Ruby

Установка: apt install ruby-full

rbenv -

Все объекты. Без аргументов функция не требует наличия скобок. Доступа к свойствам объекта нет, через методы. Динамическое типизирование.

```
1.class # Выводит класс объекта, в данном случае Fixnum
```

Типы данных

Целые числа

3.times {print "r"}	повтор действия 3 раза
1.upto(9) { |x| print x }	вывод 123456789

Строки. Одинарные кавычки - все символы, в двойных можно подставлять значения.

Многострочный текст

```
message = <<~TEXT
  This is a multi-line string
  that preserves formatting
  in a tidy way.
TEXT
```

Есть символы (Symbols) - строки, но фиксированные. Начинаются с :. Например :fucktheruby. Используются в ключах словарей для быстрого доступа.

Массивы (аналог списков python)

a[1] - обращение к 1 элементу массива.

```
a = [3, 2, 1]
a.each do |elt|
```

```
  print elt + 1
end
```

a.each	Для каждого элемента
a,map {}	Изменение массива по правилам. `b = a.map { |x| x*x } # Возведение в квадрат каждого элемента`
a.select	выбор элементов в соответствии с правилами. `b = a.select { |x| x%2==0 }`

Хэши Аналог словарей в python.

```
h = {
  :one => 1,
  :two => 2
}
h[:one]
h.each do |key, value|
 print "#{value} у ключа #{key} " # выведет 1 у ключа one 2 у ключа two
end

instrument_section = {
"cello" => "string",
"clarinet" => "woodwind",
"drum" => "percussion",
"oboe" => "woodwind",
"trumpet" => "brass",
"violin" => "string"
}

Блядь, если использовать такой синтаксис, то оно создаст ключ Symbol.
instrument_section = {
cello: "string"
}
x = instrument_section[:cello]
```

Методы

print puts	Вывод значений
ARGV[0]	Первый аргумент при вызове скрипта

Операторы

```
x = 1
x, y = 1, 2
x += 1
1..3 # набор значений от 1 до 3
1...3 # набор значений от 1 до 2
generation = case birthyear
              when 1946..1963: "Поколение 1"
              else nil
              end
# Регулярки
def areyoushure?
  while true
    print "Вы уверены"
    response = gets
    case response
      when /^[Yy]/
        return true
      when /^Nn/, /^$/
        return false
      end
  end
end

line = gets
if line =~ /Ruby|Rust/
puts "Programming language mentioned: #{line}"
end

line = gets
if line.match?(/Ruby|Rust/)
puts "Scripting language mentioned: #{line}"
end
```

```
line = gets
newline = line.sub(/Python/, 'Ruby') # replace first 'Python' with 'Ruby'
newerline = line.gsub(/Python/, 'Ruby') # replace every 'Python' with 'Ruby'
```

Условные операторы

```
today = Time.now
if today.saturday?
puts "Do chores around the house"
elsif today.sunday?
puts "Relax"
else
puts "Go to work"
end

puts "Danger, Will Robinson" if radiation > 3000
```

Блоки. Без комментариев. Являются аргументами для методов.

```
def call_block
puts "Start of method"
yield
yield
puts "End of method"
end
call_block { puts "In the block" }

Выведет:
Start of method
In the block
In the block
End of method
```

В | | передаются параметры блока. Например

```
def who_says_what
yield("Dave", "hello")
yield("Andy", "goodbye")
end
who_says_what { |person, phrase| puts "#{person} says #{phrase}" }
```

Похоже

Классы

Все классы расширяемы внутри приложения. Термин Instance = Object

.new Конструктор объекта.

Методы (функции) Если это вне класса, то глобальная функция. Возвращает последнее вычисляемое значение.

```
def square(x)
  x*x
end

def multireturn
  x, y = 1,2
  [x, y]
end
```

Если метода нет, вызывается специальный метод method_missing. По умолчанию исключение. Его можно переопределить.

```
class Greeting
  def method_missing(name, *args)
    "Привет из #{name}"
  end

  def respond_to_missing?(name, include_private = false)
    true
  end
end

g = Greeting.new
g.respond_to?(:hello)   # => true
puts g.hello            # => "Привет из hello"
```

Префиксы и постфиксы. Вроде упрощает язык, но походу хуета на уровне выебнуться. Отказались от одного, ввели другое.

Имена классов, модулей и констант с большой буквы. Локальные переменные, параметры методов и имена методов только с маленькой буквы. Глобальные переменные - префикс $. переменные объекта - @, переменные класса - @@

Если имя метода заканчивается на равно, то можно использовать синтаксис

```
# некий класс, в котором есть метод x= и нужно туда передать значение 1
o.x=(1) # как оно обычно вызывается
o.x = 1 # так тоже можно. Типа крутая фича. Закрыли свойства, зато сделали ЭТО
```

На знак вопроса - возвращается только булево значение

На восклицательный знак - метод меняет объект. Например метод .sort массива возвращает отсортированный массив, тогда как метод .sort! сортирует существующий объект.

Синглтоны - методы, добавляемые к существующему классу или единичному объекту. (Ебанутая херня, хотя позиционируется как ключевая).

```
def Math.square(x)
  x*x
end
```

Описание класса

Пример класса, создающего элементы с шагом.

```
class Sequence
  include Enumerable

  def initialize(from, to, delta)
    @from, @to, @delta = from, to, delta
  end

  def each
    x = @from
    while x <= @to
      yield x
      x += @delta
      end
```

```
  end

  def length
    return 0 if @from > @to
    Integer((@to - @from)/@delta) + 1)
  end

  alias size length # алиас на метод size

  def [](index) # переопределение метода получения доступа к элементам массива
    return nil if index < 0
    v = @from + index * @delta
    if v <= @to
      v
    else
      nil
    end
  end

  def *(factor) # переопределение операции умножения над объектом
    Sequence.new(@from*factor, @to*factor, @delta*factor)
  end
```

Модули

Набор функций.

```
module Sequence
  def self.fromtoby(from, to, delta)
    x = from
    while x <= by
      yield x
      x += delta
      end
  end
end
```

Revision #3
Created 31 October 2025 15:12:51 by Admin
Updated 31 October 2025 18:54:10 by Admin