Лучший друг программиста

автор: Гусева Е. Ю.

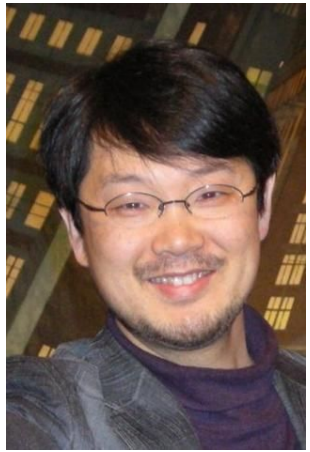
Ruby - это...

Динамический язык программирования с открытым исходным кодом с упором на простоту и продуктивность. Он обладает элегантным синтаксисом, который приятно читать и писать.

“Человек создан для творчества, и я всегда знал, что люблю творить. Увы, я обделен талантом художника или музыканта. Зато умею писать программы.”

“Я хочу, чтобы компьютер был моим слугой, а не господином, поэтому я должен уметь быстро и эффективно объяснить ему, что делать.”

Юкихиро Мацумото



Немного истории...

Создатель Ruby — Юкихиро Мацумото — начал разрабатывать Ruby 24 февраля 1993 года, и в 1995 году Ruby вышел в свет.

Название навеяно языком Perl, многие особенности синтаксиса и семантики из которого заимствованы в Ruby: англ. *pearl* — «жемчужина», *ruby* — «рубин».

Мацумото, фанат объектно-ориентированного программирования, мечтал о языке, более мощном, чем Perl, и более объектно-ориентированном, чем Python. **Основное назначение** Ruby — создание простых и в то же время понятных программ, где важна не скорость работы программы, а малое время разработки, понятность и простота синтаксиса.

Основные свойства

Ру́би — интерпретируемый язык программирования высокого уровня. Обладает независимой от операционной системы реализацией многопоточности, строгой динамической типизацией, «сборщиком мусора» и многими другими возможностями, поддерживающими много разных парадигм программирования, прежде всего классово-объектную.

Основные свойства

Интерпретируемый язык:

- Возможность прямых системных вызовов.
- Мощная поддержка операций со строками и правилами (регулярными выражениями).
- Мгновенное проявление изменений во время разработки.
- Отсутствие стадии компиляции.

Простое и быстрое программирование:

- Не надо объявлять переменные.
- Переменные динамически типизированы.
- Простой и последовательный синтаксис.
- Автоматическое управление оперативной памятью.

Основные свойства

Объектно-ориентированное программирование:

- Всё есть объект. Даже имя класса есть экземпляр класса Class.
- Классы, методы, наследование, полиморфизм, инкапсуляция и так далее.
- Методы-одиночки.
- Примеси при помощи модулей (возможность расширить класс без наследования).
- Итераторы и замыкания.
- Широкие возможности метапрограммирования.

Основные свойства

Удобства:

- Неограниченный диапазон значений целых чисел.
- Модель обработки исключений.
- Все операторы возвращают значения, даже управляющие структуры.
- Динамическая загрузка.
- Механизм перехвата исключений.
- Поддержка потоков; как собственных, так и систем семейства UNIX.

Недостатки:

- Неуправляемость некоторых процессов (таких, как выделение памяти), невозможность задания низкоуровневых структур данных или подпрограмм;
- Невозможность компиляции и сопутствующей ей оптимизации программы;
- Открытость исходного кода даже в готовой программе (есть средство упаковки исходного кода в .exe-файл под Windows);
- Следствие двух первых недостатков — весьма низкая скорость запуска и выполнения программ.

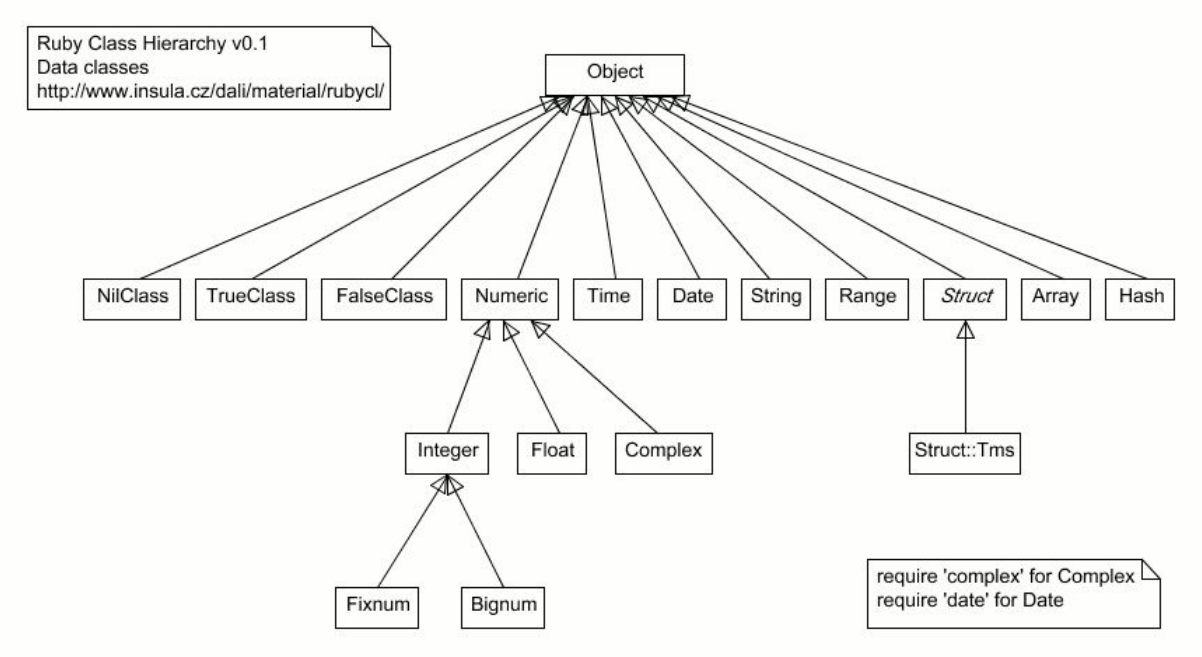
Базовые типы данных

Данные любого типа в Ruby суть объекты тех или иных классов. Самые используемые встроенные типы данных:

- Fixnum (целые числа, меньшие 2^{30}),
- Bignum (целые числа, большие 2^{30}),
- Float (числа с плавающей запятой),
- Array (массивы),
- String (строки),
- Hash (ассоциативные массивы)

Естественно, что это только базовые типы, но их вполне хватает для широкого спектра задач.

Все абстрактные типы данных



Логический тип

Логический (булевый) тип — это вариация на тему «да» или «нет». В Ruby он представлен двумя предопределёнными переменными true («истина» или «да») и false («ложь» или «нет»). Появляется логический тип в результате логических операций или вызова логических методов.

Чаще всего логический тип возникает как результат сравнения.

Логические операции		
Название операции	**Символ операции**	**Литерное обозначение**
Или	||	or
И	&&	and
Не	!	not
Исключающее или	^	xor

Логический тип

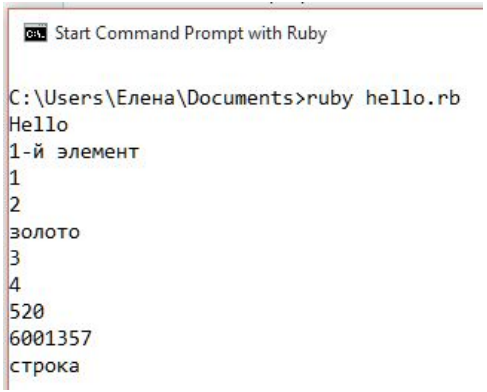
- Традиционно имена логических методов заканчиваются на ? (знак вопроса).
- В качестве false может выступать nil, а в качестве true — любой объект.
- nil — это символ пустоты.

Методы сравнения	
Название метода	**Символ**
Равно	==
Не равно	!=
Меньше	<
Больше	>
Меньше или равно	<=
Больше или равно	>=

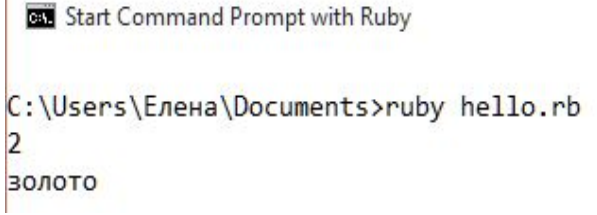
Массивы

Разработчики Ruby решили не реализовывать особых классов для динамических массивов, списков, стеков и тому подобного. Они все это реализовали в массивах — структурах данных типа (или класса — в Ruby всё равно) Array. Сделано это путём добавления специальных методов; например, методы .push и .pop для стека.

```
array = ["1-й элемент", 1, [2, "золото"],
    [3, 4, 520, 6001357, "строка"]]

puts array
```




```
puts array[2]
```



Массивы

Особенности массивов в Ruby:

- Нет ограничений (это общий принцип языка). Массивы могут быть сколь угодно длинными.
- Динамичность: размер массива легко меняется.
- Гетерогенность: один массив может хранить данные разных типов.
- Библиотека итераторов на каждый случай жизни. Эта возможность позволяет не использовать циклы для обработки данных в массивах, а, следовательно, избегать множества ошибок, связанных с неосторожным обращением с циклами. Итераторы реализуются на высочайшем уровне.
- Много других методов. Все элементарные задачи для массивов решаются вызовом нужного метода.

Строки

Стро́ки — это ряды букв и других символов. В Ruby стро́ки используют наработки языка Perl. Вот небольшой список их возможностей:

- Нет ограничений. Длина строки́ может достигать поистине фантастических размеров.
- Динамичность. Стро́ки можно расширять или уменьшать (для этого есть методы + и [ ]).
- Любой объект преобразуется в строку (методы .inspect и .to_s есть у любого объекта).
- Строка обладает обширной библиотекой методов, которые работают с правилами (это методы .gsub, .match, .scan, .split).
- Можно вставлять произвольные переменные заключив их между #{ и }. Действует для строк начинающихся и заканчивающихся ". После выполнения код заместится результатом.

Стро́ки начинаются и заканчиваются `"` (программистскими кавычками) или `'` (машинописным апострофом).

Стро́ки подобны массивам символов, поэтому их часто преобразуют к массивам, чтобы использовать богатый набор методов, а потом результат делают строкой.

Ассоциативные массивы

Ассоциативные массивы подобны массивам упорядоченных пар. Работают они подобно словарям: стрелка => показывает связь каждой сущности с какой-то другой.

```
array = {"Агент" => "007", 03 => "Скорая"}
puts array["Агент"], array[03], array[3]
```

```
Start Command Prompt with Ruby
C:\Users\Елена\Documents>ruby hello.rb
007
Скорая
Скорая

C:\Users\Елена\Documents>
```

Ассоциативные массивы оставляют возможность хранения данных разного типа только в ассоциативном виде.

Диапазоны значений

Чтобы было удобней получать подмассив или подстроку, был введён простенький тип данных — диапазон (класс Range). Диапазон формируется тремя элементами: начало, конец и тип протяжённости (символ .. или ...). Начало и конец должны быть одного типа данных (одного класса) и быть *перечислимыми*, что значит, иметь метод .succ (*succedent* — «последующему»).

Пример диапазонов:

1..100

1...100 # то же, что и 1..99

Методом `to_a` очень удобно создавать из диапазона массив, содержащий упорядоченные элементы данного диапазона.

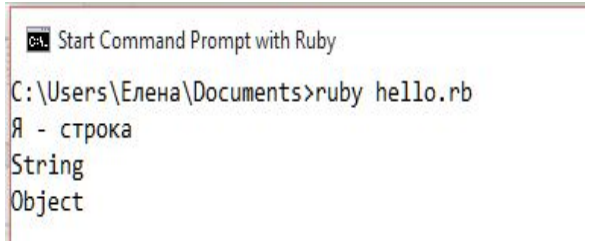
```
p (1..10).to_a
```

```
Start Command Prompt with Ruby
C:\Users\Елена\Documents>ruby hello.rb
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

C:\Users\Елена\Documents>
```

Классы и объекты

Самодельные и неабстрактные составные типы данных называются классами. Вообще, в Ruby *всё* в конечном счёте принадлежит классу Object.

```
puts str = "Я - строка"
puts str.class
puts str.class.superclass
```



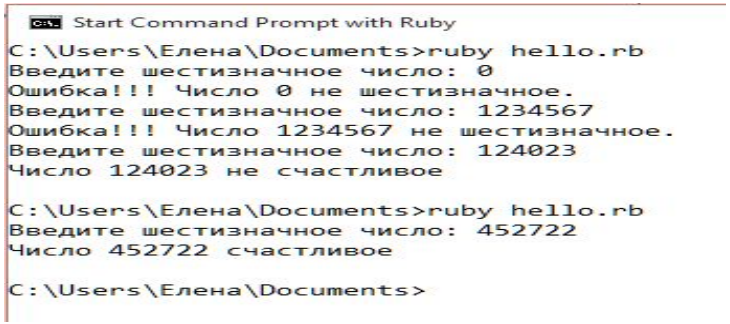
Классы можно определять и создавать по ним объекты. Стоит отметить интересную особенность языка - описание класса здесь это не просто некая описательная конструкция, не имеющая типа (как, например, в C++), а это объект (!) класса Class, дополненный пользовательскими методами и полями. Внутри класса может быть много всего интересного, и у него может быть фамильное дерево, то есть классы Ruby поддерживают наследование. Однако, множественное наследованиев Ruby не разрешается. И ещё много всего интересного можно сделать с классами и объектами.

Примеры

Числа. Пример 1

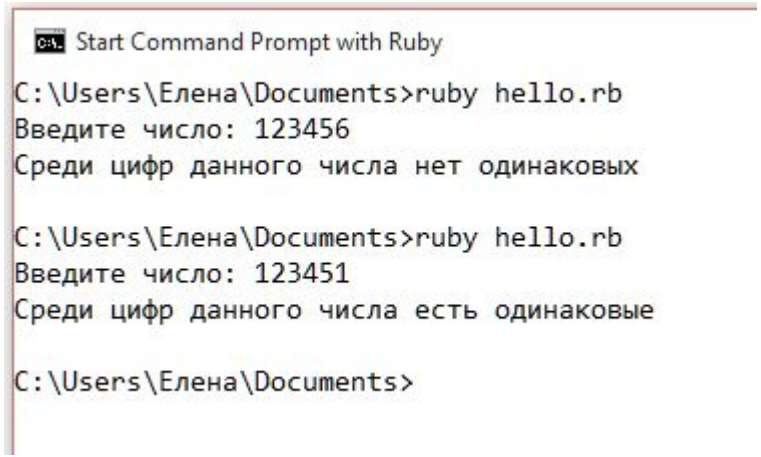
```
=begin
Дано целое шестизначное число.
Необходимо определить, является ли оно счастливым.
=end
l = 0
while l != 6 do
        print("Введите шестизначное число: ")
        ch = gets.chop
        l  = ch.length
        if l != 6 then
                puts "Ошибка!!! Число #{ch.to_i} не
шестизначное."
        end
end
```

```
ml = ch[0].to_i + ch[1].to_i + ch[2].to_i
mr = ch[3].to_i + ch[4].to_i + ch[5].to_i
if ml == mr then
  puts "Число #{ch.to_i} счастливое"
else
  puts "Число #{ch.to_i} не счастливое"
end
```



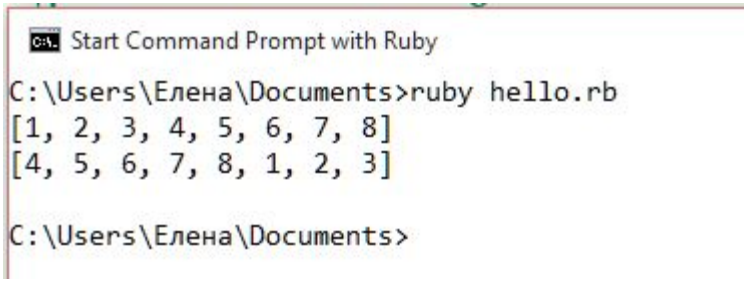
Числа. Пример 2

```
=begin
Дано число.
Необходимо определить есть ли среди цифр этого
числа хотя бы две одинаковых.
=end

print("Введите число: ")
ch = gets.chop
m = ch.to_s.split(//)
puts (m.size == m.uniq.size)? "Среди цифр данного
числа нет одинаковых" : "Среди цифр данного
числа есть одинаковые"
```



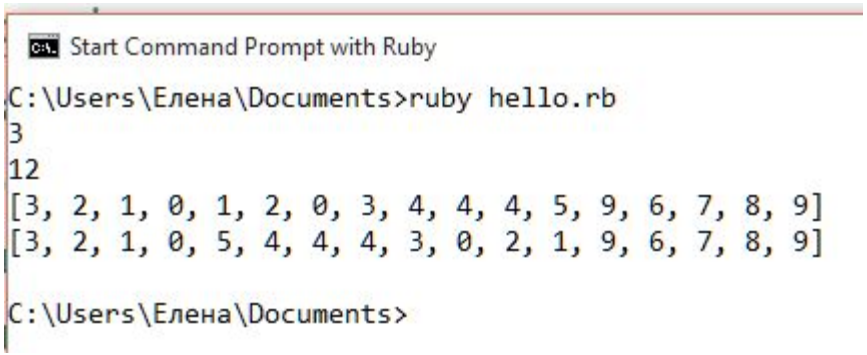
Массивы. Пример 1

```
=begin
Дан целочисленный массив.
Необходимо осуществить циклический сдвиг
элементов массива влево на три позиции.
=end

a = [1, 2, 3, 4, 5, 6, 7, 8]
p a
p a[3...a.size] + a[0..2]
```



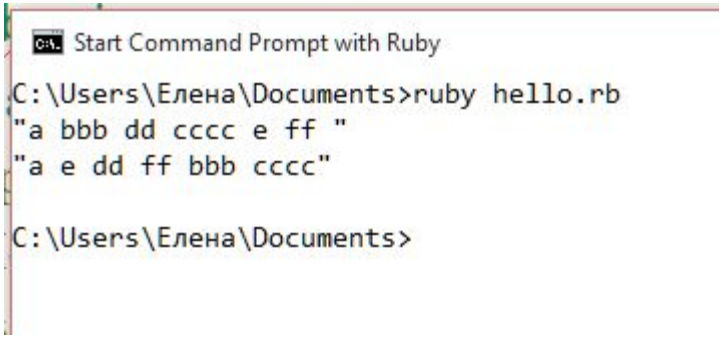
Массивы. Пример 2

```
=begin
Дан целочисленный массив.
Необходимо переставить в обратном
порядке элементы массива,
расположенные между его минимальным и
максимальным элементами.
=end

a = [3,2,1,0,1,2,0,3,4,4,4,5,9,6,7,8,9]
min, max = a.index(a.min), a.index(a.max)
min,max = max,min if min > max
p min,max,a
a [min+1...max]=a[min+1...max].reverse
p a
```



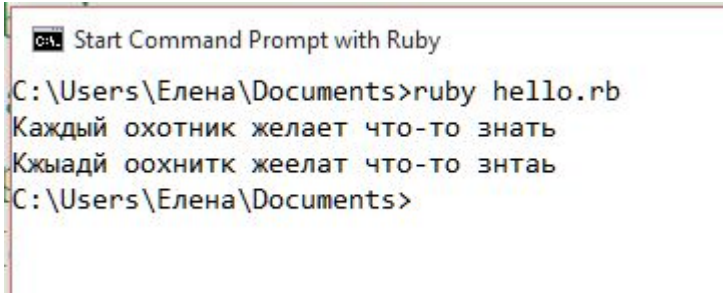
Строки. Пример 1

```
=begin
Дана строка в которой записаны слова через пробел.
Необходимо упорядочить слова по количеству букв в каждом слове.
=end

s="a bbb dd cccc e ff "
p s
p s.split(/ /).sort_by{|i| i.size}.join(" ")
```



Строки. Пример 2

```
=begin
Дана строка в которой записаны слова через пробел.
Необходимо перемешать в каждом слове все символы в случайном порядке кроме первого и последнего.
=end

str = "Каждый охотник желает что-то знать"
print str, "\n"
print str.split(" ").map{ |w| w[0] + w.split(//)[1..-2].shuffle.join + w[-1] } * " "
```



Интересности

Интересные моменты

- Многострочные комментарии помещаются между словами `=begin` и `=end`
- Переменная может иметь имя не только латинское, но и русское. Для этого, правда, требуется, чтобы весь текст программы был написан в кодировке UTF-8, а интерпретатор запускался с параметром `-KU`.
- Переменные *указывают* на объект.
- В масивах имеется не только, привычная нам, нумерация с положительной индексацией, но и с отрицательной.

$$["a"\ {}^{0}_{-4},\ "b"\ {}^{1}_{-3},\ "c"\ {}^{2}_{-2},\ "d"\ {}^{3}_{-1}]$$

```
array = ["a", "b", "c", "d"]
puts array[0] #1ый элемент массива, нумерация начинается с 0
puts array[-1] #1ый с конца элемент массива
```

```
Start Command Prompt with Ruby
C:\Users\Елена\Documents>ruby hello.rb
a
d
```

Идеология

Почему скорость написания программы важнее скорости ее выполнения?

Ответ заключается в следующем вопросе: вам платят за результат, который достигнут в срок или за качество, которое достигнуто с опозданием?

Печально, но программистам (как и переводчикам) платят за скорость написания кода, а не за его качество и скорость выполнения. Если скорость выполнения программы вполне приемлема (не идеальна, а приемлема), то, скорее всего, заказчика это устроит. Если заказчику необходим быстро работающий код, то он явно укажет это в задании, но даже в этом случае, сначала будет написана работоспособная программа и лишь потом она будет оптимизироваться по быстродействию. Иначе, программа может быть не написана вообще!

Вот именно поэтому код стоит писать быстро. И только, если он будет неприемлемо медленный, его нужно будет переделать (оптимизировать), и, возможно, делать это будете не вы, а программисты, которые специализируются на подобного рода задачах. Есть над чем задуматься?

Спасибо за внимание.