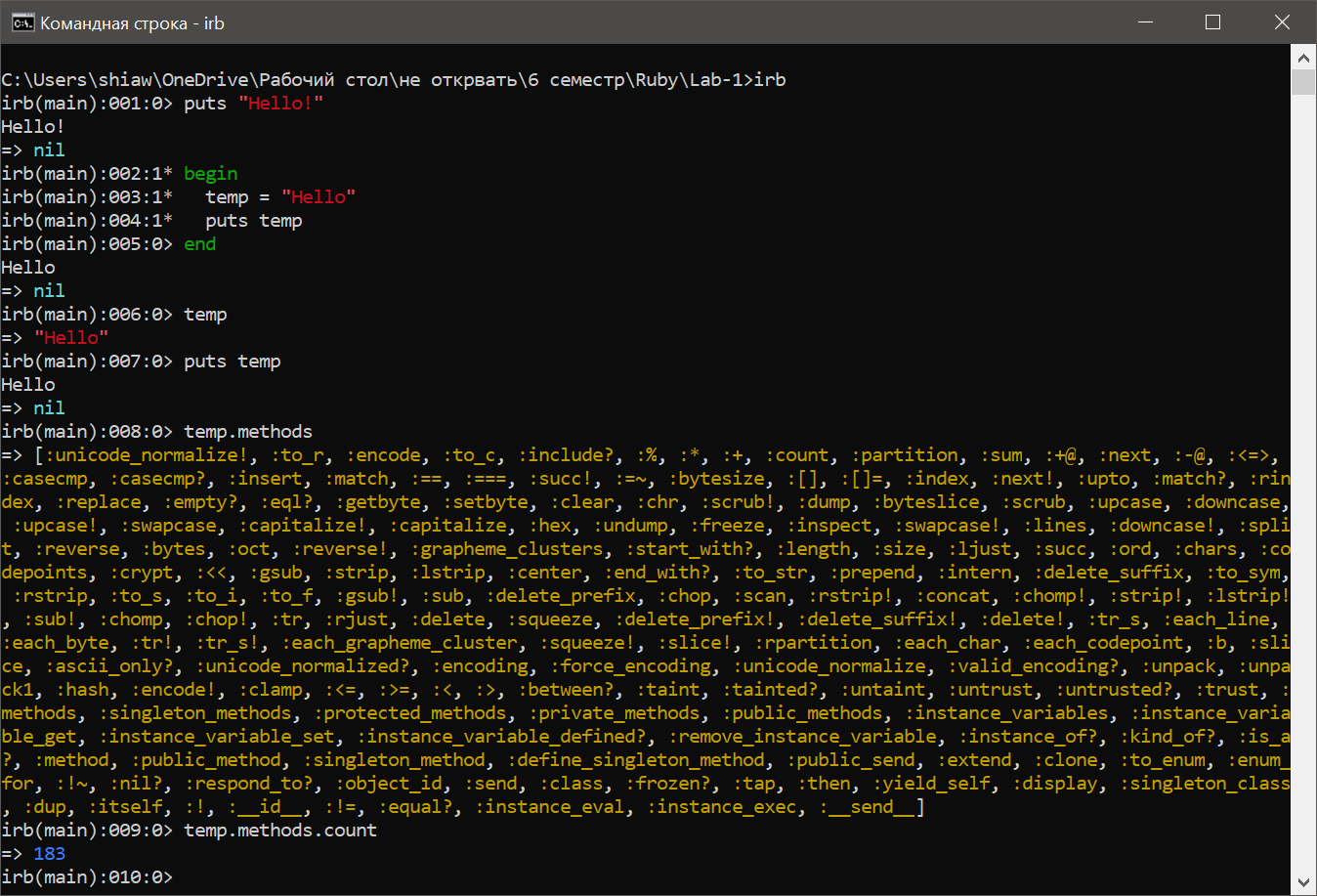
Лабораторная работа №1

IRB (interactive ruby shell) или «интерактивная руби консоль» — инструмент, благодаря которому вы можете писать любой руби код и видеть результаты его выполнения сразу же, в том же окне.

Для запуска интерактивной руби консоли достаточно набрать irb в командной строке.

На скриншоте ниже представлен переход в irb-режим с выполнением некоторых команд, в том числе просмотр всех методов строки и их количества.



Некоторые из выведенных методов (93 шт.) опишем ниже с примерами:

1. str \* integer #-> new\_str

Повторение — возвращает новую строку, которая состоит из integer-копий строки str.

"Hello! " \* 3 #-> Hello! Hello! Hello!

1. str + other\_str #-> new\_str

Возвращает новую строку, состоящую из строк str и other\_str.

"Hello " + "!" #-> Hello!

1. str << fixnum #-> str

str << obj #-> str

Добавление — присоединяет аргумент к str. Если аргумент типа Fixnum в диапазоне от 0 до 255, то он перед присоединением конвертируется в символ.

"Hello " << 33 #-> Hello!

1. str <=> other\_str #-> -1, 0, +1

Сравнение — возвращает -1, если other\_str меньше str; возвращает 0, если other\_str и str равны; возвращает +1, если other\_str больше str. Если строки различаются по длине, но эквивалентны на длине самой короткой из двух, то большей считается та, которая длиннее. Если переменная $= равна false, то сравнение базируется на сравнении двоичных значений каждого символа в строке.

"Hello" <=> "Hello" #-> 0

"Hell" <=> "Hello" #-> -1

"Hello" <=> "Hell" #-> 1

"Hello" <=> "HELLO" #-> 1

"Hello" <=> "world" #-> -1

1. str == obj #-> true или false

Эквивалентность — если строки str и obj не совпадают, то возвращается false. Возвращается true только в случае, когда код str <=> obj возвращает 0.

"Hello" == "Hello" #-> true

"Hello" == "world" #-> false

"hello" == "Hello" #-> false

1. str[index] #-> символ или nil

str[start, length] #-> new\_str или nil

str[range] #-> new\_str или nil

str[regexp] #-> new\_str или nil

str[regexp, index] #-> new\_str или nil

str[other\_str] #-> new\_str или nil

Получение подстроки или символа — возвращает код символа с индексом index, или подстроку длины length, начиная с индекса start, или подстроку, которая располагается в диапазоне range. Во всех этих вариантах вызова отрицательная индексация подразумевает отсчет с конца строки str.

Возвращается nil, если индекс index выходит за пределы допустимого диапазона, размер length запрашиваемой подстроки отрицательный или начало диапазона range попадает после конца строки str.

Если передается правило regexp, то возвращается фрагмент строки str, который удовлетворяет правилу regexp (или nil, если такого совпадения найти не удалось).

Если задан дополнительный параметр index, то возвращается содержимое группировки правила с номером index (если index равен 0, то возвращается фрагмент строки, который удовлетворяет правилу regexp).

Если в качестве аргумента передается строка other\_str, то возвращается строка other\_str, если она является подстрокой строки str (или nil, иначе).

"hello"[0] #-> "h"

"hello"[0,3] #-> "hel"

"hello"[0..3] #-> "hell"

"hello"[-3..-1] #-> "llo"

"hello"[-1..-3] #-> ""

"hello"[6] #-> nil

"hello"["he"] #-> "he"

"hello"["help"] #-> nil

"hello"[/[ah](.)\w/] #-> "hel"

"hello"[/[ah](.)\w/,0] #-> "hel"

1. str.ascii\_only? #-> true или false

Возвращает true для строки, которая содержит только ASCII символы. В противном случае возвращает false.

"a3{f%\*~".force\_encoding("UTF-8").ascii\_only? #-> true

'⬆'.ord.to\_s(16) #->2b06

"\u2b06".force\_encoding("UTF-8").ascii\_only? #-> false

1. str.capitalize #-> new\_str

Возвращает копию строки str в которой первый символ преобразуется в верхний регистр, а остальные — в нижний.

"hello".capitalize #-> "Hello"

"HELLO".capitalize #-> "Hello"

"1HeLLo".capitalize #-> "1hello"

1. str.capitalize! #-> str или nil

Модифицирует строку str по правилу: первый символ преобразуется в верхний регистр, а остальные — в нижний. Возвращает nil, если изменения не требуются.

temp = "HeLLo!"

temp.capitalize!

temp #-> "Hello!"

temp.capitalize! #-> nil

1. str.casecmp(other\_str) #-> -1, 0, +1

Регистронезависимая версия метода <=>.

"Hell".casecmp("Hello") #-> -1

"Hello".casecmp("Hell") #-> 1

"Hello".casecmp("HELLO") #-> 0

1. str.casecmp?(other\_str) → true, false, or nil

Возвращает истину, если str и other\_str равны после сворачивания регистра Unicode, и false, если они не равны.

"Hell".casecmp?("Hello") #-> false

"Hello".casecmp?("HELLO") #-> true

nil возвращается, если две строки имеют несовместимые кодировки или если other\_str не является строкой.

"Hello".casecmp?(1) #-> nil

1. str.center(integer, padstr) #-> new\_str

Если integer больше, чем str.[length](https://ru.wikibooks.org/wiki/Ruby/%D0%A1%D0%BF%D1%80%D0%B0%D0%B2%D0%BE%D1%87%D0%BD%D0%B8%D0%BA/String#String#length), то возвращает новую строку, длина которой равна integer, строка str располагается посередине, обитая символами строки padstr; иначе, возвращает str.

"Hello".center(4) #-> "center"

"Hello".center(10) #-> " center "

"Hello".center(10, '13') #-> "13Hello131"

1. str.chars #-> array

Возвращает массив, содержащий символы строки, из которой вызывается метод.

"Hello".chars #-> ["H", "e", "l", "l", "o"]

1. str.chomp(separator=$/) #-> new\_str

Возвращает новую строку, которая является копией строки str в которой удален последний символ separator. Если системная переменная $/ не была изменена и не передается параметр separator, то метод chomp удалит завершающие символы строки (такие как \n, \r и \r\n).

"Hello\n".chomp #-> "Hello"

"Hello\r".chomp #-> "Hello"

"Hello\r\n".chomp #-> "Hello"

"Hello\n\n\r".chomp #-> "Hello\n\n"

"Hello".chomp("lo") #-> "Hel"

1. str.chomp!(separator=$/) #-> str или nil

Модифицирует строку str по алгоритму, описанному в методе [chomp](https://ru.wikibooks.org/wiki/Ruby/%D0%A1%D0%BF%D1%80%D0%B0%D0%B2%D0%BE%D1%87%D0%BD%D0%B8%D0%BA/String#String#chomp). Возвращает обработанную строку str или nil, если изменения не требуются.

temp.replace("Hello!\n")

temp.chomp!

temp #-> "Hello!"

temp.chomp! #-> nil

1. str.chop #-> new\_str

Возвращает копию строки str из которой удален последний символ. Если строка заканчивается комбинацией символов \r\n, то будет удалена вся комбинация. Вызов метода chop от пустой строки возвращает пустую строку.

"Hello\r\n".chop #-> "Hello"

"Hello".chop #-> "Hell"

"".chop #-> ""

1. str.chop! #-> new\_str или nil

Модифицирует строку str по алгоритму, описанному в методе [chop](https://ru.wikibooks.org/wiki/Ruby/%D0%A1%D0%BF%D1%80%D0%B0%D0%B2%D0%BE%D1%87%D0%BD%D0%B8%D0%BA/String#String#chop). Возвращает обработанную строку str или nil, если изменения не требуются (например, если метод chop! был вызван от пустой строки).

temp.replace("Hello!")

temp.chop!

temp #-> "Hello"

1. str.chr #-> string

Возвращает односимвольную строку с начала строки.

"Hello".chr #-> "H"

1. str.clear #-> empty string

Делает строку, из которой вызывается, пустой.

"Hello".clear #-> ""

1. str.concat(fixnum) #-> str

str.concat(obj) #-> str

Добавление — присоединяет аргумент к str. Если аргумент типа Fixnum в диапазоне от 0 до 255, то он перед присоединением конвертируется в символ.

"Hello ".concat(33) #-> Hello!

1. str.count(other\_str\*) #-> fixnum

Каждый параметр other\_str преобразуется в множество символов. Метод подсчитывает количество символов str, которые принадлежат этому множеству. При помощи символа "галочка" (^) задаются исключения из множества. Выражения типа c1-c2 задают множество символов, которые располагаются между символами c1 и c2.

"hello world".count("el") #-> 4

"hello world".count("elo", "^o") #-> 4

"hello world".count("dh-m") #-> 4

1. str.delete(other\_str\*) #-> new\_str

Возвращает копию строки str, в которой удалены все символы, передаваемые параметром.

"hello world".delete("el") #-> "ho word"

"hello world".delete("elo", "^o") #-> "ho word"

"hello world".delete("dh-m") #-> "eo wor"

1. str.delete!(other\_str\*) #-> str или nil

Удаляет из строки str все символы, передаваемые параметром. Возвращает модифицированную строку str или nil, если строка str не была модифицирована.

temp.replace("hello")

temp.delete!("lo") #-> "ho"

temp.delete!("lo") #-> "nil"

1. delete\_prefix(prefix) → new\_str

Возвращает копию str с удаленным начальным префиксом

"hello".delete\_prefix("he") #-> "llo"

"hello".delete\_prefix("llo") #-> "hello"

1. delete\_prefix!(prefix) → self or nil

Удаляет начальный префикс из str, возвращая nil, если не было сделано никаких изменений.

temp.replace("hello")

temp.delete\_prefix!("he") #-> "llo"

temp.delete\_prefix!("llo") #-> "nil"

1. delete\_suffix(suffix) → new\_str

Возвращает копию str с удаленным конечным суффиксом.

"hello".delete\_suffix("llo") #-> "he"

"hello".delete\_suffix("hel") #-> "hello"

1. delete\_suffix!(suffix) → self or nilclick to toggle source

Удаляет завершающий суффикс из str, возвращая nil, если не было сделано никаких изменений.

temp.replace("hello")

temp.delete\_suffix!("llo") #-> "he"

temp.delete\_suffix!("hel") #-> "nil"

1. str.downcase #-> new\_str

Возвращает копию строки str в которой все символы вернего регистра заменены на соответствующие символы нижнего.

"HeLLo".downcase #-> "hello"

1. str.downcase! #-> str или nil

Модифицирует строку str по правилу: все символы верхнего регистра преобразовываются в соответствующие символы нижнего. Возвращает nil, если изменения не требуются.

temp.replace("HeLLo")

temp.downcase! #-> "hello"

temp.downcase! #-> "nil"

1. str.dump #-> new\_str

Создает версию строки str в которой все напечатные символы заменены на \nnn нотацию и все специальные символы экранированы.

"hello".dump #-> "\"hello\""

"\r\nhello\n".dump #-> "\"\\r\\nhello\\n\""

1. str.each\_byte {|fixnum| block } #-> str

Передает каждый байт строки str в блок. В качестве результата итератор возвращает исходную строку str

"hello".each\_byte {|a| print a, " "} #-> 104 101 108 108 111

1. str.each\_char {|char| block } #-> str

Передает каждый символ строки str в блок. В качестве результата итератор возвращает исходную строку str.

"hello".each\_char {|a| print a, " "} #-> h e l l o

1. str.empty? #-> true или false

Возвращает true если строка str имеет нулевую длину (то есть, если строка str — пустая).

"hello".empty? #-> false

"".empty? #-> true

1. str.eql?(other\_str) #-> true или false

Две строки называются эквивалентными, если они имеют одинаковое содержимое и длину.

"hello".eql?("hello") #-> true

"hello".eql?("Hello") #-> false

"hello".eql?("hell") #-> false

1. str.getbyte(index) → 0 .. 255

Возвращает индексный байт строки как целое число.

temp.replace("hello")

temp.getbyte(0) #-> 104

1. str.hash #-> fixnum

Вычисляет хеш-код для строки str. Две строки с одним и тем же содержимым будут иметь одинаковый хеш-код (именно его использует метод eql?).

"hello".hash #-> -698491128

1. str.hex #-> integer

Трактует строку str как строку шестнадцатиричных цифр (с необязательным указанием знака или необятельным префиксом 0x) и возвращает cоотвествующее число. Если преобразование не удается, то возвращает ноль.

"a".hex #-> 10

"0x0a".hex #-> 10

"hello".hex #-> 0

1. str.include? other\_str #-> true или false

str.include? fixnum #-> true или false

Возвращает true, если строка str содержит передаваемые параметром строку other\_str или символ fixnum.

"hello".include? "lo" #-> true

"hello".include? "ol" #-> false

"hello".include? ?l #-> true

1. str.index(substring [, offset]) #-> fixnum или nil

str.index(fixnum [, offset]) #-> fixnum или nil

str.index(regexp [, offset]) #-> fixnum или nil

Возвращает индекс первого вхождения передаваемой параметром подстроки (substring), символа (fixnum) или совпадения с правилом (regexp) в строке str. Возвращает nil, если вхождения нет. В качестве второго параметра передается индекс (offset), с которого следует начинать поиск вхождений.

"hello".index("e") #-> 1

"hello".index("llo") #-> 2

"hello".index("z") #-> nil

"hello".index(/[gheo]/, 2) #-> 4

1. str.insert(index, other\_str) #-> str

Вставляет строку other\_str после указанного индекса index в строку str. Отрицательная индексация подразумевает нумерацию с конца строки.

"hello".insert(-1, "!") #-> "hello!"

"hello".insert(3, " ") #-> "hel lo"

1. str.intern #-> symbol

str.to\_sym #-> symbol

Возвращает объект класса Symbol, который соответствует строке str.

Этот метод может быть использован для создания символов, которые не могут быть созданы в :xxx нотации.

"hello".intern #-> ":hello"

1. str.length #-> integer

Возвращает размер строки str.

"hello".length #-> 5

1. lines(separator=$/ [, getline\_args]) → an\_array

Возвращает массив строк в str split с использованием предоставленного разделителя записей (по умолчанию $ /). Это сокращение для str.each\_line (separator, getline\_args) .to\_a.

"hello".lines #-> ["hello"]

"hello\nworld".lines #-> ["hello\n", "world"]

"hello world".lines " " #-> ["hello ", "world"]

"hello world".lines "l" #-> ["hel", "l", "o worl", "d"]

"hello world".each\_line("l").to\_a #-> ["hel", "l", "o worl", "d"]

1. str.ljust(integer, padstr=' ') #-> new\_str

Если параметр integer больше, чем длина строки str, то возвращается новая строка длины integer со строкой str, которая выравнена по левому краю, а возникшее пустое место заполняется символами padstr; иначе, возвращается строка str.

"hello".ljust 4 #-> "hello"

"hello".ljust 10 #-> "hello "

"hello".ljust 10, "13" #-> "hello13131"

1. str.lstrip #-> new\_str

Возвращает копию строки str, в которой удалены все ведущие пробельные символы.

" hello ".lstrip #-> "hello "

"hello".lstrip #-> "hello"

1. str.lstrip! #-> self или nil

Удаляет ведущие пробелы из строки str. Возвращает nil, если в результате работы метода строка str осталась неизменной.

temp = " hello "

temp.lstrip! #-> "hello "

temp.lstrip! #-> nil

1. str.match(pattern) #-> matchdata или nil

Преобразует параметр pattern в [правило](https://ru.wikibooks.org/wiki/Ruby/%D0%A1%D0%BF%D1%80%D0%B0%D0%B2%D0%BE%D1%87%D0%BD%D0%B8%D0%BA/Regexp) (если он таковым не был) и осуществляет сопоставление этого правила со строкой str.

"hello".match '(.)\w' #-> <MatchData "he" 1:"h">

"hel3 lo".match(/\w\d\s\w/) #-> #<MatchData "l3 l">

1. match?(pattern) → true or false

match?(pattern, pos) → true or false

Преобразует шаблон в Regexp (если это еще не так), затем возвращает true или false, указывает, соответствует ли регулярное выражение str или нет, без обновления $ ~ и других связанных переменных. Если присутствует второй параметр, он указывает позицию в строке для начала поиска.

"hello".match? 'h' #-> true

"hello".match? 'h', 1 #-> false

1. str.succ #-> new\_str

str.next #-> new\_str

Рассматривает строку str как элемент символьной последовательности и возвращает следующий за строкой str элемент. Следующий элемент вычисляется увеличением кода крайнего правого элемента строки str на единицу. В результате увеличения символа, который является цифрой — получится цифра, а для символа, который является буквой — буква. Увеличение остальных символов происходит с использованием базовой символьной упорядоченной последовательности.

Если в результате увеличения возникает необходимость «переноса», символ левее увеличиваемого в данный момент — тоже увеличивается. Этот процесс повторяется для всех требуемых «переносов». Если необходимо, то к строке str будет добавлен дополнительный символ.

"hello".next #-> "hellp"

"hello3".next #-> "hello4"

"hello+\*".next #-> "hellp+\*"

"+\*".next #-> "++"

"hello ".next #-> "hellp "

"<hello>".next #-> "<hellp>"

"19Z5ZZZ99".next #-> "19Z6AAA00"

1. str.next! #-> str

Эквивалентен методу next, но меняет строку str на результат своей работы.

temp = "hello"

temp.next! #-> "hellp"

1. str.oct #-> integer

Трактует строку str как строку восьмеричных цифр (с необязательным указанием знака или необятельным префиксом 0) и возвращает cоотвествующее число. Если преобразование не удается, то возвращает ноль.

"hello".oct #-> 0

"-10".oct #-> -8

"010hello".oct #-> 8

1. str.ord #-> integer

Возвращает целочисленный порядковый номер строки из одного символа.

"hello".ord #-> 104

"a".ord #-> 97

1. str.partition(sep) → [head, sep, tail]

str.partition(regexp) → [head, match, tail]

Ищет sep или шаблон (regexp) в строке и возвращает часть перед ним, совпадение и часть после него. Если он не найден, возвращает две пустые строки и строку.

"hello".partition "l" #-> ["he", "l", "lo"]

"hello".partition "a" #-> ["hello", "", ""]

"hello".partition /.ll/ #-> ["h", "ell", "o"]

1. str.prepend(other\_str1, other\_str2, ...) → str

Добавляет указанные строки к строке str.

"hello".prepend "world" #-> "worldhello"

1. str.replace #-> str

Заменяет содержимое str соответствующими значениями из other\_str.

a = "hello"

a.replace "world" #-> "world"

1. str.reverse #-> new\_str

Возвращает новую строку, в которой символы строки str переставлены в обратном порядке.

"hello".reverse #-> "olleh"

1. str.reverse! #-> str

Изменяет порядок символов строки str на обратный.

temp.replace "hello"

temp.reverse! #-> "olleh"

1. str.rindex(substring [, fixnum]) #-> fixnum или nil

str.rindex(fixnum [, fixnum]) #-> fixnum или nil

str.rindex(regexp [, fixnum]) #-> fixnum или nil

Возвращает индекс последнего вхождения передаваемой параметром подстроки (substring), символа (fixnum) или совпадения с правилом (regexp) в строке str. Возвращает nil, если вхождения нет. В качестве второго параметра передается индекс (fixnum), на котором следует закончить поиск.

"hello".rindex("e") #-> 1

"hello".rindex("llo") #-> 2

"hello".rindex("z") #-> nil

"hello".rindex(/[heo]/, 3) #-> 1

1. str.rjust(integer, padstr=' ') #-> new\_str

Если параметр integer больше, чем длина строки str, то метод возвращает новую строку длины integer со строкой str выравненной по правому краю, а возникшее пустое место заполняется символами padstr; иначе, возвращается строка str.

"hello".rjust 4 #-> "hello"

"hello".rjust 10 #-> " hello"

"hello".rjust 10, "13" #-> "13131hello"

1. str.rstrip #-> new\_str

Возвращает копию строки str, в которой удалены все замыкающие пробельные символы.

" hello ".rstrip #-> " hello"

"hello".rstrip #-> "hello"

1. str.rstrip! #-> self или nil

Удаляет замыкающие пробельные символы из строки str. Возвращает nil, если в результате работы метода никаких изменений сделано не было.

temp.replace "hello "

temp.rstrip! #-> "hello"

temp.rstrip! #-> nil

1. str.setbyte(index, integer) # => integer

Изменяет индексный байт в строке str как целое число.

temp.replace "hello"

temp.setbyte 0, 65

temp #-> "Aello"

1. str.size #-> integer

Возвращает размер строки str.

"hello".size #-> 5

1. str.slice(index) #-> fixnum или nil

str.slice(start, length) #-> new\_str или nil

str.slice(range) #-> new\_str или nil

str.slice(regexp) #-> new\_str или nil

str.slice(regexp, index) #-> new\_str или nil

str.slice(other\_str) #-> new\_str или nil

Получение подстроки или символа — возвращает код символа с индексом index, или подстроку длины length, начиная с индекса start, или подстроку, которая располагается в диапазоне range. Во всех этих вариантах вызова отрицательная индексация подразумевает отсчет с конца строки str.

Возвращается nil, если индекс index выходит за пределы допустимого диапазона, размер length запрашиваемой подстроки отрицательный или начало диапазона range попадает после конца строки str.

Если передается правило regexp, то возвращается фрагмент строки str, который удовлетворяет правилу regexp (или nil, если такого совпадения найти не удалось).

Если задан дополнительный параметр index, то возвращается содержимое группировки правила с номером index (если index равен 0, то возвращается фрагмент строки, который удовлетворяет правилу regexp).

Если в качестве аргумента передается строка other\_str, то возвращается строка other\_str, если она является подстрокой строки str (или nil, иначе).

"hello".slice 0 #-> "h"

"hello".slice 0,3 #-> "hel"

"hello".slice 0..3 #-> "hell"

"hello".slice -3..-1 #-> "llo"

"hello".slice -1..-3 #-> ""

"hello".slice 6 #-> nil

"hello".slice "he" #-> "he"

"hello".slice "help" #-> nil

"hello".slice /[ah](.)\w/ #-> "hel"

"hello".slice /[ah](.)\w/,0 #-> "h"

1. str.slice!(fixnum) #-> fixnum или nil

str.slice!(fixnum, fixnum) #-> new\_str или nil

str.slice!(range) #-> new\_str или nil

str.slice!(regexp) #-> new\_str или nil

str.slice!(other\_str) #-> new\_str или nil

Удаляет определенный кусок текста из строки str и возвращает этот кусок в качестве результата.

Если в качестве параметра передается число (Fixnum), то возможно возникновение ошибки типа IndexError, когда значение этого числа находится вне допустимого диапазона.

Если в качестве параметра передается диапазон (Range), то возможно возникновение ошибки типа RangeError, когда диапазон выходит за рамки допустимых значений. В случае с параметрами типа Regexp и String метод молча реагирует на недопустимые значения.

temp.replace "hello world"

temp.slice 0 #-> "h"

temp.slice 3..4 #-> "lo"

temp.slice /o.\wo/ #-> "lo"

1. str.split(pattern=$;, [limit]) #-> anArray

Делит строку str на подстроки по разделителю pattern (который может быть как правилом, так и строкой). Если разделитель pattern не указан, то деление происходит по пробельному символу (если иное не присвоено специальной переменной $;). В результате деления возвращается массив, который содержит фрагменты строки str (сам разделитель в результат не входит).

Если разделитель pattern является правилом, то деление производится по подстрокам, подходящим под данное правило. Если pattern — строка, то деление производится по подстрокам, которые совпадают с разделителем.

Если задан необязательный параметр limit, то результирующий массив будет иметь количество фрагментов строки str равное limit. Последний элемент будет содержать остаток, который, возможно, еще можно поделить (то есть в строке есть еще разделители).

"hello world".split #-> ["hello", "world"]

"hello world".split(//) #-> ["h", "e", "l", "l", "o", " ", "w", "o", "r", "l", "d"]

"hello world".split(//,4) #-> ["h", "e", "l", "lo world"]

1. str.squeeze(del=nil) #-> new\_str

Создает множество символов из строки (или строк) del, переданных в качестве параметра. Возвращает новую строку, где идущие подряд одинаковые символы, которые принадлежат множеству (переданному в качестве параметра), заменяются на один такой символ. Если метод вызван без параметра, то все идущие подряд повторяющиеся символы будут заменены на соответствующий единичный.

"hello".squeeze #-> "helo"

"hhhello".squeeze "h" #-> "hello"

"hhheeello".squeeze "h-l" #-> "heeelo"

1. str.squeeze!(del=nil) #-> new\_str

Работает точно также, как и метод squeeze, но результат своей работы записывает в исходную строку.

temp.replace "hello"

temp.squeeze! #-> "helo"

1. str.strip #-> new\_str

Возвращает копию строки str в которой удалены ведущие и замыкающие пробельные символы.

" hello \n\r ".strip #-> "hello"

1. str.strip! #-> str или nil

Удаляет ведущие и замыкающие пробельные символы из строки str. Возвращает nil, если строка str таковых не содержит.

temp.replace " hello \n\r "

temp.strip! #-> "hello"

temp.strip! #-> nil

1. str.sub(pattern, replacement) #-> new\_str

str.sub(pattern) {|match| block } #-> new\_str

Возвращает копию строки str, где первое совпадение с шаблоном (или строкой) pattern заменено на строку replacement или результат выполнения блока (которому передается параметром результат совпадения).

В результате работы метода, найденное совпадение с шаблоном pattern записывается в специальную переменную $& (наследие языка Perl). В строке replacement возможно использование последовательностей вида \1, \2 и так далее до \9, которые являются ссылками на совпадения с группировками (номер группировки считается слева направо). Внутри блока на группировки можно ссылаться при помощи специальных переменных вида $1, $2 и так далее до $9. Также, выражению в блоке доступны специальные переменные $`, $& и $', которые позволяют получить доступ к подстроке до совпадения, совпадению и подстроке после совпадения, соответственно.

"hello".sub(/[eao]/,"-") #-> "h-llo"

"hello".sub(/[eao]/) {|c| c[0].to\_s + " 1334 "} #-> "he 1334 llo"

1. str.sub!(pattern, replacement) #-> str или nil

str.sub!(pattern) {|match| block } #-> str или nil

Выполняет замену подобно sub, но изменяет исходную строку str. Возвращает результат замены или nil, если замена невозможна.

temp.replace "hello"

temp.sub!(/[ea]/,"-") #-> "h-llo"

temp.sub!(/[ea]/,"-") #-> nil

1. str.succ #-> new\_str

Рассматривает строку str как элемент символьной последовательности и возвращает следующий за строкой str элемент. Следующий элемент вычисляется увеличением кода крайнего правого элемента строки str на единицу. В результате увеличения символа, который является цифрой — получится цифра, а для символа, который является буквой — буква. Увеличение остальных символов происходит с использованием базовой символьной упорядоченной последовательности. Если в результате увеличения возникает необходимость «переноса», символ левее увеличиваемого в данный момент — тоже увеличивается. Этот процесс повторяется для всех требуемых «переносов». Если необходимо, то к строке str будет добавлен дополнительный символ.

"hello".succ #-> "hellp"

"hello3".succ #-> "hello4"

"hello+\*".succ #-> "hellp+\*"

"+\*".succ #-> "++"

"hello ".succ #-> "hellp "

"<hello>".succ #-> "<hellp>"

"19Z5ZZZ99".succ #-> "19Z6AAA00"

1. str.succ! #-> str

Эквивалентен методу succ, но меняет строку str на результат своей работы.

temp.replace "hello"

temp.succ! #-> "hellp"

1. str.sum(n=16) #-> integer

Возвращает простую n-битную контрольную сумму символов строки str, где n опциональный целочисленный (Fixnum) параметр, по-умолчанию равный 16. Результат – это просто суммирование двоичных значений каждого символа строки str по модулю 2n - 1.

"h".getbyte 0 #-> 104

"e".getbyte 0 #-> 101

"he".sum #-> 205

1. str.swapcase #-> new\_str

Возвращает копию строки str в которой все символы нижнего регистра заменены на соответствующие символы верхнего и все символы верхнего регистра заменены на соответствующие символы нижнего.

"HellO".swapcase #-> "hELLo"

1. str.swapcase! #-> str или nil

Модифицирует строку str по правилу: все символы нижнего регистра заменены на соответствующие символы верхнего и все символы верхнего регистра заменены на соответствующие символы нижнего. Возвращает nil, если изменения не требуются.

temp.replace "HellO"

temp.swapcase! #-> "hELLo"

"1".swapcase! #-> nil

1. str.to\_c → complex

Возвращает комплекс, обозначающий строковую форму. Парсер игнорирует начальные пробелы и мусор в конце. Любые последовательности цифр можно разделять знаком подчеркивания. Возвращает ноль для нулевой или мусорной строки.

"10".to\_c #-> (10+0i)

"-i".to\_c #-> (0-1i)

"3-4i".to\_c #-> (3-4i)

"a".to\_c #-> (0+0i)

1. str.to\_f #-> float

Возвращает результат интерпретации строки str, как дробного числа. Символы после последнего числового — игнорируются. Если строка str не является дробным числом, то возвращается число с добавлением в конец ".0", а если в строке нет цифр, возвращается 0.0.

"111.1hg1".to\_f #-> 111.1

"111".to\_f #-> 111.0

"a".to\_f #-> 0.0

1. str.to\_i(base=10) #-> integer

Возвращает в качестве результата интерпретацию символов строки str, как целого числа с основанием base (2, 8, 10, 16 или любого другого). Символы после последнего числового — игнорируются. Если строка str не является числом, то возвращается 0.

"111.1hg1".to\_i #-> 111

"10".to\_i 2 #-> 10

"10".to\_i 8 #-> 10

"10".to\_i 10 #-> 10

"10".to\_i 16 #-> 10

"a".to\_i #-> 0

1. str.to\_r → rational

Возвращает результат интерпретации начальных символов в str как рациональных. Начальные пробелы и посторонние символы после конца допустимого числа игнорируются. Последовательности цифр могут быть разделены знаком подчеркивания. Если в начале str нет действительного числа, возвращается ноль. Этот метод никогда не вызывает исключения.

"111.1hg1".to\_r #-> 1111/10

"3".to\_r #-> 3/1

"10.5".to\_r #-> 21/2

"10/5".to\_r #-> 2/1

1. str.to\_s #-> str

Возвращает строку str в качестве результата. Никаких преобразований не производится.

54.to\_s #-> "54"

"54".to\_s #-> "54"

1. str.to\_str #-> str

Возвращает строку str в качестве результата. Никаких преобразований не производится.

54.to\_str #-> error

"54".to\_str #-> "54"

1. str.to\_sym #-> symbol

Возвращает объект класса [Symbol](https://ru.wikibooks.org/wiki/Ruby/%D0%A1%D0%BF%D1%80%D0%B0%D0%B2%D0%BE%D1%87%D0%BD%D0%B8%D0%BA/Symbol), который соответствует строке str.

Этот метод может быть использован для создания [символов](https://ru.wikibooks.org/wiki/Ruby/%D0%A1%D0%BF%D1%80%D0%B0%D0%B2%D0%BE%D1%87%D0%BD%D0%B8%D0%BA/Symbol), которые не могут быть созданы в :xxx нотации.

"hello".to\_sym #-> :hello

1. str.tr(from, to) #-> new\_str

Возвращает копию строки str в которой символы из строки from заменены на соответствующие символы из строки to. Если строка to короче, чем from, то строка дополняется своим последним символом до длины строки from. Оба строковых параметра (from и to) могут использовать нотацию c1-c2, которая разворачивается в последовательность символов в диапазоне от с1 до с2. Если первым символом в строковом параметре указать символ ^, то это будет означать все множество символов, кроме указанных.

"hello".tr "aeo","-" #-> "h-ll-"

"hello".tr "^aeo","-" #-> "-e--o"

"hello".tr "he","a" #-> "aallo"

"hello".tr "a-y","b-z" #-> "ifmmp"

1. str.tr!(from, to) #-> new\_str или nil

Преобразует строку str, используя алгоритм, описанный для метода tr. Возвращает результат преобразования или nil, если в результате работы метода изменений сделано не было.

temp.replace "hello"

temp.tr! "aeo","-" #-> "h-ll-"

temp.tr! "aeo","-" #-> nil

1. str.tr\_s(from, to) #-> new\_str

Создает копию строки str, которая преобразована по алгоритму, описанному в методе tr, но с предварительным удалением дубликатов символов, которые описаны в строке from.

"hello".tr\_s "l","-" #-> "he-o"

"hello".tr\_s "hl","-" #-> "-e-o"

"hello".tr\_s "el","le" #-> "hleo"

1. str.tr\_s!(from, to) #-> new\_str или nil

Преобразует строку str по алгоритму описанному в методе tr\_s. Возвращает результат преобразования или nil, если в результате работы метода преобразований сделано не было.

temp.replace "hello"

temp.tr\_s! "l","-" #-> "he-o"

1. str.undump #-> new\_str

Работает противоположно dump. Возвращает неэкранированную версию строки str.

"\"hello\"".undump #-> "hello"

"\"\\r\\nhello\\n\"".undump #-> "\r\nhello\n"

1. str.upcase #-> new\_str

Возвращает копию строки str в которой все символы нижнего регистра заменены на соответствующие символы верхнего.

"hEllO".upcase #-> "HELLO"

1. str.upcase! #-> str или nil

Модифицирует строку str по правилу: все символы нижнего регистра преобразовываются в соответствующие символы верхнего. Возвращает nil, если изменения не требуются.

temp.replace "hello"

temp.upcase! #-> "HELLO"

temp.upcase! #-> nil

1. str.upto(other\_str) {|s| block } #-> str

Данный итератор проходит все значения между str и other\_str включительно, передавая их в блок в качестве параметра.

"hello".upto("hellq") {|s| print s, ' ' } #-> hello hellp hellq

1. valid\_encoding? → true or false

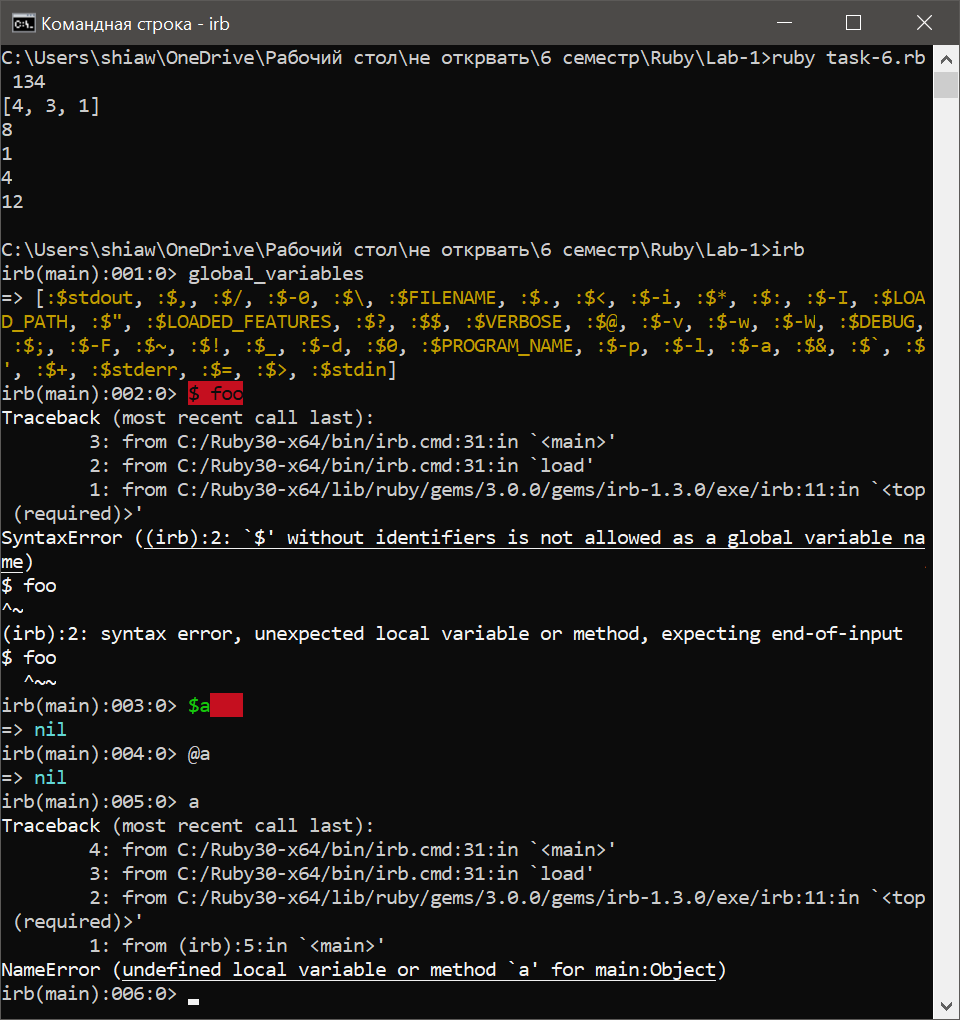
Возвращает истину для строки, которая закодирована правильно.

"a3{f%\*~".force\_encoding("UTF-8").valid\_encoding? #-> true

"\x80".force\_encoding("UTF-8").valid\_encoding? #-> false

Встроенные переменные.

Существует набор специальных переменных, имена которых состоят из знака доллара ($), за которым следует один символ. Вот основные системные переменные и их значения:



$! Последнее сообщение об ошибке.

$@ Место ошибки.

$. Номер строки, последний раз прочитанной интерпретатором.

$ & Строка, соответствующая последнему успешному совпадению.

$` Строка слева от последнего успешного совпадения.

$' Строка справа от последнего успешного совпадения.

$+ Наивысшая группа, соответствующая последнему удачному совпадению.

$1 N-я группа последнего удачного совпадения. Может быть> 1.

$~ Последнее совпадение регулярного выражения в виде массива подвыражений.

$= Флаг нечувствительности к регистру.

$/ Разделитель входной записи, по умолчанию перевод строки.

$\ Разделитель выходной записи, по умолчанию nil.

$, Разделитель выходных полей для Kernel#printи Array#join. Не-nil $ будет устаревшим.

$; Разделитель по умолчанию для String#split. Не-ноль $; будет устаревшим. С псевдонимом $ -F.

$. Номер текущей строки ввода последнего прочитанного файла.

$ < То же, что и ARGF.

$> Выходной поток по умолчанию для Kernel#printи Kernel#printf. По умолчанию $ stdout.

$\_ Последняя строка ввода строки с помощью get или readline.

$0 Содержит имя выполняемого скрипта. Может быть назначен.

$\* То же, что и ARGV.

$$ Номер процесса Ruby, на котором запущен этот скрипт. То же, что и Process.pid.

$? Состояние последнего выполненного дочернего процесса (локальный для потока).

$ LOAD\_PATH Путь загрузки для поиска скриптов Ruby и библиотек расширений, используемых Kernel#loadи Kernel#require.

$ LOADED\_FEATURES Массив содержит имена модулей, загруженные с помощью require. Алиас для $“.

$ DEBUG Флаг отладки, который устанавливается -d переключателем. При включении вывода отладки печатается каждое исключение, возникшее в $stderr (но не его обратная трассировка). Установка этого значения в истинное значение включает вывод отладки, как если бы -d был задан в командной строке. Установка этого значения в значение false отключает вывод отладки. Алиас для $ -d.

$ FILENAME Текущее входное имя файла из ARGF. То же, что и ARGF.filename.

$ stderr Текущий стандартный вывод ошибок.

$ stdin Текущий стандартный ввод.

$ stdout Текущий стандартный вывод.

$ VERBOSE Флаг подробности, который устанавливается переключателем -w или -v. Установка этого параметра в истинное значение включает предупреждения, как если бы -w или -v были заданы в командной строке. Установка этого значения nil отключает предупреждения, в том числе от Kernel#warn. Алиас для $ -v и $ -w.

$ -a Истинно, если опция -a установлена. Переменная только для чтения.

$-i В режиме редактирования на месте эта переменная содержит расширение, в противном случае nil.

$ -l Истинно, если опция -l установлена. Переменная только для чтения.

$-p Истинно, если опция -p установлена. Переменная только для чтения.

Предопределенные глобальные константы:

STDIN Стандартный ввод. Значение по умолчанию для $stdin.

STDOUT Стандартный вывод. Значение по умолчанию для $stdout.

STDERR Стандартный вывод ошибок. Значение по умолчанию для $stderr.

ENV Хеш содержит текущие переменные среды.

ARGF Виртуальная конкатенация файлов, заданных в командной строке (или из $stdin, если файлы не заданы).

ARGV Массив аргументов командной строки заданных для скрипта.

DATA Файловый объект скрипта, указывающийся сразу после \_\_END\_\_.

TOPLEVEL\_BINDING Привязка области верхнего уровня.

RUBY\_VERSION Версия языка Ruby.

RUBY\_RELEASE\_DATE Строка даты выпуска.

RUBY\_PLATFORM Идентификатор платформы.

RUBY\_PATCHLEVEL Уровень патча для этого Ruby. Если это разрабатываемая сборка Ruby, уровень исправлений будет -1.

RUBY\_REVISION Хеш фиксации GIT для этого Ruby.

RUBY\_COPYRIGHT Строка авторских прав для Ruby.

RUBY\_ENGINE Имя реализации Ruby.

RUBY\_ENGINE\_VERSION Версия реализации Ruby.

RUBY\_DESCRIPTION То же ruby --version, строка, описывающая различные аспекты реализации Ruby.

Способы форматирования строки:

1. С использованием printf или sprint:

sprintf("%d %04x", 123, 123) #=> "123 007b"

sprintf("%08b '%4s'", 123, 123) #=> "01111011 ' 123'"

sprintf("%1$\*2$s %2$d %1$s", "hello", 8) #=> " hello 8 hello"

sprintf("%1$\*2$s %2$d", "hello", -8) #=> "hello -8"

sprintf("%+g:% g:%-g", 1.23, 1.23, 1.23) #=> "+1.23: 1.23:1.23"

sprintf("%u", -123) #=> "-123"

1. Включением кода или переменной внутрь строки:

puts "three plus three is #{3+3}" #=> "three plus three is 6"

user = “Leo”

puts "Your username is #{user}" #=> "Your username is Leo"

1. С помощью процентных литералов:

% q[] Неинтерполированная строка (кроме \\ \ [и \])

% Q[] Интерполированная строка (по умолчанию)

%р[ ] Интерполированное регулярное выражение (флаги могут появляться после закрывающего разделителя)

%s[ ] Неинтерполированный символ

% w[] Неинтерполированный массив слов, разделенных пробелами

% W[] Интерполированный массив слов, разделенных пробелами

%x[ ] Интерполированная команда оболочки

Пример:

p %w{hello world} #=> ['hello','world']

p %s{hello} #=> :hello