**Московский государственный технический**

**университет им. Н.Э. Баумана**

Факультет «Радиотехнический»

Кафедра «Системы обработки информации и управления»

Курс «Парадигмы и конструкции языков программирования»

Домашняя работа

«Знакомство с языком программирования Ruby»

|  |  |  |
| --- | --- | --- |
| Выполнил: |  | Проверил: |
| студент группы РТ5-31Б: |  | преподаватель кафедры ИУ5 |
| Шарафутдинов М.Э. |  | Гапанюк Ю.Е. |
| Подпись и дата: |  | Подпись и дата: |

**Описание задания**

1. Выберите язык программирования (который Вы ранее не изучали) и (1) напишите по нему реферат с примерами кода или (2) реализуйте на нем небольшой проект (с детальным текстовым описанием).
2. Реферат (проект) может быть посвящен отдельному аспекту (аспектам) языка или содержать решение какой-либо задачи на этом языке.
3. Необходимо установить на свой компьютер компилятор (интерпретатор, транспилятор) этого языка и произвольную среду разработки.
4. В случае написания реферата необходимо разработать и откомпилировать примеры кода (или модифицировать стандартные примеры).
5. В случае создания проекта необходимо детально комментировать код.
6. При написании реферата (создании проекта) необходимо изучить и корректно использовать особенности парадигмы языка и основных конструкций данного языка.
7. Приветствуется написание черновика статьи по результатам выполнения ДЗ. Черновик статьи может быть подготовлен группой студентов, которые исследовали один и тот же аспект в нескольких языках или решили одинаковую задачу на нескольких языках.

**Основная часть**

Язык программирования Ruby начал разрабатываться 23 февраля 1993 года и вышел в свет в 1995 году.  Он был разработан японским программистом Юкихиро Мацумото (также известным как Matz).

* Сайт проекта: <https://www.ruby-lang.org/ru/>
* Документация: <https://www.ruby-lang.org/ru/documentation/>

Ruby — это высокоуровневый, интерпретируемый язык программирования общего назначения, созданный с акцентом на простоту и продуктивность. Ruby сочетает в себе элементы объектно-ориентированного, функционального и императивного программирования. Ruby часто используется для веб-разработки, особенно в сочетании с фреймворком Ruby on Rails, который значительно упрощает создание веб-приложений. Ruby on Rails позволяет быстро разрабатывать приложениях благодаря своим соглашениям и принципам, которые минимизируют количество решений, которые разработчики должны принимать.

Особенности Ruby:

* Интерпретируемость.
* Объектно-ориентированный подход.
* Поддержка блоков.
* Динамическая типизация.
* Многопоточность.

Достоинства Ruby:

* Принцип наименьшего удивления.
* Широта применения.
* Активное сообщество.

Недостатки Ruby:

* Невысокая производительность.
* Нет обратной совместимости версий.
* Относительная сложность освоения.

Язык программирования был установлен согласно инструкциям из сайта <https://www.ruby-lang.org/ru/documentation/installation/>. В работе используется среда разработки Visual Studio Code.

**Типы данных**

Ruby реализует идеологию «всё — объект», то есть любая единица данных является объектом — экземпляром некоторого класса, к которому применимы все синтаксические средства, предназначенные для работы с объектами.

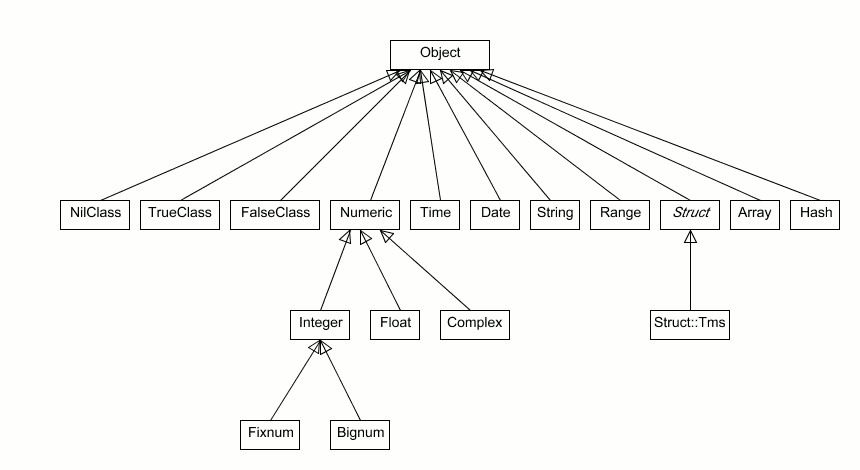
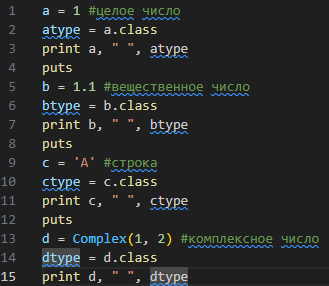


Рис. 1 Иерархия типов данных

Самые используемые встроенные типы данных:

* Fixnum (целые числа, меньше 230),
* Bignum (целые числа, больше 230),
* Float (числа с плавающей запятой),
* Array (массивы),
* String (строки),
* Hash (ассоциативные массивы).

Программа, демонстрирующая работу с типами данных представлена на рисунке 2:



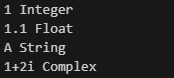


Рис. 2 Числовые типы данных

Метод class возвращает название типа (класса). Функция print не производит перенос строки по умолчанию, для этого используем функция puts. Функция Complex формирует представляет число в комплексном виде.

Комментирование одной строки происходит с помощью символа #, многострочного комментирования возможно только при использовании символа # перед каждой строкой.

Язык Ruby чувствителен к регистру, то есть переменные с именами a и A являются разными. Ruby позволяет использовать символы кодировки Unicode (в формате UTF-8).





Рис. 3

**Арифметические операции**

В языке программирования Ruby существуют следующие арифметические операторы:

* + - \* / \*\* % - базовые арифметические операции (сложение, вычитание, умножение, деление, возведение в степень и получение остатка от деления);
* a+=1 – эквивалентная запись выражения a = a+1,
* a-=1 – эквивалентная запись выражения a = a-1,
* a\*=1 – эквивалентная запись выражения a = a\*1,
* a/=1 – эквивалентная запись выражения a = a/1,

Методы класса Numeric:

* +num - унарный плюс — возвращает число *num* в качестве результата.
* -num - унарный минус — возвращает число *num* с противоположным знаком в качестве результата.
* first <=> second - возвращает:

first > second *#=> 1*

first == second *#=> 0*

first < second *#=> −1.*

* num.abs – возвращает абсолютное значение числа num.
* num.ceil – возвращает наименьшее целое число, которое больше или равно num.
* num.coerce(numeric) – если numeric такого же типа, что и num, то возвращает массив, состоящий из numeric и num. Иначе, возвращает массив с numeric и num преобразованных в дробные числа.
* num.div(numeric) - использует оператор / для выполнения деления числа num на число numeric, после чего конвертирует результат в целое число.
* num.divmod (aNumeric) - возвращает массив, состоящий из результата целочисленного деления и остатка от деления числа num на число aNumeric.
* num.eql?(numeric) - возвращает true, если num и numeric одного и того же типа и имеют одинаковые значения.
* num.floor - возвращает наибольшее целое число, которое меньше или равно num.
* num.integer? – возвращает true, если num является целым числом.
* num.modulo(numeric) – получение остатка от деления числа num на число numeric.
* num.nonzero? - возвращает num, если num не является нулем, если num это ноль, то возвращает nil.
* num.quo(numeric) – эквивалентен оператору /, но переопределяется в подклассах.
* num.remainder(numeric) - если num и numeric имеют разные знаки, то возвращает разность mod-numeric; иначе, возвращает mod.
* num.round - округляет число num до ближайшего целого.
* num.truncate – возвращает целую часть числа num.
* num.zero? – возвращает true, если num равен нулю.

Базовые математические функции:

* Math.acos(x) – вычисляет арккосинус числа x.
* Math.asin(x) – вычисляет арксинус числа x.
* Math.atan(x) – вычисляет арктангенс числа x.
* Math.cos(x) – вычисляет косинус угла x.
* Math.sin(x) – вычисляет синус угла x.
* Math.tan(x) – вычисляет тангенс угла x.
* Math.exp(x) – возвращает e\*\*x.
* Math.hypot(x, y) – возвращает sqrt(x\*\*2 + y\*\*2).
* Math.log(numeric) – возвращает натуральный логарифм числа numeric.
* Math.log10(numeric) – возвращает десятичный логарифм числа numeric.
* Math.sqrt(numeric) – извлекает квадратный корень из неотрицательного числа numeric.

Логические операторы:

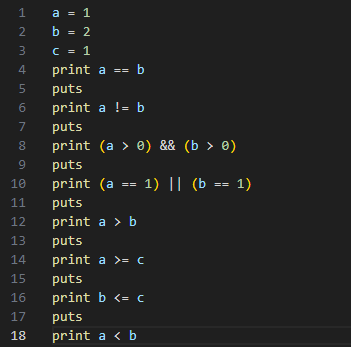
* ! – не.
* && - и.
* || - или.
* == - проверка эквивалентности объектов.
* > - больше.
* >= - больше или равно.
* < - меньше.
* <= - меньше или равно.

Битовые операторы:

* ~ - битовое отрицание not.
* & - битовое and.
* | - битовое or.
* ^ - битовое xor.
* << - оператор побитового сдвига влево.
* >> - оператор побитового сдвига вправо.

Тернарный оператор:

* ?x:y - если условие верно, тогда значение x. В противном случае значение y



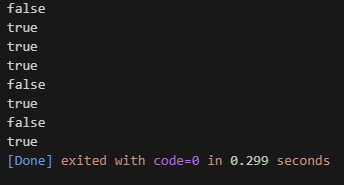
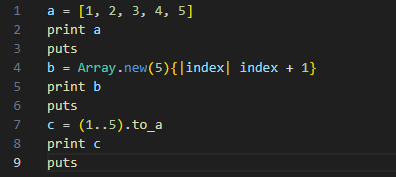


Рис. 4 Демонстрация логических операторов

**Массивы.**

В Ruby массив можно создать как минимум тремя способами:

1. a = [1, 2, 3, 4, 5]
2. Array.new(5){ |index| index + 1} - Параметром метода .new является количество элементов будущего массива (в данном случае это число 6). В фигурных скобках указано, как мы будем заполнять массив. В данном случае значение элемента массива будет больше на единицу его индекса.
3. a = (1..5).to\_a – метод .to\_a преобразует объект типа Range в массив.



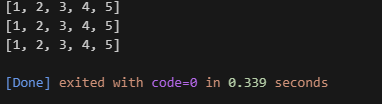
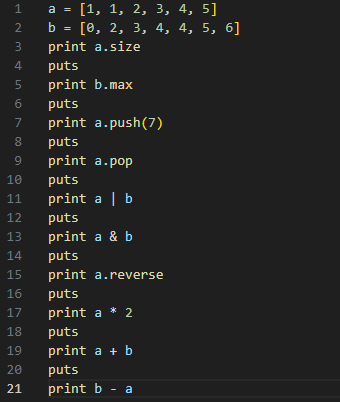


Рис. 5 Демонстрация создания массивов

Методы работы с массивами:

* Определить размерность массива: a.size
* Поиск максимального по значению: a.max
* Поиск максимального по размеру строки: a.max\_by{|elem| elem.size}
* Поиск минимального по значению: a.min
* Поиск минимального по размеру строки: a.min\_by{|elem| elem.size}
* Сортировка по значению: a.sort
* Сортировка по размеру строки: a.sort\_by{|elem| elem.size}
* Обращение массива: a.reverse
* Сложение массивов: [1, 2] + [3, 4, 5] = [1, 2, 3, 4, 5]
* Вычитание массивов: [1, 2, 2, 3, 3, 3, 4, 5, 5] – [2, 3] = [1, 4, 5, 5]
* Объединение массивов: [1, 2, 4, 5, 6] | [0, 1, 3, 5, 7] = [1, 2, 4, 5, 6, 0, 3, 7]
* Пересечение массивов: [1, 2, 3, 3, 5] & [1, 3, 4, 4, 5] = [1, 3, 5]
* Удаление дубликатов: [1, 1, 2, 2, 2, 3, 4, 0, 1, 2, 3, 4].uniq = [1, 2, 3, 4, 0]
* Удаление неопределённых (nil) элементов: a.compact
* Размножение массивов: [1, 2, 3] \* 2 = [1, 2, 3, 1, 2, 3]
* Добавление элемента в конец массива: a.push(x)
* Удаление элемента из конца массива: a.pop
* Добавление элемента в начало массива: a.unshift(x)
* Удаление элемента из начала массива: a.shift
* Проверка наличия в массиве элемента: a.include?(x)
* Проверка массива на пустоту: a.empty?
* Проверка массива на не пустоту: a.any?



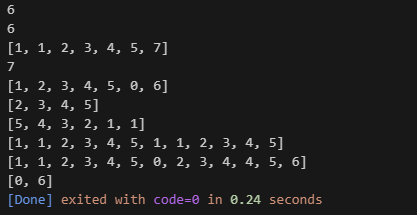


Рис. 5 Демонстрация работы с массивами

**Ассоциативные массивы.**

Способы создания ассоциативного массива:

1. Из одномерного массива:

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

Hash[\*array] #=> {1=>4, 5=>3, 2=>2}

1. Из двумерного массива:

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

Hash[\*array.flatten] #=> {1=>4, 5=>3, 2=>2}

1. Нет данных:

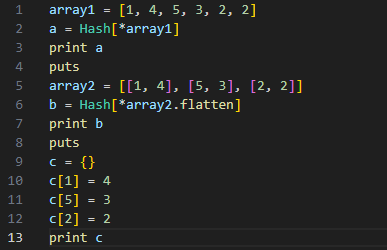
hash = {}

hash[1] = 4

hash[5] = 3

hash[2] = 2

hash #=> {1=>4, 5=>3, 2=>2}



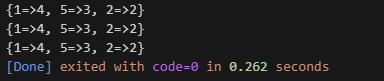
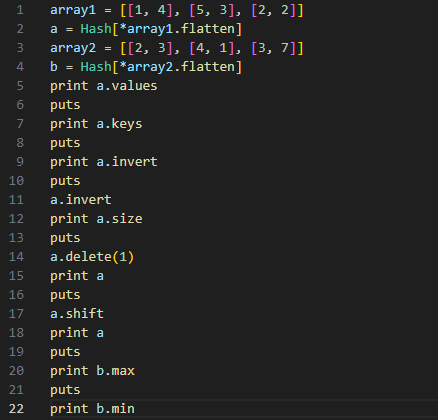


Рис. 6 Демонстрация создания ассоциативных массивов

Методы работы с ассоциативными массивами:

* Получение массива значений: a.values
* Получение массива ключей: a.keys
* Замена ключей на значения: a.invert
* Слияние двух массивов: a.merge!(b)
* Размер ассоциативного массива: a.size
* Удаление пары по ключу: a.delete(x)
* Удаление первой пары: a.shift
* Преобразовать в индексный массив: a.to\_a
* Поиск максимальной пары: a.max
* Поиск минимальной пары : a.min
* Проверка на пустоту: a.empty?
* Есть такой ключ?: a.include?(x)
* Есть такое значение?: a.value?(x)



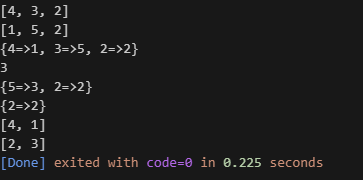
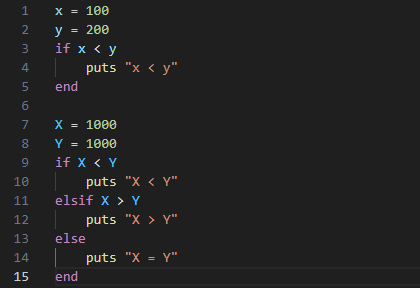


Рис. 7 Демонстрация работы с ассоциативными массивами

Условные операторы.

Условные операторы рассмотрим на примере использования связки:



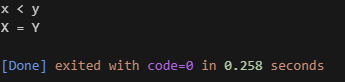
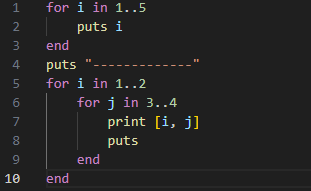


Рис. 8

Циклы

Примеры вывода на печать:



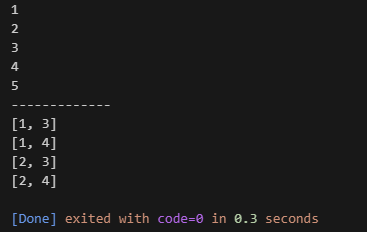


Рис. 9

Чтение и запись данных

Чтение из файла:

* 1. Чтение всего файла за раз:

Метод File.read позволяет прочитать весь файл целиком в строку.

* 1. Чтение построчно:

Если файл слишком большой, можно читать его построчно с помощью метода File.foreach.

* 1. Чтение файла в массив:

Метод File.readlines считывает файл и возвращает его содержимое в виде массива строк.

Запись в файл:

1. Запись в файл (перезапись):

Метод File.write перезаписывает файл новым содержимым.

2. Добавление данных в файл:

Метод File.open с флагом 'a' позволяет добавлять данные в конец файла.

Подводя итог, несмотря на свои недостатки, язык программирования Ruby является перспективным языком программирования.