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. Ветвления
2. Глобальные методы
3. Циклы
4. Блоки
5. Итераторы
6. Классы

ВЕТВЛЕНИЕ [1/4]
КЛЮЧЕВЫЕ СЛОВА IF, ELSIF, ELSE

```ruby
puts 'Начало программы'

if puts 'Эта строка выведется'
  puts 'А эта уже нет'
end

k = 0
if k == 1
  puts 'Содержимое if-конструкции'
elsif k == 2
  puts 'Содержимое else'
else
  puts 'Содержимое elif'
end

puts 'Завершение программы'

puts 'if-модификатор' if k == 0
```

Output:

```
Начало программы
Эта строка выведется
Содержимое elif
Завершение программы
if-модификатор
```

ВЕТВЛЕНИЕ [1/4]

```ruby
k = 1

puts (if k == 0
        'if'
      else
        'else'
      end)

var1 = if k == 1
        'if'
      else
        'else'
      end

if k == 2
  var2 = 'if'
else
  var2 = 'else'
end

puts var1, var2
```

Output:

```
else
if
else
```

```ruby
if x = 1
  y = 'hello'
end
puts x
puts y

```

Output:

```
1
hello
```

ГЛОБАЛЬНЫЕ МЕТОДЫ [1/6]

def **+** название метода в snake-режиме **+** телом метод (любое Ruby-выражение) **+** ***end***

```ruby
def convert
5 * 1000
end
puts convert
```

```
Output:

5000
```

```ruby
def convert(value)
value * 1000
end
puts convert(11)
```

```
Output:

11000
```

```ruby
def multi_params(x, y, *params)
p x # 1
p y # 2
p params # [3, 4]
end
multi_params(1, 2, 3, 4)
```

```
Output:

1
2
[3, 4]
```

ГЛОБАЛЬНЫЕ МЕТОДЫ [2/6]

Ruby поддерживает специальное ключевое слово **return** для возврата значений из методов.

```ruby
def convert(value, factor)
return value * factor
end
puts convert(11, 1024)
```

```
11264
```

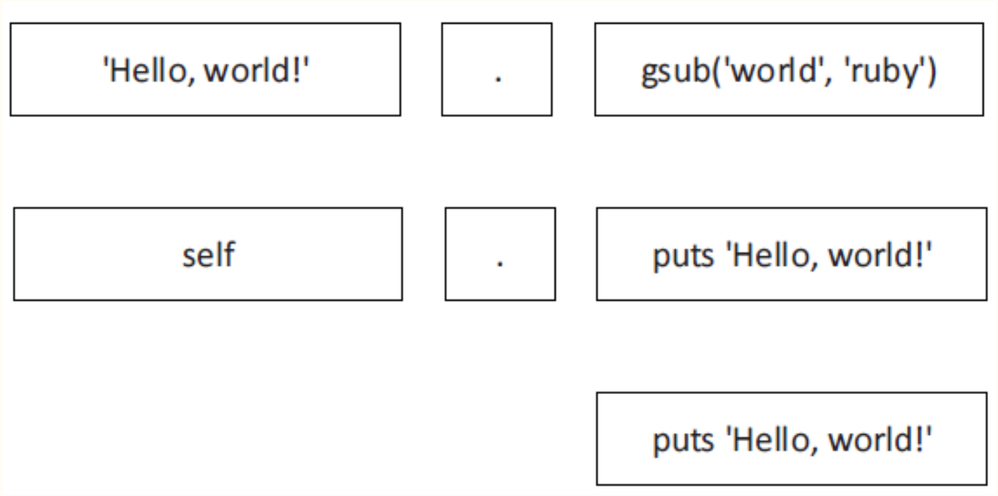
```ruby
def convert(value, factor)
value * factor
end
puts convert(11, 1024)
```

```
Output:

11264
```

Дело в том, что метод и так **возвращает результат, полученный в последнем выражении метода**. Более того, соглашения, принятые в Ruby-сообществе, требуют исключать return в последней строчке метода. Ключевое слово return находит применение в случае, когда необходимо досрочно покинуть метод

ГЛОБАЛЬНЫЕ МЕТОДЫ [3/6]



Получить ссылку на текущий объект можно при помощи специального ключевого слова **self**

```
 > self
=> main
 > self.class
=> Object
```

Глобальные методы можно (нужно) определять без использования объекта получателя. Получатель можно использовать для ограничения области действия метода.

ГЛОБАЛЬНЫЕ МЕТОДЫ [4/6]

Ruby предоставляет специальное ключевое слово **alias**, при помощи которого можно создавать для методов псевдонимы.

```ruby
def convert(value:, factor: 1000)
value * factor
end
alias kg_to_gramms convert
alias kb_to_bytes convert
puts kg_to_gramms(value: 5)
puts kb_to_bytes(value: 11, factor: 1024)
```

Output:

```
5000
11264
```

Важно помнить, что **alias** — это ключевое слово, а не метод.
Поэтому между первым и вторым аргументами запятая не ставится.

ГЛОБАЛЬНЫЕ МЕТОДЫ [5/6]

При помощи ключевого слова **undef** можно удалить метод.

```
def convert(value:, factor: 1000)
  value * factor
end

puts convert(value: 5) # 5000

undef convert

puts convert(value: 5) # undefined method `convert'
```

Рекурсия

```
def factorial(number)
return 1 if number <= 1
number * factorial(number - 1)
end
puts factorial(5)
```

```
Output:

120
```

Список предопределенных методов Ruby:

- puts, p и print
- local_variables, global_variables, instance_variables и class_variables
- require и require_relative
- sprintf и format

ГЛОБАЛЬНЫЕ МЕТОДЫ [6/6]

Для остановки программы предназначены методы **exit** и **abort:**

```
210 at_exit { puts 'Завершение программы' }
211 puts 'Эта строка выводится.'
212 abort 'Аварийная остановка программы'
213 puts 'А эта уже нет!'
214
215
216
```

```
Output:

Эта строка выводится.
Завершение программы


Аварийная остановка программы
```

Метод **sleep:**

```
puts 'Подождем 5 секунд'
sleep 5
puts 'А теперь еще 3.5 секунды'
sleep 3.5
puts 'Работа программы завершена'
```

STDIN и глобальная переменная $stdin

```
print 'Введите, пожалуйста, имя файла '
filename = gets.chomp

$stdin = File.open(filename)

puts gets('$$')
```

Чтение входного потока

val = gets
val = gets.to_i
и другие.

ЦИКЛЫ [1/1]

while - ключевое слово. Принимает логическое выражение, а конец цикла - end. Можно записать как пост-условие, но нужно использовать begin. Циклы можно делать вложенными. Прервать цикл можно словом break и закончить шаг словом next. until - ключевое слово аналог while, но логические выражение должно быть ложным. for - цикл для обхода коллекций и итераторов (не может быть пост условием).

```
Пожалуйста, введите количество повторов: 3
Hello, world!
Hello, world!
Hello, world!
Hi, world!
Hi, world!
Hi, world!
 1  2  3
 2  4  6
 3  6  9
Hello, world! only 1 time
Hello, world! only if no even
1
2
3
4
```

```ruby
print 'Пожалуйста, введите количество повторов: '
max_iterates = gets.to_i
i = 0
while i < max_iterates do
 puts 'Hello, world!'
 i += 1
end

i = 0
begin
 puts 'Hi, world!'
 i += 1
end while i < max_iterates

i = 1
size = (max_iterates * max_iterates).to_s.size + 1
while i <= max_iterates
 j = 1
 while j <= max_iterates
 print format("% #{size}d ", i * j)
 j += 1
 end
 i += 1
 puts
end

i = 0
while i < max_iterates do
 puts 'Hello, world! only 1 time'
 i += 1
 break if i >= 1
end

while i < max_iterates do
 i += 1
 next if i.even?
 puts 'Hello, world! only if no even'
end

until i >= max_iterates
 puts 'Hello, world! [until]'
 i += 1
end

for x in [1, 2, 3, 4]
 puts x
end
```

ИТЕРАТОРЫ [1/9]

Итераторы – не ключевые слова – это методы
Мы рассмотрим итераторы: loop, each, times, upto, downto и tap.
И итераторы коллекций: each, map, select, reject.

ИТЕРАТОРЫ [2/9]

Итератор loop бесконечно выполняет содержимое блока, и если его не остановить дополнительными средствами — например, ключевым словом break

```ruby
loop do
  puts 'Hello, world!'
  puts 'Hello, Ruby!'
end
```

```
Hello, world!
Hello, Ruby!
Hello, world!
Hello, Ruby!
Hello, world!
Hello, Ruby!
Hello, world!
Hello, Ruby!
Hello, world!
Hello, Ruby!
Hello, world!
Hello, Ruby!
```

ИТЕРАТОРЫ [3/9]

Итератор each полностью эквивалентен циклу for — он пробегает от первого до последнего элемента массива. И на каждой итерации в блок передается очередной элемент массива.

```ruby
rainbow = %w[red orange yellow green gray indigo violet]
for color in rainbow
 puts color
end
```

```ruby
rainbow = %w[red orange yellow green gray indigo violet]
rainbow.each { |color| puts color }
```

```
red
orange
yellow
green
gray
indigo
violet
```

ИТЕРАТОРЫ [4/9]

```ruby
rainbow = {
 red: 'красный',
 orange: 'оранжевый',
 yellow: 'желтый',
 green: 'зеленый',
 blue: 'голубой',
 indigo: 'синий',
 violet: 'фиолетовый'
}
rainbow.each { |key, name| puts "#{key}: #{name}" }
```

```
red: красный
orange: оранжевый
yellow: желтый
green: зеленый
blue: голубой
indigo: синий
violet: фиолетовый
```

ИТЕРАТОРЫ [5/9]

итератор times применяется к целым числам и позволяет выполнить блок указанное в числе количество раз

```ruby
5.times { |i| puts i }
```

```
0
1
2
3
4
```

Для итерирования от одного числа к другому применяется итератор upto

```ruby
5.upto(10) { |i| puts i }
```

```
5
6
7
8
9
10
```

Итератор downto позволяет пробегать числа с шагом -1

```ruby
10.downto(5) { |i| puts i }
```

```
10
9
8
7
6
5
```

ИТЕРАТОРЫ [6/9]

Блок each_with_index итератора принимает два параметра: первый — элемент коллекции, второй — текущий номер

```ruby
rainbow = %w[red orange yellow green gray indigo violet]
rainbow.each_with_index { |color, i| puts "#{i}: #{color}" }
```

```
0: red
1: orange
2: yellow
3: green
4: gray
5: indigo
6: violet
```

ИТЕРАТОРЫ [7/9]

map итератор возвращает новую коллекцию, количество элементов в которой совпадает с количеством элементов в исходной коллекции. Однако каждый элемент заменен на результат вычисления в блоке

```ruby
result = [1, 2, 3, 4, 5].map { |x| x + 1 }
p result
```

```
[2, 3, 4, 5, 6]
```

ИТЕРАТОРЫ [8/9]

Для того чтобы отфильтровать содержимое массива, потребуются отдельные итераторы: select и reject. Для этих итераторов также предусмотрены синонимы: find_all и delete_if соответственно.

```ruby
p [1, 2, 3, 4, 5].select { |x| x.even? } # [2, 4]
p [1, 2, 3, 4, 5].reject { |x| x.even? } # [1, 3, 5]
```

ИТЕРАТОРЫ [9/9]

tap возвращает исходный объект без изменений и используется только для побочного эффекта.

```ruby
def hash_return(params)
 params[:page] = 1
 params
end
p hash_return(per_page: 10) # {:per_page=>10, :page=>1}
```

```ruby
def hash_return(params)
 params.tap { |p| p[:page] = 1 }
end
p hash_return(per_page: 10) # {:per_page=>10, :page=>1}
```

БЛОКИ [1/5]

Для того чтобы метод мог принимать блок, в нем необходимо воспользоваться ключевым словом yield

```ruby
def my_loop
 puts 'Начало метода'
 yield
 puts 'Завершение метода'
end
my_loop { puts 'Hello, world!' }
```

```
Начало метода
Hello, world!
Завершение метода
```

БЛОКИ [2/5]

После ключевого слова yield можно указывать произвольное количество аргументов. Аргументы подставляются в соответствующие параметры блока

```ruby
def my_loop
 n = 0
 yield n += 1
 yield n += 1
 yield n += 1
end
my_loop { |i| puts "#{i}: Hello, world!" }
```

```
1: Hello, world!
2: Hello, world!
3: Hello, world!
```

БЛОКИ [3/5]

Блоки тоже возвращают значение, которое можно получить как результат вызова yield

```ruby
def greeting
 name = yield
 "Hello, #{name}!"
end

hello = greeting do
 print 'Пожалуйста, введите имя '
 gets.chomp
end
puts hello # Hello, Katya!
```

БЛОКИ [4/5]

Объект lb, полученный при помощи метода lambda, может использоваться в качестве параметров блока

```
 > lb = lambda { |n| n * n }
=> #<Proc:0x00007f987108f438@(irb):3 (lambda)>
 > lb.call(3)
=> 9
```

Чтобы подчеркнуть, что перед нами именно лямбда, можно использовать специальный синтаксис

```
 > lb = ->(x) { x * x }
=> #<Proc:0x00007febd50e0768@(irb):1 (lambda)>
 > lb.call(3)
=> 9
 > (->(x) { x * x }).call(3)
=> 9
```

БЛОКИ [5/5]

Если передать меньше параметров или больше, то будет ошибка

```ruby
lb = lambda do |fst, snd, thd|
 p fst
 p snd
 p thd
 end
lb.call('first')
```

```
NewFile1.rb:1:in `block in <main>': wrong number of arguments (given 1, expected 3) (ArgumentError)
        from NewFile1.rb:6:in `<main>'
```

КЛАССЫ [1/4]

CustomClasses.rb

```ruby
class HelloWorld_1
  def say
    'hello'
  end
end

greeting = HelloWorld_1.new
hello = HelloWorld_1.new

puts greeting.say
puts hello.say


HelloWorld_2 = Class.new do
  def say
    'hello'
  end
end

hello = HelloWorld_2.new

puts hello.say


hello_world = Class.new do
  def say
    'hello'
  end
end

hello = hello_world.new
```

Output:

```
hello
hello
hello
hello
```

```ruby
#  переопределение методов класса

class HelloWorld
  def say
  'Определяем метод say в первый раз'
  end

  def say
    'Определяем метод say во второй раз'
  end
end

hello = HelloWorld.new

puts hello.say
```

Output:

```
Определяем метод say во второй раз
```

```ruby
class HelloWorld
  def say
    'Определяем метод say в первый раз'
  end
end

class HelloWorld
  def say
    'Определяем метод say во второй раз'
  end
end

hello = HelloWorld.new

puts hello.say
```

Output:

```
Определяем метод say во второй раз
```

КЛАССЫ [2/4]

```ruby
puts 'abc'.reverse

class String
  def reverse
    'Изменяем поведение метода'
  end
end

puts 'abc'.reverse


class String
  def hello
    "Hello, #{self}!"
  end
end

puts 'world'.hello
puts 'Ruby'.hello
puts 'Igor'.hello
```

Output:

```
cba
Изменяем поведение метода
Hello, world!
Hello, Ruby!
Hello, Igor!
```

```ruby
p Car::Engine

p Engine
```

```
Car::Engine

CustomClasses.rb:136:in `<main>':
CustomClasses.rb:136:in `<main>': uninitialized constant Engine (NameError)
```

```ruby
class Car
  def title
    'BMW X7'
  end
  def description
    'Новый BMW X7 с окраской кузова...'
  end
  def engine
    @engine
  end
  def build
    @engine = Engine.new
  end
  class Engine
    def cylinders
      6
    end
    def volume
      3
    end
    def power
      250
    end
  end
end

car = Car.new
car.build

puts car.title
puts car.description
puts car.engine.cylinders
puts car.engine.volume
puts car.engine.power
```

Output:

```
BMW X7
Новый BMW X7 с окраской кузова...
6
3
250
```

КЛАССЫ [3/4]

```ruby
class Ticket
  def set_price(price)
    @pirce = price
  end
  def price
    @pirce
  end
end

first = Ticket.new
second = Ticket.new

first.set_price(500)
second.set_price(600)

puts "Цена билета first: #{first.price}"
puts "Цена билета second: #{second.price}"
```

Output:

```
Цена билета first: 500
Цена билета second: 600
```

```ruby
class Ticket
  def initialize(date:, price: 500)
    @pirce = price
    @date = date
  end
  def price
    @pirce
  end
  def date
    @date
  end
end

first = Ticket.new(date: Time.mktime(2019, 5, 10, 10, 20))
puts "Дата билета first: #{first.date}"
puts "Цена билета first: #{first.price}"
second = Ticket.new(date: Time.mktime(2019, 5, 11, 10, 20), price: 600)
puts "Дата билета second: #{second.date}"
puts "Цена билета second: #{second.price}"
```

Output:

```
Дата билета first: 2019-05-10 10:20:00 +0000
Цена билета first: 500
Дата билета second: 2019-05-11 10:20:00 +0000
Цена билета second: 600
```

МЕТОД NEW [1/1]

```
 > Array.new 3, Object.new
=> [#<Object:0x00007fc574917c38>, #<Object:0x00007fc574917c38>,
                                  #<Object:0x00007fc574917c38>]
```

```ruby
n = 0

arr = Array.new(10) do
  n += 1
  n * 10
end

p arr

arr_1 = Array.new(10) do
  @n = 0 unless @n
  @n += 1
  @n * 10
end

p arr_1
```

Output:

```
[10, 20, 30, 40, 50, 60, 70, 80, 90, 100]
[10, 20, 30, 40, 50, 60, 70, 80, 90, 100]
```

```ruby
block = proc do
  @n ||= 0
  @n += 1
  @n * 10
end
arr = Array.new(10, &block)
```

```
def new
  allocate
  initialize
end
```

GETTER, SETTER, ACCESSOR [1/1]

```ruby
class Ticket
  def initialize(date:, price: 500)
    @price = price
    @date = date
  end
  def price=(price)
    @price = price
  end
  def price
    @price
  end
  def date=(date)
    @date = date
  end
  def date
    @date
  end
end

ticket = Ticket.new(
    date: Time.new(2019, 5, 10, 10, 20),
    price: 500
  )
ticket.price = 600
ticket.date = Time.new(2019, 5, 11, 10, 20)

puts "Цена билета: #{ticket.price}"
puts "Билет на дату: #{ticket.date}"
```

```ruby
class Ticket_acc
  attr_accessor :date, :price
  def initialize(date:, price: 500)
    @pirce = price
    @date = date
  end
end

ticket_acc = Ticket_acc.new(
    date: Time.new(2019, 5, 10, 10, 20),
    price: 500
  )
ticket_acc.price = 600
ticket_acc.date = Time.new(2019, 5, 11, 10, 20)

puts "Цена билета ticket_acc: #{ticket_acc.price}"
puts "Билет на дату ticket_acc: #{ticket_acc.date}"
```

Output:

```
Цена билета: 600
Билет на дату: 2019-05-11 10:20:00 +0000
Цена билета ticket_acc: 600
Билет на дату ticket_acc: 2019-05-11 10:20:00 +0000
```