Игорь Борисов

Ruby. Уровень 1
Основы программирования

Кто хочет, тот ищет возможность, кто не хочет — ищет причину

---

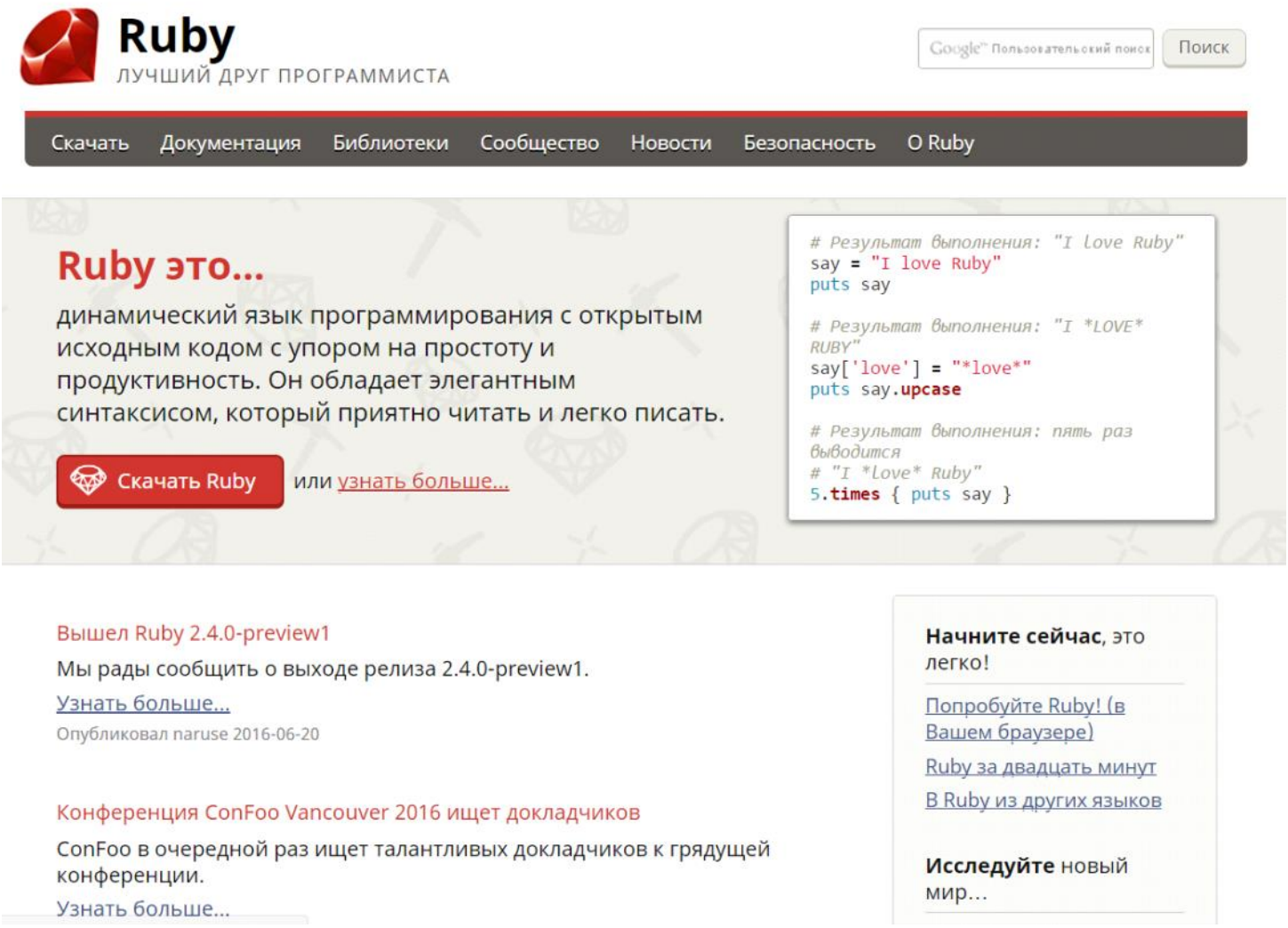
http://igor-borisov.ru

Темы курса

- Основные принцпы программирования
- Управление кодом
- Коллекции и основные структуры данных
- Методы
- Объектно-ориентированное программирование
- Практический Ruby
- Регулярные выражения
- Алгоритмы

Где живёт Ruby?

https://www.ruby-lang.org/ru



Установка Ruby

https://rubyinstaller.org/

http://railsinstaller.org/ru-RU

Использование консоли

- Уточняем версию Ruby

```
$ ruby -v
ruby 2.4.3p205 (2017-12-14 revision 61247) [x64-mingw32]
```

- Используем интерактивный Ruby

```
$ irb
irb(main):001:0>
```

Модуль 1

Ruby. Уровень 1
Основные принципы программирования

Темы модуля

- Базовые понятия программирования
- Базовые классы Ruby
- Методы объектов
- Операции над объектами
- Алгоритм написания программы

Языки программирования

- Синтаксис
- Семантика
- Реализация
  - **Интерпретация**
- Парадигма
  - **Объектно-ориентированная**
- Типы данных
  - **Динамическая типизация**

Базовые понятия программирования

- Выражения
- Данные
- Объекты
- Операторы
- Сообщения
- Типы
- Классы
- Переменные
- Константы
- Комментарии

Иерархия классов Ruby

- Object
  - Numeric
    - Integer
    - Float
  - String
  - TrueClass
  - FalseClass
  - NilClass

Класс Numeric

- +
  - +1
  - Унарный плюс. Возвращает число.
-
  - -1
  - Унарный минус. Возвращает число с противоположным знаком.
- <=>
  - 1 <=> 1 # 0
    1 <=> 2 # nil
  - Возвращает 0, если число слева равно числу справа, в противном случае – nil.
- abs
  - 1.abs # 1
    -1.abs # 1
  - Возвращает абсолютное значение числа.
- ceil
  - 1.2.ceil # 2
  - Округление вверх. Возвращает целое число.
- div
  - 5.div(2) # 2
  - Возвращает целое число результата деления.
- fdiv
  - 5.fdiv(2) # 2.5
  - Возвращает результата деления в виде вещественного числа.
- eql?
  - 1.eql?(1) # true
    1.eql?(1.0) # false
  - Возвращает true, если числа одного и того же типа и имеют одинаковые значения.
- floor
  - 1.7.floor # 1
  - Округление вниз. Возвращает целое число.
- integer?
  - 5.integer? # true
    5.3.integer? # false
  - Возвращает true, если число является целым числом.
- nonzero?
  - 5.nonzero? # 5
    0.nonzero? # nil
  - Возвращает число, если число не является 0, иначе - возвращает nil.
- round
  - 2.3.round # 2
    2.6.round # 3
  - Округляет число до ближайшего целого.
- to_int
  - 2.3.to_int # 2

    - Преобразование в целое число.
- truncate
    - 5.2.truncate # 5
    - Возвращает целую часть числа.
- zero?
    - 4.zero? # false
      0.zero? # true
    - Возвращает **true**, если число 0, иначе - **false**

Класс Integer < Numeric

- %
  - 5 % 2 # 1
    4 % 2 # 0
  - Возвращает целочисленный остаток от деления.
- *
  - 5 * 3 # 15
  - Возвращает результат умножения двух чисел.
- **
  - 2 ** 3 # 8
  - Возвращает результат возведения первого числа в степень другого.
- +
  - 2 + 3 # 5
  - Возвращает результат сложения двух чисел.
- -
  - 5 - 3 # 2
  - Возвращает результат вычитания двух чисел.
- /
  - 5 / 2 # 2.5
    4 / 2 # 2
  - Возвращает результат деления двух чисел.
- <
  - 5 < 6 # true
    5 < 3 # false
  - Возвращает **true**, если значение первого числа меньше, чем значение второго числа, иначе - **false**.
- <=
  - 5 <= 6 # true
    5 <= 5 # true
  - Возвращает **true**, если значение первого числа меньше либо равно значению второго числа, иначе - **false**.
- <=>
  - 5 <=> 6 # -1
    5 <=> 5 # 0
    5 <=> 4 # 1
  - Возвращает -1, 0 или 1, если значение первого числа , равно или больше значения второго числа.
- ==
  - 5 == 5 # true
    5 == 3 # false
  - Возвращает **true**, если значение чисел равны, иначе - **false**.
- >
  - 5 > 6 # false
    5 > 3 # true
  - Возвращает **true**, если значение первого числа больше, чем значение второго числа, иначе - **false**.
- >=

    - `5 >= 6 # false`
      `5 >= 5 # true`
    - Возвращает **true**, если значение первого числа больше либо равно значению второго числа, иначе - **false**.
- to_f
    - `5.to_f # 5.0`
    - Преобразует целое число в вещественное.
- to_s(base=10)
    - `216.to_s # "216"`
      `216.to_s(8) # "330"`
      `216.to_s(2) # "11011000"`
    - Преобразует целое число в строку по основанию системы счисления.
- chr
    - `65.chr # "A"`
    - Возвращает строку, состоящую из ASCII-символа с кодом равным числу.
- gcd
    - `54.gcd(126) # 18`
    - Возвращает наибольший общий делитель двух чисел.
- next
    - `5.next # 6`
    - Возвращает целое число увеличенное на 1.
- pred
    - `5.pred # 4`
    - Возвращает целое число уменьшенное на 1.

Класс Float < Integer

- **%**
  - `5.0 % 2.0 # 1.0`
    `4.0 % 2.0 # 0.0`
  - Возвращает целочисленный остаток от деления.
- *****
  - `5.0 * 3.0 # 15.0`
  - Возвращает результат умножения двух чисел.
- ******
  - `2.0 ** 3 # 8.0`
  - Возвращает результат возведения первого числа в степень другого.
- **+**
  - `2.0 + 3.0 # 5.0`
  - Возвращает результат сложения двух чисел.
- **-**
  - `5.0 - 3.0 # 2.0`
  - Возвращает результат вычитания двух чисел.
- **/**
  - `5.0 / 2.0 # 2.5`
    `4.0 / 2.0 # 2.0`
  - Возвращает результат деления двух чисел.
- **<**
  - `5.0 < 6.0 # true`
    `5.0 < 3.0 # false`
  - Возвращает **true**, если значение первого числа меньше, чем значение второго числа, иначе - **false**.
- **<=**
  - `5.0 <= 6.0 # true`
    `5.0 <= 5.0 # true`
  - Возвращает **true**, если значение первого числа меньше либо равно значению второго числа, иначе - **false**.
- **<=>**
  - `5.0 <=> 6.0 # -1`
    `5.0 <=> 5.0 # 0`
    `5.0 <=> 4.0 # 1`
  - Возвращает -1, 0 или 1, если значение первого числа , равно или больше значения второго числа.
- **==**
  - `5.0 == 5.0 # true`
    `5.0 == 3.0 # false`
  - Возвращает **true**, если значение чисел равны, иначе - **false**.
- **>**
  - `5.0 > 6.0 # false`
    `5.0 > 3.0 # true`
  - Возвращает **true**, если значение первого числа больше, чем значение

второго числа, иначе - false.

- >=
  - 5.0 >= 6.0 # false
    5.0 >= 5.0 # true
  - Возвращает true, если значение первого числа больше либо равно значению второго числа, иначе - false.
- to_f
  - 5.3.to_f # 5.3
  - Преобразует вещественное число в вещественное.
- to_s
  - 5.2.to_s # "5.2"
    216.to_s(8) # "330"
    216.to_s(2) # "11011000"
  - Преобразует вещественное число в строку.

Класс String

- *
  - "hello " * 3 # "hello hello hello "
  - Повторение строки определённое числом. Возвращает новую строку.
- +
  - "hello " + "world" # "hello world"
  - Конкатенация строк. Возвращает новую строку.
- <<
  - "hello " + "world" # "hello world"
  - Добавление строки к строке. Возвращает изменённую строку.
- ==
  - "hello" == "hello" # true
    "hello" == "Hello" # false
  - Сравнение строк. Возвращается **true**, если строки эквивалентны, иначе - **false**.
- []
  - "hello world"[1] # "e"
    "hello world"[1, 3] # "ell"
    "hello world"[-3, 2] # "rl"
    "hello world"[12..-1] # nil
    "hello world"[-2..-4] # ""
    "hello world"["lo"] # "lo"
    "hello world"["test"] # nil
  - Возвращает подстроку(один символ) с указанным индексом, или подстроку начиная с указанного индекса указанной длины length. Если в качестве аргумента передается строка, то возвращается строка, если она является подстрокой строки str, иначе - **nil**.
- []=
  - "hello world"[0]= "Y" # "Yello world"
    "hello world"[1, 3] = "xxx" # "Yxxxo world"
  - Изменение строки. Возвращает изменённую строку.
- capitalize
  - "hello".capitalize # "Hello"
    "HELLO".capitalize # "Hello"
  - Возвращает копию строки, в которой первый символ преобразуется в верхний регистр, а остальные — в нижний.
- capitalize!
  - Что и capitalize, но возвращает изменённую строку или **nil**, если изменения не требуются.
- chomp(separator=$/)
  - "hello\n".chomp # "hello"
    "hello".chomp("lo") # "hel"
  - Возвращает новую строку, в которой удален последний символ separator. Если системная переменная **$/** не была изменена и не передается параметр separator, удалятся завершающие символы строки (\n, \r и \r \n).
- chomp!

- Что и chomp, но возвращает изменённую строку или **nil**, если изменения не требуются.
- chop
  - "hello".chop # "hell"
  - Возвращает новую строку из которой удален последний символ. Если строка заканчивается комбинацией символов \r\n, то будет удалена вся комбинация.
- chop!
  - Что и chop, но возвращает изменённую строку или **nil**, если изменения не требуются.
- count
  - "hello world".count("lo") # 5
    "hello world".count("lo", "o") # 2
  - Возвращает количество символов или подстрок в строке
- delete
  - "hello".delete "l","lo" # "heo"
    "hello".delete "lo" # "he"
  - Возвращает новую строку без символов переданых в качестве параметра
- delete!
  - Что и delete, но возвращает изменённую строку или **nil**, если изменения не требуются.
- downcase
  - "HEllO".downcase # "hello"
  - Возвращает новую строку, в которой все символы вернего регистра заменены на соответствующие символы нижнего.
- downcase!
  - Что и downcase, но возвращает изменённую строку или **nil**, если изменения не требуются.
- empty?
  - "hello".empty? # false
    "".empty? # true
  - Возвращает **true** если строка имеет нулевую длину, иначе - **false**.
- encode
  - "привет".encode("utf-8")
  - Возвращает строку в указанной кодировке
- include?
  - "hello".include? "lo" # true
    "hello".include? "ol" # false
  - Возвращает **true**, если строка str содержит передаваемую подстроку, иначе - **false**.
- index
  - "hello".index("e") # 1
    "hello".index("lo") # 3
    "hello".index("a") # nil
  - Возвращает индекс первого вхождения передаваемой подстроки. Если вхождения нет, возвращает **nil**.
- length
  - "hello".length # 5
    "".length # 0

- Возвращает размер строки.
- lstrip
  - " hello".lstrip #-> "hello"
  - Возвращает новую строку, в которой удалены все ведущие пробельные символы.
- lstrip!
  - Что и lstrip, но возвращает изменённую строку или **nil**, если изменения не требуются.
- reverse
  - "hello".reverse # "olleh"
  - Возвращает новую строку, в которой символы строки переставлены в обратном порядке.
- reverse!
  - Что и reverse, но возвращает изменённую строку
- rstrip
  - "hello ".rstrip #-> "hello"
  - Возвращает новую строку, в которой удалены все замыкающие пробельные символы.
- rstrip!
  - Что и rstrip, но возвращает изменённую строку или **nil**, если изменения не требуются.
- strip
  - " hello ".strip # "hello"
  - Возвращает новую строку, в которой удалены ведущие и замыкающие пробельные символы.
- strip!
  - Что и strip, но возвращает изменённую строку или **nil**, если изменения не требуются.
- swapcase
  - "Hello".swapcase # "hELLO"
  - Возвращает новую строку, в которой все символы нижнего регистра заменены на соответствующие символы верхнего и все символы верхнего регистра заменены на соответствующие символы нижнего.
- swapcase!
  - Что и swapcase, но возвращает изменённую строку или **nil**, если изменения не требуются.
- to_f
  - "12.34".to_f # 12.34
    "1.25 inch".to_f # 1.25
    "in inch 1.25".to_f # 0.0
  - Возвращает результат приведения строки в вещественное число. Символы после последнего числового игнорируются. Если строка не похожа на вещественное число, то возвращается 0.0.
- to_i(base=10)
  - "12".to_i # 12
    "12 inch".to_i # 12
    "FF".to_i(16) # 255
    "hello".to_i # 0
  - Возвращает результат приведения строки в целое число, как целого

числа. Символы после последнего числового игнорируются. Если строка не похожа на число, то возвращается 0.

- upcase
  - "hEllO".upcase # "HELLO"
  - Возвращает новую строку, в которой все символы нижнего регистра заменены на соответствующие символы верхнего.
- upcase!
  - Что и upcase, но возвращает изменённую строку или nil, если изменения не требуются.

Логические операции

- Правила
  - все объекты приводятся к булеву типу **true**
  - только объекты **NilClass** и **FalseClass** приводятся к **false**
- **and, &&**
  - `true and true   # true`
  - `1 and "hello"   # "hello"`
  - `nil and "hello" # "hello"`
  - `1 and false     # false`
- **or, ||**
  - `true or false   # true`
  - `1 or "hello"    # 1`
  - `nil or "hello"  # "hello"`
  - `1 or false      # 1`
  - `nil or false    # false`
- **not, !**
  - `not false     # true`
  - `not true      # false`
  - `not "hello"   # false`

Приоритет операций

- `!`, унарный `+`
- `**`
- унарный `-`
- `*`, `/`, `%`
- `+`, `-`
- `<<`
- `>`, `>=`, `<`, `<=`
- `<=>`, `==`, `!=`
- `&&`
- `||`
- `=`, `+=`, `-=`, итд.
- `not`
- `or`, `and`

Алгоритм создания программы

- Шаг 1: Уяснение проблемы.
- Шаг 2: Описание решения на натуральном языке.
- Шаг 3: Перевод решения на язык программирования.
- Шаг 4: Тестирование кода.

Спросить пользователя "Как вас зовут?" и вывести приветствие

- Шаг 1
  - Сколько переменных понадобится?
  - Имя и фамилия должны храниться отдельно?
  - Отчество будем запрашивать?
  - Будем ограничивать длину имени? На сколько?
- Шаг 2
  - Запрашиваем имя: "Как вас зовут?".
  - Сохраняем имя в переменной name.
  - Выводим фразу "Привет, _значение переменной name_!"
- Шаг 3
  - `print "Как вас зовут?: "`
    - печатаем строку в стандартный вывод
  - `name = gets.chomp`
    - принимаем строку из стандартного ввода
    - обрезаем символ перевода строки
    - присваиваем строку переменной
  - `puts "Привет, #{name}!"`
    - подставляем в строку значение переменной
    - печатаем строку в стандартный вывод с добавлением перевода строки

Задание

- Изучить методы изученных классов
- Написать программу, которая принимает два числа (длины сторон) и считает площадь прямоугольника
- Написать программу, которая принимает одно число (радиус) и считает площадь круга
- Написать программу, которая принимает два числа (внешний и внутренний радиусы) и считает площадь бублика

Что мы изучили?

- Познакомились с базовыми понятиями программирования
- Изучили базовые классы Ruby
- Научились выполнять операции с объектами с помощью методов
- Узнали, как составлять алгоритм программы

Модуль 2

Ruby. Уровень 1
Управление кодом

Темы модуля

- Условные выражения
- Перебор условий
- Циклы

Условные выражения

- ```
  if [выражение]
    # исполняется, если !![выражение] == true
  end
  ```
- ```
  [переменная] = if [выражение] [выражение] end
  ```
- ```
  [переменная] = if [выражение] then [выражение] end
  ```
- ```
  [переменная] = [выражение] if [выражение]
  ```

- ```
  if [выражение]
    # исполняется, если !![выражение] == true
  else
    # исполняется, если не выполнилось условие
  end
  ```
- ```
  [переменная] = if [выражение] then [выражение] else [выражение] end
  ```
- ```
  [переменная] = [выражение] ? [выражение] : [выражение]
  ```

- ```
  if [выражение-1]
    # исполняется, если !![выражение-1 == true
  elsif [выражение-2]
    # исполняется, если
    # не выполнилось предыдущее условие и
    # !![выражение-2 == true
  ...
  else
    # исполняется, если не выполнились предыдущие условия
  end
  ```

Перебор условий

```
case [базовое-выражение]
  when [выражение-1]
    # исполняется, если базовое-выражение == выражение-1
  when [выражение-2], [выражение-3]
    # исполняется, если
    # базовое-выражение == выражение-2 или
    # базовое-выражение == выражение-3
  ...
  else
    # исполняется, если не сработали предыдущие блоки
end
```

Циклы

- ```
  while [выражение]
    # исполняется пока !!выражение == true
  end
  ```

- ```
  loop do
    if [выражение-1]
      # прерываем весь цикл, если !!выражение-1 == true
      break
    end
    if [выражение-2]
      # прерываем текущую итерацию, если !!выражение-2 == true
      next
    end
  end
  ```

- ```
  1.upto 5 do |[переменная]|
    [выражение]
  end
  ```
- `5.downto 1 do |n| puts line * n end`
- `5.downto(1) { |[переменная]| [выражение] }`
  ```

Задание

- Парадокс Монти Холла — одна из известных задач теории вероятностей, решение которой, на первый взгляд, противоречит здравому смыслу.
- Парадокс основан на американском телешоу «Let's Make a Deal» и назван в честь ведущего этой передачи.
- Правила игры
  - Участнику игры нужно выбрать одну из трёх дверей. За одной из дверей находится приз, за двумя другими дверями ничего нет. Участнику выбирает одну из дверей, например, номер 1, после этого ведущий, который знает, где находится приз, открывает одну из оставшихся дверей, например, номер 3, за которой ничего нет. После этого он спрашивает участника: не желаете ли он изменить свой выбор и выбрать дверь номер 2? Участник может согласиться либо остаться при своём выборе.
- Вопрос
  - Увеличатся ли шансы участника выиграть приз, если он примет предложение ведущего и измените свой выбор?
- Предположения
  - Математики утверждают, что увеличатся на 60%.
  - Разум подсказывает, что нет.
- Напишите программу, которая подтвердит или опровергнет доводы математиков.
- Для решения задачи вам потребуется глобальный метод **rand**, который генерирует:
  - `rand`
    - случайное число в диапазоне от 0 до 1
  - `rand( число )`
    - случайное число в диапазоне от 0 до число-1
  - `rand( число-1 .. число-2 )`
    - случайное число в диапазоне от число-1 до число-2
    - например, `rand(2 .. 4) # либо 2, либо 3, либо 4`

Что мы изучили?

- Познакомились с условными выражениями
- Научились перебирать условия
- Узнали, как исполнять разнообразные цикличные конструкции

Модуль 3

Ruby. Уровень 1
Коллекции и основные структуры данных

Темы модуля

- Понятие коллекций
- Основные структуры данных
  - Array
  - Hash
- Основные операции со структурами данных
- Стандартные методы итераций

Коллекции

- Класс Range
  - https://ruby-doc.org/core-2.3.0/Range.html

- ```
  for num in (1..5)
    # что-то делаем с текущим значением коллекции
  end
  ```
- ```
  (1...5).each do |num|
    # что-то делаем с текущим значением коллекции
  end
  ```

- ```
  "hello".each_char do |char|
    # что-то делаем с текущим символом строки
  end
  ```
- ```
  "hello\nworld".each_line do |line|
    # что-то делаем с текущей строкой
  end
  ```

Класс Array

- *
  - [1, 2, 3] * 3 # [1, 2, 3, 1, 2, 3, 1, 2, 3]
    [1, 2, 3] * "," # "1,2,3"
  - Повторение, которое возвращает новый массив.
- +
  - [1, 2] + [3, 4] # [1, 2, 3, 4]
  - Кокатенация, которая возвращает новый массив, созданный из двух массивов путем добавления одного к другому.
- -
  - [1,2,2,3,3,4] - [1,3] # [2, 2, 4]
  - Вычитание, которое возвращает новый массив.
- <<
  - [1, 2] << "a" << "b" << [3, 4] # [1, 2, "a", "b", [3, 4]]
  - Добавляет передаваемый объект в конец массива. Возвращает изменённый массив с уже добавленным элементом.
- <=>
  - [1, 2] <=> [1, 2, 3] # -1
    [1, 2] <=> [1, 2] # 0
    [1, 2, 3] <=> [1, 2] # 1
  - Возвращает целое число -1, 0, или 1, если текущий массив меньше, равен или больше другого массива.
- ==
  - [1, 2] == [1, 2] # true
    [2, 1] == [1, 2] # false
  - Возвращает **true**, если количество элементов и соответствующие пары элементов равны, иначе - **false**.
- at
  - ["a", "b", "c"].at(0) # "a"
    ["a", "b", "c"].at(-1) # "c"
    Возвращает элемент массива с переданным индексом. Отрицательная индексация подразумевает отсчет с конца массива. Возвращает **nil**, если индекс выходит за пределы допустимого дианазона.
- clear
  - [1, 2, 3].clear # []
  - Удаляет все элементы из массива array. Возвращает изменённый массив!
- compact
  - [1, 2, **nil**, 3, **nil**].compact # [1, 2, 3]
  - Возвращает копию массива из которого удалены все элементы **nil**.
- compact!
  - Что и compact, но изменяет оригинальный массив
- concat
  - [1, 2].concat([3, 4]) # [1, 2, 3, 4]
  - Добавляет к массиву array элементы массива other_array.
- count
  - [1, 2, 4, 2].count # 4
    [1, 2, 4, 2].count(2) # 2

- Без аргументов возвращает количество элементов массива. Если задан аргумент, считает количество элементов которые равны obj через ==.
- delete
  - [1, 2, 4, 2].delete(2) # 2 ([1, 4])
  - Удаляет все элементы из оригинального массива, которые равны переданному параметру и возвращает удалённое значение. Если ничего не было удалено, то возвращает nil.
- delete_at
  - [1, 2, 4, 2].delete_at(2) # 4 ([1, 2, 2])
  - Удаляет элемент из оригинального массива array, который имеет переданный индекс. Возвращает значение удаленного элемента или nil, если index находится вне допустимого диапазона.
- empty?
  - [].empty? # true
  - Возвращает true, если массив не содержит элементов, иначе - false
- first
  - [1, 2, 3].first # 1
    [1, 2, 3].first(2) # [1, 2]
  - Возвращает первый элемент или массив первых элементов массива. Если массив пуст, то для первой формы вызова возвращается nil, а для второй – пустой массив.
- flatten
  - [[1, 2], [3, 4, [5, 6]], 7].flatten # [1, 2, 3, 4, 5, 6, 7]
  - Преобразует многомерный массив array в одномерный.
- flatten!
  - Что и flatten, но изменяет оригинальный массив
- include?
  - ["a", "b", "c"].include?("b") # true
    ["a", "b", "c"].include?("в") # false
  - Возвращает true, если переданный объект является элементом массива, иначе – false.
- index
  - ["a", "b", "c"].index("b") #=> 1
    ["a", "b", "c"].index("d") #=> nil
  - Возвращает индекс первого вхождения элемента в массиве. Возвращает nil, если такой элемент не был найден.
- insert
  - [1,2,3,4].insert(2, 5) # [1, 2, 5, 3, 4]
  - Вставляет полученные значения перед элементом с переданным индексом.
- join
  - ["a", "b", "c"].join #=> "abc"
    ["a", "b", "c"].join(",") #=> "a,b,c"
  - Возвращает строку, созданную путем преобразования каждого элемента массива в строку, разделенных строкой переданной в качестве аргумента.
- last
  - [1, 2, 3].last # 3
    [1, 2, 3].last(2) # [2, 3]
  - Возвращает последний элемент или массив последних элементов массива.

Если массив пуст, то для первой формы вызова возвращается **nil**, а для второй – пустой массив.

- length
  - `[1, 2, 3].length # 3`
  - Возвращает количество элементов в массиве.
- pop
  - `[1, 2, 3].pop # 3`
  - Удаляет последний элемент из массива и возвращает его. Изменяет исходный массив.
- push
  - `[1, 2, 3].push(4, 5) # [1, 2, 3, 4, 5]`
  - Добавляет передаваемые объекты в конец массива. Возвращает изменённый массив.
- reverse
  - `[1, 2, 3].reverse # [3, 2, 1]`
  - Возвращает новый массив, в котором элементы массива стоят в обратном порядке.
- reverse!
  - Что и reverse, но изменяет оригинальный массив
- shift
  - `[1, 2, 3].shift # 1`
  - Удаляет первый элемент из массива и возвращает его. Изменяет исходный массив.
- sort
  - `[2, 1, 3].sort # [1, 2, 3]`
  - Возвращает новый массив, который получен путем сортировки массива по возрастанию.
- sort!
  - Что и sort, но изменяет оригинальный массив
- to_s
  - Что и join
- uniq
  - `[1, 2, 2, 3, 1, 3].uniq # [1, 2, 3]`
  - Возвращает новый массив удаляя все повторяющиеся элементы.
- uniq!
  - Что и uniq, но изменяет оригинальный массив
- unshift
  - `[1, 2, 3].unshift(4, 5) # [4, 5, 1, 2, 3]`
  - Добавляет элементы в начало массива со сдвигом вправо уже существующих. Изменяет исходный массив.
- zip
  - `[1,2].zip([3, 4], [5, 6]) # [[1, 3, 5], [2, 4, 6]]`
  - Преобразует аргументы в массивы при помощи метода to_a, объединяя каждый элемент массива с соответствующим массивом, переданным в качестве аргумента.

Итерирование массива

- ```ruby
  array.each do |item|
    # что-то делаем с текущим элементом массива
  end
  ```
- ```ruby
  array.each_index do |index|
    # что-то делаем с текущими элементом массива и его индексом
  end
  ```

- ```ruby
  array.map do |item|
    # результат последнего выражения возвращается в новый массив
    # array.map! изменяет исходный массив
  end
  ```
- ```ruby
  array.map.with_index do |item, index|
    # результат последнего выражения возвращается в новый массив
  end
  ```

- ```ruby
  array.select do |item|
    # если результат последнего выражения приводится к TrueClass,
    # то текущий элемент массива возвращается в новый массив
    # array.select! изменяет исходный массив
  end
  ```
- ```ruby
  array.reject do |item|
    # если результат последнего выражения приводится к FalseClass,
    # то текущий элемент массива возвращается в новый массив
    # array.reject! изменяет исходный массив
  end
  ```

Класс Hash

- ==
  - {"a" => 1, "b" => 2} == {"b" => 2, "a" => 1} # true
    {"a" => 1, "b" => 2} == {"a" => 2, "b" => 1} # false
  - Равенство – два хеша считаются равными, если они содержат одинаковое число ключей, и если каждая пара ключ-значение эквивалентна соответствующим элементам в другом хеше.
- clear
  - { "a" => 1, "b" => 2 }.clear # {}
  - Удаляет все пары ключ-значение из хеша.
- default=
  - h = {"a" => 1, "b" => 2}
    h.default = 0 h["a"] # 1
    h["c"] # 0
  - Устанавливает значение по-умолчанию, если ключ не существует в данном хеше
- delete
  - {"a" => 1, "b" => 2}.delete("a") # 1
  - Удаляет пару ключ-значение из хеша. Возвращается значение, соответствующее ключу. Если ключ не был найден, тогда возвращается значение по-умолчанию
- empty?
  - {}.empty? # true
  - Возвращает **true**, если хеш не содержит пар ключ-значение, иначе - **false**
- invert
  - {"a" => 1, "b" => 2}.invert # {1 => "a", 2 => "b"}
  - Возвращает новый хеш, созданный путем использования значений хеша в качестве ключей, а ключей в качестве значений.
- key
  - {"a" => 1, "b" => 2}.key(1) # "a"
  - Возвращает ключ для заданного значения. Если значение не найдено, возвращает **nil**.
- key?
  - {"a" => 1, "b" => 2}.key?("a") # true
  - Возвращает **true**, если заданный ключ находится в хеше, иначе - **false**
- keys
  - {"a" => 1, "b" => 2}.keys # ["a", "b"]
  - Возвращает новый массив, состоящий из ключей данного хеша.
- length
  - {"a" => 1, "b" => 2}.length # 2
  - Возвращает количество пар ключ-значение в данном хеше.
- merge
  - {"a"=>1, "b"=>2}.merge({"b"=>3, "c"=>4}) # {"a"=>1, "b"=>3, "c"=>4}
  - Возвращает новый хеш, который состоит из содержимого данного хеша и хеша, переданного в качестве аргумента. Если в результате слияния обнаружатся одинаковые ключи, то для него будет записано значение из

последнего хеша.
- merge![править]
  - Что и merge, но изменяет оригинальный хеш
- shift
  - {"a"=>1, "b"=>2}.shift # ["a", 1]
  - Удаляет первую пару ключ-значение из хеша и возвращает эту пару в виде массива [key, value]. Если хеш пуст, то возвращает значение по-умолчанию.
- to_a
  - {"a"=>1, "b"=>2}.to_a # [["a", 1], ["b", 2]]
  - Конвертирует хеш в массив, состоящий из массивов [key, value].
- to_s
  - {"b"=>2, "a"=>1}.to_s # "a1b2"
    Преобразует хеш в строку путем преобразования хеша в массив массивов [key, value], и преобразования этого массива в строку, используя метод join класса Array со стандартным разделителем
- value?
  - {"a"=>1, "b"=>2}.value?(1) # true
  - Возвращает **true**, если заданное значение принадлежит некоторому ключу в хеше
- values
  - {"a"=>1, "b"=>2}.values # [1, 2]
  - Возвращает массив, состоящий из значений данного хеша.

Итерирование хеша

- ```
  hash.each do |key, value|
    # что-то делаем с текущей парой ключ:значение хеша
  end
  ```
- ```
  hash.each_key do |key|
    # что-то делаем с текущим ключом хеша
  end
  ```
- ```
  hash.each_value do |value|
    # что-то делаем с текущим значением хеша
  end
  ```

- ```
  hash.select do |key, value|
    # если результат последнего выражения приводится к TrueClass,
    # то текущая пара ключ:значение возвращается в новый массив
    # hash.select! изменяет исходный хеш
  end
  ```
- ```
  hash.reject do |key, value|
    # если результат последнего выражения приводится к FalseClass,
    # то текущая пара ключ:значение возвращается в новый массив
    # hash.reject! изменяет исходный хеш
  end
  ```

- ```
  hash.keep_if do |key, value|
    # тоже, что и hash.select!, но
    # возвращает nil в случае отсутствия изменений в исходном хеше
  end
  ```
- ```
  hash.delete_if do |key, value|
    # тоже, что и hash.reject!, но
    # возвращает nil в случае отсутствия изменений в исходном хеше
  end
  ```
  ```

Класс Symbol

- inspect
  - `:hello.inspect # ":hello"`
- to_s
  - `:hello.to_s # "hello"`
- capitalize
  - Что и `:symbol.to_s.capitalize.to_sym`
- downcase
  - Что и `:symbol.to_s.downcase.to_sym`
- empty?
  - Что и `:symbol.to_s.empty?`
- length
  - Что и `:symbol.to_s.length`
- swapcase
  - Что и `:symbol.to_s.swapcase.to_sym`
- upcase
  - Что и `:symbol.to_s.upcase.to_sym`

Задание

- Парадокс дней рождения
- В группе, состоящей из 23 или более человек, вероятность совпадения дней рождения (число и месяц) хотя бы у двух людей превышает 50 %
- Для 60 и более человек вероятность такого совпадения превышает 99 %
- Утверждение не является парадоксом в строгом научном смысле: логического противоречия в нём нет, а парадокс заключается лишь в различиях между интуитивным восприятием ситуации человеком и результатами математического расчёта
- Предположения
  - Математики утверждают, что увеличатся на 60%.
  - Разум подсказывает, что нет.
- Тем не менее, утверждение кажется неочевидным, поэтому напишите программу, которая его подтвердит или опровергнет.

Что мы изучили?

- Познакомились с коллекциями языка Ruby
- Узнали, как исполнять разнообразные операции над структурам данных
- Научились итерировать по структурам данных

Модуль 4

Ruby. Уровень 1
Методы

Темы модуля

- Методы
- Блоки
- Процедуры

Методы

- ```
  def method_name
    # что-то делаем # возвращается результат последнего выражения
  end
  method_name # вызов метода
  ```

- ```
  def method_name argument [, ...]
    # ...
  end
  method_name parametr # вызов метода с параметром
  ```

- ```
  def method_name argument=value_by_default
    # аргумент имеет значение по умолчанию
  end
  ```

- ```
  def method_name argument:value_by_default
    # использование аргумента в качестве ключевого слова
  end
  method_name argument:value # параметр ключевое слово
  ```

- ```
  def method_name *params
    # params - массив аргументов
  end
  ```

- ```
  def method_name
    return # явный возврат значения
  end
  ```

- Глобальная переменная
  - **$x**
  - доступна внутри метода

Многоликий блок

- ```
  def method_name
    # исполнение анонимного ожидаемого блока
    yield
  end
  method_name{[содержимое блока]}
  ```

- ```
  def method_namea argument, &block
    # block - явно ожидаемый именованный блок
    # исполнение именованного блока с передачей ему параметра
    block.call argument
  end
  method_name(parameter) {[содержимое блока]}
  ```

- ```
  # Создание процедура из блока
  ```
  - ```
    procedure = proc{[содержимое блока]}
    ```
  - ```
    def method_namea argument, block
      # block - блок как процедура
      # исполнение процедуры с передачей ей параметра
      block.call argument
    end
    method_name parameter, procedure
    ```
    ```

Задание

- Игра «Поле чудес»
- Задача
  - Написать упрощённую версию игры,
    которая запускается в консоли для одного игрока.

- Задание 1
  - У вас есть список слов: “книга”, “месяц”, “ручка”, “шарик”, “олень”, “носок”. Обратите внимание на то, что длина всех слов одинакова.
  - Необходимо выбрать случайным образом слово и отрисовать его, используя вместо букв какие-либо символы, например “\u25A0”.
  - Необходимо установить счётчик “жизни” в какое-либо значение, например 5
  - Предложить игроку ввести букву или всё слово целиком.
    - Если буква правильная, то слово перерисовывается с видимой буквой.
    - Если буква неправильная, то у игрока отнимается одна “жизнь”.
    - Если игрок ввёл слово и это слово правильно, либо это последняя правильная буква, либо у игрока закончились “жизни”, то игра заканчивается.

```
■ ■ ■ ■ ■ | ❤x3
Назовите букву или слово целиком: е
■ ■ Е ■ ■ | ❤x3
Назовите букву или слово целиком: р
Неправильно. Вы теряете жизнь
■ ■ Е ■ ■ | ❤x2
Назовите букву или слово целиком: о
О ■ Е ■ ■ | ❤x2
Назовите букву или слово целиком: в
Неправильно. Вы теряете жизнь
О ■ Е ■ ■ | ❤x1
Назовите букву или слово целиком: ь
О ■ Е ■ Ь | ❤x1
Назовите букву или слово целиком: олень
О Л Е Н Ь
Вы выиграли! Приз в студию!
```

- Задание 2
  - Предложите игроку после каждого тура сыграть ещё или отказаться.
  - При согласии количество жизней восстанавливается
  - Игра ведётся до тех пор, пока не закончатся слова в списке.

- Задание 3
  - Предложите игроку выбрать уровень сложности текущей игры. Например, при лёгком уровне игроку будет выдаваться 7 “жизней”, при среднем - 5, а при сложном - 3 “жизни”.

- Задание 4
  - Добавьте в список слова разной длины.

Что мы изучили?

- Научились определять и использовать методы
- Познакомились с "анонимными функциями" - блоками
- Узнали, как создавать из блоков процедуры
- Уяснили, когда лучше использовать блоки, методы и процедуры

Модуль 5

Ruby. Уровень 1
Объекто-ориентированное программирование

Темы модуля

- Классы
- Объекты
- Модули
- Примеси

Классы и их экземпляры

```ruby
class ClassName
  # пустой класс
end
obj = ClassName # создание экземпляра класса ClassName
```

```ruby
class ClassName
  # создание метода
  def method_name() end
end
obj = ClassName.new # создание экземпляра класса ClassName
obj.method_name # вызов метода
```

```ruby
class ClassName
  # создание "конструктора"
  def initialize argument
    # инициализация переменной экземпляра класса
    @property = argument
  end
end
obj = ClassName.new parameter
```

```ruby
class ClassName
  # создание "конструктора"
  def initialize argument
    # инициализация переменной экземпляра класса
    @property = argument
  end
  # определение методов для доступа к переменной экземпляра класса
  def property; @property; end
  def property=(value); @property = value; end
end
obj = ClassName.new parametr
obj.property = value # доступ к переменной экземпляра класса
```

```ruby
class ClassName
  # создание "конструктора"
  def initialize
    # инициализация переменных экземпляра класса
    @property1 = value @property2 = value @property3 = value
  end
  # автоматическое определение методов для доступа к переменным экземпляра
  attr_accessor :property1
  attr_reader :property2
  attr_writer :property3
end
obj = ClassName.new
obj.property1 = value # доступ к переменной экземпляра класса
```

```ruby
class ClassName
  # определение псевдонима метода method_name
  alias other_name method_name
  # создание метода
  def method_name() end
end
obj = ClassName # создание экземпляра класса ClassName
obj.method_name # вызов метода
obj.other_name # вызов метода через псевдоним
```

Класс как объект

- ```ruby
  class ClassName
    # создание методов класса
    def ClassName.method_name1() end
    def self.method_name2() end
  end
  ClassName.method_name2 # вызов метода класса
  ```

- ```ruby
  class ClassName
    # создание константы класса
    CONSTANT = value
  end
  ClassName::CONSTANT # обращение к константе класса
  ```

- ```ruby
  class ClassName
    # создание переменной класса
    @@property = value
    # определение методов для доступа к переменной класса
    def self.property; @@property; end
    def self.property=(value); @@property = value; end
  end
  ClassName.property = value # доступ к переменной класса
  ```
  ```

Задание

- Написать игру «Математические вопросы»
- Исходные данные
  - Создайте несколько классов, которые описывают геометрические фигуры, например Circle(окружность) и Rectangle(прямоугольник)
  - В каждом из классов определите методы
    - initialize, который принимает необходимые параметры для данной фигуры
    - get_description, который возвращает описание объекта
    - get_area, который возвращает площадь фигуры
    - get_perimeter, который возвращает периметр фигуры
- Ход игры
  - В начале игры создайте случайным образом объект того или иного класса
  - Передайте в "конструктор" случайные значения, например Rectangle.new 3, 5
  - Задайте Игроку вопрос: Какова площадь данной фигуры?
  - Примите ответ и сравните с тем, который должен получиться. Сообщите об этом Игроку
  - Задайте Игроку вопрос: Каков периметр данной фигуры?
  - Примите ответ и сравните с тем, который должен получиться. Сообщите об этом Игроку
  - Если Игрок хочет продолжить игру, повторите вышеописанные действия
- Шаблон сценария игры
  - "Привет! Хочешь играть? Да/Нет: " Да
  - "Я - прямоугольник 3 x 5"
  - "Какова моя площадь?: " 15
  - "Правильно!"
  - "Каков мой периметр?: " 25
  - "Ошибка. Правильный ответ: 16"
  - "Продолжаем игру?"

Наследование

```ruby
class BaseClass
  # создание метода
  def method_name() end
end
class ClassName < BaseClass
end
obj = ClassName # создание экземпляра класса ClassName
obj.method_name # вызов метода
obj.is_a? BaseName # проверка цепочку предков
obj.instance_of? ClassName # проверка родителя
```

```ruby
class ClassName < BaseClass
  def initialize(param1, param2)
    @param = param1
    # вызов метода супер-класса
    super(param2)
  end
end
```

```ruby
class BaseClass
  def public_method() end
  def private_method() end
  # определение закрытого метода
  private :private_method
end
```

```ruby
class ClassName
  # метод вызываемый при клонировании
  def initialize_copy(orig)
    # orig - оригинальный объект
  end
end
```

```
obj = ClassName.new other = obj.clone
```

Модули

- Подключение файлов
  - load "file.rb"
  - require "./file"
- Полезные переменные
  - $LOAD_PATH или $: # массив
  - ENV # объект

- Модуль как пространство имён
  - ```
    module ModuleName
      CONSTANT = value
      def self.method_name() end
    end
    ModuleName.method_name
    ModuleName::CONSTANT
    ```

- Модуль как примеси
  - ```
    module MixinName
      def method_name() end
    end
    class ClassName
      include MixinName
    end
    obj = ClassName.new
    obj.method_name
    ```

  - ```
    module MixinName
      def method_name() end
    end
    class ClassName
    ```
    ```

```
  extend MixinName
end
ClassName.method_name
```

Что мы изучили?

- Познакомились с понятием класса
- Научились создавать объекты - экземпляры класса
- Узнали, как разбивать код на модули

Модуль 6

Ruby. Уровень 1
Практический Ruby

Темы модуля

- Работа с файлами
- JSON
- Перехват ошибок
- Класс Time
- Регулярные выражения
- И немного "драгоценных камней"

Работа с файлами

- Полезные методы
  - File.directory? "имя файла"
  - File.file? "имя файла"
  - File.exists? "имя файла"
  - File.size? "имя файла"
  - File.extname "имя файла"
  - File.executable? "имя файла"
  - File.readable? "имя файла"
  - File.writable? "имя файла"
- Режимы работы с файлом
  - "r": для чтения
  - "r+": для чтения/записи
  - "w": для записи, содержимое перезаписывается
  - "w+": для чтения/записи, содержимое перезаписывается
  - "a": для записи, курсор в конце файла
  - "a+": для чтения/записи, курсор в конце файла
- Чтение файла
  - ```
    f = File.open("имя файла", "r")
      f.each_line do|line|
        # line - текущая строка файла
      end
    f.close
    ```
  - ```
    File.open("имя файла") do |f|
      f.each_line do|line|
        # line - текущая строка файла
      end
    end
    ```
  - ```
    str = File.read("имя файла", :encoding=>"utf-8")
    # str - содержимое файла в строке
    ```
  - ```
    lines = File.readlines("имя файла")
    lines.each do|line|
      # line - текущая строка файла
    end
    ```
  - ```
    IO.foreach("имя файла")
    do|line|
      # line - текущая строка файла
    ```
    ```

```
end
File.open("имя файла") do |f|
  f.each_char do|char|
    # char - текущая символ в файле
  end
end
```

- Запись в файл
  - ```
    File.open("имя файла", "a") do |f|
      # Запись в конец файла
      f.puts "..."
    end
    ```
- Манипуляции с файлами
  - `File.rename "текущее имя", "новое имя"`
  - `File.delete "имя файла"`
  - ```
    require 'fileutils'
    FileUtils.cp "текущее имя", "другое имя"
    ```

JSON

- `require "json"`
- `JSON.generate hash_или_ array`
- `JSON.parse "json-строка"`

Ловим ошибки

- ```
  begin
    # Здесь может быть ошибка
    rescue
    # Перехватываем ошибку
  end
  ```

- ```
  begin
    # Здесь может быть ошибка
    rescue ZeroDivisionError
      # Перехватываем ошибку конкретного типа
    rescue NoMethodError
      # Перехватываем ошибку другого конкретного типа
  end
  ```

- ```
  begin
    # Здесь может быть ошибка Ruby
    # Здесь может быть наша ошибка
    raise
    rescue => e
      # Перехватываем ошибку
      e.backtrace
    retry #Попробовать ещё раз
    else
      # Если ошибки нет
    ensure
      # Финализатор
  end
  ```
  ```

Работаем со временем

- Класс Time
  - http://ruby-doc.org/core-2.3.0/Time.html
- Time.now
  - Объект текущих даты и времени
- Time.now.day
  - Текущая дата
- Time.now.to_i
  - Текущая временная метка
- Time.at(timestamp)
  - Объект даты и времени для текущей временной метки
- Time.at(timestamp).strftime("%d-%m-%Y %H-%M-%S")
  - Форматируем строковое представление

- Класс DateTime
  - https://ruby-doc.org/stdlib-2.3.0/libdoc/date/rdoc/DateTime.html

Задание

- Задача
  - Задача состоит в том, чтобы подсчитать количество уникальных слов в текстовом файле и записать их в другой файл в алфавитном порядке в формате JSON.
- Задание 1
  - Создайте в текстовом редакторе текстовый файл по имени “data.txt” и запишите в него любой произвольный текст. Текст можно скопировать из любого источника, например из веб-страницы.
- Задание 2
  - Создайте метод **read_file**.
  - Метод должен принимать имя файла, прочитать его и вернуть список уникальных слов из этого файла.
- Задание 3
  - Создайте метод **save_file**.
  - Метод должен принимать имя файла для записи и список уникальных слов. В методе подсчитайте количество уникальных слов и запишите его в файл вместе со всеми словами отсортированными в алфавитном порядке в формате JSON. Например:
    Всего уникальных слов: 64
    ==========
    ["А слово", "Б слово", "..."]

Регулярные выражения

- Онлайн тестирование регулярных выражений
  - http://rubular.com/
- .
  - Любой символ
- abc
  - Подстрока abc
- [abc]
  - любой из символов a, b, или c
- [^abc]
  - любой из символов кроме: a, b, или c
- [a-z]
  - Любой символ в диапазоне a-z
- [a-zA-Z]
  - Любой символ в диапазоне a-z или A-Z
- ^
  - Начало строки
- $
  - Конец строки
- \s
  - Любой пробельный символ
- \S
  - Любой непробельный символ
- \d
  - Любая цифра, то есть [0-9]
- \D
  - Любая не цифра, то есть [^0-9]
- \w
  - [a-zA-Z0-9_]
- \W
  - [^a-zA-Z0-9_]
- a?
  - 0 или 1 вхождение символа "a"

- a*
  - 0 или больше вхождений символа "a"
- a+
  - 1 или больше вхождений символа "a"
- a{3}
  - 3 вхождения символа "a"
- a{3,}
  - 3 или больше вхождений символа "a"
- a{3,6}
  - минимум 3 или максимум 6 вхождений символа "a"
- (a|b)
  - a или b
- (...)
  - Группировка

Классы Regexp и MatchData

- http://ruby-doc.org/core-2.3.0/Regexp.html

- Позиция вхождения
  - "строка" =~ /регулярное выражение/
  - /регулярное выражение/ =~ "строка"
- Экземпляр класса MatchData
  - http://ruby-doc.org/core-2.3.0/MatchData.html
  - "строка".**match** /регулярное выражение/
  - /регулярное выражение/.match "строка"
  - "строка".match(/регулярное выражение/).**captures**
  - "строка".**sub**(/регулярное выражение/, "строка")
  - "строка".**gsub**(/регулярное выражение/, "строка")

Ruby Gems

- https://rubygems.org/

- **gem** install gem_name
- **gem** uninstall gem_name

- **gem** server

- Использование
- **gem** install unicode

- require "unicode"
- Unicode::upcase "строка"

- ```
  class String
     def upcase
        Unicode::upcase(self)
     end
     def upcase!
        self.replace upcase
     end
  end
  ```
- "строка".upcase
  ```

Задание

- На железнодорожном вокзале ведётся журналирование рейсов
- Каждый рейс в файле журнала занимает одну строку:
  - Поезд *номер-поезда прибыл/отправился из/в город* в *время*
- В файле журнала может присутствовать и другая информация
- Текущий журнал выглядит как:

```
Рейс 365 прибыл из Сасово в 12:56:30
сообщение получено в 12:57:20
Сохранено в базу данных
Рейс 452 отправился в Сочи в 13:04:22
сообщение получено в 13:11:32
Ошибка записи в базу данных
```

- Необходимо
  - Зачитать файл журнала
  - Выбрать необходимую информацию
  - Представить информацию в виде:
    - [*время*]: Поезд № *номер-поезда из/в город*
  - Например:
    - `[12:56:30] - Поезд № 365 из Сасово`
      `[13:04:22] - Поезд № 452 в Сочи`

Что мы изучили?

- Научились работать с файлами
- Научились перехватывать ошибки
- Узнали, что такое JSON
- Познакомились с классами Time и DateTime
- Познакомились с регулярными выражениями
- Узнали о возможности расширения функционала Ruby с помощью "драгоценных камней"

Модуль 8

Ruby. Уровень 1
Алгоритмы и абстрактные типы данных

Темы модуля

- Сложность алгоритмов
- Алгоритмы поиска
- Структуры данных
- Абстрактные типы данных
- Рекурсия

Один результат - разные реализации

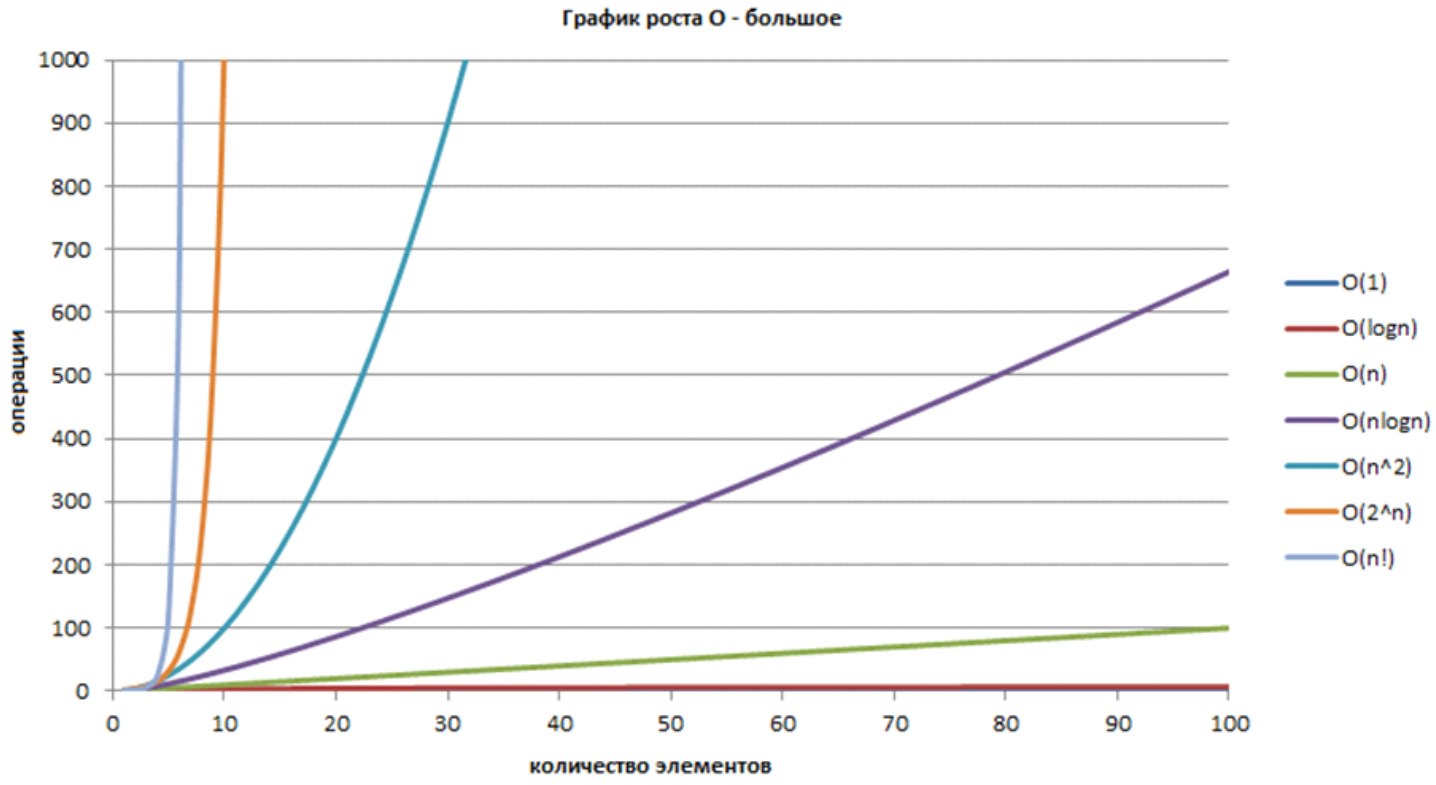
```
def palindrom_one s
    r = ""
    len = s.length - 1
    while len>=0
        r += s[len]
        len -= 1
    end
    if s.upcase == r.upcase
        return true
    end
    false
end

def palindrom_two s
    i = 0;
    j = s.length - 1
    len = s.length / 2
    while i<len
        if s[i] != s[j]
            return false
        end
        i += 1;
        j -=1
    end
    true
end

puts palindrom_one("наган")
puts palindrom_two("наган")
```

Время исполнения

```
def sum_one n
    sum = 0
    for i in (1..n)
        sum += i
    end
end

def sum_two n
    sum = (n * (n + 1)) / 2
end

require "benchmark"
puts Benchmark.measure { sum_one(100) }
puts Benchmark.measure { sum_two(100) }
```

Сложность алгоритмов

- Сложности
  - 1 Константная
  - log n Логарифмическая
  - n Линейная
  - n log n Линейно-логарифмическая (суперлинейный)
  - n^m Полиномиальный
    - n^2 Квадратичная
    - n^3 Кубическая
  - m^n Экспоненциальная
    - 2^n
  - n! Факториальный




```
def complexity n
    a = 0
    b = 0
    c = 0
    for i in n
        for j in n
            a += i * i
```

```
            b += j * j
            c += i * j
        end
    end
    x = 0
    y = 0
    for i in n
        x += a * b + i
        y += c * i
    end
    z = 42
end
```

Последовательный поиск

```
def sequential_search(list, key)
    list.each_with_index do |item, index|
        if key == item
            return index
        end
    end
    return -1
end
```

def sequential_search(list, key)

```
        list.each_with_index do |item, index|
                if key == item
                        return index
                end
        end
        return -1
end
```

Поиск в упорядоченном списке

```
def ordered_linear_search(list, key)
  list.each_with_index do |item, index|
      if key == item
         return index
      elseif item > key
         return -1
      end
   end
   return -1
end
```

```
def sequential_search(list, key)
        list.each_with_index do |item, index|
                if key == item
                        return index
                end
        end
        return -1
end
```

Бинарный поиск

```
def binary_search(list, key)
  first = 0
  last = list.length - 1

  while first <= last
    mid = (first + last) / 2
    if list[mid] == key
       return mid
    else
      if key < list[mid]
        last = mid - 1
      else
        first = mid + 1
      end
    end
  end

  return -1
end
```

Стратегия "Разделяй и властвуй"

```
def binary_search(list, key)
  if list.length == 0
    return "No"
  else
    mid = list.length / 2
    if list[mid] == key
      return "Yes"
    else
      if key < list[mid]
        return binary_search(list[0, mid], key)
      else
        return binary_search(list[mid+1, list.length-1], key)
      end
    end
  end
end
```

Рекурсия

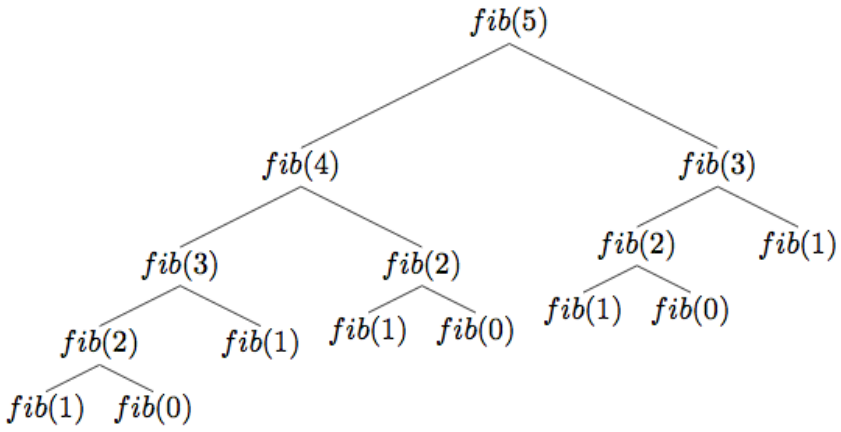
```
def question(message, answer)
    print message
    if gets.strip() == answer
       return
     else
       puts "Подумайте..."
       question(message, answer)
    end
end
question("2 * 2 = ", "4")
puts "Молодец!"

def gcd(a, b)
    return a if b == 0
    gcd(b, a%b)
end
```

Понимание рекурсии

```
def factorial n
    return 1 if n <= 1
    n * factorial(n - 1)
end

product = factorial(3)

def factorial 3
    return 1 if 3 <= 1
    3 * factorial(3 - 1)
end
    def factorial 2
        return 1 if 2 <= 1
        2 * factorial(2 - 1)
    end
        def factorial 1
            return 1 if 1 <= 1
            1 * factorial(1 - 1)
        end
    def factorial 2
        return 1 if 2 <= 1
        2 * 1
    end
def factorial 3
    return 1 if 3 <= 1
    3 * 2
end

product = 6
```

Рекурсия: за и против

```
def fib1(n)
  if n<2
    return n
  else
    return fib1(n-1) + fib1(n-2)
  end
end
```




```
def fib2(n)
  fibs=[1, 1]

  for i in 2...n
    fibs << (fibs[i-1] + fibs[i-2])
  end

  fibs[fibs.length-1]
end
```

```ruby
def fib3(n)
  a, b = 0, 1

  for _ in 1..n
    a, b = b, a + b
  end
  a
end
```

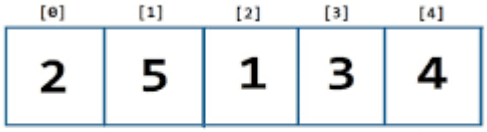
Задание

- Определите сложность следующих алгоритмов:
  - ```
    def one(list)
        result = 0
        for i in list
            for j in list
                result += i * j
            end
        end
        result
    end
    ```
  - ```
    def two(list)
        result = 0
        for i in list
            result += 1
        end
        for j in list
            result -= 1
        end
        result
    end
    ```
  - ```
    def three(n)
        while n > 0
            i += 2
            n /= 2
        end
        i
    end
    ```

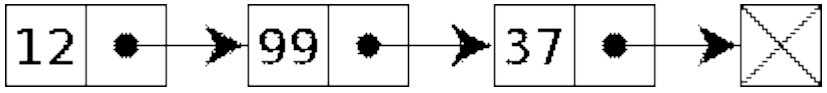
- Напишите метод **anagram**
  - Метод должен принимать в качестве параметра две строки
  - Метод должен определять, является ли одна строка анаграммой другой строки
  - Попробуйте разные варианты
  - Определите сложность алгоритмов
  - Найдите самый быстрый алгоритм
  - Помощь: Что такое анаграмма
    ```

Структуры данных

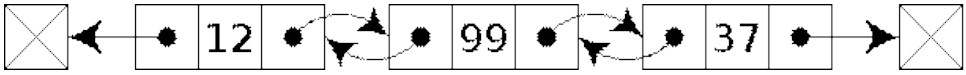
- Array



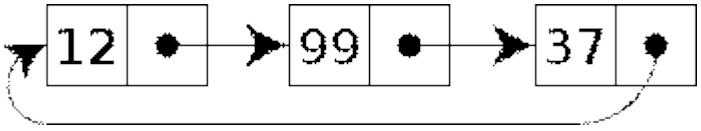
- Linked list



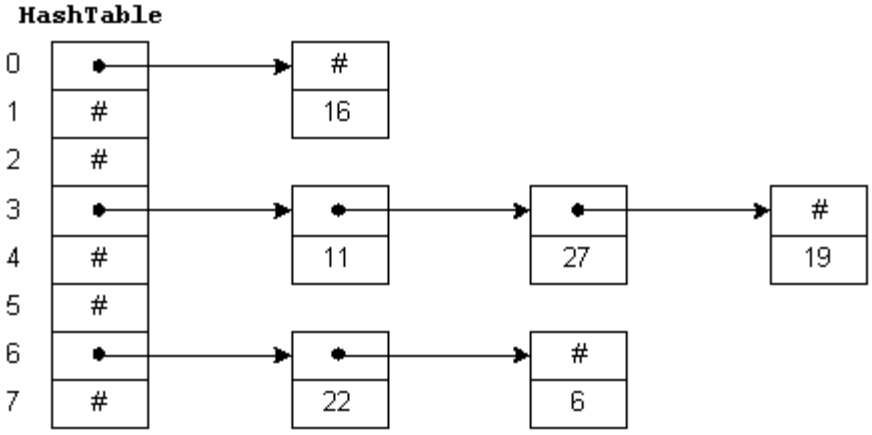
  - Doubly Linked list



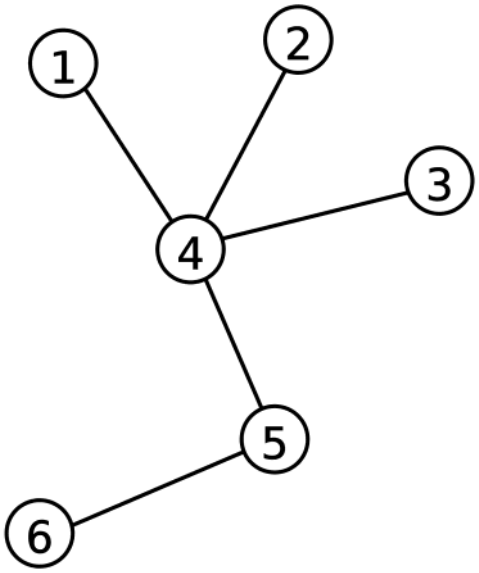
  - Circular Linked List



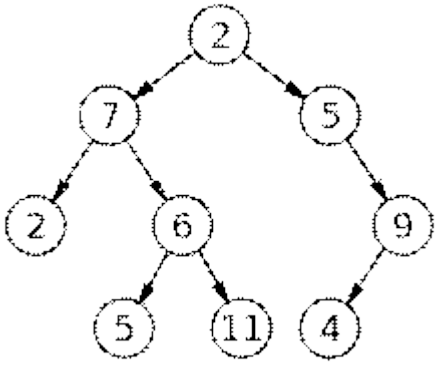
- Associative array (Hash, Map, Dictionary)



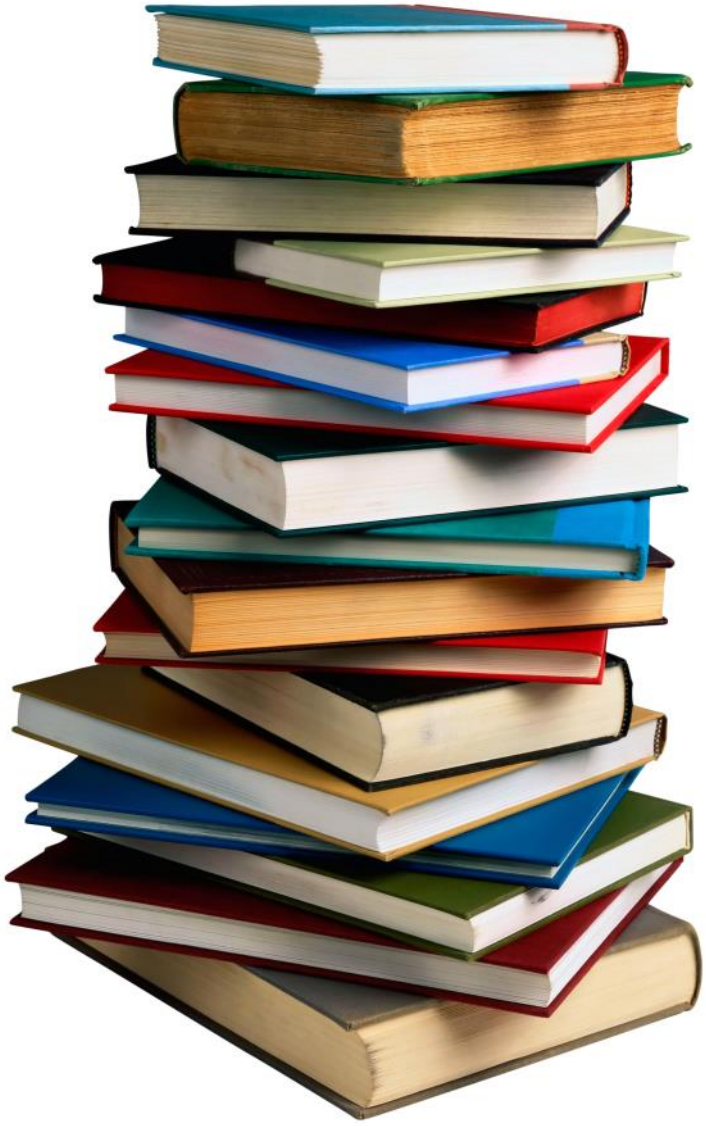
- Graph

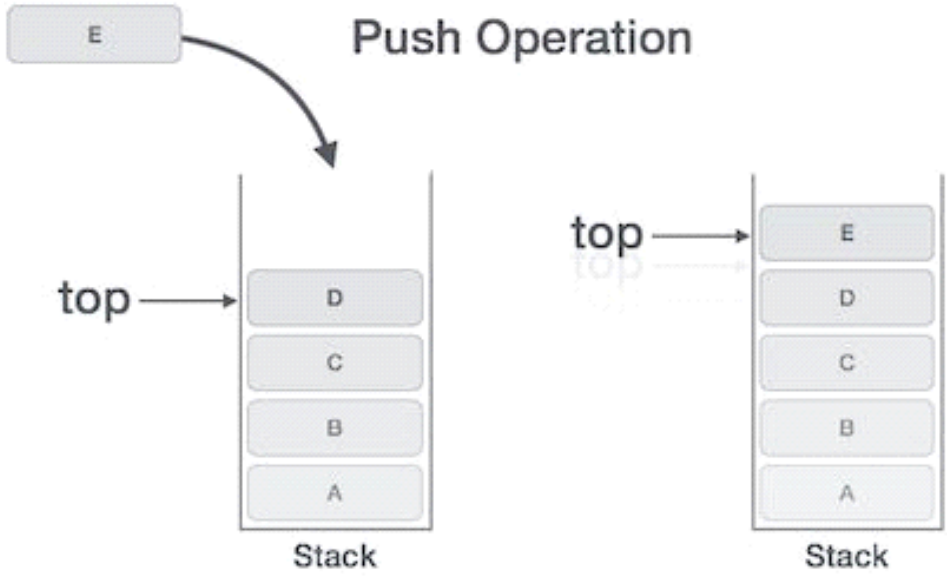


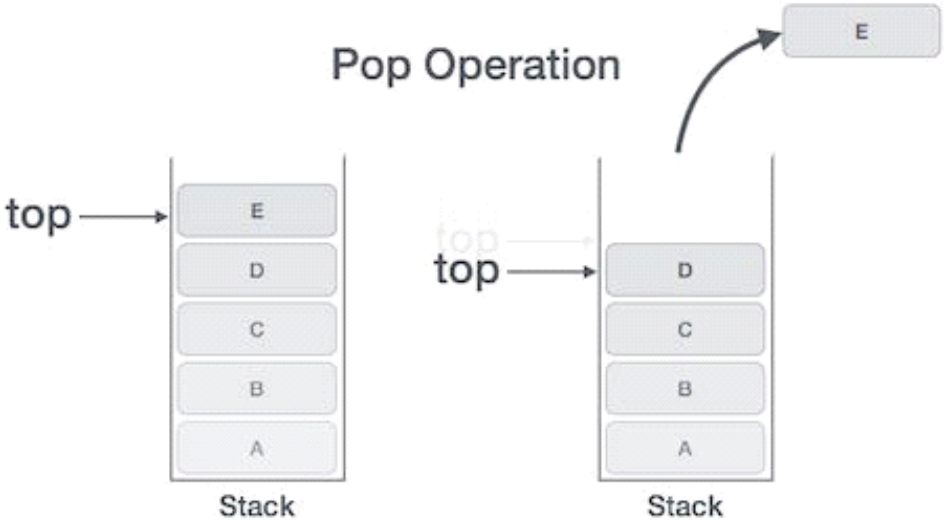
- Tree



ADT Stack





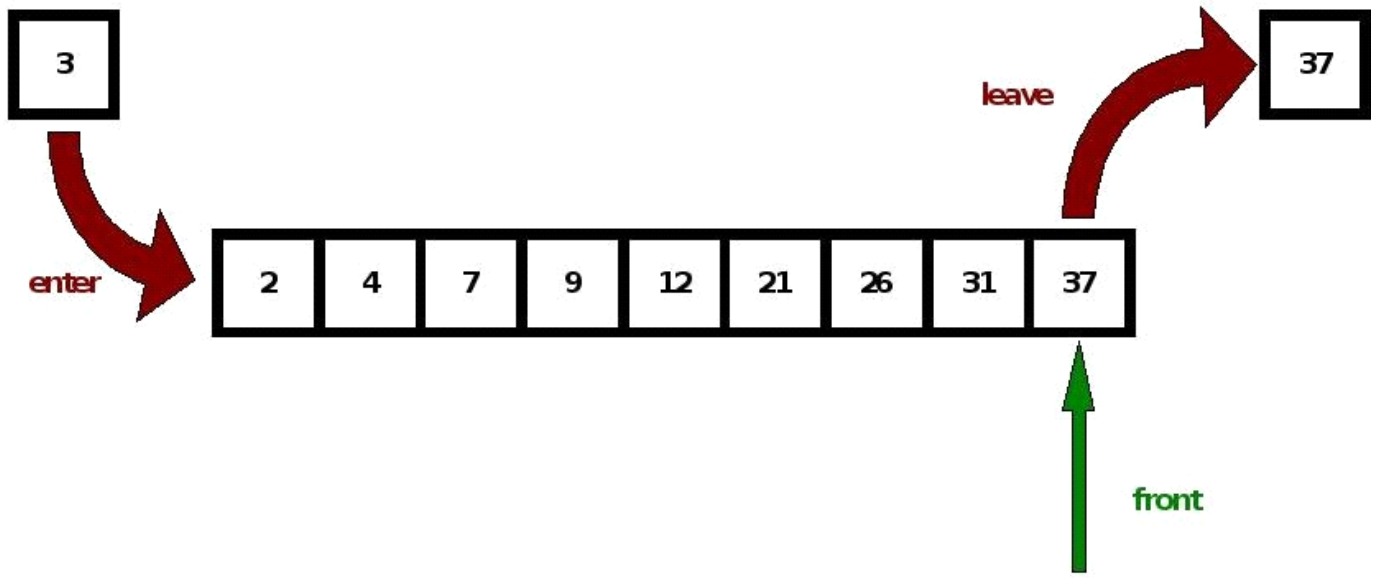


- Операции над стеком
  - initialize
  - push
  - pop
  - top
  - empty
  - size

- Примеры использования
  - Определение палиндрома
  - Сбалансированные скобки
  - Конвертация чисел между системами счисления

- Разбор арифметических выражений

ADT Queue



- Операции над очередью
  - initialize
  - enqueue
  - dequeue
  - front
  - rear
  - empty
  - size

- Примеры использования
  - Игра Hot Potato
  - Поразрядная сортировка
  - Создание пар из списков
  - Ханойская башня
  - Priority queue
    - Очередь в банк
  - Deque
    - Проверка палиндрома

Что мы изучили?

- Сложность алгоритмов
- Алгоритмы поиска
- Структуры данных
- Абстрактные типы данных
- Рекурсия

Что почитать?

- Документация Ruby
  - API documentation for Ruby 2.4.0
  - Ruby 2.4.0 Standard Library Documentation

Что дальше?

- Ruby. Уровень 2. Создание интернет - приложений в среде Ruby on Rails