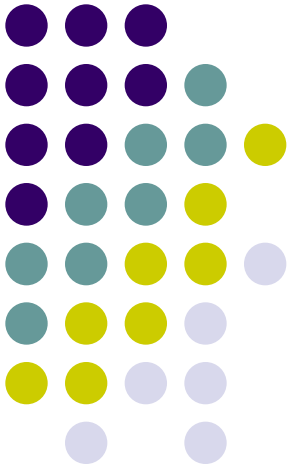
Язык Ruby

Потапова Вера

Группа 3371

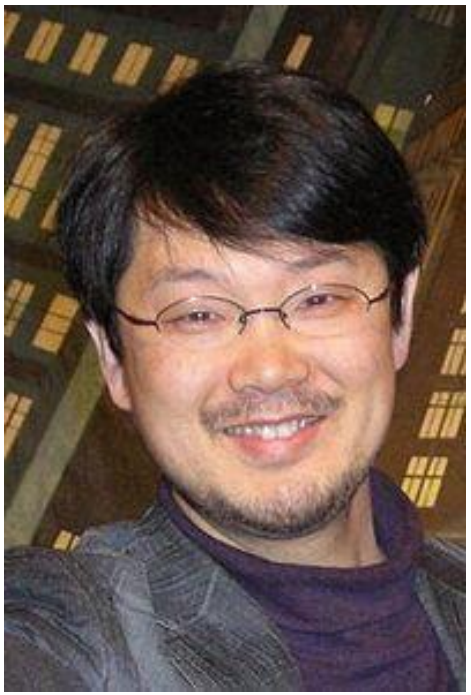


Ruby

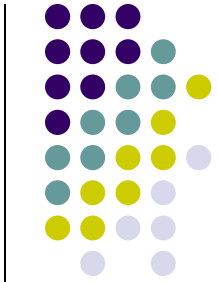
Создан Юкихиро Мацумото в 1995 г.

В основу положены элементы языков Perl, Python, Lisp, Smalltalk и др.

Основные реализации:
http://www.ruby-lang.org
http://jruby.org



Основные характеристики Ruby

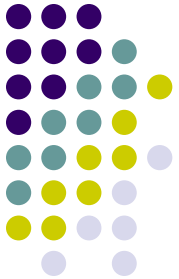


- Динамический язык
- Сквозная объектная модель
- Поддержка исключений
- Автоматическая сборка мусора
- Поддержка метапрограммирования (в т.ч. интроспекция, evaluate)
- Поддержка элементов функционального программирования (блоки/замыкания, λ-выражения)
- Встроенная поддержка регулярных выражений

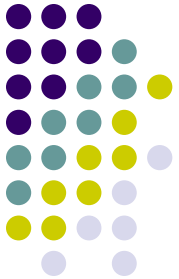
Семантика

```
a = "abcdefg"
b = a
b       #=> "abcdefg"
a[4] = 'R'
b       #=> "abcRefg"
```

При изменении значения переменной ***a***, неявно изменилось и значение ***b***, так как они содержат ссылку на один объект. То есть механизм присваивания действует одинаково для всех объектов, в отличие от языков типа C, Object Pascal,
где присваивание может означать как копирование значения, так и копирование ссылки на значение.



Семантика



•Не поддерживает множественное наследование
Поддерживает:
•Процедурный стиль
•Объектно-ориентированный
•Функциональный

Основной синтаксис

```
class SpaceMan < Dreamer
  attr_reader :name
  attr_accessor :rocket
  def initialize(name)
      @name = name
  end
  def where_i_am?
    @current_place.to_s
  end
end
```

Классы. Объявление и наследование.

class – ключевое слово для объявление класса. Символ **<** используется для наследования. Объявление в класса заканчивается ключевым словом **end**, как любой другой блок кода.

Множественное наследование запрещено.

Доступ к родительскому классу осуществляется с помощью ключевого слова **super**.

Основной синтаксис

```
class SpaceMan < Dreamer
  attr_reader :name
  attr_accessor :rocket
  def initialize(name)
      @name = name
  end
  def where_i_am?
    @current_place.to_s
  end
end
```

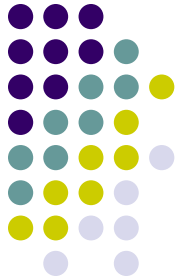
Конструктор, методы и члены класса.

Объявление метода в классе начинается с ключевого слова `def`, затем следует имя метода, и параметры.

Метод-конструктор класса должен называться `initialize`.

Любая переменная, имя которой начинающется с одного символа `@` - член класса. С двух символов `@` - статический член класса.

Основной синтаксис

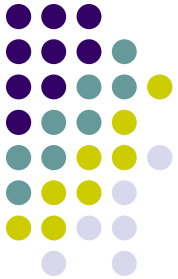


```
class SpaceMan < Dreamer
  attr_reader :name
  attr_accessor :rocket
  def initialize(name)
       @name = name
  end
  def where_i_am?
    @current_place.to_s
  end
end
```

Вызов метода.

Метод вызывается, как и в большинстве языков, через точку, скобки с перечислением параметров после вызова можно опустить, если это не вызывает недоразумений.

Основной синтаксис



```
class SpaceMan < Dreamer
  attr_reader :name
  attr_accessor :rocket
  def initialize(name)
       @name = name
  end
  def where_i_am?
    @current_place.to_s
  end
end
```

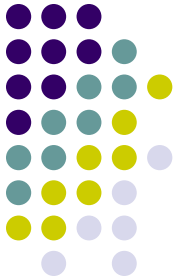
Ruby-символы.

Идентификатор, и символ двоеточия в начале – это специальный объект в ruby, ruby-символ.

В большинстве случаев можно считать, что это ссылка на строку. Точнее, что-то, что представляет строку или имя.

Два ruby-символа с одинаковым именем – это один и тот же объект.

Основной синтаксис



```
class SpaceMan < Dreamer
  attr_reader :name
  attr_accessor :rocket
  def initialize(name)
       @name = name
  end
  def where_i_am?
    @current_place.to_s
  end
end
```

Весь код в объявлении класса начинает выполнятся как только интерпретатор видит его.

attr_reader и **attr_accessor** – вызов метода класса Module, добавляющего в класс методы для чтения и доступа к членам класса **@name** и **@rocket** соответственно.

Основной синтаксис

```
class Array
      def from_place(place)
    self.select do |s|
      s.where_i_am == place
    end
  end
end

flyers << mike
flyers.from_place('Mars').each do |s|
  print s
end
```

Классы в ruby открыты и свободны для дополнения.

Здесь в стандартный класс **Array** добавляется собственный метод, который будет виден только на время действия этого кода.

Основной синтаксис

```
class Array
      def from_place(place)
    self.select do |s|
      s.where_i_am == place
    end
  end
end

flyers << mike
flyers.from_place('Mars').each do |s|
  print s
end
```

Ruby-блоки.

Ruby-блоки – специальная конструкция языка. Код, объявленный внутри **do**..**end** выполняется внутри метода, с которым используется блок.

select и **each** – стандартные методы для выбора по условию и перебора всей коллекции соответственно.

Основной синтаксис

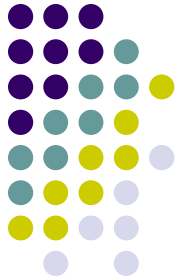
```
class Rocket
      def travel_to(place)
      planet = fly_to place
      yield planet if block_given?
      fly_back
    end
  end
end

mike = SpaceMan.new('mike')
mike.rocket = Rocket.new
mike.rocket.travel_to(:mars).do
|planet|
  mike.conquer! planet
end
```

Код, написанный внутри блока, выполняется внутри метода, принимающего его. Этот метод рассматривает блок, как функцию, в которую можно передать какие-то параметры.

Вызов блока и передача параметров происходит с помощью ключевого слова **yield**.

Разделители выражений



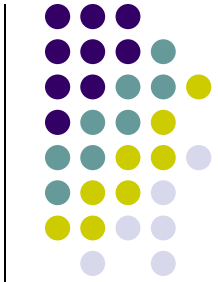
#комментарий

```
a=1; b=2    #конец выражения, «;» можно не ставить

c=a.to_f+ #выражение продолжается на следующей строке
      b.to_f

d=a.to_f\ #явный перенос на следующую строку
      -b.to_f
```

Блоки и замыкания



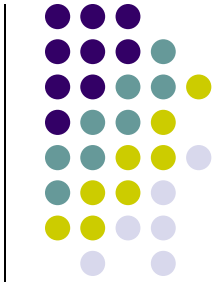
Блок – часть кода с собственным контекстом (т.е. изолированными локальными переменными).

Блок как правило объявляется в одном контексте, а исполняется в другом.

Блок может взаимодействовать с тем контекстом, в котором объявлен, т.е. является замыканием.

Аналогом блока в Java является анонимный класс.

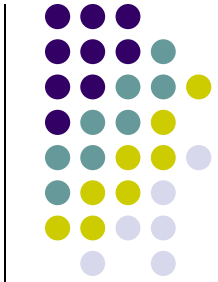
Применение блоков: функционал



```
def nTimes (n)
  for i in (0...n)
    yield                    #вызов блока
  end
end

nTimes(5) {puts “Hello, world!”}
```

Определение и вызов блока



```
def func (&block)                #функция, вызывающая блок
  yield(<params>)                #вызов блока с параметрами
  block.call(<params>)           #--//--
end
```

#передача блока в качестве параметра функции

```
func(scalar_params){|<params>|<code>}
```

#определение блока, как объекта первого класса

```
block=Proc.new {|<params>| code}
block=lambda {|<params>| code}
```

#вызов самостоятельно определеннорго блока

```
block.call(<params>)
```

Прокси-классы

```ruby
module M1
  def method
    'M1#method'
  end
end

module M2
  def method
    'M2#method'
  end
end

class C
  include M1
  include M2
end

# при вызове #method будет найден нижний модуль в цепочке наследования
# а так как M2 был включен последним, он будет стоять прямо над C
C.new.method # => "M2#method"

# также можно посмотреть всю цепочку с помощью метода Module#ancestors
C.ancestors # => [C, M2, M1, Object, Kernel, BasicObject]
```

При включении класса руби создает анонимный класс, и помещает его в цепочку наследования прямо над включившим этот модуль классом. Такие анонимные классы часто называют прокси-классами. Соответственно, при поиске метода, определенного в модуле, он будет найден в анонимном прокси-классе, и все произойдет так, как если бы метод был определен в одном из настоящих классов.

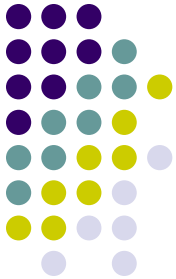
Синглтон-классы

```ruby
name = 'Vasya'
def name.spacify
  self.split('').join(' ')
end

name.spacify    # => "V a s y a"
# другие объекты класса String этот метод не получают
"Petya".spacify # => NoMethodError: undefined method `spacify' for "Petya":String
```

В ruby есть специальная форма определения метода: def object.method_name. Созданный таким образом метод называется синглтон-методом - он определен только для этого конкретного объекта.

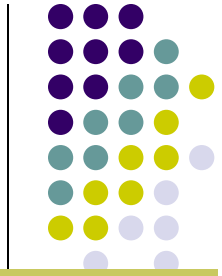
Self и текущий класс



В каждой точке кода определены так называемые “текущий объект” и “текущий класс”.
Текущий объект - это объект, который доступен через псевдопеременную self; к нему адресуются все вызовы методов без указания получателя и в нем ищутся все запрошенные инстанс-переменные.
Текущий класс - это тот класс, инстанс-методом которого становится определенный в этом месте метод.

```
1  class MyClass
2    # здесь текущий класс - MyClass
3    def my_method
4      # этот метод станет инстанс-методом класса MyClass
5    end
6  end
```

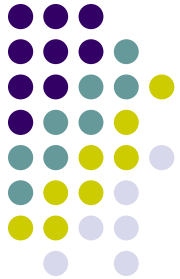
Верхний уровень



До того, как мы входим в определение какого-либо класса, self указывает на main, а текущим классом является Object. Это объясняет тот факт, что методы, определяемые на верхнем уровне, вызываются из любого участка кода, причем без получателя (такой метод будет приватным, т.е. его нельзя вызывать с явным получателем; а self указывает на объект класса Object либо его потомка, поэтому метод будет доступен).

```
1  def function
2    "I am #{self}, of class #{self.class}"
3  end
4
5  function # => "I am main, of class Object"
6
7  Object.private_instance_methods.grep(/function/) # => [:function]
```

Сравнение Ruby с Java



•Ruby - интерпретируемый язык;

•В Ruby все является объектом (в Java есть типы int и Integer, что создает определенные неудобства);

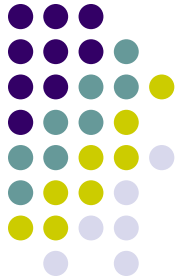
•Переменные в Ruby не являются статически типизированными и не требуют объявления;

•Модули (modules) в Ruby позволяют с помощью <<миксинов>> (mixins) конструировать подобие интерфейсов (interfaces) языка Java, допуская при этом в них реализацию методов.

Заключение

Этот язык, несомненно, является одним из лучших в качестве первого языка программирования. Быстрый цикл разработки (редактирование - запуск - редактирование), использование интерпретатора, изначальная объектно-ориентированность, нетипизированные переменные, которые не требуют объявления, - все это позволяет учащимся сконцентрировать свое внимание на общих принципах программирования.

Не менее важны мультиплатформенность Ruby и его принадлежность к миру свободно распространяемого ПО. Еще один весомый аргумент в его пользу - возможность практического использования языка в самых разных областях, что не позволит впоследствии профессионалу, который вырастет из новичка, пожалеть о напрасно потраченном времени.



Спасибо за внимание!