДУЛИ ДЛЯ РАСШИРЕНИЯ ФУНКЦИОНАЛЬНОСТИ

```ruby
class Animal
end

class Dog < Animal
end
```

НАСЛЕДОВАНИЕ

КЛАССЫ ПРЕДОСТАВЛЯЮТ СРЕДСТВА, ПОЗВОЛЯЮЩИЕ ВЫЯСНИТЬ, ЯВЛЯЕТСЯ ЛИ КЛАСС БАЗОВЫМ ИЛИ ПРОИЗВОДНЫМ В ОТНОШЕНИИ ДРУГОГО КЛАССА. ДЛЯ ЭТОГО МОЖНО ИСПОЛЬЗОВАТЬ ЛОГИЧЕСКИЕ ОПЕРАТОРЫ: >, >=, <= И <.

```ruby
class Animal
end

class Dog < Animal
end

puts Dog < Animal           # => true
puts Dog > Animal           # => false
puts Dog >= Animal          # => false
puts Dog <= Animal          # => true
puts Dog == Dog             # => true
puts Dog.ancestors          # => [Dog, Animal, Object, Kernel, BasicObject]
puts Dog.new.is_a?(Animal)  # => true
```

СПБПУ ПЕТРА ВЕЛИКОГО

НАСЛЕДОВАНИЕ

В RUBY МОЖНО НАСЛЕДОВАТЬСЯ ОТ ЛЮБОГО ВЫРАЖЕНИЯ, ВОЗВРАЩАЮЩЕГО КЛАСС, ЧТО ПОЗВОЛЯЕТ СОЗДАВАТЬ КЛАССЫ ДИНАМИЧЕСКИ.

```ruby
class Animal
end

class Dog < Animal
end

parent_class = [Animal, String].sample # choose random
class Dynamic < parent_class
end

puts Dynamic.superclass # String or Animal
```

НАСЛЕДОВАНИЕ

КОНСТАНТЫ, ОПРЕДЕЛЁННЫЕ В БАЗОВОМ КЛАССЕ, НАСЛЕДУЮТСЯ, НО ПРИ ПЕРЕОПРЕДЕЛЕНИИ СТАНОВЯТСЯ УНИКАЛЬНЫМИ ДЛЯ КАЖДОГО КЛАССА.

```ruby
class Animal
  TYPE = "Mammal"
end

class Dog < Animal
  TYPE = "Canine"
end

puts Animal::TYPE  # => "Mammal"
puts Dog::TYPE     # => "Canine"
```

НАСЛЕДОВАНИЕ

SUPER ВЫЗЫВАЕТ МЕТОД РОДИТЕЛЬСКОГО КЛАССА. SUPER БЕЗ СКОБОК ПЕРЕДАЁТ ВСЕ АРГУМЕНТЫ; С ПУСТЫМИ СКОБКАМИ НЕ ПЕРЕДАЁТ НИЧЕГО.

```ruby
class Animal
  def speak
    "Animal sound"
  end
end

class Dog < Animal
  def speak
    super + ", Woof!"  # "Animal sound, Woof!"
  end
end

puts Dog.new.speak
```

ОБЛАСТИ ВИДИМОСТИ

В ОБЪЕКТНО-ОРИЕНТИРОВАННЫХ ЯЗЫКАХ ПРОГРАММИРОВАНИЯ МЕТОДЫ ПРИНЯТО ДЕЛИТЬ НА:
PUBLIC — ОТКРЫТЫЕ;
PRIVATE — ЗАКРЫТЫЕ;
PROTECTED — ЗАЩИЩЕННЫЕ

МЕТОДЫ МОЖНО ОБЪЯВИТЬ С ИСПОЛЬЗОВАНИЕМ PUBLIC, PROTECTED И PRIVATE. ПО УМОЛЧАНИЮ ВСЕ МЕТОДЫ ЯВЛЯЮТСЯ ОТКРЫТЫМИ.

```ruby
class Animal
  def speak
    "Animal sound"
  end
end

class Dog < Animal
  def speak # Public method
    super + ", Woof!"  # "Animal sound, Woof!"
  end

  protected

  def growl
    "Grrr"     # Protected method
  end

  private

  def sleep
    "zzz"      # Private method
  end
end
```

ОБЛАСТИ ВИДИМОСТИ

```ruby
class Animal
  def speak
    "Animal sound"
  end
end

class Dog < Animal
  def speak # Public method
    super + ", Woof!"  # "Animal sound, Woof!"
  end

  protected

  def growl
    "Grrr"    # Protected method
  end

  private

  def sleep
    "zzz"     # Private method
  end

  def get_sleep
    sleep       # Ok
    self.sleep  # Error
  end
end
```

ЗАКРЫТЫЕ (PRIVATE) МЕТОДЫ НЕЛЬЗЯ ВЫЗВАТЬ С ПОЛУЧАТЕЛЕМ, ДАЖЕ С SELF.

ОБЛАСТИ ВИДИМОСТИ

КОНСТРУКТОР МОЖНО СДЕЛАТЬ ЗАКРЫТЫМ С PRIVATE_CLASS_METHOD, ОГРАНИЧИВАЯ СОЗДАНИЕ ЭКЗЕМПЛЯРОВ КЛАССА. (ТАКЖЕ МОЖНО В РУЧНУЮ ЗАПРИВАТИТЬ КОНСТРУКТОР, НО ТАК КРАСИВЕЕ)

```ruby
class SingletonDog
  private_class_method :new

  def self.instance
    @instance ||= new
  end
end

dog = SingletonDog.instance
```