

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. Предопределенные классы

- a. Строки
- b. Символы
- c. Целые числа
- d. Вещественные числа
- e. Диапазоны
- f. Массивы
- g. Хэши
- h. Спец. объекты

## 2. Переменные

- a. Локальные переменные
- b. Глобальные переменные
- c. Предопределенные переменные
- d. Переменные класса

## 3. Константы

- a. Создание и определение констант
- b. Предопределённые константы
- c. Ключевые слова `_LINE_` и `_FILE_`
- d. Метод `require`

## 4. Операторы

- a. Арифметические операторы
- b. Операторы строк и  
форматирование строк
- c. Операторы сравнения
- d. Поразрядные операторы
- e. Спец. операторы
- f. Порядок операторов

# ПРЕДОПРЕДЕЛЕННЫЕ КЛАССЫ [1/3]

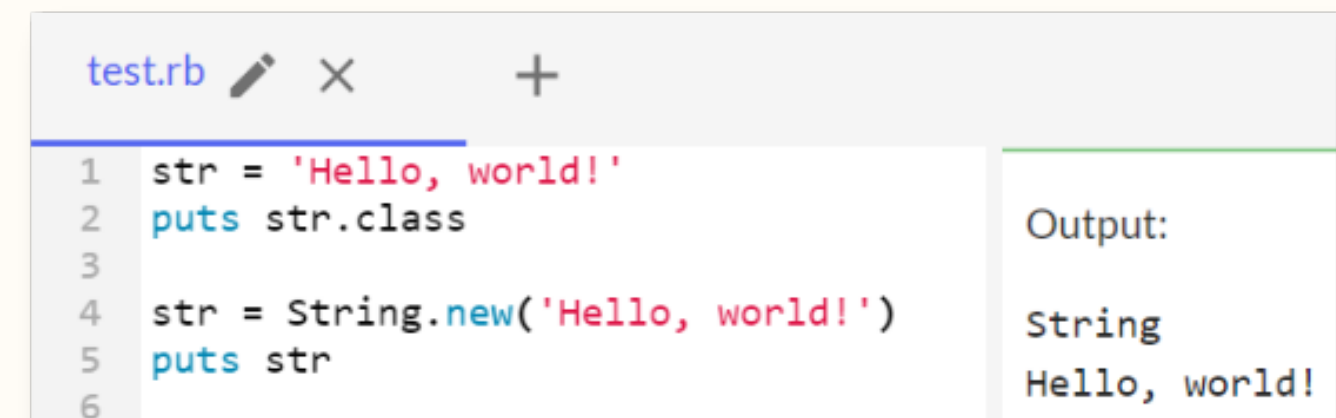
- В Ruby нет типов — поведение всех объектов задается их классами. У объекта может быть только один класс, узнать который можно при помощи метода `class`.
- Свои собственные классы можно создавать ключевым словом **class**.
- Существует множество готовых классов, например, класс `Object`, получить объект которого можно при помощи метода `new`.

test.rb		+
1	<code>puts 3.class</code>	Output:  Integer
2		
3	<code># o = Object.new</code>	
4	<code># puts o.object_id</code>	

test.rb		+
1	<code># puts 3.class</code>	Output:  60
2		
3	<code>o = Object.new</code>	
4	<code>puts o.object_id</code>	

## ПРЕДОПРЕДЕЛЕННЫЕ КЛАССЫ [2/3]

- Строка — это объект класса String. Строки можно создавать через знакомый уже метод **new**.
- Специальный способ создания при помощи укороченного синтаксиса (кавычек в случае класса String) называется **синтаксическим конструктором**.



The screenshot shows a Ruby REPL window titled 'test.rb'. It contains two code snippets. The first snippet (lines 1-3) creates a string 'Hello, world!' and prints its class, resulting in 'String'. The second snippet (lines 4-6) creates a string using the String.new method and prints it, resulting in 'Hello, world!'.

```
test.rb ✎ × +
1 str = 'Hello, world!'
2 puts str.class
3
4 str = String.new('Hello, world!')
5 puts str
6
```

Output:

String  
Hello, world!

# ПРЕДОПРЕДЕЛЕННЫЕ КЛАССЫ [3/3]

## Таблица синтаксических конструкторов

Класс	Синтаксический конструктор	Описание
String	'Hello world!'	Строки
Symbol	:white	Символы
Integer	15	Целые числа
Float	3.14159	Числа с плавающей точкой
Range	1..10	Диапазон
Array	[1, 2, 3, 4]	Массив
Hash	{hello: 'world', lang: 'ruby'}	Хэш
Proc	->(a, b) { a + b }	Прок-объекты
Regexp	//	Регулярное выражение
NilClass	nil	Неопределенное значение
TrueClass	true	Логическая истина
FalseClass	false	Логическая ложь

# СТРОКИ [1/2]

## КЛАСС STRING

```
test.rb ✎ × +
1 puts "Hello, world!".object_id
2 puts 'Hello, world!'.object_id
3
4
5
6
```

Output:

60  
80

```
test.rb +
1 puts '2 + 2 = #{ 2 + 2 }'
2 puts "2 + 2 = #{ 2 + 2 }"
3
4
5
6
```

Output:

2 + 2 = #{ 2 + 2 }  
2 + 2 = 4

```
test.rb +
1 # puts '2 + 2 = #{ 2 + 2 }'
2 # puts "2 + 2 = #{ 2 + 2 }"
3
4 puts %q(Hello, world!)
5 name = 'Ruby'
6 puts %Q(Hello, #{name}!)
```

Output:

Hello, world!  
Hello, Ruby!

```
test.rb + 42t8ex
1 str = <<here
2 В строке, которая формируется
3 heredoc-оператором,
4 сохраняются переводы строк.
5 Это идеальный инструмент
6 для ввода больших текстов.
7 here
8 puts str
9
10
```

Output:

В строке, которая формируется  
heredoc-оператором,  
сохраняются переводы строк.  
Это идеальный инструмент  
для ввода больших текстов.



# СТРОКИ [2/2]

## Другие методы класса:

- `str = 'Hello, world!'`  
`puts str[7..11] # world`
- `puts str[7..-1] # world!`
- `puts str[-6..-2] # world`
- `puts 'Hello,world!'.sub('l', '-')`  
`# "He-lo, world!"`
- `puts 'Hello, world!'.gsub('l', '-')`  
`# "He-o, wor-d!"`
- `> 'Hello, world!'.size # 13`
- `> 'Hello, world!'.length # 13`

```
test.rb +
1 puts <<~here
2 2 + 2 = #{ 2 + 2 }
3 here
4
5 puts <<~'here'
6 2 + 2 = #{ 2 + 2 }
7 here
```

Output:

```
2 + 2 = 4
2 + 2 = #{ 2 + 2 }
```

```
test.rb +
1 `ls -la`
2 # Windows
3 # `dir`
4
5 puts %x(ls -la)
6 # Windows
7 # puts %x(dir)
```

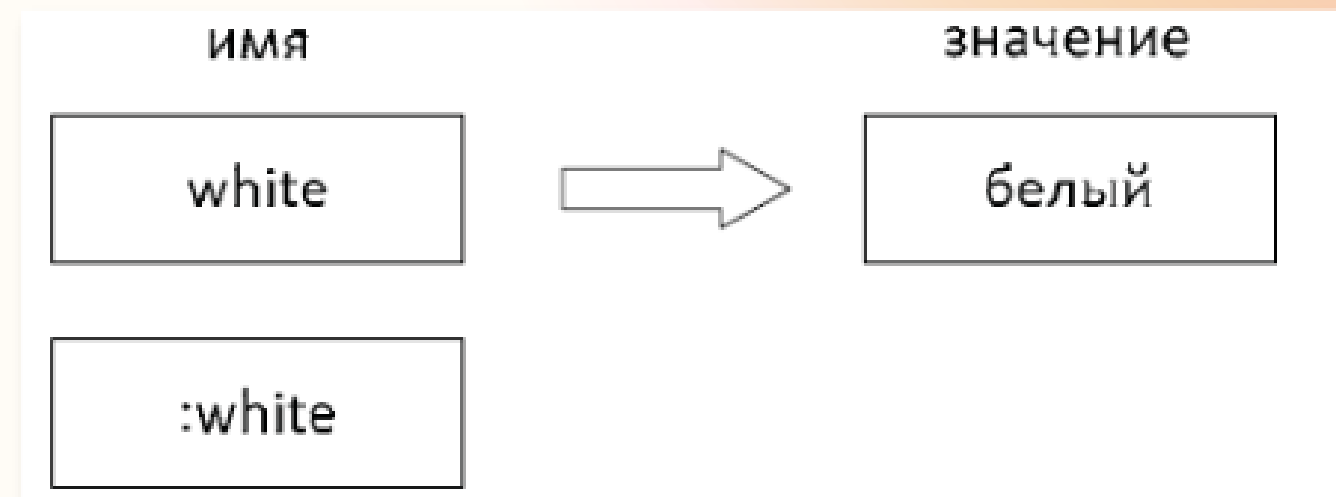
0	1	2	3	4	5	6	7	8	9	10	11	12
H	e	l	l	o	,		w	o	r	l	d	!

# СИМВОЛЫ - КЛАСС SYMBOL

```
test.rb ✎ ✕ +
1 # Symbol.new
2 # NoMethodError (undefin
3
4 p:white
5
6 puts :white.to_s|
7 puts 'white'.to_sym
8
```

Output:

```
:white
white
white
```



```
test.rb + 42t8expuf 42t8expuf ✎ NEW RUBY ▾ RU
1 p:white
2
3 puts 'white'.respond_to? :to_sym
4
5 puts %Q("#{'white'.methods}")
6
7
8
9
```

Output:

```
:white
true
[:unicode_normalize, :unicode_normalize!, :ascii_only?, :to_r, :unpack, :to_c, :%, :include?, :*,
```



# ЦЕЛЫЕ ЧИСЛА - КЛАСС INTEGER

test.rb		+
1	puts -42	
2	puts +42	
3	puts 42	
4		
5	puts 2000000 + 1900200	
6	puts 2_000_000 + 1_900_200	
7		
8	puts 0b1010101	
9	puts 0o755	
10	puts 0xffcc00	
11		
12	puts 242.to_s(2)	
13	puts 242.to_s(8)	
14	puts 242.to_s(16)	
15		
16		
17		

Output:  
-42  
42  
42  
3900200  
3900200  
85  
493  
16763904  
11110010  
362  
f2

# ВЕЩЕСТВЕННЫЕ ЧИСЛА

## КЛАСС FLOAT

```
test.rb +
1 puts 0.00012
2 puts 1.2e-4
3 puts 346.1256
4 puts 3.461256e+2
5 puts 1.8e307
6 puts 1.8e308
7 puts 1.8e308 - 1.0e307
8 puts 0 / 0.0
9 puts 1 - 0 / 0.0
10 number = 42.0
11 infpos = 100 / 0.0
12 infneg = -100 / 0.0
```

Output:

```
0.00012
0.00012
346.1256
346.1256
1.8e+307
Infinity
Infinity
```

```
13 nan = 0 / 0.0
14 p number.infinite?
15 p infpos.infinite?
16 p infneg.infinite?
17 p number.nan?
18
19 puts 2.class
20 puts 2.0.class
21 puts (2 + 2.0).class
22
23 puts 2.0.to_i
24 puts 2.to_f
25
26
```

```
NaN
NaN
nil
1
-1
false
Integer
Float
Float
2
2.0
```

# ДИАПАЗОНЫ КЛАСС RANGE

```
test.rb +
1 p 1..5
2 p 1...5
3 p (1..5).class
4 p Range.new(1, 5)
5 p Range.new(1, 5, true)
6
7 puts (1..5).first
8 puts (1..5).last
9
10 range = 1..5
11 p range.include? 3
12 p range.include? 7
13
```

Output:

```
1..5
1...5
Range
1..5
1...5
1
5
true
```

```
14 p (1..).include? 100500
15 p (1..).include? -1
16
17 p (1..).include? 100500
18 p (1..).include? -1
19
20 range = 1..5
21 p range.cover?(3)
22 p range.cover?(10)
23 p range.include?(2..3)
24 p range.cover?(2..3)
25 p range.cover?(3..7)
26
```

```
true
false
true
false
true
false
true
false
true
false
```

# МАССИВЫ КЛАСС ARRAY

```
test.rb + 42t8expuf NEW RUBY RUN
```

```
1 p [1, 2, 3, 4, 5]
2 p Array.new
3 arr = [1, 2, 3, 4, 5]
4 p arr[0]
5 p arr[-1]
6 p arr[2, 2]
7 p arr[2..3]
8
9 p [1, 'hello', 3, ['first', 'second']]
10
11 colors = [
12   'красный', 'оранжевый', 'желтый', 'зеленый',
13   'голубой', 'синий', 'фиолетовый'
14 ]
15 p colors
16
17 colors = %w[красный оранжевый желтый зеленый голубой синий фиолетовый]
18 p colors
19
20 p %w(Hello, \ world! Hello, \ Ruby!)
21
22 p colors = [:red, :orange, :yellow, :green, :blue, :indigo, :violet]
23 colors = %i[red orange yellow green blue indigo violet]
24 p colors
```

Output:

```
[1, 2, 3, 4, 5]
[]
1
5
[3, 4]
[3, 4]
[1, "hello", 3, ["first", "second"]]
["красный", "оранжевый", "желтый", "зеленый", "голубой", "синий", "фиолетовый"]
["красный", "оранжевый", "желтый", "зеленый", "голубой", "синий", "фиолетовый"]
["Hello, world!", "Hello, Ruby!"]
[:red, :orange, :yellow, :green, :blue, :indigo, :violet]
[:red, :orange, :yellow, :green, :blue, :indigo, :violet]
```



# ХЭШИ

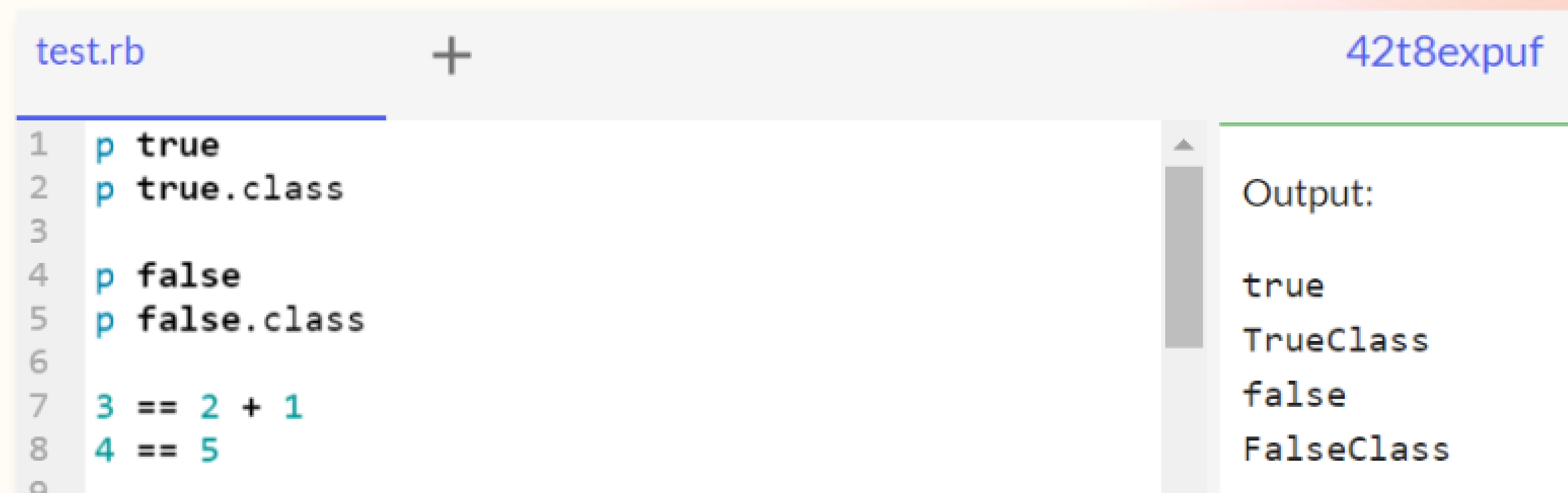
## КЛАСС HASH

test.rb + 42t

```
1 h = { 'first' => 'hello', 'second' => 'world' }
2 puts h['second']
3
4 h = { :first => 'hello', :second => 'world' }
5
6 # актуальный синтаксис
7 h = { first: 'hello', second: 'world' }
8 puts h[:first]
9
```

Output:  
  
world  
hello

# ЛОГИЧЕСКИЕ ОБЪЕКТЫ TRUE И FALSE



The screenshot shows a Ruby REPL window with a tab labeled 'test.rb'. The code entered is as follows:

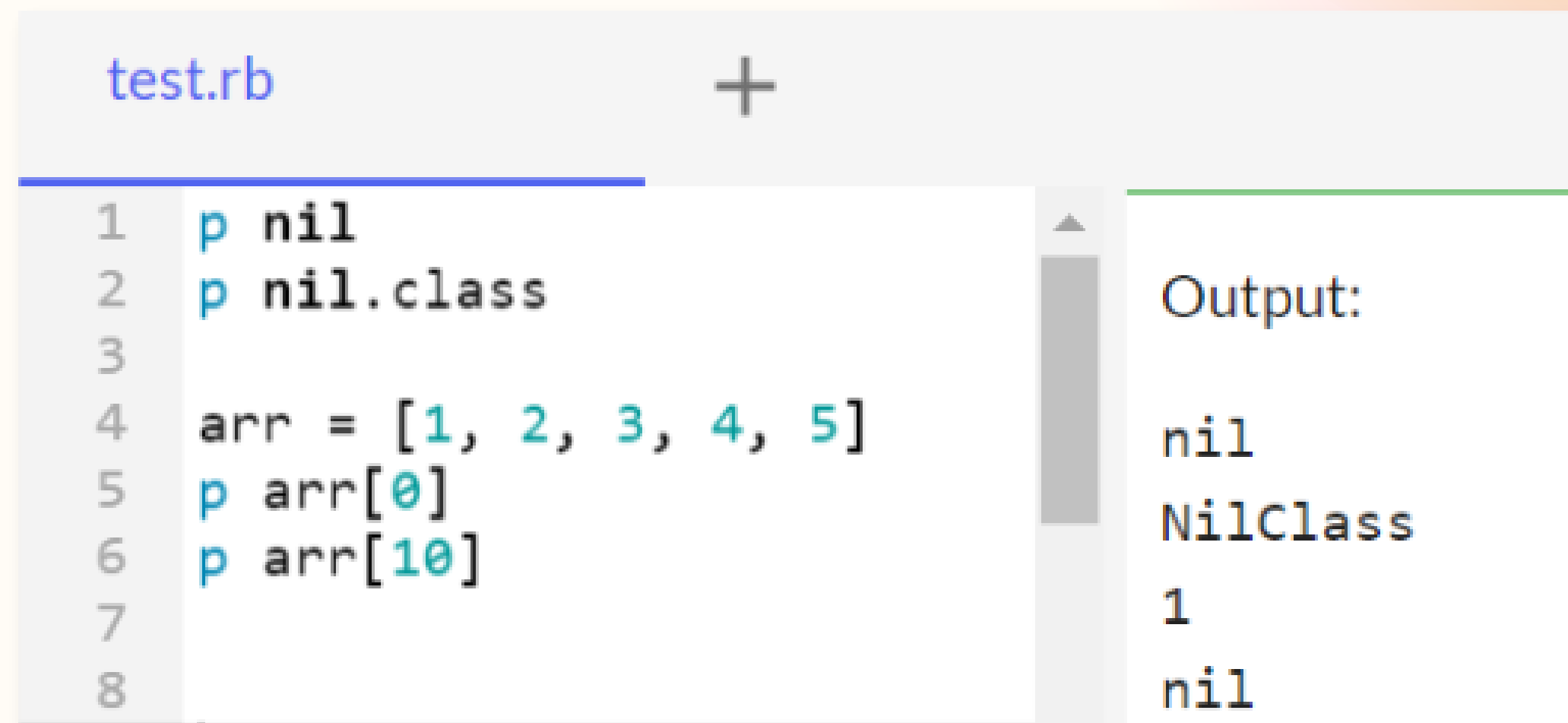
```
1 p true
2 p true.class
3
4 p false
5 p false.class
6
7 3 == 2 + 1
8 4 == 5
9
```

The output on the right side of the window is:

```
Output:
true
TrueClass
false
FalseClass
```



# ОБЪЕКТ NIL



The screenshot shows a Ruby REPL window with a tab labeled 'test.rb'. The code is as follows:

```
1 p nil
2 p nil.class
3
4 arr = [1, 2, 3, 4, 5]
5 p arr[0]
6 p arr[10]
7
8
```

The output on the right is:

```
Output:
nil
NilClass
1
nil
```

СПБПУ ПЕТРА ВЕЛИКОГО

# ТИПЫ ПЕРЕМЕННЫХ

**LOCAL**

**\$GLOBAL**

**@OBJECT**

**@@KLASS**

## ЛОКАЛЬНЫЕ ПЕРЕМЕННЫЕ [1/2]

```
1 ▾ def say_bye
2   x = 'переменная x из say_bye'
3 end
4
5 ▾ def start
6   x = 'переменная x из start'
7   puts x # переменная x из start
8   say_bye
9   puts x # переменная x из start
10 end
11
12 x = 'переменная из глобальной области видимости'
13 puts x # переменная из глобальной области видимости
14 start
15 puts x # переменная из глобальной области видимости
```

Output:

переменная из глобальной области видимости  
переменная x из start  
переменная x из start  
переменная из глобальной области видимости

## ЛОКАЛЬНЫЕ ПЕРЕМЕННЫЕ [2/2]

**local\_variables** - метод для получения списка локальных переменных

```
1 greeting = 'Hello, world!'
2 number = 100_000
3 p local_variables # [:greeting, :number]
4
5
```

Output:

```
[:greeting, :number]
```

# ГЛОБАЛЬНЫЕ ПЕРЕМЕННЫЕ [1/2]

```
1 def say_bye
2   $x = 'переменная $x из say_bye'
3 end
4
5 def start
6   $x = 'переменная $x из start'
7   puts $x # переменная $x из start
8   say_bye
9   puts $x # переменная $x из say_bye
10 end
11
12 $x = 'переменная из глобальной области видимости'
13 puts $x # переменная из глобальной области видимости
14 start
15 puts $x # переменная $x из say_bye
```

Output:

переменная из глобальной области видимости  
переменная \$x из start  
переменная \$x из say\_bye  
переменная \$x из say\_bye

# ГЛОБАЛЬНЫЕ ПЕРЕМЕННЫЕ [2/2]

**global\_variables** - метод для получения списка локальных переменных

```
$x = 'переменная из глобальной области видимости'  
p global_variables
```

Output:

```
[ :$?, :$$, :$&, :$`, :$', :$+, :$=, :$KCODE, :$-K, :$,, :$/,
```



# ПРЕДОПРЕДЕЛЕННЫЕ ПЕРЕМЕННЫЕ

- **\$LOAD\_PATH** - содержит список путей для поиска библиотек и классов Ruby
- **\$stdout** - ссылается на текущий поток вывода, связанный по умолчанию с консолью. Именно в этот поток отправляется информация, которая выводится методом puts.
- **\$PROGRAM\_NAME** - позволяет получить название процесса

# ИНСТАНС-ПЕРЕМЕННЫЕ

**instance\_variables** - метод для получения списка локальных переменных

```
1 class Ticket
2   def set_date(date)
3     @date = date
4   end
5   def date
6     @date
7   end
8   def set_price(price)
9     @price = price
10  end
11  def price
12    @price
13  end
14 end
15
16 first = Ticket.new
17 second = Ticket.new
18 first.set_price(2000)
19 first.set_date('31.10.2019')
20 second.set_price(3000)
21 second.set_date('02.12.2019')
22
23 puts 'Первый билет'
24 puts "Цена: #{first.date}"
25 puts "Дата: #{first.price}"
26 puts 'Второй билет'
27 puts "Цена: #{second.date}"
28 puts "Дата: #{second.price}"
29
30 p first.instance_variables # [:@price, :@date]
```

Output:

Первый билет  
Цена: 31.10.2019  
Дата: 2000  
Второй билет  
Цена: 02.12.2019  
Дата: 3000  
[:@price, :@date]

# ПЕРЕМЕННЫЕ КЛАССА

**class\_variables** - метод для получения списка локальных переменных

```
1 ▾ class HelloWorld
2   @@counter = 0
3 ▾   def counter
4     @@counter
5   end
6 ▾   def set_counter(counter)
7     @@counter = counter
8   end
9 end
10
11 first = HelloWorld.new
12 first.set_counter 10
13 second = HelloWorld.new
14 puts second.counter # 10
15
16 p HelloWorld.class_variables # [:@@counter]
```

Output:

10

[:@@counter]

# ПРИСВАИВАНИЕ И СРАВНЕНИЕ

```
1 number = 1
2 print "number = ", number, "\n"
3 num = (number = 2)
4 print "num = ", num, ", number = ", number, "\n"
5 num = number = 3
6 print "num = ", num, ", number = ", number, "\n"
7
```

Output:

```
number = 1
num = 2, number = 2
num = 3, number = 3
```

```
first = Object.new
second = Object.new
copy_first = first
print "first: ", first, "\n"
print "second: ", second, "\n"
print "copy_first: ", copy_first, "\n"
p first==second
p first==copy_first
```

Output:

```
first: #<Object:0x0000561db6d2b840>
second: #<Object:0x0000561db6d2b818>
copy_first: #<Object:0x0000561db6d2b840>
false
true
```

# КЛОНИРОВАНИЕ

```
first = Object.new
clone_first = first.dup
print "first: ", first, "\n"
print "clone_first: ", clone_first, "\n"
p first==clone_first
```

Output:

```
first: #<Object:0x0000560a5ea9b898>
clone_first: #<Object:0x0000560a5ea9b6b8>
false
```

```
class Rainbow
  def set_colors(colors)
    @colors = colors
  end
  def colors
    @colors
  end
end

rainbow = Rainbow.new
colors = %i[red orange yellow green blue indigo violet]
rainbow.set_colors(colors)

copy = rainbow.dup

p copy.colors
p rainbow.object_id # 70315732997580
p copy.object_id # 70315732997540
p rainbow.colors.object_id # 70315732997560
p copy.colors.object_id # 70315732997560
```

Output:

```
[:red, :orange, :yellow, :green, :blue, :indigo, :violet]
60
80
100
100
```

# КОНСТАНТЫ

```
CONST = 12
puts CONST # 12
CONST = 15
puts CONST # 15
puts RUBY_VERSION # 2.6.0
```

Output:

```
12
15
2.7.0
```

```
test.rb:3: warning: already initialized constant CONST
test.rb:1: warning: previous definition of CONST was here
```

```
def hello
  COUNT = 1
end
```

```
hello
```

Output:

```
test.rb:2: dynamic constant assignment
  COUNT = 1
  ^~~~~~
```



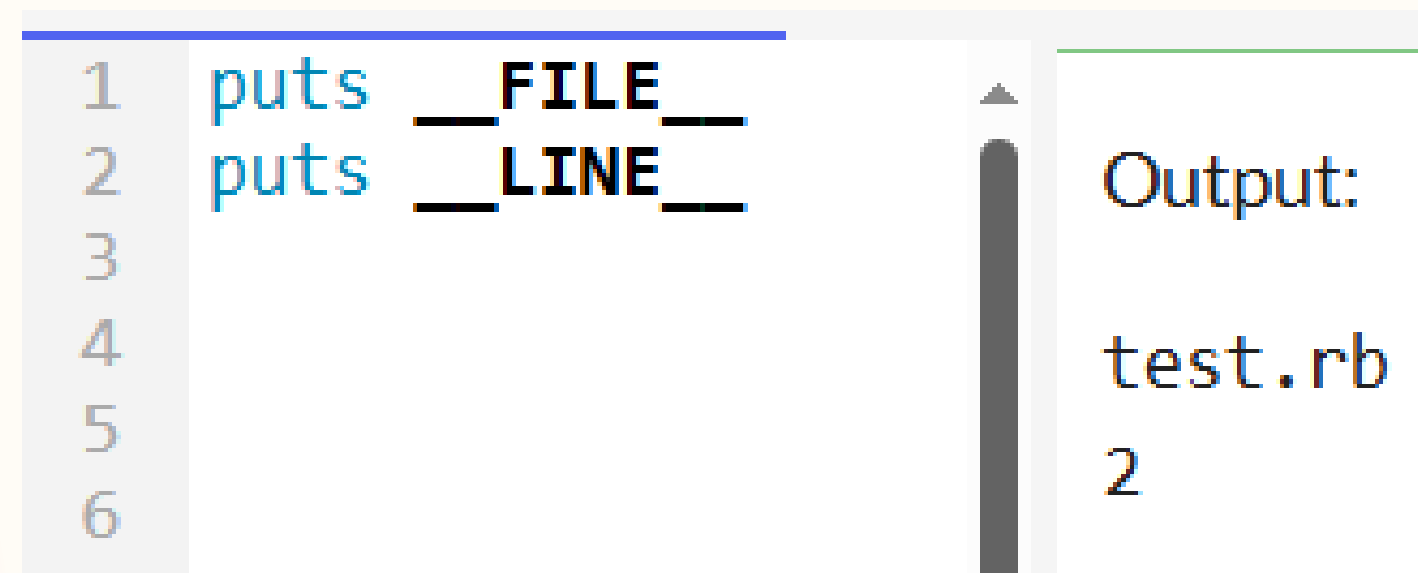
## ПРЕДОПРЕДЕЛЕННЫЕ КОНСТАНТЫ

Константа	Описание
RUBY_VERSION	Версия Ruby
RUBY_RELEASE_DATE	Строка с датой релиза Ruby
RUBY_PLATFORM	Строка с идентификатором операционной системы
ARGV	Аргументы, переданные программе
ENV	Переменные окружения
STDOUT	Поток стандартного вывода
STDIN	Поток стандартного ввода
STDERR	Поток ошибок
DATA	Содержимое Ruby-программы после ключевого слова <code>__END__</code>

## КЛЮЧЕВЫЕ СЛОВА

**`__LINE__`** - получение текущего номера строки в файле

**`__FILE__`** - возвращает имя файла, в которой  
расположена программа



```
1 puts __FILE__
2 puts __LINE__
3
4
5
6
```

Output:

test.rb  
2

## МЕТОД REQUIRE

файл **hello\_world.rb**

```
1 class HelloWorld
2   def greeting
3     puts 'Hello, world!'
4   end
5 end
6
7 test = 'test'
```

2

```
1 $LOAD_PATH << '.'
2 require 'hello_world'
3 hello = HelloWorld.new
4 hello.greeting
5 puts test
```

3

```
1 require_relative 'hello_world'
2 hello = HelloWorld.new
3 hello.greeting
4 puts test
```

1

```
require './hello_world'
hello = HelloWorld.new
hello.greeting
puts test
```

Output:

Hello, world!

test.rb:5:in `test': wrong number of arguments (given 0, expected 2..3) (ArgumentError)  
from test.rb:5:in `'

# ПОДКЛЮЧЕНИЕ КЛАССОВ И ГЕМОВ

## Подключение классов

```
1 require 'date'
2 p Date.new
3
4 require 'erb'
5 template = 'Текущее время <%= Time.now %>'
6 puts ERB.new(template).result
```

Output:

```
#<Date: -4712-01-01 ((0j,0s,0n),+0s,2299161j)>
Текущее время 2024-10-03 15:55:59 +0000
```

установка утилиты gem:

**\$ GEM INSTALL PRY**

## Подключение гемов

```
1 require 'pry'
2 class HelloWorld
3   def greeting
4     binding.pry
5     puts 'Hello, world!'
6   end
7 end
8 hello = HelloWorld.new
9 hello.greeting
```



# АРИФМЕТИЧЕСКИЕ ОПЕРАТОРЫ

Оператор	Описание
+	Сложение
-	Вычитание
*	Умножение
/	Деление
%	Остаток от деления
**	Возведение в степень

Сокращенная форма	Полная форма
var += 10	var = var + 10
var -= 10	var = var - 10
var *= 10	var = var * 10
var /= 10	var = var / 10
var %= 10	var = var % 10
var **= 10	var = var ** 10

<pre> 1 puts (5 + 3) * (6 + 7) / 2 2 puts 23 % 2 3 p 23.even? 4 p 23.odd? 5 6 fst, (f, s), thd = 'Hello', ['world', 'Ruby'], '!' 7 p fst # "Hello" 8 p f # "world" 9 p s # "Ruby" 10 p thd # "!" 11 12 fst, snd, thd = 'Hello, world!' 13 p fst # Hello, world! 14 p snd # nil 15 p thd # nil 16 17 fst, snd, thd = 'Hello', ['world', '!'] 18 p fst # "Hello" 19 p snd # ["world", "!"] 20 p thd # nil 21 fst, snd, thd = 'Hello', *['world', '!'] 22 p fst # "Hello" 23 p snd # "world" 24 p thd # "!" 25 26 fst, *snd = *['Hello', 'world', '!'] 27 p fst # "Hello" 28 p snd # ["world", "!"] 29 30 p *[1, 2, 3, 4, 5] 31 </pre>	<p>Output:</p> <pre> 52 1 false true "Hello" "world" "Ruby" "!" "Hello, world!" nil nil "Hello" ["world", "!"] nil "Hello" "world" "!" "Hello" ["world", "!"] 1 2 3 4 5 </pre>
---	--

# ОПЕРАТОРЫ СТРОК

Оператор умножения строки на число `puts 'Hello' * 3 # HelloHelloHello`

Оператор `[]` (оператор доступа к отдельным элементам строки)

Оператор `<<` (Оператор добавления подстроки к строке)

Оператор `#` (Оператор интерполяции)

Оператор `+` (Оператор Сложения строк)



# ФОРМАТИРОВАНИЕ СТРОК

Определитель	Описание
%a	Определитель вещественного числа для подстановки в строку в шестнадцатеричном формате
%A	Аналогичен %a, однако префикс 0x и экспонента p используют прописные, а не строчные буквы
%b	Определитель целого, которое выводится в виде двоичного числа. При использовании альтернативного синтаксиса %#b двоичное число предваряется префиксом 0b
%B	Аналогичен %b, однако при использовании альтернативного синтаксиса %#B в префиксе используется прописная буква 0B
%c	Спецификатор символа строки используется для подстановки в строку формата одного символа — например: 'a', 'w', '0', '\0'
%d	Спецификатор десятичного целого числа используется для подстановки целых чисел — например: 0, 100, -45. Допускается использование синонимов %i и %u
%e	Спецификатор числа в экспоненциальной нотации — например, число 1200 в этой нотации записывается как 1.2e+03, а 0.01, как 1e-02
%E	Аналогичен %e, однако для обозначения экспоненциальной нотации используется прописная буква E: 1.2E+03 и 1E-02

Определитель	Описание
%f	Спецификатор вещественного числа — например, 156.001
%g	Спецификатор десятичного числа с плавающей точкой, который ведет себя как %f, однако для чисел меньше $10^{-4}$ (0.00001) ведет себя как спецификатор экспоненциальной нотации %e
%G	Аналогичен %g, однако для обозначения экспоненциальной нотации используется прописная буква E
%o	Спецификатор для подстановки в строку формата восьмеричного числа без знака
%p	Вызывает для вставляемого значения var метод var.inspect и подставляет результат в строку
%s	Спецификатор для подстановки в строку формата строки
%x	Спецификатор для подстановки в строку формата шестнадцатеричного числа (строчные буквы для a, b, c, d, e, f)
%X	Спецификатор для подстановки в строку формата шестнадцатеричного числа (прописные буквы для A, B, C, D, E, F)
%%	Обозначение одиночного символа % в строке вывода

# ОПЕРАТОРЫ СРАВНЕНИЯ [1/2]

Оператор	Описание
>	Оператор «больше» возвращает <code>true</code> , если левый операнд больше правого
<	Оператор «меньше» возвращает <code>true</code> , если левый операнд меньше правого
>=	Оператор «больше равно» возвращает <code>true</code> , если левый операнд больше или равен правому операнду
<=	Оператор «меньше равно» возвращает <code>true</code> , если левый операнд меньше или равен правому операнду
==	Оператор равенства возвращает <code>true</code> , если сравниваемые операнды равны
!=	Оператор неравенства возвращает <code>true</code> , если сравниваемые операнды не равны
<=>	Возвращает <code>-1</code> , если левый операнд меньше правого, <code>0</code> — в случае, если операнды равны, и <code>1</code> , если левый операнд больше правого
==~	Возвращает <code>true</code> , если операнд соответствует регулярному выражению (см. главу 30)
!~	Возвращает <code>true</code> , если операнд не соответствует регулярному выражению (см. главу 30)
===	Оператор равенства, предназначенный для перегрузки в классах

```
p (1..10) === 5           # true
p (1..10) === 11          # false
p String == 'Hello, world!' # false
p String === 'Hello, world!' # true
p /\d+/ == '12345'         # false
p /\d+/ === '12345'        # true
```

```
> (1..10) == 5
=> true
> 5 == (1..10)
=> false
```

```
> (1..10) .==(5)
=> true
> 5.==(1..10)
=> false
```

## ОПЕРАТОРЫ СРАВНЕНИЯ [2/2]

```
fst = 4 / 3.0 - 1
snd = 1 / 3.0

p fst == snd # false
p (fst - snd).abs < Float::EPSILON # true
```

```
> 'Hello' < 'Hello, world!'
=> true
```

```
> 'a' > 'b'
=> false
> 'a' < 'b'
=> true
```

```
> 'a'.ord
=> 97
> 'b'.ord
=> 98
```

```
> 'Hello, ruby!' > 'Hello, world!'
=> false
> 'Hello, ruby!' < 'Hello, world!'
=> true
```



# ПОРАЗРЯДНЫЕ ОПЕРАТОРЫ

## [1/2]

Оператор	Описание
&	Поразрядное (побитовое) пересечение — И (AND)
	Поразрядное (побитовое) объединение — ИЛИ (OR)
^	Поразрядное ИСКЛЮЧАЮЩЕЕ ИЛИ (XOR)
~	Поразрядное отрицание (NOT)
<<	Сдвиг влево битового представления значения левого целочисленного операнда на количество разрядов, равное значению правого целочисленного операнда
>>	Сдвиг вправо битового представления значения левого целочисленного операнда на количество разрядов, равное значению правого целочисленного операнда

# ПОРАЗРЯДНЫЕ ОПЕРАТОРЫ

## [2/2]

Сокращенная запись	Полная запись
<code>number &amp;= var;</code>	<code>number = number &amp; var;</code>
<code>number  = var;</code>	<code>number = number   var;</code>
<code>number ^= var;</code>	<code>number = number ^ var;</code>
<code>number &lt;&lt;= var;</code>	<code>number = number &lt;&lt; var;</code>
<code>number &gt;&gt;= var;</code>	<code>number = number &gt;&gt; var;</code>

## СПЕЦИАЛЬНЫЕ ОПЕРАТОРЫ

"&." - оператор безопасного вывода

"defined?" - ключевое слово defined?

Метод	Описание
<code>local_variable_defined?</code>	Проверяет существование локальной переменной <code>hello</code>
<code>instance_variable_defined?</code>	Проверяет существование инстанс-переменной <code>@hello</code>
<code>class_variable_defined?</code>	Проверяет существование переменной класса <code>@@hello</code>
<code>const_defined?</code>	Проверяет существование константы, например, <code>CONST</code>



# ПОРЯДОК ОПЕРАТОРОВ

Оператор	Описание
! ~ +	Логическое отрицание !, поразрядное отрицание ~, унарный плюс + (например, +10)
**	Возведение в степень
-	Унарный минус - (например, -10)
* / %	Умножение *, деление /, взятие остатка %
+ -	Арифметические плюс + и минус -
>> <<	Поразрядные сдвиги вправо >> и влево <<
&	Поразрядное И
^	Поразрядное ИСКЛЮЧАЮЩЕЕ ИЛИ, поразрядное ИЛИ
<= < > >=	Операторы сравнения с более высоким приоритетом
<=> == === != =~ !~	Операторы сравнения с более низким приоритетом
&&	Логическое И
	Логическое ИЛИ
.. ...	Операторы диапазона
z ? y : z	Тернарный оператор
= %= { /= -= +=  = &= >>= <<= *= &&=   = **=	Операторы присваивания
not	Логическое отрицание
or and	Логические ИЛИ и И