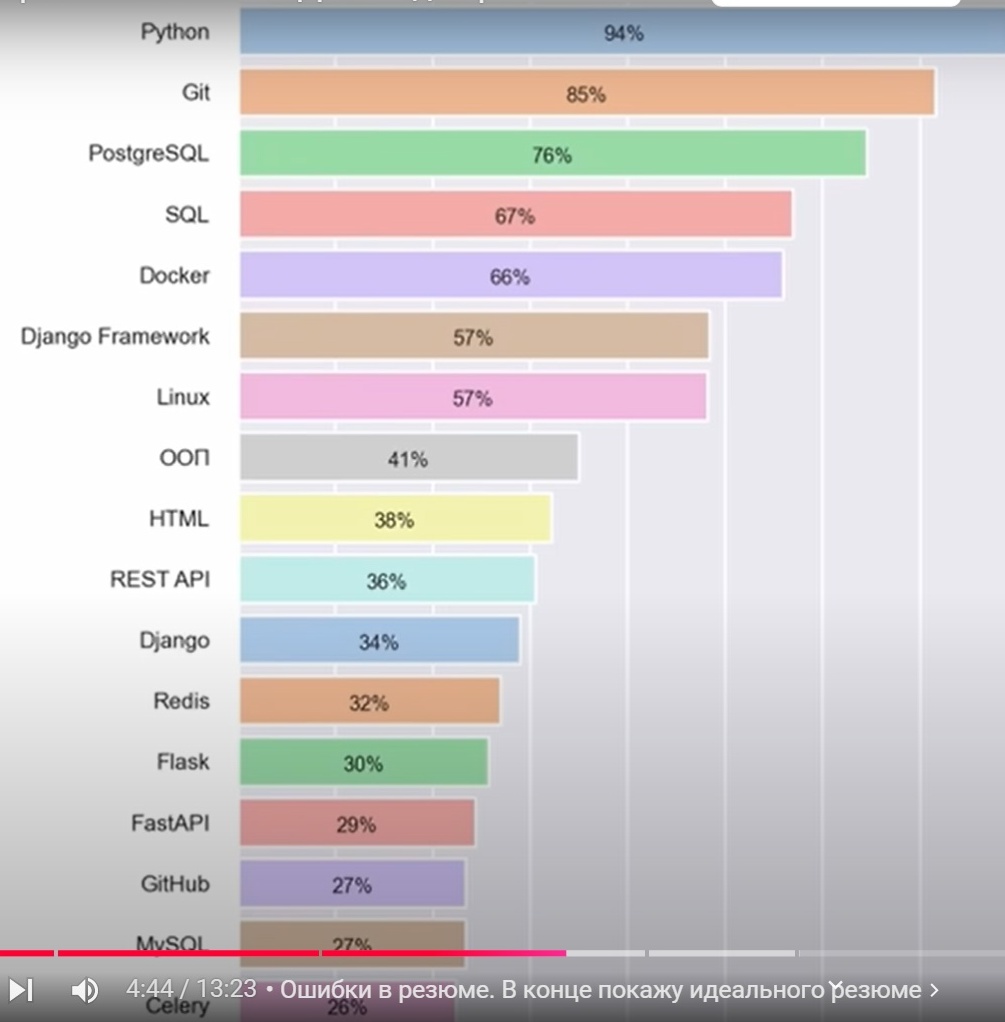
Ruby начал учить в начале 2019, но оказалось, что к Ruby надо знать Javascript, верстку, библиотеку на js,



**Урок 1**

Ruby - язык программирования, написанный на Си.

Ruby on rails - фреймворк, для быстрого создания веб приложений. Это гем (пакет) для расширения возможностей языка Ruby.

Что необходимо для изучения языка ruby

интерпретатор ruby

редактор. Например sublime text, notepad++, встроенный редактор far commander.

shell утилиты для linux так и windows

программа git. Позволяет хранить разные версии кода

rubymine (при возможности)

sql базы данных (SQLite, MySQL, PostgreSQL)

sql оболочка

терминал

установить файловый менеджер far: Windows Linux

RubyInstaller - это ruby окружение для Windows

Если планируйте перейти на Linux то необходимо выбрать Linux Mint Cinnamon

**PostgreSQL** - система управления базами данных - СУБД

**Docker -** это платформа, предназначенная для разработки, доставки и запуска приложений с помощью контейнеров. Контейнеры позволяют разработчикам упаковывать приложение с его всеми зависимостями в стандартизированную единицу для разработки программного обеспечения

**DevOps** (от англ. development & operations) — методология автоматизации технологических процессов сборки, настройки и развёртывания программного обеспечения.

Она предполагает активное взаимодействие специалистов по разработке со специалистами по информационно-технологическому обслуживанию и взаимную интеграцию их технологических процессов друг в друга для обеспечения высокого качества программного продукта.

Методология фокусируется на стандартизации окружений разработки с целью быстрого переноса программного обеспечения через стадии жизненного цикла ПО, способствуя быстрому выпуску версий программного продукта.

Преимущества DevOps:

— сокращение времени выхода на рынок;

— улучшение удовлетворенности клиентов;

— улучшение качества продукции;

— более надёжные выпуски;

— повышение производительности и эффективности;

— увеличение способности создавать правильный продукт путём быстрого экспериментирования.

**Heroku** — это популярная платформа для развёртывания веб-приложений, которая позволяет быстро и легко запускать приложения даже тем пользователям, которые не знакомы с DevOps. Стартапы обратились к Heroku, когда он только появился, потому что он позволял небольшим командам тестировать, развёртывать и запускать свои приложения без специального инженера DevOps. Сейчас Heroku поддерживает несколько языков, но изначально он поддерживал только Ruby.

**Vagrant** — это инструмент для управления виртуальными машинами из командной строки. Он позволяет разработчикам создавать и запускать проекты, разработанные для конкретной операционной системы, в любой операционной системе. Например, разработчик может создавать сервисы, предназначенные для работы в Linux, на своём ноутбуке Mac или Windows и быть уверенным, что они будут корректно работать при развёртывании. Vagrant написан на Ruby, и среды разработки Vagrant должны быть настроены с помощью Ruby.

**Chef и Puppet** — это инструменты, которые инженеры DevOps используют для автоматизации и управления конфигурацией веб-приложений и серверов. С помощью Chef или Puppet инженеры могут определять среды приложений с помощью простого файла конфигурации, а затем использовать эту конфигурацию для запуска экземпляров приложения без каких-либо действий вручную на любой платформе.

**Passenger, Unicorn и Puma** — это веб-серверы, написанные на Ruby. Они обрабатывают необработанные входящие HTTP-запросы, отправляют их в соответствующие серверные веб-приложения, а затем обрабатывают HTTP-ответ, отправленный приложением. Серверы веб-приложений на Ruby работают в паре с фреймворками веб-разработки, написанными на Ruby.

**Vessel**, используются для сканирования и загрузки веб-страниц.

**Nokogiri** ещё одна популярная библиотека Ruby, используется для анализа HTML-кода на сканированных страницах, чтобы извлечь определённые части страницы в структурированном виде и использовать их для анализа данных

**Урок 2 Тема урока: переменные**

Комментарий (однострочный) в ruby - #

Если идёт дублирование данных, то лучше вынести эти данные в переменную.

Переменная это значение которое можно менять или использовать в разных частях программы.

Переменные задаются до вывода на экран.

gets (от англ. get string) - метод получения строки из стандартного устройства ввода из консоли, например, клавиатуры.

Спец.символы:

\n - перевод строки

После получения строки в переменную методом gets также передаётся символ перевода строки (\n). Его можно убрать при помощи метода chomp.

name = gets.chomp

print - оператор не переводит строку после своей работы, в отличие от puts, то есть

puts "Hello" тоже самое, что print "Hello\n"

#{foo} - вызов переменной внутри кавычек

функция = метод = процедура

объект = экземпляр класса

**Урок 3**

**Ruby** — динамический, строго типизированный язык с глубоким уклоном в объектно-ориентированную и функциональную парадигму программирования.  
# Строгая типизация, число нельзя умножить на строку.  
  
В Ruby очень много метапрограммирования, встроенного во все части языка. Это значит, что программа может менять свое поведение прямо во время работы практически в любую сторону

функция = метод = процедура.

Методы определяются с помощью ключевого слова def (сокращение от «define» - определять)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

работа в irb (если ввести x.methods – выведутся все доступные методы, которые можно применить для х)

ctrl+d – выйти из irb

nil – нулевой объект

типы данных: string – строка, float – плавающий (2,2), integer – простое число (2), array [..] – массив, hash {..} хэш

в

новый файл: шифт +f4

новый каталог f7

просмотр файла f3

редактировать f4

копия программы f5

контр+О - запустить

.chomp - удаляет завершающий символ перевода новой строки (\n)

.class – определить тип данных в irb. (3.14 – float)

irb - работа в редакторе (интерпритатор), exit - выход

Преобразование типов данных:

.to\_i - в число

.to\_f - в число с плавающей точкой

.to\_s - в строку

puts 'result= ' +x.to\_s (если пишется result, то х нужно присваивать значение строки, чтоб можно было вывести строку и число вместе)

Основные типы данных в Ruby:

string - "2"

float - 2.1

array - [...]

hash - {...}

Integer - 2

**Урок 4**

!-удаляет символ новой строки у переменной (пример x=x.chomp равняется x.chomp!)

#{ } - значение переменной (интерполяция), если кавычки одинарные интерполяция строк не получится!

Следующие строки равнозначны

# Склеивание

puts "x = " + x + " y = " + y.to\_s + " z = " + z.to\_s

# с интерноляцией строк

puts "x = #{x} y = #{y} z = #{z}"

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

x.times {puts ‘y’} – х раз вывести у

|  |  |
| --- | --- |
| то же самое | |
| 10.times { print "Chunky bacon!" } то же само что и | 10.times do print "Chunky bacon!" end |

for k in 1..3

print k

end

выведет от 1 до 3

|  |  |
| --- | --- |
| то же самое | |
| i = 3  while i > 0 do  print i  i -= 1  end | j = 3  until j == 0 do  print j  j -= 1  end |

**Цикл**

m=0

loop do

m += 1 # тоже самое, что и m=m+1

print "Ruby!"

break if m == 30

end

# выведет Руби 30 раз

10.times {|i| puts i.to\_s}

выведет от 0 до 9

\_\_\_\_\_\_\_\_\_\_\_\_\_

|| - pipe (труба)

Конкатенация – это объединение двух или более строк в одну новую строку. Чтобы объединить строки, используйте оператор конкатенации +

**урок 5**

def square(n)

puts n \*\* 2

end

square(12) # prints "144"

параметр n (для «числа»), аргумент 12 при вызове/

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

def by\_three?(number)

if number % 3 == 0

return true

else

return false

end #если число делится на 3, то будет тру, если нет, то фалс

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

аргументы-разделители. Аргументы-разделители — это аргументы, перед которыми стоит звёздочка \*. Это сообщает программе, что метод может получить один или несколько аргументов.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_

strip - удаляет пробелы

\t - табуляция

\r - возврат "каретки" (печатной машинки)

**урок 6**

sleep

10.times {puts "aaa"}

10.times do

end

rand - случайное число

**урок 7**

**==**, который является оператором сравнения (также называемым оператором отношения). == означает «равно». Если мы уже использовали = для присваивания.

is\_true = 2 != 3 (2 не равно 3)

is\_false = 2 == 3 (2 равно 3)

компараторы должны сравнивать два значения друг с другом, чтобы получить true или false

Вы также можете использовать логические или булевы операторы. В Ruby их три: и (&&), или (||), и не (!). Булевы операторы возвращают булевы значения: true или false.

Логический оператор и, && приводит к true только в том случае, если оба выражения по обе стороны && являются true. Вот как работает &&:

Пример:

true && true # => true

true && false # => false

false && true # => false

false && false # => false

В Ruby также есть оператор или (||). В Ruby || называется инклюзивным или, потому что он вычисляет true тогда, когда одно или другое или оба выражения истинны. Проверьте сами:

Пример:

true || true # => true

true || false # => true

false || true # => true

false || false # => false

в Ruby есть логический оператор not (!). ! делает значения true значениями false, и наоборот.

Пример:

!true # => false

!false # => true

if a==1 (a равно 1)

if a!=1 (a не равно 1)

counter = counter + 1 то же самое что и counter + =1

n<1 (n меньше 1)

n>1 (n больше 1)

n<=1 (n меньше или равно 1)

n>=1 (n больше или равно 1)

if a==3 && b>4

exit-завершение программы

**урок 8**

когда вы знаете, сколько раз будете выполнять цикл, и в этом случае вам понадобится цикл for. пример: for num in 1...10  
puts num  
end

Причина, по которой эта программа считала до 9, а не до 10, заключалась в том, что мы использовали три точки в диапазоне; это говорит Ruby исключить конечное число при подсчете: for num in 1...10 означает “перейти к, но не включать 10”. Если мы используем две точки, например, for num in 1..10, это указывает Ruby включить наибольшее число (10) в диапазон

Оператор if в Ruby принимает выражение, которое является просто красивым словом для обозначения чего-то, что имеет значение, равное true или false. Если это выражение равно true, Ruby выполняет блок кода, следующий за if. Если оно не истинно (то есть равно false), Ruby не выполняет этот блок кода: он пропускает его и переходит к следующему.  
  
Параллельным оператором для if является оператор else. Оператор if/else говорит Ruby: «Если это выражение истинно, выполните этот блок кода; в противном случае выполните код после оператора else.  
  
Оператор elsif может добавить любое количество альтернатив к оператору if/else  
  
Оператор unless (если не/за исключением)  
пример:  
hungry = true  
unless hungry  
puts "I'm writing Ruby programs!"  
else  
puts "Time to eat!"  
end

\_\_\_\_\_\_\_\_\_\_\_\_

итератор else - иначе (только одно условие)

elsif - иначе (много условий)

while - пока (while 2+2==4 - делать что-то\_

оператор **case** - будет определять, какие действия выполнять в зависимости от того, что введёт пользователь.

case language

when "JS"

puts "Websites!"

when "Python"

puts "Science!"

else

puts "I don't know!"

end

else — это то, что сделает оператор case, если ни один из его операторов when не соответствует case (в данном случае значению language)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

метод reverse - переворачивает строку (hello - olleh)

итератор each (каждый) (1..5).each = 1,2,3,4,5 а если (1...5).each = 1,2,3,4

**урок 9**

метод def print\_1

end (вызов процедуры 1)

return - возврат значения из функции (выход из функции)

@-глобальная переменная

instance - экземпляры класса (объекты)

символ : (для ускорения и оптимизации)

|  |  |
| --- | --- |
| 1 способ | 2 способ |
| old\_hash={ :one => 1, :two => 2, :three => 3 } | new\_hash = { one: 1, two: 2, three: 3} |

Вы ставите двоеточие в конце символа, а не в начале. Вам больше не нужна хэш-ракета.

Важно отметить, что, несмотря на то, что у этих клавиш двоеточие стоит в конце, а не в начале, это всё равно символы!

array - массив (набор объектов (строки, числа и тд))

arr[] –создать массив

пример

demo\_array = [100, 200, 300, 400, 500]

print demo\_array[2]

#вывод 300

\_\_\_\_\_\_\_\_\_\_

arr.sort! – сортировка по порядку (есть два метода сортировки: .sort или sort!. Первый метод, .sort просто возвращает отсортированный массив, не изменяя исходный. Второй метод, .sort! изменяет исходный массив)

arr.uniq - вывод без повторов

arr.reverse - вывод в обратном порядке

arr<<1 - добавить в массив 1

\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**вывести только key или value**

movie\_ratings = {  memento: 3,  primer: 3.5,  the\_matrix: 3,  truman\_show: 4, red\_dawn: 1.5, skyfall: 4, alex\_cross: 2, uhf: 1, lion\_king: 3.5}

movie\_ratings.each\_key {|k|puts k} #выведет только названия фильмов

**пример сортировки**:

books = ["Charlie and the Chocolate Factory", "War and Peace", "Utopia", "A Brief History of Time", "A Wrinkle in Time"]

books.sort! { |firstBook, secondBook|   secondBook<=>firstBook}#<=> это оператор комбинированного сравнения

puts books

**Сортировка в обратном порядке**

Оператор комбинированного сравнения выглядит так: <=> (Он возвращает 0, если первый операнд (сравниваемый элемент) равен второму, 1 если первый операнд больше второго, и -1 если первый операнд меньше второго)

books = ["Charlie and the Chocolate Factory", "War and Peace", "Utopia", "A Brief History of Time", "A Wrinkle in Time"]

# To sort our books in ascending order, in-place

books.sort! { |firstBook, secondBook|   secondBook<=>firstBook}

# Sort your books in descending order, in-place below

puts books

**Сортировка в алфавитном и обратном порядке**

def alphabetize(arr, rev=false) # определяем метод alphabetize, arr – это массив, параметр rev – означает, что у метода  alphabetize будет второй параметр для обратного порядка, который в начале равен false

  if rev

    arr.sort { |item1, item2| item2 <=> item1 }

  else

    arr.sort { |item1, item2| item1 <=> item2 }

  end

end

books = ["Heart of Darkness", "Code Complete", "The Lorax", "The Prophet", "Absalom, Absalom!"]

puts "A-Z: #{alphabetize(books)}"

puts "Z-A: #{alphabetize(books, true)}"

вывод

A-Z: [«Авессалом, Авессалом!», «Код завершен», «Сердце тьмы», «Лоракс», «Пророк»]

Z-A: [«Пророк», «Лоракс», «Сердце тьмы», «Код завершен», «Авессалом, Авессалом!»]

метод **.zip** - преобразует любые элементы в массивы, затем объединяет элементы объекта с соответствующими элементами каждого из аргументов. Пример:

string\_AZ = Hash[("a".."z").to\_a.zip((1..26).to\_a)]#связывает строковые буквы с их местом в алфавите (“a” с 1, “b” с 2 ...)

метод **.select -** отфильтровавает хэш по значениям, которые соответствуют определённым критериям. Пример:

grades = { alice: 100, bob: 92, chris: 95, dave: 97}

grades.select { |name, grade| grade < 97 } # ==> { :bob => 92, :chris => 95 }

grades.select { |k, v| k == :alice } # ==> { :alice => 100 }

\_\_\_\_\_\_\_\_\_\_\_\_\_

languages = ["HTML", "CSS", "JavaScript", "Python", "Ruby"]

languages.each {|x|puts x} #выведет 5 языков прогр.

\_\_\_\_\_\_\_\_\_\_\_\_\_

вывод двумерного массива

s = [["ham", "swiss"], ["turkey", "cheddar"], ["roast beef", "gruyere"]]

s.each { |sub\_array| sub\_array.each { |element| puts element }} #выведет 6 наименований

\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**урок 10**

arr.each do |item| - выполнить что то для каждого элемента массива. item – элемент массива

**создать массив**

|  |  |
| --- | --- |
| 1 способ (если каждый элемент выражается одним словом). Метод %w{} позволяет опустить кавычки и запятые между элементами. | 2 способ |
| days = %w{Monday Tuesday Wednesday Thursday Friday Saturday Sunday} | days =  ["Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday", "Sunday"] |

strings = ["HTML", "CSS", "JavaScript", "Python", "Ruby"]

symbols = [] #создаем новую переменную и пустой массив

strings.each do |s| #перебираем массив строк из стринг в символы

 symbols.push(s.to\_sym) #добавляем символы в пустой массив симболс

end

print symbols # [:HTML, :CSS, :JavaScript, :Python, :Ruby]

to\_sym, то же самое что и .intern – преобразование строки в символ. Пример: "hello".intern

# :hello

\_\_\_\_\_\_\_\_\_\_\_\_\_

Четыре известных вашей программе глагола — добавлять, отображать, обновлять и удалять — универсальны. Эта аббревиатура более известна как **CRUD** - create, read, update и delete (соответственно). Это действия, которые вы предпринимаете, когда обновляете запись в базе данных, запрашиваете информацию на веб-сайте или пишете сообщение в блоге. Хорошо быть знакомым с этой настройкой, потому что вы увидите ее во всем, от вызовов API до веб-фреймворков, таких как Ruby on Rails.

movies = {StarWars: 4.8, Divergent: 4.7}

puts "What would you like to do? "

choice = gets.chomp

case choice

when "add"

  puts "What movie would you like to add? "

  title = gets.chomp

  if movies[title.to\_sym].nil?

    puts "What rating does the movie have? "

    rating = gets.chomp

    movies[title.to\_sym] = rating.to\_i

  else

    puts "That movie already exists! Its rating is #{movies[title.to\_sym]}."

  end

when "update"

  puts "What movie would you like to update? "

  title = gets.chomp

  if movies[title.to\_sym].nil?

    puts "That movie does not exist."

  else

    puts "What is the new rating? "

    rating = gets.chomp

    movies[title.to\_sym] = rating.to\_i

  end

when "display"

  movies.each do |title, rating|

    puts "#{title}: #{rating}"

  end

when "delete"

  puts "What movie would you like to delete? "

  title = gets.chomp

  if movies[title.to\_sym].nil?

    puts "That movie does not exist."

  else

    movies.delete(title.to\_sym)

  end

else

  puts "Error!"

end

arr[a,b,c,d,e]

arr [1,2][1] - создается новый массив (b,c)из массива (a,b,c,d,e) и выводится b

\_\_\_\_\_\_\_\_\_\_\_\_\_\_

.delete\_at 0 - удалить из массива

.delete item - удалить сам объект из массива

\_\_\_\_\_\_\_\_\_\_\_\_\_\_

[[mike, jassie],[sasha, nisha]] - объединение массивов

рефакторинг - улучшение программы

игра камень ножницы бумага – я не дописал (не понял item)

**урок 11**

a, b = b,a – поменять переменные местами ( задания на собес) у меня не получилось

\_\_\_\_\_\_\_\_\_\_\_\_\_

arr.each\_with\_index do |item,i| - метод заменяющий

puts “#{i+1} #{item}”

end

\_\_\_\_\_\_\_\_\_\_\_\_

**хэши**

ключи должны быть уникальными, но значения могут повторяться

key value storage

key (ключ (Mike (имя)

value(значение(76647-номер тел)

h или hh {} – хэш

hh={‘Mike’=>’ 76647’, ‘Jassie’=>’66557’}

puts hh

=> hash rocket

\_\_\_\_\_\_\_\_\_\_\_\_

в методе хэш идет после параметров, например

def mm par1, hash

end

\_\_\_\_

hh=[‘Mike’] =76647, где Майк – это key (ключ), а 76647 – это value (значение)

puts number [‘mike’] – показать телефон майка

\_\_\_\_\_\_\_

hh.each do |key, value| #each – это метод, который принимает блок кода и запускает этот блок кода для каждого элемента в списке

end

**урок 12**

**словарь:** hh={'dog'=>['собака','пес']'cat'=>['кошка','кот'] 'girl'=>['девушка']} , где dog – это ключ (key), а собака – массив (value)

hh=[‘mike’]<<’eee’ – добавить ‘еее’ в массив. Второй вариант hh=[‘mike’].push…

ассоциативные массивы (hashtable)

|  |  |
| --- | --- |
| массив | хэш |
| arr=[] | hh={} |
| содержатся элементы (от 0 до N) | содержатся пары (ключ->значение) key ->value |
| arr=[1] | hh={key} |

----------

arr.[rand(0..(arr.size-1))] – обратиться к случайному элементу массива

\_\_\_\_\_\_\_

hh.keys – перечисление ключей

hh.value – перечисление значений

\_\_\_\_\_\_\_\_\_\_\_\_\_\_

.each do |key, value| - – перечисление ключей, значений

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

.each\_key do |key| - другой вариант

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Вывести количество значений (ответ 5)

hh={'dog'=>['собака','пес'],'cat'=>['кошка','кот'], 'girl'=>['девушка']}

x=0

hh.each\_value do |value|

x=x+value.size

end

puts x

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

проверка значения (value) в словаре

hh={'dog'=>'собака'}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

hh.has\_key? 'cat' – функция покажет в irb истину или ложь (если в функции знак вопроса, то ответ будет истина или ложь), если написать puts hh.has\_key? 'cat' – то выведет на экран

**первый способ проверки Есть ли ключ в хэше**

if hh.has\_key? 'cat'

puts ‘в словаре есть слово кошка’

end

\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**второй способ**

if hh [‘cat’]

puts ‘в словаре есть слово кошка’

end

\_\_\_\_\_\_\_\_\_\_\_

третий способ проверки значения

\_\_\_\_\_\_\_\_\_\_\_\_

**бесконечный цикл**

Итератор — это просто метод Ruby, который многократно вызывает блок кода. Блок кода — это фрагмент, содержащий инструкции, которые нужно повторять.

Самый простой итератор — это метод loop . Вы можете создать базовый (но бесконечный!) цикл, просто набрав

loop { print "Hello, world!" }  
В Ruby фигурные скобки ({}) обычно взаимозаменяемы с ключевыми словами do (для открытия блока) и end (для его закрытия). Зная это, мы можем написать более эффективный цикл, чем тот, что приведён выше:

i = 0  
loop do  
  i += 1  
  print "#{i}"  
  break if i > 5  
end

\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Ключевое слово break — это наша карта «Выход из тюрьмы»: она прерывает цикл, как только выполняется условие.

Итератор loop является самым простым, но и одним из наименее мощных. Более полезным итератором является метод .each, который может применять выражение к каждому элементу объекта по очереди. Синтаксис выглядит следующим образом:

object.each { |item| }  
Вы также можете использовать do ключевое слово вместо {}:

object.each do |item|   
end  
Имя переменной между | | может быть любым: это просто заполнитель для каждого элемента объекта, для которого вы используете .each

пример

array = [1,2,3,4,5]

array.each do |x|

  x += 10

  print "#{x}"

end

\_\_\_\_\_\_\_\_\_\_\_

Есть два способа зациклить

|  |  |
| --- | --- |
| 1. способ  loop do  break  end | 2. способ  while true (2+2==4 или !=nil)  break  end |

**Вычисляем сумму чисел из массива:**

numbers = [5, 2, 8]

sum = 0

numbers.each do |n|

sum += n

end

puts sum

**Урок 13**

Метод def say\_hi name, age

puts “hi #{name}”

return

end

\_\_\_\_\_

**создать хэш**

|  |  |
| --- | --- |
| 1 способ | 2 способ |
| hh={‘mike’=>66”} | hh=Hash.new #Hash должно быть написано с заглавной буквы  hh=[‘dand’]=’dog’ |

Пример вывода значения:

matz = { "First name" => "Yukihiro",

  "Last name" => "Matsumoto",

  "Age" => 47,

  "Nationality" => "Japanese",

  "Nickname" => "Matz"

}

matz.each do |key, value|

  puts value

end

**Массив** — это просто список элементов в квадратных скобках, например: [1, 2, 3, 4]

arr=[] и второй способ arr=Array.new

\_\_\_\_\_\_

**инициализация хэша**

первый способ

hh={}

hh[‘mike’]=66

hh[‘jassie’]=26

второй способ

hh={‘mike’=>66, ‘jassie’=>26}

**способы вывести хэш на экран**

1. puts hh

2. puts hh.inspect (как написано в программе (в строку с кавычками и тд))

3. hh.each do(k,v)

puts “#{k}…#{v}”

end

4. puts hh.keys – вывести все ключи

5. puts hh.keys.inspect

6. puts hh.value – вывести все значения

7. puts hh.value.inspect

\_\_\_\_\_\_\_\_\_

hh={}

hh.keys.each do |key|

value=hh[key]

end

hh.value.each

\_\_\_\_\_\_\_\_\_\_\_\_\_\_

очистить хэш

hh.clear

\_\_\_\_\_\_\_\_\_\_\_\_\_\_

проверка хэша

ih hh.has\_key? ‘mike’

puts hh [‘mike’]

end

\_\_\_\_\_\_\_\_\_\_\_\_

то же самое в сокращенном синтаксисе

ih hh.key? ‘mike’

\_\_\_\_\_\_\_\_\_\_\_\_

еще вариант проверки значения хэша

if hh [‘mike’]

end

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**удаление**

можно удалить только пару

hh.delete ‘mike’

\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**задание Записная книжка**

@hh={}

def add\_person name,age

#добавляет в пару хэш

@hh[name]=age

end

def show\_hash

#показывает хэш

@hh.keys.each do |key|

age=@hh[key]

puts "name: #{key}, age: #{age}"

end

end

loop do

#добавляет, пока не введена пустая строка

print "enter name: "

name=gets.strip

if name == ''

show\_hash

exit

end

print "enter age: "

age=gets.to\_i

add\_person name,age

end

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**if**

puts ‘ok’ if (true) – например a==10, b==5, пишем puts ‘ok’ if a+b>7, выйдет ok, т.к. 15>7 (это true)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

def add\_person name, age – метод деф, параметры name и age, если захотим добавить третий параметр или поменять местами name и age, то придётся и в других методах все исправлять, поэтому, нужно вместо name и age, написать options, который будет являться хэшем для параметров name и age,

end

def add\_person options

options = {:name=>’mike’, :age=>66}

-------------------------

**merge –** объединяет два хэша

book=book1.merge book2 (объединяет две записные книжки в одну)

def show\_book book

puts "============================"

book.keys.each do |key|

age=book[key]

puts "name: #{key}, age: #{age}"

end

puts "============================"

end

book1 ={'mike'=>65, 'gus'=>55}

show\_book book1

book2 ={'walt'=>50, 'jessie'=>25}

show\_book book2

book=book1.merge book2 #если добавить ! знак после merge, то book= можно не использовать,т.е.

book1.merge! book2

show\_book book

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**>** - символ перенаправления ввода

в консоли пишем название руби файла 2.merge.rb > book.txt и в директории появится текстовый файл

============================

name: mike, age: 65

name: gus, age: 55

name: walt, age: 50

name: jessie, age: 25

============================

**>>** - дописывает в конец файла, а не замещает его содержимое

\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**html**

<body> - значит что дальше пойдет html код

<br> - перевод на новую строку

<br/> - слэш значит, что тэг закрыается

<b> - открывается тэг bold (жирный)

</b> - закрывается тэг bold, пример <b>text</b>

<i> - открывается тэг italic (курсив)

</i> - закрывается тэг курсив

тэг heading (заголовок)

<h1>text</h1>

<h2>text</h2>

<h3>text</h3>

<h4>text</h4>

<h5>text</h5>

**Урок 14**

**ООП**



**инкапсуляция -** это принцип, согласно которому внутреннее устройство сущностей объединяют в специальной «оболочке» и скрывают от вмешательств извне. Доступ к объектам возможен через специальные открытые методы, а напрямую обратиться к их содержимому нельзя. 1

Принцип инкапсуляции гласит, что все важные данные, необходимые для работы класса или объекта, должны в нём же и храниться. Только определённая информация будет доступна для внешних функций и объектов. Данные конкретного объекта или класса хранятся исключительно в пределах этого самого класса или объекта. Вносить в них корректировки, пользуясь другими классами, нельзя. Окружение имеет право запрашивать только публичные (общие) методы и атрибуты. 2

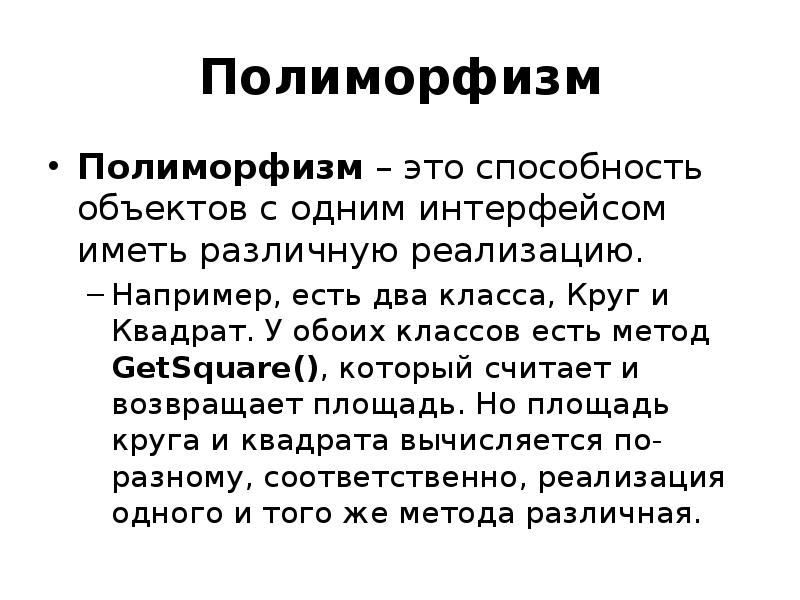
Преимущества инкапсуляции:

Безопасность данных. Инкапсуляция позволяет защитить данные объекта от некорректного доступа и изменения извне. 3

Сокрытие реализации. Когда класс инкапсулирует свою реализацию, то изменения внутри класса не отражаются на внешнем коде. Это позволяет менять реализацию объекта, не нарушая функциональность клиентского кода, что облегчает поддержку и эволюцию программы. 3

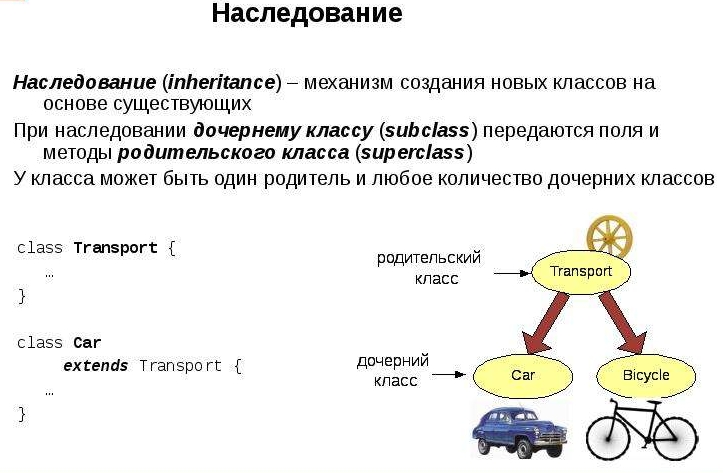
Упрощение интерфейса. Инкапсуляция позволяет предоставить простой и понятный интерфейс для работы с объектами. Клиентский код взаимодействует только с публичными методами класса, не требуя знания деталей его внутренней реализации. 3

Модульность. Инкапсуляция помогает создавать модульные системы, где каждый класс представляет собой отдельный модуль со своими данными и методами. Модули могут взаимодействовать друг с другом через публичные интерфейсы, что способствует повышению читаемости и понимаемости кода.



**полиморфизм -**

**наследование –**



**Класс –** это шаблон (чертеж) какой либо детали, а **объект (экземпляр**) – это деталь класса. т.е. шаблон один деталей много.

Создать класс:

class Animal

def run

puts :’бегу’

end

end

aa=Animal.new

aa=run

где def это метод, а new – это объект

Пример 2.

В классе обязательно объявлять глобальные переменные, если они будут использоваться в разных методах

class Animal

def run

@name = 'dog'

puts "#{@name} is running"

end

def stop

puts "#{@name} is stopping"

end

end

a=Animal.new

a.run

a.stop

Пример 3. Модификация записной книжки

class Book

def initialize #метод инишалайз(конструктор), при инициализации создается глобальная переменная (@hh) которая будет доступна только внутри класса. Этот метод нужен, во-первых чтобы определить массив, а во-вторых, когда нужно вывести методы, но мы еще не знаем в какой последовательности эти методы будут выведены.

@hh={}

end

def add\_person options #объявляем метод в классе add\_person

puts 'alredy exists' if @hh[options[:name]]

@hh[options[:name]] = [options[:age]]

end

def show\_all

@hh.keys.each do |key|

age=@hh[key]

puts "name: #{key}, age: #{age}"

end

end

end

b=Book.new #создает записную книжку и вызывает метод инишалайз, это его особенность, другие методы нужно вызывать, а именно этот вызывается при создании Book

b.add\_person :name =>"walt", :age=>50 #добавляет человека в з.к.

b.show\_all #выводит

\_\_\_\_\_\_\_\_\_\_\_

обращаемся к переменной класса (hh), через метод (def), return писать не обязательно

def aaa

return @hh #обращаемся к переменной класса (hh), через метод (def), return писать не обязательно

end

b=Book.new

puts b.aaa

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**атрибуты**

|  |  |
| --- | --- |
| **1. старый вариант** | **2. с атрибутом** |
| class Book  def initialize  @hh={}  @last\_person = ''  end  def add\_person options  @last\_person = options [:name]  puts 'alredy exists' if @hh[options[:name]]  @hh[options[:name]] = options[:age]  end  def show\_all  @hh.keys.each do |key|  age=@hh[key]  puts "name: #{key}, age: #{age}"  end  end  **def last\_person**  **@last\_person**  **end**  end  b=Book.new  b.add\_person :name =>"jessie", :age=>25  b.add\_person :name =>"mike", :age=>65  b.show\_all  puts "last person: #{b.last\_person}" | class Book  **attr\_reader :last\_person** #добавляет класс last\_person, в код, для того чтобы на мне писать лишние строки  def initialize  @hh={}  @last\_person = ''  end  def add\_person options  @last\_person = options [:name]  puts 'alredy exists' if @hh[options[:name]]  @hh[options[:name]] = options[:age]  end  def show\_all  @hh.keys.each do |key|  age=@hh[key]  puts "name: #{key}, age: #{age}"  end  end  end  b=Book.new  b.add\_person :name =>"jessie", :age=>25  b.add\_person :name =>"mike", :age=>65  b.show\_all  puts "last person: #{b.last\_person}" |

**attr\_reader :aa** (переменная аа доступна только для чтения) прим. называют не переменная, а свойство

**attr\_accessor :aa** (переменная аа доступна для чтения и записи)

**attr\_writter:aa** (переменная аа доступна для записи)

Программа про самолет

class Airplane

attr\_reader :model #атрибут ридер, модель самолета, не изменяется

attr\_accessor :altitude #атрибут акцессор, модель самолета меняется

attr\_accessor :speed

def initialize(model) #передаем параметры инициализации модели

@model = model

@speed = 0 #начальная скорость

@altitude = 0 #начальная высота

end

def fly #метод (самолет может летать)

@speed = 800

@altitude = 10000

end

def land #метод (самолет может садиться)

@speed = 0

@altitude = 0

end

def moving? #метод, показывает движется самолет или нет (true или false, для этого в конце метода нужен знак вопроса)

return @speed >0 # это выражение будет true, если скорость выше 0 и false, если скорость меньше или равно 0, return можно не печатать

end

end

models = ['airbus-320', 'boeng-777', 'superjet-100'] #создаем массив из моделей самолетов

planes = [] #создаем массив самолетов

10.times do

model = models[rand(0..2)] #модель выбирается рандомно

plane = Airplane.new(model)

planes << plane

if rand(0..1)==1 #создаем рандом, если 1, то самолет летит

plane.fly

end

plane.speed = rand (50..700)

plane.altitude = rand (0..10000)

end

planes.each do |plane| #для каждого самолета выводим, что с ним происходит

puts "Model: #{plane.model}, Speed: #{plane.speed}, Altitude: #{plane.altitude}" #вывести основную информацию

puts "Is moving: #{plane.moving?}" #с помощью интерполяции обращаемся к методу мувинг?

end

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Урок 15**

новая программа

class Song

attr\_accessor :name, :duration

def initialize name, duration #конструктор, принимает значение имя и длина

@name = name

@duration = duration

end

end

song1 = Song.new 'the show...', 6 #создаем объект (название песни и ее длительность (6мин)), словом new вызвается инишилайз

puts song1.name #выводим на экран

puts song1.duration

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Сущности**

1 \*

Airplane

Airport

Значит, что 1 аэропорт может содержать много самолетов (\*)

Значит, нужно создать массив аэропорт, который будет содержать много объектов (самолетов)/

#класс аэропорт

class Airport

attr\_reader :name

attr\_reader :planes

def initialize(name)

@name = name #переменная нэйм

@planes = []

end

def add\_plane plane

@planes << plane

end

end

class Plane

attr\_reader :model

def initialize(model)

@model = model #переменная модель

end

end

airports = [] #создаем массив аэропортов

airport1 = Airport.new 'new york'

airport2 = Airport.new 'chikago'

airports << airport1 #добавляем аэропорт нью-йорк в массив аэропортов

airports << airport2 #добавляем аэропорт чикаго в массив аэропортов

#создаем самолеты для 1 аэропорта

plane1=Plane.new 'boeing'

plane2=Plane.new 'airbus'

plane3=Plane.new 'IL'

#добавляем самолеты в 1 аэропорт

airport1.add\_plane plane1

airport1.add\_plane plane2

airport1.add\_plane plane3

#создаем самолеты для 2 аэропорта

plane\_a=Plane.new 'aaa'

plane\_b=Plane.new 'superget'

plane\_c=Plane.new 'TU'

#добавляем самолеты во 2 аэропорт

airport2.add\_plane plane\_a

airport2.add\_plane plane\_b

airport2.add\_plane plane\_c

airports.each do |airport| #в первом цикле выводим название аэропорта

puts "airport: #{airport.name}"

airport.planes.each do |plane|

puts "plane: #{plane.model}"

end

end

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

программа песни в альбом

class Album

attr\_reader :name

attr\_reader :songs

def initialize name

@name = name

@songs = [] #создаем массив песен

end

def add\_song song

songs << song #добавляем песню в массив песен

end

end

#===========================

class Song

attr\_reader :name, :duration #задаем св-во

def initialize name, duration

@name = name

@duration = duration

end

end

#===========================

#создаем альбом

album = Album.new 'white'

#создаем песню

song1 = Song.new 'one', 4

song2 = Song.new 'two', 4.6

song3 = Song.new 'three', 4.3

album.add\_song song1

album.add\_song song2

album.add\_song song3

puts album.songs[0].name #выводим название песни 1

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Наследование**

class Animal #суперкласс (его наследуют)

def initialize name # конструктором определяем параметры (можно масив создать в конструкторе)

@name = name

end

def run

puts "#{@name} is running"

end

def jump

puts "#{@name} isjumping"

end

end

class Cat < Animal #класс Кошка унаследует все функции класса Энимал (run, jump)

def initialize

super 'cat' #передаем в суперкласс имя животного

end

def say #определяем свои методы

puts 'meow'

end

end

class Dog < Animal #класс dog унаследует все функции класса Энимал (run, jump)

def initialize

super 'dog' #передаем в суперкласс имя животного

end

def say #определяем свои методы (отдельная функциональнасть)

puts 'woof'

end

end

cat=Cat.new #создаем объект

cat.run

cat.say

dog=Dog.new #создаем объект

dog.run

dog.say

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Урок 16**

**Повторение**

class Animal #суперкласс (его наследуют)

def initialize name # конструктором определяем параметры, создаем массив, хэш

@a=part1

@b=part2

arr=[]

hh={}

end

end

animal=Animal.new 5 #запускаем класс

уровень классов –

public(стоит по умолчанию)

protected

privat (ограничивает доступ к методам (только внутри класса)) пример задача ниже

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

class Animal #суперкласс (его наследуют)

def initialize name #определяем конструктор

@name = name

end

def run

puts "#{@name} is running"

end

def jump

eat

puts "#{@name} is jumping"

sleep

end

private #этот метод будет вызываться, только через другой метод, в данном случае через метод jump (10 строка)

def eat

puts "#{@name} is eating"

end

def sleep

puts "#{@name} is sleeping"

end

end

animal=Animal.new 'cat'

animal.jump

#animal eat - мы может вызвать метод eat этой строчкой, но, чтоб вызывать его только через метод jump (в 11 строке) можно написать private

#animal.sleep

­­­\_\_\_\_\_\_\_\_\_\_\_

**статические методы**

в небольших программах

class Manager

def self.say\_hi #static - статический метод, чтобы не создавать объект Man.say\_hi

puts 'hi'

end

end

class Hipster

def self.say\_hi

puts 'hiii, yo'

end

end

class Jessie\_Pinkman

def self.say\_hi

puts 'hi, bitch'

end

end

Manager.say\_hi

Hipster.say\_hi

Jessie\_Pinkman.say\_hi

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**yield** - вызов (пропускать вперед, сдавать)

def run\_5\_times

5.times do

yield

end

end

run\_5\_times {puts 'hghgdkkd'}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**lambda** – вызов функции. Если лямбда содержит больше 3-х операторов, то обычно используют простые функции.

|  |  |
| --- | --- |
| **1. способ** | **2.способ** |
| x=lambda {|a|} | x= lambda do  end |
| x.call #вызывает функцию | |

Пример

say\_hi=lambda {puts 'hi'}

say\_hi. call

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

пример 2

sub\_10=lambda do |x| # отнимает 10 от переданного параметра

return x-10

end

a=sub\_10.call 1000 #отнимаем 10 от 1000

puts a #ответ 990

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

пример 3

say\_hi=lambda {puts 'hi'}

say\_bye=lambda{puts 'bye'}

week = [say\_hi,say\_hi,say\_hi,say\_hi,say\_hi,say\_bye,say\_bye]

week.each do |f|

f.call

end

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

пример 4

say\_hi=lambda {puts 'hi'} #создаем функции

say\_bye=lambda{puts 'bye'}

week = [say\_hi,say\_hi,say\_hi,say\_hi,say\_hi,say\_bye,say\_bye] #создаем массив

week.each do |f| #для каждого для неделе, создаем параметр

f.call #последовательно из массива вызываются функции

end

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

sub\_10=lambda do |x| # определяем лямбда функции. Она будет отнимать 10 от переданного параметра

return x-10 #возврат

end

#можно верхние три строчки написать в одну строку sub\_10=lambda {|x| return x-10}, можно ключевое слово ретёрн удалить sub\_10=lambda {|x| x-10}

a=sub\_10.call 1000 #переменная - отнимаем 10 от 1000

puts a

\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Модули (module=namespace(пространство имен)) –** позволяют делать множественные наследования. Для обращения к модулю используют ::

#модуль - это следующий уровень логического разделения

module Humans

class Manager #класс

def say\_hi #static - статический метод, чтобы не создавать объект Man.say\_hi

puts 'hi'

end

end

class Hipster

def say\_hi

puts 'hiii, yo'

end

end

class Jessie\_Pinkman

def say\_hi

puts 'hi, bitch'

end

end

end

module Animals

class Dog

end

class Cat

end

end

hipster = Humans::Hipster.new #создаем

hipster.say\_hi

cat=Animals::Cat.new #создаем кошку

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**require** (требовать)-модуль из другого файла ./file1.rb (точка и слэш). Обычно все классы разбиваются на файлы. Это нужно когда, несколько программистов пишут в разных файлах

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Глобальная переменная $** - переменная которая начинается со знака доллар, доступна во всей программе

**Переменная внутри класса @**

**Локальная переменная** – будет доступна только внутри метода

**CONSTANT** – константа, которая никогда не изменяется (PI= 3,14)

Пример:

class Math

PI = 3.14

end

**Переменная класса @@**

**Урок 17**

**Метапрограммирование**

**send** (перенаправление)

пример

def mm

puts

end

есть 3 варианта вызова функции mm

1. mm

2. send :mm

3. send “mm”

второй и третий способ – метапрограммирование

def mm par1

puts par1

end

send :mm, 1

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

пример с метапрограммированием и без

|  |  |
| --- | --- |
| class Something #создали класс    attr\_accessor :x, :y  def initialize hash  hash.each do |key, value| #Ич проходится по каждой паре ключ\значение  send "#{key}=", value #с помощью сэнд устанавливаем значение для переменной с помощью метапограмиирования  end    end  end  s=Something.new :x =>1, :y=>2 #инициализируем с помощью хэша, переменной С присваиваем новый объект Самфинг.Икс и игрек будет передан в хэш в 4 строку  puts s.x | class Something #создали класс    attr\_accessor :x, :y  def initialize hash  hash.each do |key, value| #Ич проходится по каждой паре ключ\значение  **def initialize x,y**  **@x=x**  **@y=y**  **end**  end  end  end  s=Something.new :x =>1, :y=>2 #инициализируем с помощью хэша, переменной С присваиваем новый объект Самфинг.Икс и игрек будет передан в хэш в 4 строку  puts s.x |

**method\_missing** (метод отсутсвия) – резервированное название метода (я еще не знаю что это значит)

пример

class Something

def method\_missing name #

puts "calling #{name}"

end

end

s=Something.new #создаем объект

s.dfjdlj #присваиваем имя объекту

s.yyyuuu

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Пример 2

class Albuquerque

def initialize actions #передаем параметр в экщемпляр

@actions= actions #глобальная переменная класса

end

def method\_missing name

value=@actions[name]

puts "if you want to #{name}, you must call #{value}"

end

end

a=Albuquerque.new :cook => "walt", :take\_a\_ride =>"jessie", :die =>"gus" #создаем хэш - экземпляр города

a.cook #в Классе Альбукерке у нас нет метода Кук, но когда мы его вызваем его через метод миссинг с параметром кук из массива экшенз по параметру кук у нас встанет значение Волт (это метапрограммирование)

a.take\_a\_ride

a.die

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**define\_method** – определение метода по имени

|  |  |
| --- | --- |
| send :define\_method, "aaa" do #дефайн - это один из параметров функции сенд. с помощью дефайн метод определяем метод ааа  puts "hello..."  end  aaa #вызов метода | def aaa  puts "hello..."  end  aaa |

пример 2

**Режим открытия файлов**

|  |  |
| --- | --- |
| r | read |
| w | write |
| r+ | read and write |
| w+ | read and write |
| a | append – дозапись (открыть файл для записи и передвинуть указатель на конец файла) |

output=File.open “test.txt”, “w”

output.write “Something”

output.close

\_\_\_\_\_\_\_\_\_\_\_\_\_

input=File.open"test2.txt", "r" #создаем тестовый файл, будем его чиать

while (line=input.gets) #чтение файла строка за строкой

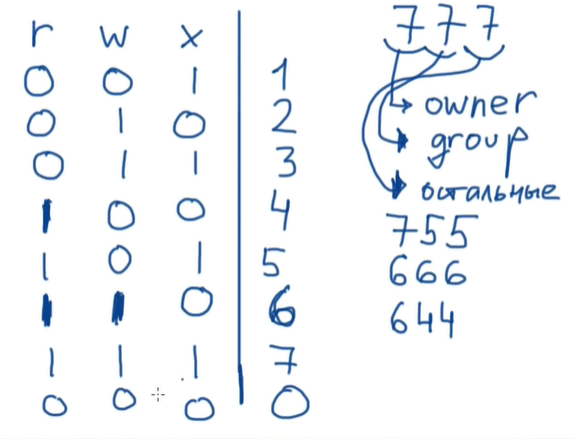
puts line

end

input.close

\_\_\_\_\_\_\_\_\_\_\_\_\_

**права доступа файлов в linux**



записывается chmod 777 config.php

puts Dir.pwd (print working directry – напечатать рабочую директорию)

**Урок 18**

input=File.open "passwords.txt", "r"

while line=input.gets #это нужно запомнить

line.strip! #обрезаем символ white space (пустые строки, табуляция, пробелы)

if line.size ==6 #выводим все пароли длиной 6 символов

puts line #выводим все пароли

end

end

перенаправляем пароли длиной 6 символов в отдельный тхт файл, для этого в Фаре пишем C:\projects\1.output.rb > passwords2.txt

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

print "Enter your passwords: "

my\_password = gets.strip

input=File.open "passwords.txt", "r"

while line=input.gets #это нужно запомнить

line.strip! #обрезаем символ white space (пустые строки, табуляция, пробелы)

if line == my\_password

puts "Your password is weak"

end

end

puts "Your password isn't weak"

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

сделать функцию которая проверяет пароль weak или нет

def is\_password\_weak? password #создаем функцию (если функция начинается с is, то всегда знак вопроса ставим)

input=File.open "passwords.txt", "r" #это нужно запомнить

while line=input.gets #это нужно запомнить

line.strip! #обрезаем символ white space (пустые строки, табуляция, пробелы)

if password.include? line

return true

end

end

return false

end

print "Enter your passwords: "

my\_password = gets.strip

if is\_password\_weak? my\_password

puts "Your password is weak"

else

puts "Your password isn't weak"

end #это программа у меня не работает

|  |  |
| --- | --- |
| get (получить) | post (отправить) |
| браузер получает данные с сервера | браузер отправляет данные на сервер |

Программа fiddler web

|  |  |
| --- | --- |
| 1 способ | 2 способ |
| require 'net/http' #реквая - (требуется)подгружаем в нашу программу net http  response = Net::HTTP.get 'rubyschool.us','/router' # получаем страницу. Нет - пространство имен. HTTP - название класса. Функция Гет принимает два параметра (1. - нзвание домена).Роутер - путь на котором расположена сама страница. И все это возвращается в переменную Респонс  puts response | require 'net/http'  require 'uri' #переменная ури  aaa=URI.parse 'http://rubyschool.us/router'#создаем новый объект с помощью статического метода (УРИ.парсе). Новый объект помещается в aaa (universal resourse identificator)  response = Net::HTTP.get aaa  puts response |

**функция post\_form**

require 'net/http'

require 'uri' #переменная ури

aaa=URI.parse 'http://rubyschool.us/router'#создаем новый объект с помощью статического метода (УРИ.парсе). Новый объект помещается в aaa (universal resourse identificator)

response = Net::HTTP.post\_form (aaa, :login => "aaa", :password => "123123").body #отправляем форму

puts response

\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**подбор пароля**

require 'net/http'

require 'uri' #переменная ури

aaa=URI.parse 'http://rubyschool.us/router'#создаем новый объект с помощью статического метода (УРИ.парсе). Новый объект помещается в aaa (universal resourse identificator)

response = Net::HTTP.post\_form(aaa, :login => "aaa", :password => "123123").body #отправляем форму

puts response.include? "denied" #содержится ли в коде страницы денай? Если тру - присутствует, значит доступ запрещен

input=File.open "passwords.txt", "r"

while line=input.gets #это нужно запомнить, в переменную лайн каждый раз записывается новый пароль из текстового файла

line.strip! #обрезаем символ white space (пустые строки, табуляция, пробелы)

end

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Cygwin – набор юникс команд

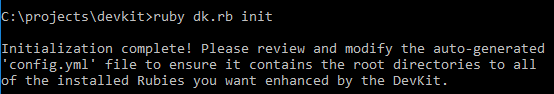
GOW – набор программ из юникса

| grep (фильтрует строки, по определенным словам)

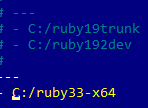
**Урок 19**

Установка devkit

в Фар пишем ruby dk.rb init



в файле config.yml дописываем путь –c:/Ruby33-x64



в Фар пишем ruby dk.rb install



Обновляем

gem update --system

**Урок 20**

Устанавливаем Sinatra

**gem install Sinatra –** это упрощенный рейлс

require 'sinatra' #подключение модуля синатра

get '/' do #методом Гет - получаем данные c браузера и мы будет обслуживать этот запрос

'hi'

end

В браузере пишем <http://localhost:4567/> (4567 – это порт компьютера)

создаем файл index.erb

require 'sinatra' #подключение модуля синатра

get '/' do #методом Гет - получаем данные c браузера и мы будет обслуживать этот запрос. Слэш - обозначает корневой сайт

erb :index #erb-tamplane engine (движок шаблонов)

end

get '/contacts' do

'phone 222'

end

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

scraping

**Установка msysGIT**

установка nodejs – это графическая оболочка GIT.

Добавляем его в path. Win+E – св-ва – доп.параметры системы-переменные среды-добавляем c:\program files\nodejs

перезапускаем far и пишем node

Установка Ungit – это пакет. Gem если использовать термин руби.

В фаре пишем npm install -g ungit

Проверяем ungit. Создаем папку тест и вводим ungit. Он работает как маленький сервер и для обращения к нему, используем браузер. Для того чтобы Ангит не занимал Фар, пишем в фаре start ungit и он запустится в отдельном окне

create repository – создать место для хранения кода. Это просто отдельный каталог с разрешением .git и в этот каталог

создаем коммит (вклад)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**barbershop программа**

создаем файл index.erb в каталоге view

<h1> welcome barb </h1>

<p1> we are open </p1>

<form action="/"method="POST">

<input type="text" name="useer name"/> <br /> #добавляем поля

<input type="text" name="phone"/> <br />

<input type="text" name="date"/> <br />

<input type="submit"/> #добавляем кнопку

</form>

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

в синатра создаем папку Паблик, чтоб можно было ее просматривать из браузера

**css** (**Cascading Style Sheets)** — язык таблиц стилей, который позволяет прикреплять стиль (например, шрифты и цвет) к структурированным документам (например, документам HTML и приложениям XML). Придает сайту оформление. Размещается в файле с расширением .css.

Вводные слова

могут начинаться с точки, с хэша и без точки

|  |  |  |
| --- | --- | --- |
|  | . something (пишется только в css файле!) | #slogan (пишется только в css файле!) |
| пусто (правила применяются к тэгу) | с точкой (применяется к множеству элементов с помощью атрибута class) | с хэша (применяется только к одному элементу с помощью атрибута id) Причем идентификатор (slogan) должен быть уникальным и больше не повторяться |
|  | <p class=”something”> welcome</p> | <p id=”slogan”> welcome</p> |

Подключается в поле <head>

**Урок 21**

**bootstrap –** набор css стилей, которые придумала компания твиттер, документация на сайте <https://getbootstrap.com>

устанавливаем бутстрэп на синатра через гитхаб - <https://github.com/bootstrap-ruby/sinatra-bootstrap>

Заходим в code – копируем ssh. в фаре в нужной директории вводим git clone и вставляем из буфера ssh (У меня получилось скопировать через https)

**Sinatra**

configure do #механизам сессий нужен, для того чтобы хранить текущее имя пользователя в этой сессии

enable :session

end

**Урок 22**

Создаем репозиторий в Git

Чтоб посмотреть что в репозитории, в фар вводим команду git clone

ungit – master – указывает на локальную копияю. Originmaster – находится на гитхабе.

commit (вклад), знак минус – строчка удалена. Знак плюс – строчка прибавлена

contributions – что и в какой день ты коммитил.

Правильно запускать через строку – bundle exec app1.rb (у меня не получилось)

Код из файла app.rb будет автоматически помещаться в файл layout.erb в часть с ключевым словом yield

gem install sinatra-reloader – чтоб не перегружать постоянно сервер, после изменения файла. Затем в начале app.rb прописываем require 'sinatra/reloader'

Затем в фаре прописываем start ruby app.rb

И app откроется в отдельном окне.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

|  |  |
| --- | --- |
| 1 вариант | 2 вариант |
| def get\_website\_contents  puts '<body>'  **yield** #команда yield вызывает что находится в скобках {}  puts '</body>'  end  get\_website\_contents {puts "something"} | get\_website\_contents do  puts "something"  end |

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_