Лекция 4

ХЭШи

- al Hash. [] ("flat", 3, "curved", 2)
- a2 Hash. [] ("flat"=>3, "curved"=>2)
- bl Hash["flat",3,"curved",21
- b2 Hash["flat"=>3,"curved"=>2]
- cl = {"flat",3,"curved",21}
- c2 = {"flat"=>3,"curved"=>21

ХЭШи

- d = Hash.new # Создать пустой хэш
- e = Hash.new(99) # Создать пустой хэш
- f = Hash.new("a"=>3) # Создать пустой хэш
- e["angled"J # 99
- e.inspect # {}
- f["b"J # {"a"=>3} (значением по умолчанию
- # является тоже хэш)
- f.inspect # !}

ХЭШи

- a= Hash.new("missing")
- a["hello“]
- a.default="nothing"
- a["hello"]
- a["good"] << "Ьуе"
- a.defalt
- # объект по умолчанию - строка "missing"
- # "мissing"
- # "nothing"
- # "noth::.ngbye"
- # "noth:.ngbye"

ХЭШи

- a.store("angled",5)
- a.fetch("flat") # 3
- a.[] ("flat") # 3
- a ["flat"] # 3
- a["bent"] # nil

ХЭШи

- a = (1=>2, 3=>4}
- b = a.shift
- a = {1=>1, 2=>4, 3=>9, 4=>16}
- a.delete(3) 9
- a.delete(5)
- a.delete(6) { "не найдено" }
- delete_if
- reject
- reject!

ХЭШи

- {“a”=>3,"Ь"=>2} .each do |key, val|
- print val, "из", key, ";“ # 3 из a ; 2 из Ь ;
- end
- each_key
- each_value
- Ь = a.invert
- a = {"a"=>l,"b"=>2}
- a.has_key? ''c'' # false
- a.include? ''a'' # true
- a.key? 2 # false
- a.memЬer? "Ь" # true

ХЭШи

- a.empty? # false
- a. length # 2
- a.has_value? 2 # true
- a.value? 99 # false
- h = {"a"=> 1, "b" => 2)
- b=h.to_a # [["a", 1], ["b", 2]]
- a=b.to_h

ХЭШи

- h.keys
- h.values
- h = {l=>"one",2=>"two",3=>"three",4=>"four","cinco"=>"five"!
- h.values_at(3,"cinco",4)
- h.values_at(l,3)

ХЭШи

- names.detect { |k, v | v=="tucker" }
- **# [“joe”, “tucker”]**
- names. find {|k, v | v==v. upcase }
- select
- find_all
- list = names.sort
- list1 =list.to_h
- new_dict = dict.merge(added)
- new_dict = dict.rnerge(added) { |key,old,new| old < new? old: new }

ХЭШи

- a = {"a" => 1,"b" => 2,"z" => 3}
- b = {"x" => 99,"y" => 88,"z" => 77}
- intersection = a.keys & b.keys
- difference = a.keys - b.keys
- c = a.dup.update(b)
- inter = {}
- intersection.each { |k| inter[k]=c[k]}
- # inter равно {"z"=>77}
- diff={}
- difference.each { |k| diff[k]=c[k]}
- # diff равно {"a"=>l, "b"=>2}

Перечислимые структуры

- sum = names.inject(0) { |x,n| x+n}
- sum = O
- nums.each { |n| sum += n }
- words = %w[alpha beta gamma delta epsilon eta theta]
- longest_word = words.inject do |best,wl
- w.length > best.length? w : best
- end

Перечислимые структуры

- flag1 = nums.any? { |x| x % 2 == O}
- flag1 = nums.all? { |x| x % 2 == O}
- flag1= list.all?
- squares = nums.partition do |x|
- Math.sqrt(x) .to_i**2 == x
- end
- mod3 = nums.group_by { |x| x % 3}
- arr.each_slice(3) do …
- arr.each_cons(3) do

Перечислимые структуры

- each
- getline
- each_line
- list.each.with_index do |x,i|…
- find_index
- first
- last
- one?
- none?
- count
- minmax

блоки

Блоки

Перечислимые структуры

- hash = {1 => 2, 3 => 6, 4 => 8, 3 => 10, 7 -> 14}
- arr1 = hash.take(2)
- arr2 = hash.take_while {|k,v| v <= 8}
- arr1 = hash.drop(2)
- range.reduce(2,:*)
- array.to_set
- array.to_json

Перечислимые структуры

- hash = {1 => 2, 3 => 6, 4 => 8, 3 => 10, 7 -> 14}
- arr1 = hash.take(2)
- arr2 = hash.take_while {|k,v| v <= 8}
- arr1 = hash.drop(2)
- range.reduce(2,:*)
- array.to_set
- array.to_json

Множества

- s1 = Set[3,4,5]
- arr = (3, 4, 5]
- s2 Set.new(arr)
- s3 = Set.new(arr) { |x| x.to_s }
- a = x.union(y)
- b = x l y
- c = x + y
- a = x.intersection(y)
- b = x & y
- diff = Set[l,2,3] - Set[3,4,5]

Множества

- member?
- include?
- empty?
- clear
- x = Set[3,4,5]
- y= Set[3,4]
- x.subset?(y) # false
- y.subset?(x)
- proper_ subset?
- superset?
- classify блок

Файлы

- file1 = File.new("one")
- file2 = File.new("two", "w")
- out = File.new("captains.log", "w")
- out.close
- trans = File.open("transactions", "w")
- File.open ("sor.iefi1.e", "w") do |file|
- file.puts "Строка l"
- file.puts "Строка 2"
- file.puts "Третья "
- end
- File.new("filel", "r+")
- File.new("file2", "w+")
- tell
- pos

Файлы

- diskfile = File.new (“foofile", "w")
- puts "Привет ... "
- $stdout = diskfile
- puts "Пока!"
- diskfile. close
- $stdout = STDOUT
- puts "Это все.“
- readline
- readlines

Файлы

- diskfile = File.new (“foofile", "w")
- puts "Привет ... "
- $stdout = diskfile
- puts "Пока!"
- diskfile. close
- $stdout = STDOUT
- puts "Это все.“
- readline
- readlines

Ввод вывод

- print "Привет ... "
- STDOUT.flush
- sleep 10
- print "Пока'\n«

БД

- require 'mysql2'
- client = Mysql2: :Client.new(
  :host => "localhost",
  :username => “root”)

- client.query("CREATE DATABASE list")

- client.query("USE list“)

- client.query("CREATE TABLE members (
  name varchar(1024), email varchar(:C24))")
- client.query <<-SQL
- INSERT INTO members VALUES
- ('John Doe', 'jdoe@rubynew.com'),
- ('Fred Smith', 'smithf@rubyexpert .com')

ООП

- class ColoredRectangle

```
def initialize(r, g, b, sl, s2)
@r, @g, @b, @sl, @s2 - r, g, b, sl, s2
end


def self.white_rect(sl, s2)
        new(Oxff, Oxff, Oxff, s:, s2)
end


def self.gray_rect(sl, s2)
    new(Ox88, Ox88, Ox88, sl, s2)
end
```

ООП

```
class PersonalComputer
        attr_accessor :manufacturer,
                :mociel, :processor,
:clock,
                :ram, :disk, :monitor,
                :colors, :vres, :hres,
::1et
        def initialize &block
                instance_eval &ьlock
        end
end
```

```
desktop = PersonalComputer.new do
self.manufacturer = "Acme"
self.model = "THX-1138"
self.processor = "986"
self.clock = 9.6
self.ram = 16
self.disk = 20
self.monitor = 25
self.colors = 16777216
self.vres = 1280
self.hres = 1600
self.net = "T3 11
end
```

ООП

```
class Metal
        @@current_temp = 70
        attr accessor :atomic_number

        def self.current_temp=(x)
                @@current_temp = x
        end
        def self.current_temp
                @@current_temp
        end
        def liquid?
                @@current_temp >= @melting
        end
        def initialize(atnum, melt)
                @atomic number = atnum
                @melting = melt
        end
end
```

ООП

```
class Person
        attr reader :name, :age, :pay_scale
        protected :age
        private :pay_scale

        def initialize(name, age, pay_scale)
                @name, @age, @pay_scale = name, age, pay_scale
        end

        def <=>(other)
                age <=> other.age
        end

        def same_rank?(other)
                pay_scale == other.pay_scale
        end
        def rank
                case pay_scale
                        when 1 .. 3
                                “lower”
                        when 4 .. 6
                                "middle"
                        when 7 ..9
                                "high"
                end
        end
end
```