1НАЦІОНАЛЬНА АКАДЕМІЯ УПРАВЛІННЯ
ФАКУЛЬТЕТ КОМП'ЮТЕРНИХ НАУК
КАФЕДРА ІНТЕЛЕКТУАЛЬНИХ СИСТЕМ

ОБ`ЄКТНО-ОРІЄНТОВАНЕ ПРОГРАМУВАННЯ

ЧАСТИНА 1. ПРОГРАМУВАННЯ НА МОВІ RUBY

Навчально-методичні матеріали

КИЇВ - 2009

ОБ`ЄКТНО-ОРІЄНТОВАНЕ ПРОГРАМУВАННЯ. ЧАСТИНА 1. ПРОГРАМУВАННЯ НА МОВІ RUBY. Навчально методичні матеріали / Укл. Баклан І.В., Степанкова Г.А. - К.: НАУ, 2009, 164 с.

Матеріали присвячені вивченню об`єктно-орієнтованої мови програмування Ruby, одної з найпопулярніших мов сучасності, яка отримала прикладне застосування у Web-технології — Ruby-on-Rails.

Затверджено на засіданні Ради факультету комп'ютерних наук.

Протокол №1 від 10 вересня 2009 р.

ЗМІСТ

ЛАБОРАТОРНА РОБОТА 1....4
ОБ'ЄКТНО-ОРІЄНТОВАНА МОВА RUBY. ОСНОВНІ ВЛАСТИВОСТІ МОВИ....4
1.1. ОСНОВНІ ВІДОМОСТІ ПРО МОВУ RUBY....4
1.2. ПОЧАТКОВІ ДАНІ....4
1.3. ОБ'ЄКТИ ТА МЕТОДИ....6
1.4. ВИВЕДЕННЯ ДАНИХ....12
1.5. ЗМІННІ ТА КОНСТАНТИ....13
1.6. МАСИВИ....16
1.7. ВВЕДЕННЯ ДАНИХ....22
ЛАБОРАТОРНА РОБОТА 2....29
ОБ'ЄКТНО-ОРІЄНТОВАНА МОВА RUBY. ОСНОВНІ ВЛАСТИВОСТІ МОВИ (ПРОДОВЖЕННЯ)....29
2.1. МЕТОДИ....29
2.2. УМОВНІ ОПЕРАТОРИ....32
2.3. ОПЕРАТОРИ ЦИКЛУ....36
ЛАБОРАТОРНА РОБОТА 3....42
КОМПЛЕКСНЕ ПРОГРАМУВАННЯ МОВОЮ RUBY....42
3.1. БІБЛІОТЕКИ....42
3.2. ПРИКЛАДИ ПРОГРАМ....45
ДОВІДНИК ПО RUBY....68
Клас Class < Module....68
Класс Array....71
Клас Hash....95
Клас Matrix....108
Клас Proc....118
Клас Range....121
Клас String....126
Клас Struct....156
Клас GC....161
ПРИКЛАД ОФОРМЛЕННЯ ПРОТОКОЛУ З ЛАБОРАТОРНОЇ РОБОТИ....162

ЛАБОРАТОРНА РОБОТА 1. ОБ'ЄКТНО-ОРІЄНТОВАНА МОВА RUBY. ОСНОВНІ ВЛАСТИВОСТІ МОВИ.

1.1. ОСНОВНІ ВІДОМОСТІ ПРО МОВУ RUBY

Ruby - одна з наймолодших мов програмування. Своїм ім'ям вона зобов'язана дорогоцінному каменю рубіну (за аналогією з іншою широко розповсюдженою мовою програмування Perl - перли). От як описує Ruby його творець, японський програміст Юкихиро Мацумото (Yukihiro Matsumoto): "Це потужна і динамічна об'єктно-орієнтована мова з відкритими вихідними кодами, яку я почав розробляти в 1993 році. Ruby працює на багатьох платформах, включаючи Linux і багато реалізацій Unix, MS-DOS, Windows 9x/2000/NT, BeOS й MacOS. Головна мета Ruby - ефективність розробки програм. Користувачі знайдуть, що програмування на ній ефективне й навіть забавне".

В Японії Ruby сильно потіснила такі відомі мови як Python й Perl (а книга "Ruby the Object-Oriented Scripting Language" стала бестселером) та почала поширюватися в усьому світі. За останній рік з'явилися три англомовних книги, присвячені Ruby (на жаль, поки не мають українського перекладу). У цієї мови дуже непогані шанси стати дійсно популярною - адже вона увібрала у себе достоїнства інших мов, врахувавши їхні недоліки. Вона входить у стандартну поставку ОС Linux (починаючи з версії 7.2), а користувачам MS Windows для першого знайомства варто порекомендувати його трохи застарілу версію (1,2 Мбайт), що включає, крім інтерпретатора мови й бібліотек, керівництво користувача, FAQ та безліч прикладів. Ruby є вільно розповсюджуваним продуктом, тому ви можете не турбуватися ні про його вартість, ні про обмеження в його використанні.

Ця мова безсумнівно є одною з кращих як перша мова програмування, яку вивчатимуть студенти й школяри. Швидкий цикл розробки (редагування - запуск - редагування), використання інтерпретатора, споконвічна об'єктно-орієнтованість мови, нетипізовані змінні, які не вимагають оголошення - все це дозволяє учням сконцентрувати свою увагу на загальних принципах програмування. Надалі ми будемо орієнтуватися на роботу в ОС Linux. Використання Ruby в інших операційних системах практично нічим не відрізняється, а результати виконання завдань не залежать від використовуваної ОС.

1.2. ПОЧАТКОВІ ДАНІ

Спочатку перевіримо, чи встановлений інтерпретатор Ruby у системі чи ні. У вікні **shell** введіть команду `ruby -v` (ключик `-v' вимагає вказівки інтерпретатором версії мови). Якщо наступне повідомлення з'явиться, то Ruby встановлений (версія, дата й платформа можуть відрізнятися):

```
ruby 1.6.4 (2001-06-04) [i386-linux-gnu]
```

Файли, що містять програми мовою Ruby, звичайно мають розширення **`*.rb`**. По давній програмістській традиції наша перша програма буде друкувати фразу "Hello, World!". За

допомогою будь-якого редактора *plain*-тексту (emacs, kwrite, notebook і т.п.) створимо файл `hello.rb`, у якій помістимо текст

```
puts "Hello, World!"
```

Для виконання цієї програми в командному рядку введіть

```
ruby hello.rb
```

У результаті виконання програми в командному вікні буде надрукована необхідна фраза.

Другий спосіб виконання програм доступний користувачам не всіх операційних систем, у яких функціонує Ruby. Користувачам ОС Linux варто помістити в початок файлу з текстом програми наступний рядок:

```
#!/usr/bin/env ruby
```

Вона **обов'язково** повинна починатися з першої позиції. Потім потрібно змінити права доступу файлу із програмою, зробивши його таким, що виконується:

```
chmod +x hello.rb
```

Тепер для запуску програми досить увести команду

```
./hello.rb
```

Для того, щоб зробити програму більш зрозумілою людині, що її читає, вставляються коментарі. Однорядкові коментарі починаються із символу `#` і тривають до кінця рядка. Багаторядкові коментарі вкладають у спеціальні "дужки" - усе, що розташовується між рядками `=begin` й `=end`, вважається коментарем. Наприклад,

```
#!/usr/bin/env ruby
=begin
Це
 коментар
=end
puts "Hello, World!"
# Це теж коментар
```

Програма мовою Ruby, яку часто звуть скриптом, є послідовністю інструкцій (тверджень, пропозицій). Кожна інструкція за замовчуванням закінчується кінцем рядка. Якщо ж за якихось причин потрібно розмістити кілька інструкцій на одному рядку, то їх потрібно розділяти символом `;` (крапка з комою). З іншого боку, іноді інструкція не міститься на одному рядку. У цьому випадку символ `\` сигналізує про те, що її продовження розташовується в наступному рядку.

Приклад

```
#!/usr/bin/env ruby
# Інструкція закінчується кінцем рядка
puts "Hello, World!"

# Кілька інструкцій в одному рядку
```

```
puts "Це тест, "; puts "який демонструє роботу Ruby."

# Незакінчена інструкція,
# продовження якої на наступному рядку
puts "Програмування на Ruby - " +
 "приємне заняття."


# Твердження, розділене на кілька рядків
puts \
    "І ми обов'язково цьому навчимося!"
```

Інструкція (твердження) найчастіше являє собою якусь послідовність операторів, застосованих до різних виразів, і (можливо) викликів функцій.

Далі рядок

```
#!/usr/bin/env ruby
```

у тексти програм не включається.

1.3. ОБ'ЄКТИ ТА МЕТОДИ

При описі Ruby було відзначено, що він є об'єктно-орієнтованою мовою програмування. Це означає що все, з чим працює програма, є об'єктом, при цьому обчислення здійснюється за допомогою взаємодії об'єктів. Кожен об'єкт є представник (екземпляр) деякого класу і його поводження (функціональність) визначається саме класом. Тим самим всі об'єкти, які є екземплярами одного класу, можуть виконувати ті самі дії, називані методами. Для того, щоб застосувати метод до деякого об'єкта використовується оператор виклику методу, який позначається символом `.` (крапка): після вказівки об'єкта ставиться крапка, а потім указується застосований метод.

Поки що ми не будемо вдаватися в особливості об'єктно-орієнтованого програмування (тим більш, що в різних мовах програмування по різному трактується концепція ООП), а лише познайомимося з деякими "найпростішими" об'єктами, без яких неможливо приступати до подальшого вивчення програмування. Це представники так званих "вбудованих" класів. Серед них:

`true`
: логічна величина, що означає істину; єдиний представник класу `TrueClass;`

`false`
: логічна величина, що означає неправду; єдиний представник класу `FalseClass;`

`числа`
: представники класу `Numeric,` що як підкласи містять дробові (`Float`) і цілі числа (`Integer`);

`строки`
: представники класу `String`;

`nil`
: "ніщо"; єдиний представник класу `NilClass`.

моменти часу

представники класу `Time`.

1.3.1. Числа

Знайомство із представниками різних класів почнемо із чисел. Цілі числа в Ruby є екземплярами класу **`Integer`**, що поєднує два підкласи: **`Fixnum`** або **`Bignum`**. Об'єктами класу Fixnum є ті цілі числа, чиє двійкове подання здатне розміститися в машинному слові (31 біт на більшості комп'ютерів). Якщо число виходить за межі зазначеного діапазону, то воно **автоматично** перетвориться в об'єкт класу Bignum, чий діапазон змін обмежується тільки об'ємом доступної пам'яті. Якщо в результаті роботи з об'єктами класу Bignum підсумкове значення попадає в діапазон, що задається класом Fixnum, то вихідний результат перетвориться в екземпляр класу Fixnum. При записі цілих чисел спочатку вказується його знак (знак `+` звичайно опускається). Далі йде основа системи числення, у якій задається число (якщо вона відмінна від 10): 0 - для восьмеричної, 0x - для шістнадцятирічної, 0b - для двійкової. Потім іде послідовність цифр, що виражає число в даній системі числення. При записі більших чисел можна використати символ підкреслення, що ігнорується при обробці числа. У таблиці представлені цілі числа й зазначена їхня приналежність до того або іншого класу.

Число	Клас
123456	`Fixnum`
123_456	`Fixnum` (підкреслення ігнорується)
-543	`від'ємне Fixnum`
123_456_789_123_456	`Bignum`
0xaabb	`шістнадцятирічне`
0377	`восьмеричне`
-0b1010	`двійкове (негативне)`
0b001_001	`двійкове`

Дробові числа задаються в десятковій системі числення, при цьому для відділення дробової частини використовується символ `.` (крапка). Такі числа є екземплярами класу **`Float`**, наприклад, `12.34`. Для завдання дробових чисел може бути застосована й *експонентна* форма запису: два різних подання `-.1234e2` й `1234e-2` задають одне й теж число 12.34.

Для обчислення арифметичних виразів застосовуються наступні оператори: `+` (додавання), `-` (вирахування), `*` (множення), `/` (ділення), `%` (остача від ділення), `**` (піднесення в ступінь). Порядок обчислення визначається стандартними пріоритетами операцій, а для його зміни використаються круглі дужки. Помітимо, що якщо всі аргументи виразу цілі числа, то й результат буде цілим, якщо ж хоча б одне із чисел, що входять у вираз - дробове, то й результат буде екземпляром класу `Float`.

Приклад

Створіть файл із ім'ям `object.rb`, у який помістить наступний текст

```
puts 5/8                        # 0
puts 5.0/8                      # 0.625
puts 2**1000
puts ((2*500+1)*(2**500-1))
```

Виконайте програму й поясніть результат.

В Ruby є кілька методів, що дозволяють перетворювати об'єкти одного класу в іншій.

Метод	Призначення методу	Приклад використання	Результат
to_f	**Перетворити об'єкт в екземпляр класу Float**	`1234.to_f`	`1234.0`
to_i	**Перетворити об'єкт в екземпляр класу Fixnum або Bignum**	`-12.34. to_i`	`-12`

В останньому прикладі перша крапка відокремлює дробову частину числа, а друга є оператор виклику методу.

Відзначимо ще кілька методів, які використовуються при роботі з числами (тобто представниками класу `Numeric`). Серед них: `ceil` (знаходження найменшого цілого не меншого, ніж дане), `floor` (найбільше ціле, не більше даного), `round` (округлення до найближчого цілого), одержання абсолютної величини числа `abs`. Нижче наведені приклади використання цих методів.

```
puts 12.34.ceil                 #  13
puts 12.34.floor                #  12
puts -12.ceil                   # -12
puts -12.floor                  # -12
puts 12.34.round                #  12
puts 12.54.round                #  13
puts -34.56.abs                 #  34.56
```

1.3.2. Рядки

Іншим настільки ж розповсюдженим класом є клас **`String`**. До цього класу відносяться довільні рядки символів, вкладені в апострофи або лапки, наприклад, `'hello'`, `'раз, два, три'`, `"привіти всім"`. Є кілька альтернативних способів завдання кожного із цих видів рядків.

Для завдання рядка в апострофах можна використати кожної зі способів представлених нижче.

```
puts 'hello'         # hello
puts %q/hello/       # hello
puts %q(hello)       # hello
```

Два символи ****, що йдуть один за одним, усередині такого рядка заміняються на один.

```
puts 'hell\\o'       #  hell\o
puts %q(hell\\o)     #  hell\o
```

Для того, щоб усередину рядка, взятого у лапки, вставити апостроф, треба попередньо "екранувати" його символом ****.

```
puts 'hell\'o'       # hell'o
puts %q(hell\'o)     # hell\'o
puts 'hel"l"o'       # hel"l"o
```

Для створення рядка, взятого в лапки, треба або просто взяти його в лапки, або використати конструкції **%Q** або **%**, після яких указується рядок, обрамлений з двох сторін тим самим символом, відмінним від цифр і букв (можна використати пару будь-яких дужок). Нижче перераховано кілька способів завдання рядка "hello":

```
"hello"  %Q/hello/ %Q{hello} %Q(hello) %Q!hello!
 %<hello> %(hello)  %!hello!   %*hello*  %+hello+
```

Для включення усередину рядка, взятого в лапки, символу **"**, варто екранувати його символом \:

```
puts "Say \"Hello\""   #  Say "Hello"
```

Рядок, взятий у лапки, дозволяє інтерпретувати послідовності символів, що починаються із символу **** (зворотний слеш), такі, наприклад, як **\n** (символ переходу на новий рядок) і **\t** (табуляція). Іншою особливістю рядків, взятих у лапки, є можливість використання **підстановки**: якщо рядок містить деякий вираз, обмежений символами **#{** й **}**, те він замінюється на результат його обчислення. Додайте в програму наступні рядки й проаналізуйте отриманий висновок.

```
puts "a \t b"; puts 'a \t b'
puts "вираз 3*5+8 дорівнює  #{3*5+8}"
puts 'вираз 3*5+8 дорівнює  #{3*5+8}'
puts "робота із цілими числами: 5/8 = #{5/8}"
puts "переведення у клас Float: 5/8 = #{5/8.to_f}"
```

До рядків також можуть застосовуватися методи **to_i** й **to_f**. При перетворенні до цілого числа відкидається кінцева частина рядка, як тільки зустрічається символ відмінний від цифри (виключення - знак плюс або мінус у першій позиції рядка). Аналогічні правила застосовуються й при перетворенні до дробового числа. Єдиною відмінністю є те, що перша знайдена крапка розцінюється як символ відділення дробової частини. Наступний фрагмент ілюструє сказане:

```
puts "-12.34".to_i
puts "12.34".to_f
puts "+12:34".to_i
puts "12qq34".to_f
```

Для одержання рядка, що містить символ із заданим ASCII кодом, використається метод **chr**, наприклад,

```
puts 209.chr
```

Варто пам'ятати, що цей метод може бути застосований тільки до позитивного цілого числа, що не перевищує 255.

Клас `String` надає велику кількість методів для роботи з рядками, деякі з яких представлені в наступній таблиці.

Призначення й приклад використання методу		Результат
+	Зчеплення рядків `"мол" + "око"`	`"молоко"`
*****	Повтор рядка `"ab" * 3`	`"ababab"`

[ ]	Повертає ASCII-код символу, що перебуває на зазначеній позиції рядка (відлік починається з нуля)
	`"abcdef"[0]`
	`"abcdef"[0].chr`
		`97`
		`"a"`

[поч..кін]	Повертає підрядок, укладений в зазначеному діапазоні (включаючи кінці) `"abcdef"[0..3]`	`"abcd"`
[поч, дов]	Повертає підрядок, що починається із зазначеної позиції й має задану довжину `"abcdef"[0,3]`	`"abc"`
capitalize	Замінює перший символ рядка (якщо він є літерою латинського алфавіту) на заголовну `"abc".capitalize`	`"Abc"`
chop	Видаляє останній символ рядка (два символи, якщо вони є "\r\n") `"abcdef".chop`	`"abcde"`

delete	Видаляє зазначені символи з рядка, може вказуватися діапазон зміни символів
	`"abcdef".delete('ea')`
	`"abcdef".delete('a-c')`
		`"bcdf"`
		`"def"`

index	Визначає номер позиції, з якої починається зазначений підрядок; можна вказувати номер позиції, з якої починається пошук
	`"abcdabcd".index("cd")`
	`"abcdabcd".index("cd",3)`
		`2`
		`6`

length
size
	Визначають довжину рядка (у байтах)
	`"12345678".length`
	`"12345678".size`
		`8`
		`8`

ljust
rjust
center
	Доповнюють рядок пробілами до зазначеної ширини, вирівнюючи відповідно по лівому краю, по правому краю або по центру
	`"123".ljust(8)`
	`"123".rjust(8)`
	`"123".center(8)`
		`"123     "`
		`"     123"`
		`"  123   "`

reverse	Повертає рядок, що містить символи у зворотному порядку `"1234567".reverse`	`"7654321"`
strip	Видаляє пробіли на початку й кінці рядка `"   123   ".strip`	`"123"`
squeeze	Залишає в групі повторюваних символів тільки один; допускається завдання списку символів, на яких поширюється дана дія	`"2-23*"` `"22-23*"`

	"22---23**".squeeze "22---23**".squeeze('*-')	
tr	Заміняє всі знайдені входження символів на задані "22+33=55".tr('25','47')	"44+33=77"

Окремого згадування заслуговує метод **eval**, що дозволяє динамічно виконувати інші методи. Цей метод аналізує переданий йому рядок, розглядаючи його як частину програми, і виконує її. Зверніть увагу, що при виклику цього методу використовується не крапкова, а *функціональна* форма запису:

```
puts eval("2**10")
puts eval('"мол".size * "око".size')   # 9
```

1.3.3. Час і дата

Екземпляр класу **Time** у мові Ruby містить інформацію про дату, час і тимчасову зону. Для створення об'єкта цього класу, що містить поточну дату й час, використається метод **now**. Такі об'єкти можуть бути аргументами операцій **+** й **-**:

```
puts Time.now
puts Time.now+60    # додали 60 секунд до поточного часу
puts Time.now-60    # відняли 60 секунд від поточного часу
```

Відзначимо деякі методи цього класу, для виклику яких використовується крапкова нотація.

Метод	Призначення методу
sec	Одержати число секунд
min	Одержати число хвилин
hour	Одержати число годин
mday й day	Одержати день місяця
mon й month	Одержати номер місяця
year	Одержати рік
wday	Одержати номер дня тижня
yday	Одержати номер дня в році
zone	Одержати інформацію про тимчасову зону
to_i	Одержати число секунд, що пройшли з 1 січня 1970 року

У мові Ruby є метод **sleep**, що змушує програму "заснути" на число секунд, зазначене як аргумент методу. Подивіться на приклад використання методів для роботи з об'єктами класу Time.

```
puts "До Нового року залишилося #{365-Time.now.yday} днів"
puts Time.now
puts "Почекаємо 10 секунд."
```

```
sleep(10)
puts Time.now
```

1.4. ВИВЕДЕННЯ ДАНИХ

Ми вже бачили, як здійснюється вивідення інформації в Ruby. Познайомимося з іншими операторами, що забезпечують друк даних. Створіть файл `print.rb`, додайте в нього наступний текст і подивіться на результат:

```
puts "puts завжди завершується переходом до нового рядка."
print "А оператор print не робить цього за замовчуванням, "
print "як ви бачите в цьому прикладі.\n"
print "print може бути викликаний відразу  ",
  "з декількома аргументами.\n"
```

Оператор **`p`** подібний до оператора **`puts`**, він відображає об'єкти у вигляді, який зрозумілий людині. Не слід використовувати його для виводу російських букв - він друкує їхні ASCII-коди.

```
p Time.now, Time.now+3600
```

Значно більшими можливостями володіє оператор **`printf`**, що забезпечує *форматний* вивід. Відразу після імені оператора вказується рядок формату, що містить як звичайні символи, виведені на друк, так і **специфікації перетворення**, які викликають перетворення й друк інших аргументів у тому порядку, в якому вони перераховані. Кожна специфікація перетворення починається із символу `%` і закінчується символом-специфікатором перетворення. Між `%` і символом-специфікатором у порядку, перерахованому нижче, можуть розташовуватися наступні модифікатори:

- прапори (у будь-якому порядку):

-	указує на те, що перетворений аргумент повинен бути притиснутий до лівого краю поля;
+	пропонує завжди друкувати число зі знаком;
0	указує, що числа повинні доповнюватися нулями до всієї ширини поля;
пробіл	якщо перший символ, що друкується, не знак, то числу повинен передувати пробіл;

- число, що задає мінімальну ширину поля;
- крапка, що відокремлює покажчик ширини поля від покажчика точності;
- число, що задає точність.

Основні літери-специфікатори й роз'яснення їхнього змісту приводяться в наступній таблиці.

Літера	Аргумент	Вигляд друку
`d`	`Fixnum, Bignum`	друк цілого числа

s	String	друк рядка
f	Float	друк дробового числа

Включить у файл наступні оператори форматного друку й подивіться на результати виконання:

```
printf "%8s~~%-8s\n", "abcd", "abcd"
printf "%8-s::%8s\n", "abcd", "abcd"
printf "%06d\n", 2**10; printf "%+6d\n", 2**10
printf "%6d\n", 2**10;  printf "%+6d\n", 2.5*1.3
printf "%4.3f\n", 2.5*1.3
printf "-2/7=%+1.6f, -2/6=%2.15f\n",
   -2/7.to_f, -2/6.to_f
```

Зверніть увагу, що рядок формату взятий в подвійні лапки й може містити спеціальні послідовності символів. Одною з них є **\n**. Ця послідовність дає вказівку інтерпретаторові Ruby продовжити виведення інформації з наступного рядка (newline). Іншою, часто використовуваною послідовністю, є **\t** (табуляція), що пересуває фокус висновку до наступної позиції табуляції.

```
puts "*\t*\t*\t*\t*"
puts " 1234567 1234567 1234567 1234567 "
```

1.5. ЗМІННІ ТА КОНСТАНТИ

Як й у багатьох інших мовах програмування в Ruby змінні є основними "будівельними блоками". У цьому курсі ми будемо використовувати так називані **локальні змінні.** Імена локальних змінних починаються з рядкової латинської букви (від a до z) або символу підкреслення й продовжуються будь-якою комбінацією латинських букв, цифр і символу підкреслення. Тіж правила поширюються й на імена методів. Є невелика кількість зарезервованих слів, які не можна використовувати як імена змінних і методів.

Зарезервовані слова							
`__FILE__`	and	def	end	in	or	self	unless
`__LINE__`	begin	defined?	ensure	module	redo	super	until
BEGIN	break	do	false	next	rescue	then	when
END	case	else	for	nil	retry	true	while
alias	class	elsif	if	not	return	undef	yield

Крім локальних змінних Ruby підтримує й інші їхні види - глобальні змінні, змінні класів й екземплярів. Приналежність до кожного з цих видів визначається символом, що стоїть перед ім'ям змінної (префіксом): символ **$** задає глобальну змінну, **@** - змінну екземпляра, **@@** - змінну класу. Локальні змінні не мають префікса. Імена констант і класів повинні починатися із прописної латинської букви (A-Z) і, аналогічно змінним, продовжуються будь-якою комбінацією латинських букв, цифр і символу підкреслення.

Локальна змінна створюється динамічно в момент, коли їй у процесі виконання програми перший раз привласнюється якесь значення. Оператор присвоювання має вигляд знака **=** (дорівнює), наприклад, **a = 1; b = a; name = "Іван"**. Змінні можуть використовуватись у всіх виразах Ruby аналогічно відповідним об'єктам:

```
a = "Привіт усім!"

puts a
```

В Ruby змінні є **посилання** на об'єкти. Їх варто трактувати як якісь мітки. Для того, щоб краще зрозуміти цю концепцію, проведемо наступну аналогію: нехай у вас є деяка коробка (об'єкт), ви берете наклейку (змінну), підписуєте її (даєте ім'я змінної), після чого наклеюєте її на коробку (об'єкт). В Ruby змінні не мають типу, тому що вони є всього лише посиланнями на об'єкти довільних типів. Проілюструємо сказане наступним прикладом:

```
a = "one"
puts a
puts a.type
a = 23
puts a
puts a.type
```

Зверніть увагу на метод **type**, що повертає тип об'єкта. При першому використанні змінна **a** є посилання на об'єкт класу **String**, потім ми привласнюємо цій змінній інше значення (наклеюємо ту ж етикетку на новий об'єкт), після чого вона стає посиланням на екземпляр класу **Fixnum**.

Ruby підтримує також і **присвоювання з операцією**. Результат виконання наступних двох операторів однаковий:

```
a+=12
a=a+12
```

Загальна форма подібних операторів виглядає так:

```
вираз1 операція= вираз2
```

що відповідає операторові

```
вираз1 = вираз1 операція вираз2
```

Між знаком операції й символом рівності не повинне бути пробілу. Такий синтаксис допустимий для наступних операцій:

```
+, -, *, /, %, **, &, |, ^, <<, >>, &&, |
```

Коли потрібно привласнити значення декільком змінним, можна використати **множинне присвоювання**:

```
a, b, c = 1, 2, 3
```

Приклад

Нехай потрібно обміняти значення двох змінних, не вводячи допоміжну змінну. З використанням множинного присвоювання це можна записати так:

```
a, b = b, a
```

Даний приклад пояснює, чому цю форму оператора присвоювання називають також "паралельним присвоюванням".

Ми вже згадували одну із чудових можливостей Ruby - **підстановку**. Вона дозволяє виконувати різноманітні трюки з рядком, взяті в подвійні лапки, наприклад,

```
magic = 42
puts "Секретне число дорівнює #{magic - 3}!!"
```

Коли Ruby знаходить конструкцію **#**`{вираз}` усередині рядка, взятого в подвійні лапки, то змінює цю її частину на результат обчислення виразу.

Приклад

Дане тризначне число. Надрукувати число, що виходить при перестановці цифр десятків й одиниць заданого числа.

```
number = 342
puts "Вихідне число дорівнює #{number}"
n1 = number/100                  # число сотень
n2 = (number/10)%10              # число десятків
n3 = number%10                   # число одиниць
answer = n1*100 + n3*10 + n2
puts "Результат #{answer}"
```

Іноді виникає необхідність вказати, що той або інший об'єкт у програмі є незмінним. Такі об'єкти називаються константами. В Ruby імена констант, також як й імена класів, повинні починатися із заголовної букви (є певна традиція задавати імена констант, що складаються з одних заголовних букв). Розглянемо наступний фрагмент програми:

```
NUM = 234
p NUM      # 234
NUM = "qq" # warning: already initialized constant NUM
p NUM      # "qq"
```

Як бачимо, інтерпретатор видав попередження про те, що константа NUM вже була визначена раніше. Проте значення константи було змінено.

Завдання № 1.1

1. **Напишіть програму, що друкує кількість квадратів зі стороною 130 мм, які можна відрізати від прямокутника зі сторонами 543x130 мм.**

2. **Напишіть програму, що по даному числу a, не використовуючи ніяких арифметичних операцій крім множення, одержує a^8 за три операції.**

1.6. МАСИВИ

Раніше ми вже говорили про "найпростіші" об'єкти Ruby - числа та рядки. До їхнього числа відносяться й **масиви** - екземпляри класу **`Array`**. Масив є просто набір (колекція) елементів, у якому доступ до кожного елемента можливий по його номері (цілому числу).

Масив може складатися з елементів, що належать різним класам. Зверніть увагу на те, що *нумерація елементів масиву починається з нуля.*

Щоб створити екземпляр класу **`Array`**, його елементи, які розділені комами, беруть у квадратні дужки, наприклад, `[1, 2, 3]`.

Для завдання *масиву рядків* можна використати більш зручну форму з використанням виразу **`%w`**: запис **`%w(раз два три)`** еквівалентний запису **`["раз", "два", "три"]`**. Між символом **`w`** і відкриваючою дужкою не повинне бути пробілу. Якщо кілька слів повинні стати одним елементом масиву, то для їхнього поділу перед пробілом додається символ `\` (backslash):

```
%w(раз\ два три\ чотири)  # ["рази два", "три чотири"]
```

Іноді потрібно тільки створити масив, але не заповнювати його елементами. У цьому випадку використовується одна з наступних конструкцій:

```
a = [ ]
b = Array.new
```

Якщо потрібно створити "порожній" масив заданого розміру, то він указується як аргумент методу `new`, наприклад,

```
myArray = Array.new(10)

# коли у функції тільки один аргумент,
# дужки можна опустити
myArray = Array.new 10
```

Необов'язковий другий аргумент методу **`new`** задає клас, екземпляри якого передбачається розміщувати в масиві:

```
timeArray = Array.new(3, Time)
p timeArray            #[Time, Time, Time]
```

Одним з безсумнівних достоїнств Ruby є наявність у ньому великої кількості методів для роботи з масивами. Нижче у таблиці перераховані ті, які найбільше використовуються та які ілюструють можливість роботи з масивом як із множиною, стеком й іншими структурами даних.

Призначення та приклад використання методу		Результат

[ ]
at
	Одержання елемента із зазначеним індексом; якщо аргумент є негативним числом, то індекс відраховується з кінця
	`a = ["a", "b", "c", "d"]; a[0]; a.at(0)`
	`a[-2]; a.at(-2)`
		`"a"`
		`"c"`

+	Додавання одного масива до іншого `[1, 2, 3] + [4, 5]`	`[1, 2, 3, 4, 5]`
*****	Повторення `[1, 2] * 2`	`[1, 2, 1, 2]`
|	Об'єднання множин `["a", "b", "c"] | ["c", "d", "a"]`	`["a", "b", "c", "d"]`
&	Перелік множин `[1, 1, 3, 5] & [1, 2, 3]`	`[1, 3]`
-	Різниця множин `[1, 2, 2, 3, 3, 3, 4, 5] - [1, 2, 4]`	`[3, 5]`

<<
push
	Додавання у кінець масива
	`[1, 2] << "c" << [3, 4]`
	`a = ["a", "b"]; a.push("c", "d")`
		`[1, 2, "c", [3, 4]]`
		`["a", "b", "c", "d"]`

unshift	Додавання елемента у початок масива зі здвігом залишившихся `a = ["b", "c"]; a.unshift("a")`	`["a", "b", "c"]`
clear	Видалення усіх елементів масива `a = ["a", "b", "c", "d"]; a.clear`	`[]`

collect	Виконання блоку операторів, які взяті в фігурні лапки, по разу для кожнього елементу масиву
	`a = ["a", "b", "c", "d"]`
	`a.collect {|i| i+"!"}`
		`["a!", "b!", "c!", "d!"]`

compact	Видалення всіх об'єктів `nil` з масиву `["a", "b", nil, "c", nil].compact`	`["a", "b", "c"]`
empty?	Перевірка на відсутність елементів масиву `[].empty?`	`true`
length	Визначення кількості елементів масиву `[ 1, 2, 3, 4, 5 ].length`	`5`

pop	Видалення останнього елементу масиву
	`a = [ "a", "m", "z" ]; a.pop`
	`a`
		`"z"`
		`["a", "m"]`

each	Виклик блока операторів по одному разу для кожнього масиву
	`a = ["a", "b", "c"]`
	`a.each {|x| print x, "--" }`
		`a--b--c--`

flatten	"Розгладжування" масиву — отримання одномірного масиву з масиву масивів
	`s = [ 1, 2, 3 ]`
	`t = [ 4, 5, 6, [7, 8] ]`
	`a = [ s, t, 9, 10 ]; a.flatten`
		`[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]`

index	Отримання номеру першого входження аргументу (`nil`, якщо не знайден)
	`a = [ "a", "b", "c" ]; a.index("b")`
	`a.index("z")`
		`1`
		`nil`

join	Створення єдиного рядка з масиву рядків	

	`[ "a", "b", "c" ].join` `[ "a", "b", "c" ].join("-")`	`"abc"` `"a-b-c"`
nitems	Визначення кількості елементів масиву, які відрізняються від `nil` `[ 1, nil, 3, nil, 5 ].nitems`	`3`

reverse	Інвертування масиву
	`[ "a", "b", "c" ].reverse`
	`[ 1 ].reverse`
		`["c", "b", "a"]`
		`[1]`

sort	Сортировка масиву
	`a = ["d", "a", "c"]`
	`a.sort`
	`a.sort {|x,y| y <=> x }`
		`["a", "c", "d"]`
		
		`["d", "c", "a"]`

Будьте уважні при сортуванні рядків, що містять букви російського алфавіту. Впорядкування рядків відбувається по ASCII-кодах їхніх символів, а таблиця кодування koi8-r, прийнята в ОС Linux, розміщує російські букви не за абеткою (див. "Практична інформатика", частина 1).

```
a2 = %w(собака вовк кінь альбатрос)
puts a2.sort   # альбатрос кінь собака вовк
```

Приклад

Створіть файл, у який помістить наступну програму.

```
a = [1, "cat", 3.14] # створення масиву із трьох елементів
puts a
puts a[0]          # друк першого елементу масиву
puts a.type
print  a[0].type, "\t", a[1].type, "\t", a[2].type, "\n"

a[2] = "dog"      # зміна значення третього елемента
puts a
print  a[0].type, "\t", a[1].type, "\t", a[2].type, "\n"
```

Простежте за змінами типу елементів масиву **a**.

Будьте уважні при присвоюванні значень змінним, якщо вони є посилання на масиви. Розглянемо наступний приклад, коментуючи дії зі змінними в термінах "наклейок":

```
# "упаковуємо" масив Ruby
magic = [1, 2]
# обидві мітки наклеєні на ту саму упаковку
hocusPocus = magic
# їх значення збігаються
puts magic
puts hocusPocus
```

```
puts "---змінюємо---"

# змінимо  вміст упаковки з ім'ям hocusPocus
hocusPocus[0] = 6
puts magic
# подивимося...
puts hocusPocus
# і помітимо, це обидві змінні змінилися
```

Так як масиви в мові Ruby поводяться майже всі об'єкти. Виключенням є числа (Fixnum), логічні величини **true** й **false**, а також спеціальна величина **nil** (нуль, нічого), яка використовується для посилання "у нікуди". Коли ви привласнюєте значення змінній, яка вказує на об'єкт одного з перерахованих вище типів, іншій змінній, ці дві змінні починають посилатися на *два різних об'єкти*, що мають однакове значення, а не на той самий об'єкт. Наприклад,

```
# "упакували" об'єкт типу FixNum
magic = 42
# тепер маємо дві упаковки, кожна зі своєю міткою
hocusPocus = magic
# обидві упаковки мають однаковий вміст
puts magic
puts hocusPocus

# змінимо вміст упаковки з ім'ям hocusPocus
hocusPocus = 0
# подивимося...
puts magic
# і помітимо, що вони різні
puts hocusPocus
```

Значення, що привласнюють змінним, можуть бути виразами або масивами. Якщо число елементів у правій частині більше числа змінних у лівій, то зайві значення ігноруються. Якщо число змінних ліворуч перевищує число значень праворуч, то змінні, які залишились, приймають значення **nil**. Символ ***** перед ім'ям *останньої* змінної в списку вказує, що вона є масивом, у якому містяться значення, що залишилися.

```
a, b = [1, 2]      # a = 1; b = 2
a, b = 1, 2        # a = 1; b = 2
a, b, c = 1, 2     # a = 1; b = 2; c = nil
a, b = 1, 2, 3     # a = 1; b = 2
a, *b = 1, 2, 3    # a = 1; b = [2, 3]
```

В Ruby, як й у більшості інших сучасних мов, немає багатомірних масивів, але вони легко моделюються створенням масиву масивів, тобто масиву, елементами якого є також

масиви. Як наслідок, такі масиви не зобов'язані бути прямокутними (тобто число елементів у різних рядках може відрізнятися), що дозволяє значно економити пам'ять. Наприклад, двовимірний масив розміром 3x3 можна задати так

```
a = [
   [11, 12, 13],
   [21, 22, 23],
   [31, 32, 33]
  ]
```

Аналогічно можна створити масив будь-якої необхідної розмірності, хоча масиви розмірності, більшої чим два, зустрічаються досить рідко. До елементів таких масивів можна добратися, послідовно вказуючи індекси необхідних елементів:

```
p a[1]        #  [21, 22, 23],
p a[1][1]     #  22
```

Якщо потрібно тільки створити багатомірний масив, не заповнюючи його даними, то варто використати наступну конструкцію:

```
# Порожній масив:
# створили три різних масиви й помістили їх у четвертий
a = Array.new(3)
a[0] = Array.new(3)
a[1] = Array.new(3)
a[2] = Array.new(3)
a[1][1] = 123
p a  # [[nil, nil, nil], [nil, 123, nil], [nil, nil, nil]]
```

Метод **map**, що виконує блок операторів, які йдуть за ним, для кожного елемента об'єкту, до якого він застосовується, дозволяє більш компактно реалізувати розглянуту вище конструкцію:

```
a = (0..2).map{ Array.new(3)}  # ПРАВИЛЬНО
a[1][1] = 123
p a  # [[nil, nil, nil], [nil, 123, nil], [nil, nil, nil]]
```

Спроба ж створити такий масив, аналогічно одномірному, приводить до несподіваного на перший погляд результату:

```
b = Array.new(3, Array.new(3)) # НЕПРАВИЛЬНО
# створили масив, у який помістили
# три посилання на той самий об'єкт
b[1][1] = 123
p b # [[nil, 123, nil], [nil, 123, nil], [nil, 123, nil]]
```

Приклад

Розглянемо створення масиву розміру 3x3, що повинен бути заповнений за наступною

схемою: у першому рядку масиву - поточний рік, номер місяця й дня місяця, у другому - назву місяця, день та найменування дня тижня, у третьому рядку - поточна година, хвилина та секунда.

```
myArray = (0..2).map{ Array.new 3 }

mouth = %w(січень лютий березень квітень травень червень
   липень серпень вересень жовтень листопад грудень)

dayOfWeek = %w(неділя понеділок вівторок середа
      четвер п'ятниця субота)
t = Time.now
myArray[0][0] = t.year
myArray[0][1] = t.month
myArray[0][2] = t.day
myArray[1][0] = mouth[t.month]
myArray[1][1] = t.day
myArray[1][2] = dayOfWeek[t.wday]
myArray[2][0] = t.hour
myArray[2][1] = t.min
myArray[2][2] = t.sec

print "Сьогодні #{myArray[0][2]}.",
  "#{myArray[0][1]}.#{myArray[0][0]}\n"
print "Число:  #{myArray[1][1]}, ",
  "місяць:  #{myArray[1][0]},",
  " день тижня: #{myArray[1][2]}\n"
print "Зараз  #{myArray[2][0]} година, ",
  "#{myArray[2][1]} хв,", " #{myArray[2][2]}сек\n"
```

Приклад

Найчастіше в задачах потрібно підрахувати деяку характеристику масиву, при цьому доводиться послідовно перебирати всі його елементи. У таких ситуаціях зручно використати вбудований метод **`each`**. Продемонструємо його використання на задачі знаходження суми всіх елементів масиву.

```
b = [23, -14, 45, 78, -2.5]
s = 0
b.each {|i| s +=i}
puts s
```

Завдання № 1.2

1. Напишіть програму, що друкує середнє арифметичне першого й останнього елементів масиву.

2. Напишіть програму, що знаходить перетинання й об'єднання двох множин A={1,2,3} й B={2,4,5}, використовуючи методи роботи з масивами.

1.7. ВВЕДЕННЯ ДАНИХ

Для введення даних із клавіатури можна використати метод **gets**, що поміщає в рядок всі символи, що вводяться. Введення завершується натисканням на клавішу **Enter**. Подібним же чином діє й метод **readline**.

Створіть файл із ім'ям **input.rb**, у який помістить наступний текст:

```
print "Введіть ваше ім'я: "
a=gets
print "Привіт, ", a
# ще раз
print "Введіть ваше ім'я: "
a=readline
print "Привіт, ", a
```

У першій частині нашого курсу ми вже згадували, що з погляду ОС Linux вхідний і вихідний потоки даних (за замовчуванням - введення з клавіатури й вивід у вікно **shell**) нічим не відрізняються від інших файлів - той же потік байтів. Тому введення комбінації клавіш `Ctrl+D`, що означає переривання введення даних, розцінюється аналогічно кінцю файлу. У нашому випадку дія методів **readline** й **gets** відрізняється лише реакцією на виявлення кінця файлу: у першому випадку видається помилка EOFError (End Of File Error), а в другому повертається об'єкт **nil**.

Зверніть увагу, що останній символ введеного рядка (незалежно від методу, що використовувався) є **\n**. Це є наслідком натискання на клавішу **Enter** (в інших операційних системах кінець рядка може кодуватися іншими символами).

```
print "Введіть рядок: "
a=gets
b=a.size
print "ASCII код останнього символу дорівнює ", a[b - 1], "\n"
```

Вище вже описувався метод `chop`, що дозволяє видалити останній символ рядка (або два останніх символи, якщо вони є `\r\n`).

```
print "Уведіть ваше ім'я: "
```

```
a=gets
print a, ", привіт!\n"
b=a.chop
print b, ", привіт!\n"
```

Як бачите, символ перекладу на новий рядок вилучений.

В Ruby багато методів мають своїх "двійників", що містять знак **!** після свого імені. Це так звані "небезпечні" методи, які руйнують об'єкт, до якого вони застосовуються. Так, для об'єктів типу **String** припустимий як метод **chop**, так й **chop!**. Перший з них, застосований до рядка, повертає новий об'єкт - рядок з вилученим останнім символом, а метод `chop!` змінює сам вихідний об'єкт.

```
a="123456"
b=a.chop
puts a, b
b=a.chop!
puts a, b
```

Змінимо програму, додавши до неї метод **chop!**:

```
print "Введіть ваше ім'я: "
a=gets.chop!
print "Привіт ", a, "! Як поживаєш?\n"
```

Тепер наше вітання друкується на одному рядку.

При введенні числової інформації варто примусово перетворити отриманий рядок у число за допомогою методів **to_i** та **to_f**. Нижче приводиться фрагмент програми, що запитує два числа й друкує результати їхньої обробки.

```
print "Введіть ціле число: "
num=gets.to_i
print "Остача від ділення на 5 дорівнює #{num%5}\n"
print "Введіть число (крапка відокремлює дробову частину): "
num=gets.to_f
print "При округленні вийшло число #{num.round}\n"
```

Раніше розглянутий метод **eval** дозволяє значно розширити можливості уведення даних.

Припустимо, що нам потрібно ввести не число,а арифметичний вираз (можливо, він містить операції *, /, +, -, ** або імена методів для роботи з числами). Тоді, після видалення символу переводу рядка, треба до отриманого рядка застосувати метод **eval**:

```
puts "Ruby не боїться величезних чисел."
puts "Введіть страшний арифметичний вираз, "
puts "наприклад, 14**256+3*17"
```

```
puts eval gets.chop!
```

За допомогою цього методу можна виконати команди, введені з клавіатури.

```
puts "Введіть команди, розділяючи їх крапкою з комою:"
eval gets.chop!
```

Нижче приводиться приклад виконання цієї програми:

```
Введіть команди, розділяючи їх крапкою з комою:
puts Time.now; print "2+3="; puts 2+3
Sat Mar 30 11:47:42 MSK 2002
2+3=5
```

ІНДИВІДУАЛЬНІ ЗАВДАННЯ ДО ЛАБОРАТОРНОЇ РОБОТИ № 1

Варіант	Завдання
1	Завдання 1. 1. Написати програми, що обчислюють середнє арифметичне чисел x, y й z; 2. Написати програму друкуючу третю від кінця цифру в записі позитивного цілого числа (напр., для 130986 відповідь 9). Завдання 2. Внести такі зміни в програми із завдання 1, щоб вихідні дані запитувалися з клавіатури. Завдання 3. Написати програму, що обчислює периметр і площу правильного 17-кутника, вписаного в окружність радіуса R. Що потрібно змінити, щоб програма обчислювала периметр й площю 25-кутника?
2	Завдання 1. 1. Написати програми, що обчислюють середнє арифметичне чисел x, y й z; 2. Написати програму друкуючу першу цифру в дробовому записі позитивного дробового числа (напр., для 32,567 відповідь 5). Завдання 2. Внести такі зміни в програми із завдання 1, щоб вихідні дані запитувались з клавіатури. Завдання 3. Написати програму, що обчислює периметр і площу правильного 25-кутника, вписаного в окружність радіуса R. Що потрібно змінити, щоб програма обчислювала периметр й площу 17-кутника?
3	Завдання 1. 1. Написати програми, що обчислюють середнє арифметичне чисел x, y й z. 2. Написати програму друкуючу суму цифр тризначного числа. Завдання 2. Внести такі зміни в програми із завдання 1, щоб вихідні дані запитувались з клавіатури. Завдання 3. Написати програму, що обчислює периметр і площу правильного 33-кутника, вписаного в окружність радіуса R. Що потрібно змінити, щоб програма обчислювала периметр й площу 22-кутника?
4	Завдання 1. 1. Написати програми, що обчислюють відстань між точками з координатами (x1, y1) і (x2, y2); 2. Написати програму друкуючу третю від кінця цифру в записі позитивного цілого числа (напр., для 130986 відповідь 9). Завдання 2. Внести такі зміни в програми із завдання 1, щоб вихідні дані запитувались з клавіатури. Завдання 3. Обчислити дробову частину середнього геометричного трьох введених позитивних чисел.
5	Завдання 1. 1. Написати програми, що обчислюють відстань між точками з координатами (x1, y1) і (x2, y2);

	2. Написати програму друкуючу першу цифру в дробовому записі позитивного дробового числа (напр., для 32,567 відповідь 5). Завдання 2. Внести такі зміни в програми із завдання 1, щоб вихідні дані запитувались з клавіатури. Завдання 3. Обчислити довжину окружності, площу кругу й об'єм кулі заданого радіуса.
6	Завдання 1. 1. Написати програми, що обчислюють відстань між точками з координатами (x1, y1) і (x2, y2); 2. Написати програму друкуючу суму цифр тризначного числа. Завдання 2. Внести такі зміни в програми із завдання 1, щоб вихідні дані запитувались з клавіатури. Завдання 3. Обчислити периметр і площу прямокутного трикутника за довжинами двох катетів.
7	Завдання 1. 1. Написати програми, що обчислюють відстань між точками з координатами (x1, y1) і (x2, y2); 2. Написати програму друкуючу третю від кінця цифру в записі позитивного цілого числа (напр., для 130986 відповідь 9). Завдання 2. Внести такі зміни в програми із завдання 1, щоб вихідні дані запитувались з клавіатури. Завдання 3. По координатах трьох вершин деякого трикутника знайти його площу та периметр.
8	Завдання 1. 1. Написати програми, що обчислюють відстань між точками з координатами (x1, y1) і (x2, y2); 2. Написати програму друкуючу першу цифру в дробовому записі позитивного дробового числа (напр., для 32,567 відповідь 5). Завдання 2. Внести такі зміни в програми із завдання 1, щоб вихідні дані запитувались з клавіатури. Завдання 3. По довжинах двох сторін деякого трикутника й куту між ними, заданому в градусах, знайти довжину третьої сторони й площу трикутника.
9	Завдання 1. 1. Написати програми, що обчислюють відстань між точками з координатами (x1, y1) і (x2, y2); 2. Написати програму друкуючу суму цифр тризначного числа. Завдання 2. Внести такі зміни в програми із завдання 1, щоб вихідні дані запитувались з клавіатури. Завдання 3. Знайти добуток цифр заданого чотиризначного числа.
10	Завдання 1. 1. Написати програми, що обчислюють середнє арифметичне чисел x, y й z; 2. Написати програму друкуючу третю від кінця цифру в записі позитивного цілого числа (напр., для 130986 відповідь 9). Завдання 2. Внести такі зміни в програми із завдання 1, щоб вихідні дані запитувались з клавіатури. Завдання 3. Написати програму, що обчислює периметр і площу правильного

	17-кутника, вписаного в окружність радіусу R. Що потрібно змінити, щоб програма обчислювала периметр й площу 25-кутника?
11	Завдання 1. 1. Написати програми, що обчислюють середнє арифметичне десяти чисел - x1, x2, ..., x10; 2. Написати програму друкуючу другу цифру в дробовому записі позитивного дробового числа (напр., для 32,567 відповідь 6). Завдання 2. Внести такі зміни в програми із завдання 1, щоб вихідні дані запитувались з клавіатури. Завдання 3. Написати програму, що обчислює периметр і площу правильного 36-кутника, вписаного в окружність радіуса R. Що потрібно змінити, щоб програма обчислювала периметр й площу 25-кутника?
12	Завдання 1. 1. Написати програми, що обчислюють середнє арифметичне восьми чисел - y1, y2, ..., y8. 2. Написати програму друкуючу суму цифр чотиризначного числа. Завдання 2. Внести такі зміни в програми із завдання 1, щоб вихідні дані запитувались з клавіатури. Завдання 3. Написати програму, що обчислює периметр і площу правильного 28-кутника, вписаного в окружність радіуса R. Що потрібно змінити, щоб програма обчислювала периметр й площу 19-кутника?
13	Завдання 1. 1. Написати програми, що обчислюють відстань між точками з координатами (a1, b1) і (a2, b2); 2. Написати програму друкуючу другу від кінця цифру в записі позитивного цілого числа (напр., для 130986 відповідь 8). Завдання 2. Внести такі зміни в програми із завдання 1, щоб вихідні дані запитувалисm з клавіатури. Завдання 3. Обчислити дробову частину середнього геометричного п'яти введених позитивних чисел.
14	Завдання 1. 1. Написати програми, що обчислюють відстань між точками з координатами (x1, y1) і (x2, y2); 2. Написати програму друкуючу другу цифру в дробовому записі позитивного дробового числа (напр., для 32,567 відповідь 6). Завдання 2. Внести такі зміни в програми із завдання 1, щоб вихідні дані запитувались з клавіатури. Завдання 3. Обчислити довжину периметра й площу прямокутника та об'єм паралелепіпеда по заданних розмірах - a, b, c.
15	Завдання 1. 1. Написати програми, що обчислюють відстань між точками з координатами (c1, d1) і (c2, d2); 2. Написати програму друкуючу суму цифр п'ятизначного числа. Завдання 2. Внести такі зміни в програми із завдання 1, щоб вихідні дані запитувались з клавіатури. Завдання 3. Обчислити висоту й об'єм правильного тетраедра.

16	Завдання 1. 1. Написати програми, що обчислюють відстань між точками з координатами (a, b) і (c, d); 2. Написати програму друкуючу четверту від кінця цифру в записі позитивного цілого числа (напр., для 130986 відповідь 0). Завдання 2. Внести такі зміни в програми із завдання 1, щоб вихідні дані запитувались з клавіатури. Завдання 3. По координатах трьох вершин деякого трикутника (x1,y1), (x2,y2), (x3,y3) знайти його площу й периметр.
17	Завдання 1. 1. Написати програми, що обчислюють відстань між точками з координатами (d1, p1) і (d2, p2); 2. Написати програму друкуючу першу цифру в дробовому записі позитивного дробового числа (напр., для 32,567 відповідь 5). Завдання 2. Внести такі зміни в програми із завдання 1, щоб вихідні дані запитувались з клавіатури. Завдання 3. По довжинах двох сторін деякого трикутника a,b та куту між ними - g, заданому в градусах, знайти довжину третьої сторони - c та площу трикутника - s.
18	Завдання 1. 1. Написати програми, що обчислюють відстань між точками з координатами (a1, c1) і (a2, c2); 2. Написати програму друкуючу суму цифр тризначного числа. Завдання 2. Внести такі зміни в програми із завдання 1, щоб вихідні дані запитувались з клавіатури. Завдання 3. Знайти добуток цифр заданого десятизначного числа.

ЛАБОРАТОРНА РОБОТА 2. ОБ'ЄКТНО-ОРІЄНТОВАНА МОВА RUBY. ОСНОВНІ ВЛАСТИВОСТІ МОВИ (ПРОДОВЖЕННЯ).

2.1. МЕТОДИ

Методи, поряд зі змінними, є основними будівельними блоками в Ruby. Ми вже застосовували методи, визначені для таких класів, як числа й рядки.

При написанні програм часто доводиться створювати додаткові методи (які звуться також функціями або підпрограмами). Кожен метод повинен бути визначений до свого використання за допомогою ключових слів **def** та **end**. Як ми вже відзначали, ім'я методу повинне починатися з рядкової латинської букви (від a до z). Також як і змінним, методам рекомендується давати осмислені імена. Якщо ім'я складається з декількох слів, то для їхнього поділу рекомендується використати символ підкреслення, або кожне слово, починаючи із другого, виділяти заголовною буквою.

Методам, обумовленим поза визначенням класу, привласнюється спеціальна позначка **private** (приватний), яка зветься специфікатором доступу. При звертанні до такого методу використається **функціональна** форма виклику методу, а не оператор виклику методу **.** (крапка). З іншої сторони методи, визначені усередині визначення класу, за замовчуванням позначаються як **public**. При виклику методів, визначених таким чином, використається "крапкова" форма виклику методу. Через те, що ми поки не створюємо нових класів у своїх програмах, то всі методи, розроблені нами, будуть викликатися тільки у функціональному стилі.

Давайте перепишемо нашу першу програму так, щоб вона використала виклик методу. Для цього у файл із ім'ям **method.rb** помістимо наступний текст:

```
def helloWorld
  puts "Hello, World!"
end

helloWorld
```

Частина програми між словами **def** й **end** (у нашому випадку єдиний оператор **puts**) становить тіло методу.

Змінимо метод так, щоб він міг друкувати будь-який текст:

```
def saySomething(text)
  puts text
end

saySomething("Hello, World!")
```

Ми додали **параметр** **text** у наш метод. При визначенні методу параметри завжди

беруть у круглі дужки, які йдуть відразу за ім'ям методу, та розділяються комами.

Параметри методу є локальними змінними. Це означає (у термінах наших наклейок й об'єктів), що метод оперує **копіями** змінних, переданих йому (часто говорять, що параметри передаються за значенням). Інакше кажучи, перед тим як передати мітки й створити метод, Ruby створює копії всіх цих об'єктів, тимчасово, тільки на час виконання методу, поміщає імена параметрів на них. Проілюструємо це наступним прикладом:

```
def printOneMoreThan(x)
  x += 1
  puts "Під час виконання: #{x}"
end

x = 1
puts "До: #{x}"
printOneMoreThan(x)
puts "Після: #{x}"
```

Виконаємо цю програму й переконаємося, що значення змінної **`x`** не змінилося після завершення методу **`printOneMoreThan`**.

При створенні методу Ruby дозволяє задати значення деяких аргументів. Ці, так звані **параметри за замовчуванням**, дають можливість виклику без обов'язкової вказівки значення всіх параметрів. Нижче приводиться приклад такого методу.

```
def saySomething(text = 'Hello, World!')
  puts text
end

saySomething
```

При цьому виклику методу параметр не вказувався. Ruby дозволяє викликати метод декількома різними способами:

```
saySomething
saySomething()
saySomething "Пробіл між ім'ям методу й аргументами"
saySomething(" або використання дужок")
```

Рекомендується завжди розміщати аргументи методу в круглих дужках. Це дозволить уникнути деяких помилок і непорозумінь. Загальноприйнятими виключеннями є методи друку - `puts`, `print` і т.д., хоча й при їхньому виклику аргументи також можна брати в дужки:

```
puts(2 + 3)
print("Hello, ", 3 + 4, "\n")
```

Значеннями за замовчуванням можуть бути будь-які вирази Ruby. Вони обчислюються в момент виклику методу, причому обчислення відбувається зліва направо. У цих виразах можуть використовуватись й інші параметри. Наступний приклад демонструє поводження методу залежно від кількості аргументів при виклику:

```
def options(a=99, b=a+1)
 [a, b]
```

```
end

p options          # [99, 100]
p options(1)       # [1, 2]
p options(2, 4)    # [2, 4]
```

Після всіх звичайних аргументів як аргумент може бути зазначений масив, що позначається символом `*` перед його ім'ям. При виклику в такий спосіб певного методу, Ruby підраховує кількість переданих аргументів й, якщо їхня кількість більше, ніж число звичайних параметрів, то ті, що залишилися, розміщає в новому об'єкті класу Array. Наприклад,

```
def varargs(a, *b)
 [a, b]
end

p varargs(1)        #  [1, []]
p varargs(1, 2)     #  [1, [2]]
p varargs(1, 2, 3) #  [1, [2, 3]]
```

Якщо аргумент-масив йде за аргументами, що мають значення за замовчуванням, то при недостачі аргументів використаються значення за замовчуванням, у противному випадку розміщення параметрів аналогічно попередньому. Проілюструємо сказане прикладом:

```
def mixed(a, b=99, *c)
 [a, b, c]
end

p mixed(1)            # [1, 99, []]
p mixed(1, 2)         # [1, 2, []]
p mixed(1, 2, 3)      # [1, 2, [3]]
p mixed(1, 2, 3, 4)  # [1, 2, [3, 4]]
```

Дуже часто від методу потрібне повернення значення, отриманого в результаті його виконання. Для цього використається конструкція **`return вираження`** (якщо ви опустите `вираження`, залишивши тільки `return`, то повертається значення `nil`). Якщо не використати оператор `return`, то значення, що повертає метод, буде значенням останнього виразу, обчисленого в методі.

Зупинимося трохи докладніше на останньому твердженні. Справа в тому, що в Ruby, у відмінності від багатьох інших мов програмування, *будь-який вираз має значення*. Приведемо фрагмент програми для ілюстрації цього факту:

```
def saySomething(text = 'Hello, World!')
 puts text
end

x = saySomething
puts x
puts x = 23
```

```
puts x += 1
```

Зверніть увагу, що значення, яке повертає метод `saySomething`, є `nil`. Це означає, що значення останнього твердження в методі (тобто оператора `puts text`) було `nil`. Значення, що повертає оператор `x = 23` є число 23, а оператор `x += 1` дорівнює 24. Тепер спробуйте здогадатися, який буде результат виконання програми:

```
def countUp(x = 0)
 puts "У цей момент X дорівнює #{x}"
 x += 1
end

x = countUp()
x = countUp(x)
x = countUp(x)
```

Завдання №2.1

1. Напишіть функцію `dayOfWeek`, що друкує назву поточного дня тижня.
2. Напишіть функцію, що визначає кількість слів у введеній фразі.

2.2. УМОВНІ ОПЕРАТОРИ

Умовний оператор, який також зветься оператором вибору, дозволяє залежно від *істинності* або *хибності* деякої умови виконувати ту або іншу послідовність команд.

Ruby підтримує всі конструкції вибору, традиційно використовувані в інших мовах програмування. Але перш ніж говорити про їх, зупинимося на деяких особливостях Ruby при обчисленні **істинності** окремих виразів.

Варто запам'ятати, що тільки `false` й `nil` трактуються як помилкові в логічному контексті. *Все інше, що обчислюється, є істинним.* Для того щоб краще зрозуміти зміст даного висловлення, виконаєте наступну програму:

```
if 23
  puts "23 є ІСТИНА"
end


if "qq"
  puts "И будь-який рядок є ІСТИНА"
end
```

Ruby підтримує всі стандартні логічні оператори, а також додатковий оператор `defined?`. Для кон'юнкції двох логічних виразів використовують `and` й `&&`. Їхній результат буде істинний тільки якщо істинні *обидва* операнда. Ці функції обчислюють другий операнд лише тоді, коли перший істинний, а відрізняються вони тільки своїм пріоритетом (пріоритет оператора `and` менше `&&`).

В свою чергу результатом виконання операторів `or` й `||` (диз'юнкції) буде істина, якщо *хоча б один* з операндів істинний. Вони обчислюють другий оператор тільки в тому випадку, якщо перший виявиться помилковим. Як й у випадку з оператором `and`, оператор `or` має менший пріоритет у порівнянні з `||` (відзначимо, що оператор `and` має пріоритет, який дорівнює пріоритету оператора `or`, у той час як пріоритет `&&` вище, ніж в `||`).

Оператори `not` й `!` повертають значення, протилежне своєму аргументу (якщо аргумент істинний, то результат помилковий, і навпаки). Між собою відрізняються тільки пріоритетом.

Оператор `defined?` повертає `nil`, якщо його аргумент не визначений, і його опис у противному випадку. Нижче наведені приклади застосування цього оператора.

```
puts defined? 1
puts defined? dummy
puts defined? printf
puts defined? (c, d = 1, 2)
puts defined? 42.abs
```

У логічних виразах допускаються також наступні операції порівняння: `==` `<`, `<=`, `>=`, `>` `!=`.

Синтаксис умовного оператора `if` у мові Ruby аналогічний синтаксису в більшості інших мов програмування. Наприклад,

```
# Визначення методу оцінки величини виразу
def howBig(i)
  if i < 10
    puts "менше 10"
  elsif i < 20
    puts "між 10 й 20"
  elsif i < 30
    puts "між 20 й 30"
  else
    puts "більше або дорівнює 30"
  end
end
# Використання методу ...
howBig(7);  howBig(15)
howBig(23); howBig(105)
```

Загальна форма оператора `if` така:

```
if <логічний_вираз> [then]
    тіло_оператора
elsif <логічний_вираз> [then]
    тіло_оператора
    ...
```

```
else
  тіло_оператора
end
```

Тут <логічний_вираз> може бути будь-яким фрагментом коду мовою Ruby, результатом обчислення якого є логічна величина (з урахуванням сказаного вище). Слово `then` відокремлює тіло оператора від умови. Запис його у квадратних дужках означає, що воно може бути опущено, якщо тіло починається з нового рядка. Значення, що повертає оператор `if`, є результат останнього обчисленого виразу. Змінимо наш приклад, щоб продемонструвати сказане:

```
# Наш метод оцінки величини виразу
# тепер повертає рядок
def howBig(i)
  if i < 10 then "менше 10"
  elsif i < 20 then "між 10 й 20"
  elsif i < 30 then "між 20 й 30"
  else "більше або дорівнює 30"
  end
end
# Використання методу ...
puts howBig(7); puts howBig(15)
puts howBig(23); puts howBig(105)
```

Для виконання одного варіанта із двох можливих використається наступна форма оператора `if`:

```
if <логічний_вираз> [then]
    тіло_оператора
else
    тіло_оператора
end
```

Наприклад,

```
if i < 100
  puts "Мало"
else
  puts "Значно більше"
end
```

Оператор `if` може використовуватися в правій частині оператора присвоювання, наприклад,

```
str = if i < 100 then "Мало" else "Побільше" end
puts str
```

Для аналогічних цілей використається й **тернарний** оператор

```
логічний_вираз ? вираз1 : вираз2
```

Якщо `логічний_вираз` істинний, то виконується `вираз1`, інакше - `вираз2`. Подивіться на приклади використання цього оператора:

```
i = 23
i < 100 ? puts("Мало") : puts("Значно більше")
i = 1234
s = i < 100 ? "Мало" : "Значно більше"
puts s
```

Як бачите, цей оператор особливо зручний, якщо потрібно змінної привласнити одне із двох значень.

Приклад

Напишіть програму, що визначає парність уведеного числа.

```
print "Уведіть ціле число: "
a = gets.to_i
str = a%2 == 0 ? "четно" : "нечетно"
puts "Число #{a} " + str
```

Якщо потрібно виконати послідовність операторів тільки в тому випадку, коли виконане деяка умова, то використається форма

```
if <логічний_вираз> [then]
    тіло_оператора
end
```

Для цих же цілей можна використати й **модифікований** оператор `if`:

```
вираз if <логічний_вираз>
```

Зрівняєте наступні дві форми оператора `if`:

```
if radiation > 3000
  puts "Радіаційна небезпека!"
end
# те ж саме
puts "Радіаційна небезпека!" if radiation > 3000
```

Приклад

Напишіть програму, що визначає, чи є введена фраза паліндромом (перевертишем).

```
puts "Введіть фразу:"
a = gets.chop!.delete(' ')      # видалили всі пробіли
a = a.tr('A-Z,А^-Я', 'a-z,а^-я') # заміна букв на прописні
str = "паліндром"
```

```
str = "не " + str if a != a.reverse # перевірка
puts "Введена фраза " + str + "."
```

При виборі з більшого числа альтернатив зручніше використати оператор `case`. Нижче наведений його загальний вид (частини, взяті у квадратні дужки, можуть бути опущені).

```
case <вираз>
when  <тест1> [then]
    ...
when  <тест2> [then]
    ...
when  <тест> [then]
    ...
[else
    ... ]
end
```

В операторі `case <вираз>` послідовно порівнюється з виразом `<тест>` доти поки, він не збіжиться з одним з них (порівняння виконується за допомогою операції `==`), після чого виконується відповідний фрагмент коду. Оператор `case` повертає значення останнього обчисленого виразу, або `nil`, якщо такого не було.

```
s = case i
    when   0 .. 9
       "однозначне "
    when  10 .. 99
       "двозначне "
    when 100 .. 999
       "тризначне "
    else
       "величезне "
    end
puts s
```

Тут ми знову скористалися оператором `..` (двокрапка), що повертає список цілих чисел, включених між лівим і правим операндами (включаючи їх самих).

Завдання № 2.2.

1. Напишіть програму, що друкує максимальне й мінімальне з трьох введених чисел.
2. Напишіть програму, що друкує назву місяця за порядковим номером.

2.3. ОПЕРАТОРИ ЦИКЛУ

Для завдання повторюваних дій у більшості мов програмування використаються оператори циклу. В Ruby є два таких оператори - **`while`** й **`until`**, а також велика кількість ітераторів. Оператор `while` виконує оператори, які включені у його тіло, нуль або більше раз, доти, поки **істинна** його умова, що задається деяким логічним виразом. Його загальний вид такий:

```
while <вираз> [do]
    ...
  тіло циклу
    ...
end
```

Іншим оператором циклу є `until`, що виконується доти, поки його умова **неправдива**:

```
until <вираз> [do]
  ...
end
```

Приклад

Розглянемо програму, що друкує числа від 1 до 5. Спочатку використаємо оператор `while`, потім `until`. Зверніть увагу, що умова закінчення одного оператора циклу є запереченням умови іншого оператора.

```
i=1
while i <= 5
 puts i; i += 1
end

# ще раз
i=1
until i > 5
 puts i; i += 1
end
```

Крім цих двох операторів циклу в Ruby є велике число, так званих, **ітераторів** (iterate - повторювати). Давайте подивимося на приклади їхнього використання. Конструкція

```
3.times do
 print "Агов! "
end
```

використовує ітератор **`times`**. Цикл, заданий таким чином, виконається *рівно три рази.*

У випадку, коли потрібно виконати деякі дії, що залежать від змінюваної величини, кілька разів, можна використати ітератор **`upto`**, так у процесі виконання програми

```
0.upto(9) do |x| print x, " " end
```

буде надруковано `0 1 2 3 4 5 6 7 8 9 .`

Повтор від 0 до 12 із кроком 3 можна записати за допомогою ітератора **`step`**:

```
0.step(12, 3) {|x| print x, " " } # 0 3 6 9 12
```

При роботі з масивами зручно використати ітератор **each**:

```
[1, 1, 2, 3].each {|k| print k, " " } # 1 1 2 3
```

Ітератор **for in** дуже схожий на each, наприклад, вивід, отриманий у результаті виконання наступних двох конструкцій однаковий.

```
for i in  ["one", "two", "three"]
 print i, " "
end

# те ж саме
["one", "two", "three"].each{ |i| print i, " "}
```

Итератор for звичайно використають там же, де й ітератор each - при роботі з масивами й діапазонами. Загальний вид оператора for такий:

```
for <змінна> in <вираз> [do]
  тіло_ітератора
end
```

Приклад

Перепишемо програму друку чисел від 1 до 5 з використанням оператора for:

```
for i in 1 .. 5
 puts i
end
```

У цьому прикладі ми знову скористалися оператором завдання діапазону .. (двокрапка), що дозволив нам створити список чисел від 1 до 5 (включно). Схожий оператор ... (три крапки) при створенні діапазону не включає в нього правий операнд. Фрагмент програми, розташований нижче, еквівалентний попередній.

```
for i in 1...6
 puts i
end
```

Приклад

Розглянемо кілька варіантів програми, що обчислює факторіал введеного числа.

1. Використання оператора while

```
2.    print "Введіть ціле позитивне число: "
3.    str = gets.chop! # ввели рядок
4.    num = str.to_i   # перетворили в число
5.    if num > 0
6.     i = 1
7.     fact = 1
```

```
8.     while i <= num
9.      fact *= i
10.  i += 1
11. end
12. puts "Факторіал числа #{num} дорівнює #{fact}"
13.else
14. puts "Ви ввели непозитивне число"
15.end
```

2. Використання оператора `for`

```
17.print "Введіть ціле позитивне число: "
18.num = gets.to_i # рядок відразу перетворили в число
19.if num > 0
20. fact = 1
21. for i in 1 .. num
22.  fact *= i
23. end
24. puts "Факторіал числа #{num} дорівнює #{fact}"
25.else
26. puts "Ви ввели непозитивне число"
27.end
```

3. Використання функції для визначення факторіала

```
29.def fact(n)
30. f = 1
31. 1.step(n,1) {|k| f *= k}
32. return f
33.end
34.print "Уведіть ціле позитивне число: "
35.if (num = gets.to_i) > 0
36. print "#{num}! = #{fact(num)}\n"
37.else
38. puts "Факторіал числа #{num} не визначений\n"
39.end
```

В останньому прикладі ми описали обчислення факторіала у вигляді функції й використали ітератор `step`, що забезпечив зміну змінної `k` із кроком 1.

Приклад

Наведена нижче програма запитує ціле позитивне число й визначає кількість цифр у ньому (зверніть увагу на множинне присвоювання).

```
print "Введіть ціле позитивне число: "
a, k = gets.to_i, 0
while a>0
 a /= 10; k += 1 # відкинули останню цифру
end
print "Кількість цифр у введеному числі дорівнює #{k}.\n"
```

Приклад

Нехай потрібно ввести із клавіатури масив чисел і надрукувати суму всіх елементів. Приведемо кілька рішень цієї задачі.

1. Спочатку введемо кількість елементів масиву, а потім заповнимо його елементами, один за одним. Зверніть увагу, що перший елемент масиву має індекс 0, а останній - на одиницю менший, ніж розмірність масиву.

```
print "Введіть число елементів масиву: "
sn = gets.chop!; n = sn.to_i
b = Array.new(n)    # створили масив з n елементів
s = 0               # обнулили суму
for i in 0 .. n - 1
 print "Введіть #{i+1}-й елемент масиву: "
 b[i] = gets.chop!.to_f; s = s + b[i]
end
print "Сума всіх елементів масиву дорівнює ", s, "\n"
```

2. У цій версії програми всі числа вводяться відразу у вигляді одного рядка, у якій числа відокремлюється друг від друга пробілом (наприклад, 23 -34.67 100.5). Вбудований метод `split` розділяє рядок на елементи масиву, аргументом цього методу є роздільник (якщо роздільником є пробіл, то можна викликати цей метод без аргументу). Таким чином, якби ми домовилися розділяти числа, наприклад, крапкою з комою, то виклик методу виглядав би так: `a.split(';')`. Для визначення довжини масиву ми застосували метод `length` (можна замінити на `size`).

```
puts "Введіть масив чисел (розділяючи їх пробілами):"
a = gets.chop!
b = a.split # розбили рядок на окремі числа
s = 0
for i in 0 .. b.length - 1
 s += b[i].to_f
end
puts "Сума всіх елементів масиву дорівнює #{s}"
```

В Ruby є чотири конструкції, що задаються ключовими словами **`break`**, **`redo`**, **`next`** й **`retry`**, які змінюють звичайний порядок виконання циклів. Їхня дія описана в наступній таблиці.

break	Негайно припиняє виконання циклу; керування передається на твердження, розташоване відразу за циклом
redo	Повторює тіло циклу з початку, не перераховуючи умову виконання циклу (не переходячи до наступного елемента у випадку ітератора)
next	Пропускає частину тіла циклу, яка йде за ним, та переходить до наступної ітерації
retry	Починає виконання циклу із самого початку

Розглянемо на прикладі ітератора `for` дію зазначених конструкцій.

Код	Результат

`for i in 1 .. 5`
`  print  i`
`  break if i == 3`
`  print "*"`
`end`
	Результат:
	`1*2*3`

`for i in 1 .. 5`
`  print  i`
`  redo if i == 3`
`  print "*"`
`end`
	Результат:
	`1*2*33333 ...`
	виконання циклу не припиняється

`for i in 1 .. 5`
`  print  i`
`  next if i == 3`
`  print "*"`
`end`
	Результат:
	`1*2*34*5*`

`for i in 1 .. 5`
`  print  i`
`  retry if i == 3`
`  print "*"`
`end`
	Результат:
	`1*2*31*2*31*2*...`
	виконання циклу не припиняється

Приклад

Наступна програма починає повторення циклу спочатку, якщо при введенні вказати символ `y`.

```
for i in 1 .. 5
 print "Now at #{i}. Restart?(y/n) "
 retry if gets.chop == "y"
end
```

От один з можливих варіантів виконання цієї програми:

```
Now at 1. Restart?(y/n) n
Now at 2. Restart?(y/n) y
Now at 1. Restart?(y/n) n
    ...
```

Завдання № 2.3.

1. **Напишіть програму, що обчислює суму всіх парних натуральних чисел, що не перевершують 1000.**
2. **Напишіть програму, що визначає максимальний елемент масиву чисел.**

ЛАБОРАТОРНА РОБОТА 3. КОМПЛЕКСНЕ ПРОГРАМУВАННЯ МОВОЮ RUBY.

3.1. БІБЛІОТЕКИ

Мова Ruby поставляється з великою й корисною бібліотекою модулів і класів. З поняттям класу ми вже трохи познайомилися, а модулі — це такий спосіб групування разом методів, класів і констант, на якому ми зараз не будемо зупинятися. У цьому розділі нас буде цікавити застосування бібліотечних модулів і класів.

Для того щоб скористатися яким-небудь методом одного з модулів, варто вказати ім'я модуля (воно так само, як й ім'я класу починається із заголовної функції), а потім через крапку ім'я методу, наприклад, `Math.sqrt(2)`. Для звертання до константи варто відокремити її ім'я від імені модуля або класу двокрапкою. У випадку, коли доводиться використати методи одного модуля кілька разів, зручніше підключити потрібний модуль за допомогою оператора `include`, що дозволить використати виклик методу без вказівки імені модуля:

```
puts Math.sqrt(2); puts Math::PI
# або
include Math
puts sqrt(2); puts PI
```

Почнемо з опису модуля **`Math`**, що містить методи для роботи з математичними функціями й дві константи, що задають числа пі - `PI` та е - `E`.

Метод	Призначення методу
`cos(x)`, `sin(x)` і `tan(x)`	Повертають косинус, синус і тангенс числа x, заданого в радіанах
`atan2(y, x)`	Повертає арктангенс дробу y/x, де x й y задані в радіанах й укладені в діапазоні від -PI до PI
`exp(x)`	Повертає число е, зведене в ступінь x
`log(x)`	Повертає натуральний логарифм числа x>0
`log10(x)`	Повертає логарифм числа x>0 по підставі 10
`sqrt(x)`	Повертає квадратний корінь від'ємного числа x

Іншим корисним модулем є **`Enumerable`**, що дозволяє викликати методи, пов'язані з перерахуванням, пошуком і сортуванням інформації. Продемонструємо роботу з ним на прикладах. Для виводу будемо використовувати оператор `p`.

```
include Enumerable

# одержати квадрати цілих чисел від 1 до 4
```

```
a1 = (1..4).collect {|i| i*i }
# теж саме
a2 = (1..4).map {|i| i*i }
p a1, a2
# видати чотири рази рядок  "cat"
b = (1..4).collect { "cat"  }
p b
# знайти перше входження числа,
# яке ділиться й на 5 і на 7
# пошук серед чисел від 1 до 10   - таких немає
c = (1..10).detect {|i| i % 5 == 0 and i % 7 == 0 }
p c
# пошук серед чисел від 1 до 100  - першим знайдене число 35
c1 = (1..100).detect {|i| i % 5 == 0 and i % 7 == 0 }
# те ж саме
c2 = (1..100).find  {|i| i % 5 == 0 and i % 7 == 0 }
p c1, з2
# знайти всі входження чисел, які діляться й на 5 і на 7
c = (1..100).find_all  {|i| i % 5 == 0 and i % 7 == 0 }
p c
# знайти всі цілі числа, кратні 3 і не перевищуючі 10
d1 = (1..10).find_all {|i|  i % 3 == 0 }
# те ж саме
d2 = (1..10).select {|i|  i % 3 == 0 }
# й, навпаки, видалити  всі цілі числа, кратні 3
d3 = (1..10).reject {|i|  i % 3 == 0 }
p d1, d2, d3
# перетворити об'єкт перерахувального типу у масив
l = 1..10; p l.to_a   # [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

Цей модуль містить також методи `max` - знаходження максимального значення, `min` - мінімального й метод `sort`, що виконує сортування. Їх можна застосовувати до всіх об'єктів, які можна порівнювати між собою - числам, рядкам і т.п.

```
puts "Максимум чисел #{1..10} дорівнює #{(1..10).max}"
# сортування за спаданням
p (1 .. 8).sort { |i,j| j<=>i} #  [8, 7, 6, 5, 4, 3, 2, 1]
# мінімальний елемент масиву
a1 = %w(albatross dog horse)
p a1.min          # albatross
```

А от так можна визначити самий довгий рядок у масиві:

```
puts a1.max {|a,b| a.length <=> b.length } # albatross
```

Приклад

Збережіть наведену нижче програму друку тригонометричних функцій у файлі ex1.rb і виконаєте її. При введенні даних можна використати константу `PI` і математичні операції,

наприклад, `-3*PI/4`.

```
include Math

print "Введіть кут x у радіанах: "
y = gets.chop!
print "x=", y
x = eval y

print "\nбезформатний висновок:\tcos(x)= #{cos(x)}\n"
printf "\nдрук із 4-мя знаками після",
  " коми (як у таблицях Брадиса)\n ",
  "cos(x)=%1.4f\n", cos(x)

printf "\nx=%s: cos(x)= %1.4f\tsin(x)=%1.4f\t",
 y, cos(x), sin(x)

if cos(x).abs < 0.0001
 print "tg не існує\t"
else
 printf "tg(x)=%1.4f\t", tan(x)
end

if sin(x).abs < 0.0001
 print  "ctg не існує\n\n"
else
 printf "ctg(x)=%1.4f\n\n", cos(x)/sin(x)
end
```

Приклад

Наступна програма друкує значення тригонометричних функцій для кутів від -180 градусів до 180 із кроком 15 градусів.

```
include Enumerable, Math
# склали список чисел від -PI до PI із кроком PI/12
a = (-12 .. 12).map {|i| i.to_s + "*PI/12"}
# друк заголовка
print "Кут\t   Синус\t  Косинус\t  Тангенс\tкотангенс\n"
for i in a.reverse  # для друку в порядку спадання
  x = eval i
  printf "%4s\t%9.4f\t%9.4f\t",
    eval(i + "*(15*12/PI)").round, # перевели в градуси
    sin(x), cos(x)
```

```
 if cos(x).abs < 0.0001 then print "  не сущ.\t"
 else  printf "%9.4f\t", tan(x)
 end

 if sin(x).abs < 0.0001 then print  "  не сущ.\n"
 else printf "%9.4f\n", cos(x)/sin(x)
 end
end
```

3.2. ПРИКЛАДИ ПРОГРАМ

3.2.1. НЗД і алгоритм Евкліда

При розгляді задач, пов'язаних з обробкою цілих чисел, часто доводиться зіштовхуватися з поняттям НЗД - найбільшого загального дільника чисел. Відомо багато алгоритмів обчислення НЗД, ми розглянемо лише два з них.

Задача 1

Напишіть програму, що обчислює `НЗД(a,b)` - найбільший загальний дільник двох уведених із клавіатури ненегативних цілих чисел `a` й `b`, не рівних нулю одночасно.

Варіант 1

Визначимо максимум із цих чисел й, послідовно зменшуючи його, будемо шукати число, на яке діляться й `a` й `b`. Програма обов'язково завершить свою роботу, тому що в самому невдалому випадку, коли ці числа взаємно прості, `k` стане дорівнює 1, а на 1 діляться всі числа, отже виконання циклу припиниться.

```
print "Введіть перше число: "; a = gets.to_i
print "Введіть друге число: "; b = gets.to_i

k = a >= b ? a : b # тепер k -  максимум
until (a % k == 0) and (b % k == 0)
 k -= 1
end
print "НЗД(#{a},#{b}) = #{k}\n"
```

Варіант 2 - алгоритм Евкліда

Алгоритм Евкліда заснований на наступних властивостях НЗД: для всіх `a` й `b`, більші або рівних 0, виконуються рівності

```
НЗД(a, b) = НЗД(a - b, b)= НЗД(a, b - a);
НЗД(a, 0) = НЗД(0, a) = a
```

```
print "Введіть перше число: "; a = gets.to_i
print "Введіть друге число: "; b = gets.to_i
m, n = a, b
while  !((m == 0) || (n == 0))
  if m >= n
    m = m - n
  else
    n = n - m
  end
end

k = m == 0 ? n : m
print "НОД(#{a},#{b}) = #{k}\n"
```

3.2.2. Робота з файлами

Введення даних із клавіатури в процесі виконання програми зручне тільки у випадку невеликих об'ємів. В інших ситуаціях необхідне читання інформації з заздалегідь підготовленого файлу.

Для роботи з файлом за допомогою методу `new` створюється екземпляр класу `File`. Обов'язковим аргументом цього методу є рядок, що містить ім'я файлу, наприклад,

```
f = File.new("myfile.txt")
```

Іншим (необов'язковим) аргументом є завдання режиму роботи з файлом. За замовчуванням цей параметр має значення `"r"`, що відповідає режиму "тільки читання". Якщо потрібно відкрити файл із можливістю запису в нього, то варто вказати параметр `"w"`.

Метод `readlines` зчитує весь файл, створює масив і розміщує кожен прочитаний рядок в окремому елементі масиву При подальшій обробці отриманого масиву зручно використати метод `each`.

Задача 2

Є текстовий файл `fio.txt`, що містить список прізвищ, імен та по батькові учнів. При цьому кожен рядок файлу містить дані тільки про одну людину, наприклад,

```
Петров Сергій Васильович
Сидорова Ольга Петрівна
Іванова Марія Данилівна
```

Напишіть програму, що читає інформацію з файлу й

1. друкує пронумерований список учнів;
2. друкує пронумерований список прізвищ й ініціалів учнів.

Рішення 1

```
f = File.new("fio.txt")
n = 1
student = f.readlines
student.each{ |i|
 print n, ". ", i
 n += 1
}
```

Відзначимо, що програма повинна розташовуватися в тій же директорії, що й файл з даними. У противному випадку необхідно вказувати повний шлях до файлу з даними.

Для одержання ініціалів кожен рядок отриманого масиву перетворимо в масив, елементами якого будуть прізвище, ім'я та по батькові. Потім прізвище друкуємо повністю, а замість наступних елементів масиву - тільки їхній перший символ. Для перетворення рядка в масив використовується раніше розглянута функція `split`. Нагадаємо, що параметром за замовчуванням цієї функції є пробіл.

Рішення 2

```
f = File.new("fio.txt")
n = 1
student = f.readlines
student.each{ |i|
 i.chop!
 fio = i.split
 print n, ". ", fio[0], " ", fio[0][0].chr, ". ",
  fio[1][0].chr, ".\n"
 n += 1
}
```

3.2.3. Випадкові числа

Моделювання випадкових процесів на комп'ютері - досить розповсюджене явище. Як й у більшості мов програмування в Ruby є методи, що генерують, так звані, "псевдовипадкові" числа (їх не можна вважати чисто випадковими, адже створюються вони за допомогою деякого алгоритму).

Для одержання випадкового цілого числа в діапазоні від 0 але n використається метод `rand(n)`. Для знаходження випадкового числа в діапазоні від 0 до 1 викликають цей же метод, але без параметру, наприклад,

```
a1 = (1..10).collect { |j| rand(100)}
a2 = (1..4).collect { |j| rand()}
p a1, a2
```

Нижче представлений можливий результат:

```
[64, 17, 21, 95, 58, 24, 29, 60, 47, 63]
[0.2009671596, 0.8890923676, 0.3349569312, 0.8719313448]
```

Одним із застосувань випадкових чисел є комп'ютерне моделювання методу Монте-Карло для визначення площі деякої фігури. Фігуру вписують в іншу, площа якої відома, і випадковим образом "кидають" точки, підраховуючи число влучень у фігуру. При досить великій кількості випробувань відношення числа крапок, що потрапили усередину фігури, до загального числа крапок прагне до відношення їхніх площ.

Задача

Напишіть програму, що обчислює наближене значення числа PI за допомогою методу Монте-Карло.

Рішення

Число PI дорівнює відношенню площі кругу до квадрата його радіуса. Впишемо коло одиничного радіуса із центром на початку координат у квадрат і методом Монте-Карло знайдемо його площу. Координати випадкових точок, киданих у коло, належать інтервалу (-1; 1). Величина, випадково розподілена в цьому інтервалі, задається виразом `2*rand() - 1`.

Для контролю за часом додамо два екземпляри класу Time: час початку й закінчення розрахунку. Програма, що реалізує цей алгоритм, представлена нижче. В ній використовується оператор `eval`, щоб кількість точок можна було задавати у вигляді арифметичного виразу.

```
puts "Введіть кількість точок:"
n1, n, t1  = 0, eval(gets.chop), Time.now
for i in 1 .. n
 x = 2*rand() - 1
 y = 2*rand() - 1
#      перевіряємо влучення усередину кола
 n1 += 1 if (x**2 + y**2) < 1
end
puts "PI=#{4.0*n1/n}"
t2 = Time.now
puts "Число точок #{n}, час розрахунку " +
 "біля #{(t2 - t1).round} сек."
```

Можливі результати роботи програми будуть *схожі* (випадковість!) на наступні:

```
Введіть кількість точок: 10**5    PI=3.14028
Число точок 100000, час розрахунку близько 2 сек.

Введіть кількість точок: 10**6    PI=3.142204
Число точок 1000000, час розрахунку близько 20 сек.
```

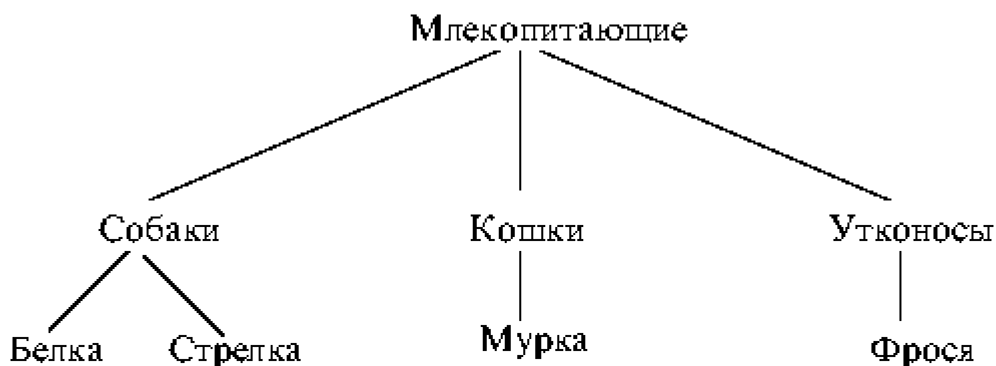
3.2.4. Об'єктно-орієнтоване програмування

Об'єктно-орієнтоване програмування (ООП) - це результат природної еволюції більш ранніх методологій програмування. Потреба в ООП зв'язана зі стрімким ускладненням розроблювальних програм й, як наслідок, їхньою недостатньою надійністю.

Можна сказати, що **ООП** - це моделювання об'єктів за допомогою ієрархічно зв'язаних класів. При цьому малозначні деталі об'єкта сховані від нас, і якщо ми даємо команду якомусь об'єкту, то він "знає", як її виконати. Фундаментальною концепцією в ООП є поняття обов'язку або *відповідальності* за виконання дії.

Всі об'єкти є представниками, або *екземплярами*, класів. Метод, який активізований об'єктом у відповідь на повідомлення, визначається класом, до якого належить одержувач повідомлення. Всі об'єкти одного класу використовують ті самі методи у відповідь на однакові повідомлення.

Класи представляються у вигляді ієрархічної деревоподібної структури, у якій класи з більше загальними рисами розташовуються в корені дерева, а спеціалізовані класи й в остаточному підсумку індивідуми розташовуються в галузях. На малюнку показана одна з можливих ієрархій класів, що включає в себе собак Білку й Стрілку, кішку Мурку й качкодзьоба Фросю.



Класи собак, кішок і качкодзьобів є дочірніми стосовно класу ссавців, отже успадковують його властивості. При програмній реалізації цієї ієрархії логічно метод "годівля дитинчати" реалізовувати в батьківському класі, замість того, щоб кілька разів дублювати його в кожному з підкласів. *Спадкування* властивостей батьківського класу дозволяє використати їх у дочірніх класах. Трохи інша ситуація з народженням дитинчат, адже качкодзьоби відкладають яйця, а не є живородними тваринами? У цій ситуації виручає властивість *полиформизма*: різні реалізації методів можуть носити однакові імена, а система сама визначить яку з реалізацій використати в тому або іншому випадку. У нашому прикладі треба в класі ссавців реалізувати метод "потомство" (народити дитинча), у класах собак і кішок цей метод буде відсутній (система буде шукати його в батьківському класі й знайде його там), а в класі качкодзьобів потрібно написати новий метод, з тим же ім'ям, але іншою реалізацією (відкласти яйця).

Таким чином, в основі ООП лежать три основні поняття:

- інкапсуляція (приховання даних у класі або методі);
- спадкування;
- поліморфізм.

Інкапсуляцію можна представити, як захисну оболонку навколо коду даних, з якими цей код працює. Оболонка задає поводження й захищає код від довільного доступу ззовні.

Спадкування - це процес, у результаті якого один тип успадковує властивості іншого типу.

Поліморфізм - це концепція, що дозволяє мати різні реалізації для того самого методу, які будуть вибиратися залежно від типу об'єкту, переданого метода при виклику.

Рішення

Нижчеподана програма ілюструє основні принципи об'єктно-орієнтованого програмування.

Помістить фрагмент коду, розташований нижче, у файл із ім'ям `life.rb`

```
class Animal
  def breath #дихання
    print "всі тварини дихають: вдихнули й видихнули\n"
  end
end
class Cat<Animal
  def bark # Подати голос
    print "Mew Mew, я кішка. \n"
 end
end
class Dog
 def bark # Подати голос
  print "Bow Wow, я собака. \n"
 end
end
class Bird
 def lay_egg
  print "Яйце знесене\n"
 end
 def fly
  print "Я птах, я лечу!!!\n"
 end
end
class Penguin<Bird
```

```
 def fly
  print "Пінгвіни не літають!!!\n"
 end
end
# Створюємо об'єкти різних класів
pochi = Dog.new
pochi.bark
tama = Cat.new
tama.breath
tama.bark
macaw  = Bird.new
macaw.lay_egg
macaw.fly
penguin  = Penguin.new
penguin.lay_egg
penguin.fly
```

Завдання 3.1.

1. Напишіть програму, що запитує з клавіатури натуральне число, більше 1, і друкує список всіх простих нескоротних дробів, укладених в інтервалі між 0 й 1, знаменники яких не перевищують введене число. Наприклад, якщо ввести число 4, то повинна бути надрукована послідовність 1/2 1/3 2/3 1/4 3/4. Підказка: використайтеся функцію знаходження НЗД.

2. Створіть текстовий файл, у якому розмістить прізвища учнів й їхній ріст у сантиметрах. Напишіть програму, яка

а) друкує прізвище й ріст найвищих учнів;

б) по введеному числу - росту в сантиметрах - друкує список всіх учнів, чий ріст не перевищує введеного числа.

3. Напишіть програму, що обчислює методом Монте-Карло площу криволінійної трапеції, яка обмежена графіками функцій y=sin(x) і y=0 (x змінюється в інтервалі від 0 до PI).

4. Запустіть програму life.rb, розглянуту вище. Якщо ви зрозуміли, який світ описує дана програма, то спробуйте виконати наведені нижче завдання:

а) створіть ще одну кішку;

б) поясніть, ким є pochi і чи зможе він виконати команду pochi.breath (дихати)? Якщо ні, то внесіть відповідні зміни в текст програми;

в) змініть код програми так, щоб і птахи в ній теж уміли дихати.

ІНДИВІДУАЛЬНІ ЗАВДАННЯ ДО ЛАБОРАТОРНОЇ РОБОТИ № 3

Варіант	**Номера задач для виконання індивідуального завдання з лабораторних робіт**																			
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**
1	1	11	19	29	37	47	55	65	73	83	91	101	109	119	127	137	145	154	163	172
2	2	12	20	30	38	48	56	66	74	84	92	102	110	120	128	138	146	155	164	173
3	3	13	21	31	39	49	57	67	75	85	93	103	111	121	129	139	147	156	165	174
4	4	14	22	32	40	50	58	68	76	86	94	104	112	122	130	140	148	157	166	175
5	5	15	23	33	41	51	59	69	77	87	95	105	113	123	131	141	149	158	167	176
6	6	16	24	34	42	52	60	70	78	88	96	106	114	124	132	142	150	159	168	177
7	7	17	25	35	43	53	61	71	79	89	97	107	115	125	133	143	151	160	169	178
8	8	18	26	36	44	54	62	72	80	90	98	108	116	126	134	144	152	161	170	179
9	9	19	27	37	45	55	63	73	81	91	99	109	117	127	135	145	153	162	171	180
10	10	20	28	38	46	56	64	74	82	92	100	110	118	128	136	146	154	163	172	181
11	11	21	29	39	47	57	65	75	83	93	101	111	119	129	137	147	155	164	173	182
12	12	22	30	40	48	58	66	76	84	94	102	112	120	130	138	148	156	165	174	183
13	13	23	31	41	49	59	67	77	85	95	103	113	121	131	139	149	157	166	175	184
14	14	24	32	42	50	60	68	78	86	96	104	114	122	132	140	150	158	167	176	185
15	15	25	33	43	51	61	69	79	87	97	105	115	123	133	141	151	159	168	177	186
16	16	26	34	44	52	62	70	80	88	98	106	116	124	134	142	152	160	169	178	187
17	17	27	35	45	53	63	71	81	89	99	107	117	125	135	143	153	161	170	179	188
18	18	28	36	46	54	64	72	82	90	100	108	118	126	136	144	154	162	171	180	189
19	19	29	37	47	55	65	73	83	91	101	109	119	127	137	145	155	163	172	181	190
20	20	30	38	48	56	66	74	84	92	102	110	120	128	138	146	156	164	173	182	191
21	21	31	39	49	57	67	75	85	93	103	111	121	129	139	147	157	165	174	183	192
22	22	32	40	50	58	68	76	86	94	104	112	122	130	140	148	158	166	175	184	193
23	23	33	41	51	59	69	77	87	95	105	113	123	131	141	149	159	167	176	185	194
24	24	34	42	52	60	70	78	88	96	106	114	124	132	142	150	160	168	177	186	195
25	25	35	43	53	61	71	79	89	97	107	115	125	133	143	151	161	169	178	187	196
26	26	36	44	54	62	72	80	90	98	108	116	126	134	144	152	162	170	179	188	197
27	27	37	45	55	63	73	81	91	99	109	117	127	135	145	153	163	171	180	189	198
28	28	38	46	56	64	74	82	92	100	110	118	128	136	146	154	164	172	181	190	199
29	29	39	47	57	65	75	83	93	101	111	119	129	137	147	155	165	173	182	191	200
30	30	40	48	58	66	76	84	94	102	112	120	130	138	148	156	166	174	183	192	201
31	31	41	49	59	67	77	85	95	103	113	121	131	139	149	157	167	175	184	193	195
32	32	42	50	60	68	78	86	96	104	114	122	132	140	150	158	168	176	185	194	196
33	33	43	51	61	69	79	87	97	105	115	123	133	141	151	159	169	177	186	195	197
34	34	44	52	62	70	80	88	98	106	116	124	134	142	152	160	170	178	187	196	198
35	35	45	53	63	71	81	89	99	107	117	125	135	143	153	161	171	179	188	197	199
36	36	46	54	64	72	82	90	100	108	118	126	136	144	154	162	172	180	189	198	200
37	37	47	55	65	73	83	91	101	109	119	127	137	145	155	163	173	181	190	199	201

Зміст завдань

1	Дан цілочисельний масив. Необхідно вивести спочатку його елементи з парними індексами, а потім - з непарними.
2	Дан цілочисельний масив. Необхідно вивести спочатку його елементи з непарними індексами, а потім - парними.
3	Дан цілочисельний масив. Вивести номер першого з тих його елементів, які задовольняють подвійній нерівності: A[0] < A[i] < A[-1]. Якщо таких елементів немає, то вивести [ ].
4	Дан цілочисельний масив. Вивести номер останнього з тих його елементів, які задовольняють подвійній нерівності: A[0] < A[i] < A[-1]. Якщо таких елементів немає, то вивести [ ].
5	Дан цілочисельний масив. Перетворити його, додавши до парних чисел перший елемент. Перший й останній елементи масиву не змінювати.
6	Дан цілочисельний масив. Перетворити його, додавши до парних чисел останній елемент. Перший й останній елементи масиву не змінювати.
7	Дан цілочисельний масив. Перетворити його, додавши до непарних чисел останній елемент. Перший й останній елементи масиву не змінювати.
8	Дан цілочисельний масив. Перетворити його, додавши до непарних чисел перший елемент. Перший й останній елементи масиву не змінювати.
9	Дан цілочисельний масив. Замінити всі додатні елементи на значення мінімального.
10	Дан цілочисельний масив. Замінити всі додатні елементи на значення максимального.
11	Дан цілочисельний масив. Замінити всі від'ємні елементи на значення мінімального.
12	Дан цілочисельний масив. Замінити всі від'ємні елементи на значення максимального.
13	Дан цілочисельний масив. Здійснити циклічний зсув масиву вліво на одну позицію.
14	Дан цілочисельний масив. Здійснити циклічний зсув елементів масиву вправо на одну позицію.
15	Дан цілочисельний масив. Перевірити, чи утворять елементи арифметичну прогресію. Якщо так, то вивести різницю прогресії, якщо ні - вивести nil.
16	Дан цілочисельний масив. Перевірити, чи утворять елементи геометричну прогресію. Якщо так, то вивести знаменник прогресії, якщо ні - вивести nil.

17	Дан цілочисельний масив. Знайти кількість його локальних максимумів.
18	Дан цілочисельний масив. Знайти кількість його локальних мінімумів.
19	Дан цілочисельний масив. Знайти максимальний з його локальних максимумів.
20	Дан цілочисельний масив. Знайти мінімальний з його локальних мінімумів.
21	Дан цілочисельний масив. Визначити кількість ділянок, на яких його елементи монотонно зростають.
22	Дан цілочисельний масив. Визначити кількість ділянок, на яких його елементи монотонно спадають.
23	Дано дробове число R і масив дробових чисел. Знайти елемент масиву, що найбільш близький до даного числа.
24	Дано дробове число R і масив дробових чисел. Знайти елемент масиву, що найменш близький до даного числа.
25	Дан цілочисельний масив. Перетворити його, вставивши перед кожним позитивним елементом нульовий елемент.
26	Дан цілочисельний масив. Перетворити його, вставивши перед кожним від'ємним елементом нульовий елемент.
27	Дан цілочисельний масив. Перетворити його, вставивши після кожного додатним елемента нульовий елемент.
28	Дан цілочисельний масив. Перетворити його, вставивши після кожного від'ємного елемента нульовий елемент.
29	Дан цілочисельний масив. Упорядкувати його за зростанням.
30	Дан цілочисельний масив. Упорядкувати його за спаданням.
31	Дан цілочисельний масив. Вивести індекси масиву в тім порядку, у якому відповідні їм елементи утворять спадаючу послідовність.
32	Дан цілочисельний масив. Вивести індекси масиву в тім порядку, у якому відповідні їм елементи утворять зростаючу послідовність.
33	Дан цілочисельний масив. Знайти індекс мінімального елемента.
34	Дан цілочисельний масив. Знайти індекс максимального елемента.
35	Дан цілочисельний масив. Знайти індекс першого мінімального елемента.
36	Дан цілочисельний масив. Знайти індекс першого максимального елемента.
37	Дан цілочисельний масив. Знайти індекс останнього мінімального елемента.
38	Дан цілочисельний масив. Знайти індекс останнього максимального елемента

39	Дан цілочисельний масив. Знайти кількість мінімальних елементів.
40	Дан цілочисельний масив. Знайти кількість максимальних елементів.
41	Дан цілочисельний масив. Знайти мінімальний парний елемент.
42	Дан цілочисельний масив. Знайти мінімальний непарний елемент.
43	Дан цілочисельний масив. Знайти максимальний парний елемент.
44	Дан цілочисельний масив. Знайти максимальний непарний елемент.
45	Дан цілочисельний масив. Знайти мінімальний додатний елемент.
46	Дан цілочисельний масив. Знайти максимальний від'ємний елемент.
47	Дан цілочисельний масив й інтервал a..b. Знайти мінімальний з елементів у цьому інтервалі.
48	Дан цілочисельний масив й інтервал a..b. Знайти максимальний з елементів у цьому інтервалі.
49	Дан цілочисельний масив. Знайти кількість елементів, розташованих перед першим мінімальним.
50	Дан цілочисельний масив. Знайти кількість елементів, розташованих перед першим максимальним.
51	Дан цілочисельний масив. Знайти кількість елементів, розташованих після першого максимального.
52	Дан цілочисельний масив. Знайти кількість елементів, розташованих після першого мінімального .
53	Дан цілочисельний масив. Знайти кількість елементів, розташованих перед останнім максимальним.
54	Дан цілочисельний масив. Знайти кількість елементів, розташованих перед останнім мінімальним.
55	Дан цілочисельний масив. Знайти кількість елементів, розташованих після останнього максимального.
56	Дан цілочисельний масив. Знайти кількість елементів, розташованих після останнього мінімального .
57	Дан цілочисельний масив. Знайти індекс першого екстремального (тобто мінімального або максимального) елемента.
58	Дан цілочисельний масив. Знайти індекс останнього екстремального (тобто

	мінімального або максимального) елемента.
59	Дан цілочисельний масив. Знайти кількість елементів, між першим й останнім мінімальним.
60	Дан цілочисельний масив. Знайти кількість елементів, між першим й останнім максимальним.
61	Дан цілочисельний масив. Знайти два найбільших елементи.
62	Дан цілочисельний масив. Знайти два найменших елементи.
63	Дан цілочисельний масив. Знайти максимальну кількість мінімальних елементів, що йдуть підряд.
64	Дан цілочисельний масив. Знайти максимальну кількість максимальних елементів,що йдуть підряд.
65	Дан цілочисельний масив. Вивести спочатку всі його парні елементи, а потім - непарні.
66	Дан цілочисельний масив. Вивести спочатку всі його непарні елементи, а потім - парні.
67	Дан цілочисельний масив. Перевірити, чи чергуються в ньому парні й непарні числа.
68	Дан цілочисельний масив. Перевірити, чи чергуються в ньому додатні й від'ємні числа.
69	Дано дробове число R і масив дробових чисел. Знайти два елементи масиву, сума яких найбільш близька до даного числа.
70	Дано дробове число R і масив дробових чисел. Знайти два елементи масиву, сума яких найменш близька до даного числа.
71	Дан цілочисельний масив. Видалити всі елементи, що зустрічаються менш двох разів.
72	Дан цілочисельний масив. Видалити всі елементи, що зустрічаються більше двох разів.
73	Дан цілочисельний масив. Видалити всі елементи, що зустрічаються рівно два рази.
74	Дан цілочисельний масив. Видалити всі елементи, що зустрічаються рівно три рази.
75	Дан цілочисельний масив. Знайти середнє арифметичне модулів його елементів.
76	Дан цілочисельний масив. Знайти середнє арифметичне квадратів його

	елементів.
77	Дано ціле число. Знайти суму його цифр.
78	Дано ціле число. Знайти добуток його цифр.
79	Дан цілочисельний масив. Піднести до квадрата від'ємні елементи й у третій ступінь - додатні. Нульові елементи - не змінювати.
80	Даний діапазон a..b. Одержати масив із чисел, розташованих у цьому діапазоні (крім самих цих чисел), у порядку їхнього зростання, а також розмір цього масиву.
81	Даний діапазон a..b. Одержати масив із чисел, розташованих у цьому діапазоні (крім самих цих чисел), у порядку їхнього спадання, а також розмір цього масиву.
82	Дано число A та натуральне число N. Знайти результат наступного виразу 1 + A + A*2 + A*3 + … + A*N.
83	Дано число A и натуральне число N. Знайти результат наступного виразу 1 - A + A*2 - A*3 + … + (-1**N)*A*N.
84	Дано натуральне число N. Знайти результат наступного добутку 1*2*…*N...
85	Дано натуральне число N. Якщо N - непарне, то знайти добуток 1*3*…*N; якщо N - парне, то знайти добуток 2*4*...*N.
86	Дан цілочисельний масив. Знайти середнє арифметичне його елементів.
87	Дан цілочисельний масив. Знайти всі парні елементи.
88	Дан цілочисельний масив. Знайти кількість парних елементів.
89	Дан цілочисельний масив. Знайти всі непарні елементи.
90	Дан цілочисельний масив. Знайти кількість непарних елементів.
91	Дан цілочисельний масив і число K. Якщо існує елемент, менший К, то вивести true; у противному випадку вивести false.
92	Дан цілочисельний масив і число K. Якщо існує елемент, більший К, то вивести true; у противному випадку вивести false.
93	Дан цілочисельний масив і число K. Якщо всі елементи масиву менше К, то вивести true; у противному випадку вивести false.
94	Дан цілочисельний масив і число K. Якщо всі елементи масиву більше К, то вивести true; у противному випадку вивести false.
95	Дан цілочисельний масив і число K. Вивести кількість елементів, менших К.

96	Дан цілочисельний масив і число К. Вивести індекс першого елемента, більшого К.
97	Дан цілочисельний масив і число К. Вивести індекс останнього елемента, меншого К.
98	Дан цілочисельний масив. Вивести індекси елементів, які менше свого лівого сусіда, і кількість таких чисел.
99	Дан цілочисельний масив. Вивести індекси елементів, які менше свого правого сусіда, і кількість таких чисел.
100	Дан цілочисельний масив. Вивести індекси елементів, які більше свого правого сусіда, і кількість таких чисел.
101	Дан цілочисельний масив. Вивести індекси елементів, які більше свого лівого сусіда, і кількість таких чисел.
102	Дан цілочисельний масив. Перевірити, чи утворить він зростаючу послідовність.
103	Дан цілочисельний масив. Перевірити, чи утворить він спадаючу послідовність.
104	Дан цілочисельний масив. Перевірити, чи утворить він упорядковану послідовність.
105	Дан цілочисельний масив. Якщо даний масив утворить спадаючу послідовність, то вивести nil, у противному випадку вивести номер першого числа, що порушує закономірність.
106	Дан цілочисельний масив, що містить принаймні два нулі. Вивести суму чисел з даного масиву, розташованих між першими двома нулями.
107	Дан цілочисельний масив, що містить принаймні два нулі. Вивести суму чисел з даного масиву, розташованих між останніми двома нулями.
108	Дан цілочисельний масив і ціле число К. Піднести до степеня К всі елементи масиву.
109	Дан цілочисельний масив. Знайти мінімальний і максимальний елемент у масиві.
110	Дан цілочисельний масив. Поміняти місцями мінімальний і максимальний елементи масиву.
111	Дан цілочисельний масив. Переставити у зворотному порядку елементи масиву, розташовані між його мінімальним і максимальним елементами.
112	Написати програму обчислення суми цифр, введеного з клавіатури натурального числа.

113	Ввести натуральне число й надрукувати його мінімальну цифру. Наприклад, для 1234076 відповіддю буде 0, а для 77777 - 7.
114	Ввести натуральне число й надрукувати кількість входжень його максимальної цифри в число. Так для числа 77777 відповідь буде 5, а для 12321 - відповідь дорівнює 1.
115	Число 2005 можна представити у вигляді суми послідовних натуральних чисел декількома способами, наприклад, 2005 = 1002+1003 = 399+400+401+402+403. Знайдіть кількість подібних розкладань для введеного з клавіатури числа.
116	Знайти таке десятичне число, що всі його цифри різні, причому число, складене з перших двох його цифр, ділиться на 2, з перших трьох цифр - на 3, із чотирьох - на 4, і так далі, а саме число ділиться на 10.
117	Яка найбільша кількість цифр може бути в числі, якщо будь-який його "початок" ділиться на кількість цифр цього початку?
118	Напишіть програму, що для заданого натурального числа n визначає кількість входжень числа 3 у розкладання заданого числа на прості співмножники.
119	Знайдіть кількість максимальних елементів у масиві.
120	У масиві знаходиться n дійсних чисел. Відомо, що серед них є хоча б одне від'ємне. Знайдіть величину найбільшого серед всіх від'ємних чисел.
121	Дано текст. Знайдіть найбільшу кількість цифрв ньому, які йдуть підряд.
122	Дано текст. Визначите максимальне з наявних у ньому чисел.
123	Дано два рядки. Визначите, скільки початкових символів першого рядка збігається з початковими символами другого. Розгляньте два випадки: а) відомо, що рядки різні; б) рядки можуть співпадати.
124	Розширте клас Time методом, що визначає кількість днів, що залишилися до Нового року.
125	Позитивне ціле число d називається найбільшим загальним дільником (НЗД) цілих чисел a й b, якщо: a) d ділить a й d ділить b, б) для будь-якого іншого дільника цих чисел c (c != d), d ділить c. Відомо, що НЗД(x, 0) = НЗД(0, x) = x і НЗД(x, y) = НЗД(y, x%y), де x%y є остача від ділення x на y. Розширте клас Integer методом gcd (greatest common divisor), що обчислює НЗД двох цілих чисел.
126	В одному масиві записана кількість м'ячів, забитих футбольною командою в кожній з 20 ігор, в іншому - кількість пропущених м'ячів у цій же грі. Для кожної гри визначите словесний результат гри (виграш, програш або нічию).
127	В одному масиві записаний ріст деяких студентів, а в іншому (з тим же числом елементів) - їхні прізвища в тім же порядку, у якому зазначений ріст. Відомо, що всі студенти різного росту. Надрукуйте прізвище найвищого студента.

128	Напишіть функцію simplify(num), призначену для винесення співмножників заданого натурального числа num з під знака квадратного кореня. Функція повинна повертати масив із двох натуральних чисел [a, b], таких, що N = a**2*b і не існує натуральних c > a й d,
129	Напишіть функцію text_simplify(num), що виводить результат роботи попередньої функції у вигляді рядка тексту.
130	Напишіть програму, що запитує із клавіатури натуральне число n і визначє кількість різних цифр у його десятковому записі.
131	Напишіть програму, що визначає кількість квитків з 6-значними номерами, у яких сума перших 3 десяткових цифр дорівнює сумі 3 останніх.
132	Напишіть програму, що вводить із клавіатури рядок символів, що містить послідовність слів, розділених пробілами, і друкує всі паліндроми (слова, які читаються зліва направо так само, як і зправа наліво), що зустрілися в рядку.
133	Напишіть програму, що запитує з клавіатури натуральне число n і друкує число n!!, де n!! = 1*3*5*...* n для непарного n й n!! = 2*4*6*...*n для парного.
134	Послідовність чисел Фібоначі задається співвідношеннями a(0) =0, a(1) =1, a(i)= a(i-1)+a(i-2), i = 2, 3, … Напишіть програму, що запитує із клавіатури натуральне число n і друкує числа Фібоначі від 0 до n.
135	Напишіть програму, що запитує з клавіатури масив чисел і друкує кількість елементів, що належать відрізку [3; 7], і їхню суму.
136	Напишіть програму, що запитує із клавіатури натуральне число n і видаляє з його десяткового подання всі 5 й 0, залишаючи порядок інших цифр колишнім. Із числа 59015504, наприклад, повинне вийти 914.
137	Напишіть програму, що запитує із клавіатури число X і масив чисел, а потім визначає, чи втримуються в цьому масиві три елементи, які йдуть підряд, рівні X.
138	Напишіть програму, що запитує із клавіатури натуральне число й друкує масив, що містить всі натуральні дільники введеного числа.
139	Напишіть програму, що вводить із клавіатури масив чисел і друкує номер першого елемента, рівного 0, або повідомлення про те, що масив не містить нульових елементів.
140	Напишіть програму, що запитує із клавіатури масив чисел a[i] і два числа N й M такі, що 0 <= N <= M < a.size, після чого визначає, чи є фрагмент масиву a[N] ... a[M] упорядкованим по зростанню.
141	Напишіть програму, що вводить із клавіатури масив чисел і друкує найбільший індекс від'ємного елемента або повідомлення про те, що масив не містить від'ємних елементів.

142	Напишіть програму, що запитує із клавіатури натуральне число k>1 і заповнює масив першими k натуральними числами, що діляться на 13 або 17.
143	Послідовність чисел Фібоначі задається співвідношеннями a(0) =0, a(1) =1, a(i)= a(i-1)+a(i-2), i = 2, 3, ... Напишіть програму, що вводить з клавіатури масив цілих чисел, що належать діапазону від 1 до 50, і визначає, скільки серед них чисел Фібоначі й скільки чисел, в яких перша значуща цифра в десятковому записі дорівнює 1 або 2.
144	У текстовому файлі містяться відомості про учнів школи: прізвище, ім'я й клас (число й буква) у наступному форматі: Іванова Дарья 7в, Сидоров Михайло 8а, Іванова Дарья 10м, … Напишіть програму, що визначає чи є "тезки" (люди зі співпадаючими прізвищами й іменами) у паралельних класах.
145	Текстовий файл містить прізвища людей, їхні ініціали й телефон у наступному форматі (двокрапка відокремлює телефон від інших даних): Сидоров И. И. : 700-89-06, Петров Н. А. : 163-67-50, ... Напишіть програму, що визначає телефон за введеними з клавіатури прізвищем та ініціалам.
146	Текстовий файл містить дані про кубики: розмір кожного кубика - довжина ребра в сантиметрах, його кольори - червоний (к), жовтий (ж), зелений (з), синій (с) і матеріал - дерево (д) або метал (м) у наступному форматі: 12.4 к д, 98 з м 78.34 с м, … Напишіть програму, що визначає кількість і сумарний об'єм дерев'яних кубиків кожного з перерахованих кольорів.
147	Напишіть програму, що знаходить і виводить на друк всі чотиризначні числа abcd, для яких виконуються наступні умови: 1) a, b, c, d - різні цифри й 2) ab-cd=a+b+c+d. Тут запис ab означає, що число складене із цифр a й b.
148	Напишіть програму, що вводить із клавіатури масив чисел і визначає максимальну кількість додатних елементів масиву, які йдуть підряд і неперериваються ні нулями, ні від'ємними елементами, і друкує знайдений фрагмент.
149	Напишіть програму, що запитує з клавіатури три пари чисел — координати трьох точок на площині - і визначає, чи утворять вони невироджений трикутник. У випадку позитивної відповіді програма повинна надрукувати його площу й тип (рівносторонній, рівнобедрений або загальний вид). Підказка: використайте формулу Герона для обчислення площі. Трикутник вироджений, якщо його площа виявляється рівною 0.
150	Напишіть програму, що за допомогою датчика випадкових чисел вибирає натуральне число, менше 100, і пропонує ввести з клавіатури загадане число. Після введення числа повідомляється більше або менше уведене число, ніж задумане. Якщо число вгадане, то програма поздоровляє гравця з виграшем, якщо ж за 7 ходів не вдалося відгадати задумане число, то програма повинна припинити гру, повідомивши про програш. Припустимо, що програма "загадала" число 38. У цьому випадку діалог може виглядати, наприклад, так: Введіть число: 64 Число велике, спробуйте ще раз: 21 Число мале, спробуйте ще раз: 44 Число велике, спробуйте ще раз: 40 Число велике, спробуйте ще

	раз: 30 ...
151	Напишіть програму для навчання переведенню чисел із двійкової системи числення в десяткову. Програма за допомогою датчика випадкових чисел повинна вибрати число, що не перевищує 20, вивести його на екран у двійковій системі числення й перевірити, чи дійсно введене після цього з клавіатури число є його десятковим записом.
152	Напишіть програму, що друкує кількість натуральних рішень нерівності x^2+y^2 < n для введеного натурального числа n.
153	Напишіть програму, що вводить з клавіатури непорожний масив чисел і друкує новий масив такий, що в ньому спочатку йдуть всі додатні, потім нульові, а потім від'ємні числа з введеного масиву зі збереженням порядку проходження чисел одного знака. Наприклад, з масиву [-3, 5, 2, 0, -4, -1, 0, 5] повинне вийти [5, 2, 5, 0, 0, -3, -4, -1].
154	Напишіть програму, що вводить з клавіатури непорожний масив цілих чисел, заміняє всі елементи масиву, крім крайніх, на напівсуму сусідніх, і друкує результат.
155	Напишіть програму, що вводить із клавіатури непорожний масив чисел і заміняє нулями всі від'ємні елементи, що передують його максимальному елементу (першому за порядком, якщо їх декілка).
156	Напишіть програму, що вводить з клавіатури непорожний масив різних чисел, міняє в ньому місцями найбільший і найменший елементи й потім друкує отриманий масив.
157	Напишіть програму, що вводить з клавіатури число A і непорожний масив чисел, після чого друкує кількість елементів, більших за A.
158	Напишіть програму, що вводить з клавіатури число x і непорожний масив чисел [a_0, a_1, a_2, ..., a_n], після чого друкує значення полінома P = a_n * x^n + a_(n-1) * x^(n-1) + ... +a_1 * x +a_0.
159	Напишіть програму, що вводить із клавіатури непорожний масив різних чисел і друкує другий по величині елемент масиву.
160	У змаганнях з фігурного катання N суддів (N > 3) незалежно виставляють оцінки спортсменові. Потім з оголошених оцінок виділяють найвищу (тільки одну, якщо найвищу оцінку виставили декілька суддів). Аналогічно діють і з найнижчою оцінкою. Для оцінок, що залишилися, обчислюється середнє арифметичне, котре й стає підсумковою оцінкою. Напишіть програму, що вводить із клавіатури ряд чисел - оцінки суддів — і друкує підсумкову оцінку спортсмена.
161	Напишіть програму, що запитує з клавіатури два цілих числа a й b і визначає, чи є в сумі цих чисел дві однакові цифри, що йдуть підряд. Наприклад, для a = 100, b = 10 відповідь "є"; для a = 111, b = 10 відповідь "ні".

162	Розглянемо довільне натуральне число й знайдемо суму його цифр, потім суму цифр отриманого числа й так далі, поки не одержимо однозначне число. Назвемо це число цифровим коренем. Напишіть програму, що запитує з клавіатури натуральне число N та обчислює його цифровий корінь.
163	При піднесенні деяких натуральних чисел у квадрат отриманий результат може кінчатися вихідним числом. Наприклад, 76*76 = 5776. Напишіть програму, що запитує із клавіатури натуральне число N і друкуючу всі N-значні числа, що володіють зазначеною властивістю.
164	У текстовому файлі міститься інформація про два прямокутники. Кожен рядок містить чотири числа, що описують один прямокутник: координат-x - координата лівого нижнього кута, координат-y - координата лівого нижнього кута, ширина прямокутника і його висота. Наприклад, 1.5 2.0 3.5 6.7, 2.1 3.2 1.0 4.5. Напишіть програму, що зчитує дані з такого файлу й визначає, чи вірно, що другий прямокутник цілком міститься в першому. У випадку позитивної відповіді програма повинна надрукувати різницю їх площ.
165	З курсу математичного аналізу відомо, що значення певного інтеграла від a до b безперервної функції f(x) дорівнює площі криволінійної трапеції, обмеженою віссю абсцис, графіком функції f(x) і прямими x = a, x = b. Напишіть програму, що запитує з клавіатури натуральне число n та обчислює певний інтеграл функції f(x) = sin(x) на відрізку від a = 0 до b = PI методом трапецій, при якому графік функції замінюється на ламану лінію, що проходить через крапки (i; f(i)), де i = a + (b - a) * k/n, а k пробігає всі значення від 0 до n.
166	Напишіть програму, що запитує з клавіатури три пари чисел — координати трьох крапок на площині - і визначає, чи утворять вони невироджений трикутник (у виродженого трикутника більша сторона дорівнює сумі двох інших сторін). У випадку позитивної відповіді програма повинна надрукувати його вид (гострокутний, прямокутний або тупокутний) і величини його кутів (у градусах).
167	Напишіть програму, що запитує з клавіатури натуральне число й друкує цифру, що зустрічається в даному числі найбільше число раз, і число її входжень. У випадку, коли таких цифр виявляється декілька, програма повинна перелічити їх усі.
168	Напишіть програму, що запитує з клавіатури два натуральних числа: число годин m (0<=12) і число хвилин n (0<=n<60). Якщо введено некоректні дані, програма, сповістивши про це, повинна знову повернутися до введення даних. Для коректних даних слід надрукувати найменший час (число повних хвилин), який повинен пройти до того моменту, коли годинна й хвилинна стрілки збіжаться.
169	Напишіть програму, що запитує з клавіатури натуральне число n і друкує всі прості дільники введеного числа.
170	Напишіть програму, що друкує чотиризначні прості числа, кожне з яких

	володіє тією властивістю, що сума першої й другої цифри запису цього числа дорівнює сумі третьої й четвертої цифр.
171	Напишіть програму, що запитує із клавіатури два натуральних числа й виводе масив, що містить всі їхні загальні дільники.
172	Напишіть програму, що запитує з клавіатури натуральне число й визначає, чи можна представити це число у вигляді суми квадратів двох натуральних чисел.
173	Натуральне число зветься зробленим, якщо воно дорівнює сумі всіх своїх дільників, за винятком самого себе. Наприклад, число 6 — зроблене, тому що 1+2+3=6, а 8 не є зробленим (1+2+4 не дорівнює 8). Напишіть програму, що запитує з клавіатури натуральне число й друкує масив, що містить всі зроблені числа, що не перевищують введеного числа.
174	Напишіть програму, що запитує із клавіатури масив чисел і визначає кількість різних його членів.
175	Текстовий файл містить інформацію про ріст у сантиметрах юнаків (м) і дівчин (ж) у наступному форматі: Єршов Володимир : 170 : м, Сидоров Михайло : 168 : м, Петрова Марья : 169 : ж, Ворошилов Олег : 183 : м, Репкина Ольга : 162 : ж, ... Напишіть програму, що запитує з клавіатури рід (чоловічий або жіночий) і друкує список кандидатів у баскетбольну команду. Дівчина є кандидатом у баскетбольну команду, якщо її ріст перевищує 160 см, а юнак, якщо 170 см.
176	Напишіть програму, що запитує з клавіатури натуральне число n і друкує перші n рядків трикутника Паскаля.
177	Два натуральних числа називаються дружніми, якщо кожне з них дорівнює сумі всіх дільників іншого, крім самого цього числа. Наприклад, 220 й 284 є дружніми числами. Напишіть програму, що запитує з клавіатури два натуральних числа N й M (N < M) і друкує всі пари дружніх чисел у діапазоні від N до M.
178	Натуральне число з n цифр є числом Амстронга, якщо сума його цифр, зведених в n-ю ступінь, дорівнює самому числу (наприклад, 153=1^3+5^3+3^3=1+125+27). Напишіть програму, що друкує всі числа Амстронга, що складаються із двох, трьох і чотирьох цифр.
179	Назвемо натуральне число паліндромом, якщо воно збігається із числом, записаним тими ж цифрами, але у зворотному порядку (наприклад, 45654). Напишіть програму, що визначає всі натуральні числа менші ста, квадрат яких є паліндромом.
180	Напишіть програму, що запитує з клавіатури натуральне число n і визначає, чи можна представити n! у вигляді добутку трьох послідовних натуральних чисел.
181	Розглянемо деяке натуральне число n. Якщо воно не є паліндромом (тобто не збігається із числом, записаним тими ж цифрами, але у зворотному порядку), то змінимо порядок його цифр на зворотний і складемо число, що вийшло, з

	вихідним. Якщо сума не є паліндромом, то будемо повторювати ці дії доти, поки не вийде паліндром. Напишіть програму, що запитує з клавіатури три натуральних числа K, L й M (K не перевищує L), і визначає, чи вірно, що для будь-якого натурального числа з діапазону від K до L процес завершиться не пізніше, ніж за M таких дій.
182	Розглянемо деяке натуральне число n>1. Якщо воно парне, то розділимо його на 2, інакше помножимо на 3 і додамо 1. Будемо повторювати цей процес доти, поки не вийде 1. Напишіть програму, що запитує з клавіатури три числа K, L, M (K не перевищує L), і визначає, чи вірно, що для будь-якого натурального числа з діапазону від K до L процес завершиться не пізніше, ніж за M таких дій.
183	Напишіть програму, що друкує всі менші 10^6 натуральні числа, які є паліндромами (збігаються із числами, записаними тими ж цифрами, але у зворотному порядку) як у десятковій, так й у двійковій системах числення.
184	Напишіть програму, що запитує з клавіатури масив чисел a[i] й упорядковує його за неспаданням (a[i]<=a[i+1]). Для рішення задачі скористайтеся алгоритмом, який зветься сортування вибором: знайдіть найменший елемент і поміняйте його місцями з першим елементом, потім знайдіть найменший серед тих, що залишилися, та обміняйте його місцями із другим елементом і т.д.
185	Напишіть програму, що вводить з клавіатури масив чисел і визначає найбільше число однакових елементів, що йдуть у ньому підряд.
186	Напишіть програму, що вводить цілі коефіцієнти багаточлена й знаходить всі його раціональні корені. Підказка: скористайтеся теоремою Безу, відповідно до якої чисельник p будь-якого раціонального кореня багаточлена x=p/q є дільником вільного члена багаточлена, а знаменник q — дільником коефіцієнта при старшому ступені багаточлена.
187	У двох текстових файлах зберігається інформація про стартовий і фінішний час всіх учасників лижної гонки на 30 км у наступному форматі: номер учасника, години, хвилини й секунди, розділені пробілами. Напишіть програму, що перевіряє коректність даних і друкує результат (годину, хвилини, секунди) і зайняте місце учасника по запитаному із клавіатури номеру.
188	Напишіть програму, що вводить із клавіатури арифметичний вираз у вигляді суми двох двозначних (або однозначних) натуральних чисел і друкує словесний вираз виду Перший Доданок плюс Другий Доданок дорівнює Сума. Коли результат додавання є тризначним числом варто друкувати його цифровий запис. Наприклад, при введенні рядка 2+17, програма повинна надрукувати рядок " два плюси сімнадцять дорівнює дев'ятнадцять", а при уведенні 56+73 - "п'ятдесят шість плюс сімдесят три дорівнює 129".
189	Напишіть програму, що вводить із клавіатури непорожний масив чисел, у якому можуть бути повторювані елементи, і друкує значення третього по величині елемента масиву або повідомлення про те, що в цьому масиві немає трьох різних чисел.

190	У текстовому файлі міститься набір двозначних чисел, розділених пробілами. Кожне число являє собою закодовану кістку доміно. Напишіть програму, що зчитує дані з файлу й визначає, чи можна їх усі викласти в ряд, не порушуючи правил гри. У випадку позитивної відповіді представити один з можливих варіантів такого розташування. Наприклад, для вхідних даних 31 00 13 одержуємо відповідь: некоректні вхідні дані; а для вхідних даних 02 04 42 відповідь така: можна, 04 42 20.
191	Напишіть програму, що по заданому раціональному числу, записаному у вигляді звичайного дробу з натуральними чисельником a і знаменником b, друкувала б відповідний цьому числу десятковий дріб. Якщо дріб є періодичним, то варто надрукувати період дробу, наприклад, 1/3 = 0,(3). Підказка. Спочатку виділіть цілу частину (може й нульову), а потім розглянете послідовність залишків, що починається з остачі k від ділення a на b (k = a % b), у якій кожен наступний елемент виходить по формулі k = (10 * k) % b. Якщо в цій послідовності зустрінеться нульова остача, то дріб є кінцевим, інакше він зобов'язан бути періодичним, що означає повторення одного із залишків на деякому кроці (з алгебри відомо, що довжина періоду не може перевищити b - 1).
192	Для поділу російського слова при переносі з одного рядка на інший найчастіше користуються трьома основними правилами: (I) дві голосні, що йдуть підряд, можна розділити, якщо перша з них передує, а за другою йде яка-небудь приголосна буква; (II) дві приголосні, що йдуть підряд, можна розділити, якщо першій з них передує гласна, а в тій частині слова, що йде за другою приголосною, є хоча б одна гласна; (III) якщо правила (I)-(II) незастосовні, то можна розділити слово так, щоб права частина містила більш, ніж одну букву, і кінчалася на гласну, а ліва містила більш, ніж одну букву, серед яких була б хоча одна гласна. Примітка: буква Й приєднується до попередньої гласної й розглядається разом з нею як єдина гласна; аналогічно, букви Ь и Ъ приєднуються до попередньої приголосної. Напишіть програму, що по заданому слову російської мови запропонує всі можливі варіанти його розбивки на частини для переносу.
193	Напишіть програму, що запитує з клавіатури фразу російською мовою й зашифровує текст, що міститься у файлі, за наступними правилами: а) після кожної гласної букви додати по дві чергові букви з введеної ключової фрази (після вичерпання букв фраза повторюється знову); б) наприкінці кожного рядка додати три довільні прописні (малі) букви російського алфавіту; в) для кожного набору із чотирьох символів рядка переставити четвертий символ з першим, а другий с третім (хвостові символи рядка, що не ввійшли в четвірки, не переставлялися).
194	Напишіть програму, що запитує з клавіатури фразу російською мовою й розшифровує текст, що міститься у файлі, який був раніше закодований за наступними правилами: а) після кожної гласної букви були додані по дві чергові букви з введеної ключової фрази (після вичерпання букв фраза повторюється знову); б) наприкінці кожного рядка додані три довільні прописні (малі) букви російського алфавіту; в) у кожному наборі із чотирьох

	символів рядка були зроблені перестановки: четвертий символ переставили з першим, а другий із третім (хвостові символи рядка, що не ввійшли в четвірки, не переставлялися).
195	Напишіть програму, що запитує з клавіатури два натуральних числа A и B - сторони прямокутника - і визначає на скільки квадратів його можна розрізати, відрізаючи щораз квадрат максимальної площі із цілою довжиною сторони. Надрукувати кількість і розмір всіх квадратів, наприклад, при A = 20, B = 10 відповідь: 2 зі стороною 10; при A = 20, B = 15 відповідь: 1 зі стороною 15, 3 зі стороною 5.
196	Пляшка лимонаду коштує K рублів, а порожня пляшка - M рублів. Родина в понеділок на всі гроші купила лимонад. Випивши все, вони наступного дня здали порожні пляшки, додали здачу попереднього дня й знову на всі гроші купили лимонад. Дана процедура повторювалася щодня, поки була можливість. Напишіть програму, що запитує з клавіатури два натуральних числа K та M і друкує мінімальну суму грошей, при якій у п'ятницю родина зможе купити хоча б одну пляшку лимонаду.
197	X людей розташовані по колу. Почавши відлік від першого, видаляють кожного K-го, стуляючи коло після кожного видалення. Напишіть програму, що запитує із клавіатури два натуральних числа X и K і друкує номер останнього що залишився.
198	Бригада із трьох роботів збирає за один день ще одного нового робота. Час життя нового робота - 5 днів, після закінчення яких він гине. Напишіть програму, що запитує з клавіатури два натуральних числа M и N, після чого визначає, скільки роботів буде існувати через N днів, якщо на момент старту було M нових роботів. Роботи, що не ввійшли в бригади, простоюють.
199	Напишіть програму, що запитує з клавіатури дійсне число $r > 0$ і натуральне qmax, після чого визначає найкраще наближення r у вигляді раціонального дробу p/q, де q не перевищує qmax.
200	Напишіть програму, що запитує з клавіатури два натуральних числа a й b, після чого визначає два цілих числа x й y такі, щоб трикутник ABC з вершинами в точках A(0;0), B(a;b) і C(x;y) був невиродженим і мав мінімально можливу площу.
201	Напишіть програму, що запитує з клавіатури два натуральних числа N й s (де s менше кількості цифр в N) і визначає, які s цифр числа N варто видалити, щоб цифри, що залишилися, склали найменше можливе число. Надрукувати в порядку зростання номера вилучених цифр. Цифри пронумеровані зліва направо, а нумерація починається з одиниці. Наприклад, при N = 2435, s = 1 варто видалити другу цифру (4), після чого вийде число 235. Це і є найменше число, тому що інші (435, 245, 243) більше його.

ДОВІДНИК ПО RUBY

Клас Class < Module

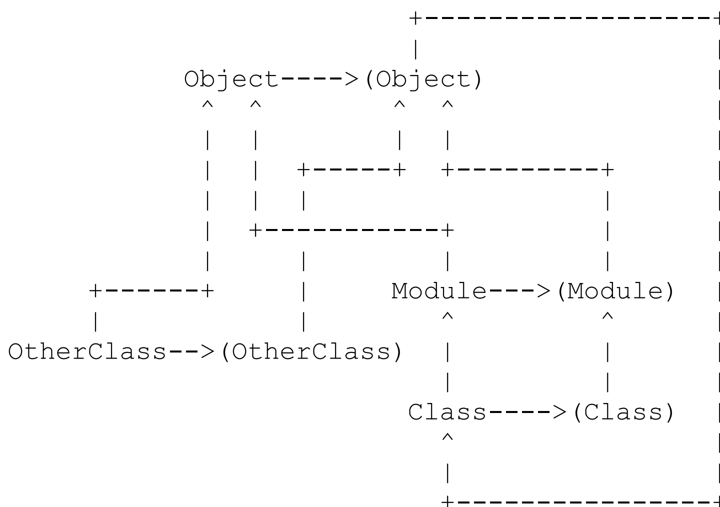
Класи в Ruby це унікальні об'єкти, кожний з яких є екземпляром класу Class. Коли новий клас створюється (звичайно використається class Name ... end), об'єкт типу Class створюється й привласнюється однойменній глобальній змінній (у цьому випадку Name). Коли викликається Name.new для створення нового об'єкта, запускається метод new класу Class. Це можна продемонструвати перевантаженням методу new у класі Class:

```
class Class
   alias oldNew  new
   def new(*args)
      print "Создается новый ", self.name, "\n"
      oldNew(*args)
   end
end

class Name
end

n = Name.new
```

результат:

Створюється новий Name

Класи, модулі й об'єкти взаємозалежні. На наступній діаграмі, вертикальними стрілками позначене спадкування, а круглими дужками - метакласи. Всі метакласи - це об'єкти класу 'Class'.



Оголошені атрибути будуть доступні в межах ієрархії спадкування, де кожен нащадок одержує копію атрибутів своїх батьків, замість істинного вказівника на нього. Це означає, що нащадок може додати елементи, для приклада, у масив без тих доповнень які він ділить зі своїм батьком, елементами одного з ним рівня (які мають загального батька) або нащадками, які не схожі на правильні атрибути рівня класу і які розділені поперек всієї ієрархії.

Методи класу
new
Методи об'єкту
allocate, inherited, new, superclass

Class::new

```
Class.new(super_class=Object)   #->    a_class
```

Створення нового анонімного (безіменного) класу успадкованого від super_class (або Object, якщо викликається без параметрів). Ви можете дати класу ім'я, привласнивши його константі

Class#allocate

```
class.allocate   #->   obj
```

Виділяє місце для нового об'єкта класу class. Об'єкт, що повертає, є екземпляром класу class.

Class#inherited

```
class.inherited( sub_class )
```

Викликається Ruby, коли відбувається спадкування від класу class. Новий підклас передається як параметр sub_class.

```
class Top
   def Top.inherited(sub)
      puts "Новый подкласс: #{sub}"
   end
end

class Middle < Top
end

class Bottom < Middle
end
```

результат:

```
Новый підклас: Middle
Новый підклас: Bottom
```

Class#new

```
class.new(args, ...)    #->  obj
```

Викликає allocate для створення нового об'єкта класу class, викликається метод initialize, якому передається args. Цей метод викликається щораз, коли створюється об'єкт за допомогою методу.new.

Class#superclass

```
class.superclass   #-> a_super_class или nil
```

Повертає суперклас (від якого відбулося спадкування) класу class, або nil.

```
File.superclass     #-> IO
IO.superclass       #-> Object
Object.superclass   #-> nil
```

Класс NilClass

Глобальне значення nil є єдиним екземпляром класу NilClass й означає «відсутність значення». У логічному контексті еквівалентно false. Методи, які хочуть сказати, що їм нема чого повернути - повертають nil. Змінні, значення яким не привласнене - мають значення nil.

Методи об'єкту

&, ^, inspect, nil?, to_a, to_f, to_i, to_s, ||

NilClass#&

```
false & obj   #-> false
nil & obj     #-> false
```

Логічне «І» завжди повертає false. obj завжди обчислюється, тому що є агрументом методу. У цьому випадку немає ніякого скороченого обчислення.

NilClass#^

```
false ^ obj    #-> true или false
nil   ^ obj    #-> true или false
```

Логічне «АБО НІ». Якщо obj дорівнює nil або false, повертає false; інакше повертає true.

NilClass#inspect

```
nil.inspect  #-> "nil"
```

Завжди повертає рядок "nil".

NilClass#nil?

```
nil.nil?   #-> true
```

Завжди повертає true.

NilClass#to_a

```
nil.to_a    #-> []
```

Завжди повертає порожній масив.

NilClass#to_f

```
nil.to_f    #-> 0.0
```

Завжди повертає нуль.

NilClass#to_i

```
nil.to_i    #-> 0
```

Завжди повертає нуль.

NilClass#to_s

```
nil.to_s    #-> ""
```

Завжди повертає порожній рядок.

NilClass#|

```
false | obj   #->   true или false
nil   | obj   #->   true или false
```

Логічне «АБО» повертає false, якщо obj дорівнює nil або false; true інакше.

Класс Array

Масив - упорядкована колекція довільних об'єктів із цілочисельною індексацією. Нумерація

елементів масиву починається з 0, як у мовах C або Java. Негативний індекс припускає відлік з кінця масиву, тобто індексу -1 відповідає останній елемент масиву, -2 - передостанній, і так далі.

Домішки

Enumerable (all?, any?, collect, detect, each_cons, each_slice, each_with_index, entries, enum_cons, enum_slice, enum_with_index, find, find_all, grep, group_by, include?, index_by, inject, map, max, member?, min, partition, reject, select, sort, sort_by, sum, to_a, to_set, zip)

Методи класу

[], new

Методи об'єкту

[]=, [], &, |, *, +, -, <<, <=>, ==, abbrev, assoc, at, clear, collect!, collect, compact!, compact, concat, delete_at, delete_if, delete, each_index, each, empty?, eql?, fetch, fill, first, flatten!, flatten, frozen?, hash, include?, indexes, index, indices, insert, inspect, join, last, length, map!, map, nitems, pack, pop, push, rassoc, reject!, reject, replace, reverse!, reverse_each, reverse, rindex, select, shift, size, slice!, slice, sort!, sort, to_ary, to_a, to_s, transpose, uniq!, uniq, unshift, values_at, zip

Array::[]

```
Array::[](...)
```

Повертає новий масив, заповнений зазначеними об'єктами.

```
Array.[]( 1, 'a', /^A/ )
Array[ 1, 'a', /^A/ ]
[ 1, 'a', /^A/ ]
```

Array::new

```
Array.new(size=0, obj=nil)
Array.new(array)
Array.new(size){|index| block }
```

Повертає новий масив. У першій формі виклику, створюється порожній масив. У другий, створюється масив розміру size, заповнений копіями obj (тобто, size посилань на obj). У третьої, створюється копія масив, переданого як параметр (масив створюється за допомогою виклику методу to_ary від масиву-параметра). В останньому випадку, створюється масив зазначеного розміру. Кожен елемент у цьому масиві обчислюється в зазначеному блоці, якому передається індекс оброблюваного елемента. Результат блоку записується як значення елемента в масив.

```
Array.new
Array.new(2)
Array.new(5, "A")

# только одна копия объекта создается
a = Array.new(2, Hash.new)
a[0]['cat'] = 'feline'
a
```

```
a[1]['cat'] = 'Felix'
a

# здесь создается несколько копий объекта
a = Array.new(2) { Hash.new }
a[0]['cat'] = 'feline'
a

squares = Array.new(5) {|i| i*i}
squares

copy = Array.new(squares)
```

Array#&

```
array & other_array
```

Перетинання множин - повертає новий масив, що складається з елементів, які є в обох масивах, але без дублікатів.

```
[ 1, 1, 3, 5 ] & [ 1, 2, 3 ]   #=> [ 1, 3 ]
```

Array#|

```
array | other_array     ->  an_array
```

Об'єднання множин - повертає новий масив, що поєднує елементи масивів array й other_array, але з вилученими дублікатами.

```
[ "a", "b", "c" ] | [ "c", "d", "a" ]
       #=> [ "a", "b", "c", "d" ]
```

Array#*

```
array * int     ->    an_array
array * str     ->    a_string
```

Повторення - зі строковим аргументом еквівалентний коду array.join(str). Інакше, повертає новий масив, що складається з int зчеплених копій array.

```
[ 1, 2, 3 ] * 3    #=> [ 1, 2, 3, 1, 2, 3, 1, 2, 3 ]
[ 1, 2, 3 ] * ","  #=> "1,2,3"
```

Array#+

```
array + other_array   -> an_array
```

Зчеплення - повертає новий масив, створений із двох масивів шляхом додавання одного до іншого.

```
[ 1, 2, 3 ] + [ 4, 5 ]    #=> [ 1, 2, 3, 4, 5 ]
```

Array#-

```
array - other_array    -> an_array
```

Вирахування масивів - повертає новий масив, що копіює оригінальний масив, але видаляє з нього елементи, які є в іншому масиві.

```
[ 1, 1, 2, 2, 3, 3, 4, 5 ] - [ 1, 2, 4 ]  #=>  [ 3, 3, 5 ]
```

Якщо вам необхідна різниця множин, а не вирахування масивів, то дивитеся убік класу Set

Array#<<

```
array << obj            -> array
```

Додати елемент - додає переданий об'єкт у кінець масиву. Повертає масив із уже доданим елементом.

```
[ 1, 2 ] << "c" << "d" << [ 3, 4 ]
        #=>  [ 1, 2, "c", "d", [ 3, 4 ] ]
```

Метод push має приблизно таку ж функціональність

Даний метод змінює вихідний масив. Будьте уважні! Щоб виключити подібного роду зміни використайте метод +

Array#<=>

```
array <=> other_array   ->  -1, 0, +1
```

Порівняння - повертає ціле число (-1, 0, або +1) якщо поточний масив менше, дорівнює або більше іншого масиву. Відповідні елементи обох масивів рівняються (використається метод <=>). Якщо яка або з пар елементів не збігається, то повертається результат цього порівняння. Якщо все пари елементів збігаються, то повертається результат порівняння по довжині. Таким чином, два масиви вважаються «рівними» (на думку методу Array#<=>) тоді й тільки тоді, коли їхні довжини й відповідні пари елементів збігаються.

```
[ "a", "a", "c" ]    <=> [ "a", "b", "c" ]   #=> -1
[ 1, 2, 3, 4, 5, 6 ] <=> [ 1, 2 ]            #=> +1
```

Array#==

```
array == other_array   ->   bool
```

Рівність - два масиви вважаються рівними, якщо кількість елементів і відповідні пари елементів рівні (використається метод ==).

```
[ "a", "c" ]    == [ "a", "c", 7 ]      #=> false
[ "a", "c", 7 ] == [ "a", "c", 7 ]      #=> true
```

```
[ "a", "c", 7 ] == [ "a", "d", "f" ]   #=> false
```

Array#[]

```
array[index]                -> obj      или nil
array[start, length]        -> an_array или nil
array[range]                -> an_array или nil
array.slice(index)          -> obj      или nil
array.slice(start, length)  -> an_array или nil
array.slice(range)          -> an_array или nil
```

Одержання значення елемента - повертає елемент із індексом index, або подмассив довжини length, починаючи з індексу start, або подмассив, що розташовується в діапазоні range. Негативна індексація припускає звіт з кінця масиву (по індексі -1 розташовується останній елемент). Повертає nil, якщо індекс виходить за діапазон припустимих значень

```
a = [ "a", "b", "c", "d", "e" ]
a[2] +  a[0] + a[1]    #=> "cab"
a[6]                   #=> nil
a[1, 2]                #=> [ "b", "c" ]
a[1..3]                #=> [ "b", "c", "d" ]
a[4..7]                #=> [ "e" ]
a[6..10]               #=> nil
a[-3, 3]               #=> [ "c", "d", "e" ]
# специальные случаи
a[5]                   #=> nil
a[5, 1]                #=> []
a[5..10]               #=> []
```

•Методи slice й [] - одне й теж!

- Існує метод at, що володіє схожої, але меншою функціональністю

Array#[]=

```
maccuB[uHgekc]        = obj                          ->  obj
maccuB[cTapT, pa3mep] = obj или an_array или nil  ->  obj или an_array или nil
maccuB[guana3oH]      = obj или an_array или nil  ->  obj или an_array или nil
```

Розподіл елементів - поміщає елемент в maccu по uHgekcy, або заміняє подмассив деякого pa3mepa, починаючи з індексу cTap, або заміняє подмассив у певному guana3oHe. Якщо індексація перевищує поточний розмір масиву, то масив збільшується автоматично. Негативна індексація має на увазі відлік з кінця масиву. Якщо для другого випадку використати pa3mep рівний 0, то зазначені елементи будуть вставлені перед елементом з індексом cTap. Якщо всі параметри для другого й третього випадку задати рівними nil, то з maccuBa будуть вилучені всі елементи. Помилка IndexError з'являється тоді, коли негативний індекс указує на елемент, розташований перед початком масиву.

```
a = Array.new
a[4] = "4";                 #=> [nil, nil, nil, nil, "4"]
a[0, 3] = [ 'a', 'b', 'c' ] #=> ["a", "b", "c", nil, "4"]
a[1..2] = [ 1, 2 ]          #=> ["a", 1, 2, nil, "4"]
a[0, 2] = "?"               #=> ["?", 2, nil, "4"]
```

```
a[0..2] = "A"                #=> ["A", "4"]
a[-1]   = "Z"                #=> ["A", "Z"]
a[1..-1] = nil               #=> ["A"]
```

Корисно знати, що додавання елементів у масив існують ще методи push й unshift

Array#abbrev

```
maccuB.abbrev(pattern = nil)
```

Обчислює набір однозначних скорочень для рядків в maccuBe. Якщо в якості pattern передається правило або рядок, то будуть оброблятися тільки рядка, які відповідають правилу або починаються з даного рядка.

```
%w{ car cone }.abbrev   #=> { "ca" => "car", "car" => "car",
                              "co" => "cone", "con" => cone",
                              "cone" => "cone" }
```

Для використань методу abbrev необхідне підключення бібліотеки abbrev.rb зі стандартної бібліотеки Руби: require 'abbrev'

Array#assoc

```
array.assoc(obj)   ->  an_array  или  nil
```

Шукає у двовимірному масиві масив, перший елемент якого дорівнює obj (для порівняння використається метод ==). Повертає перший знайдений масив (тобто, асоційований масив) або nil, якщо такий знайти не вдалося

```
s1 = [ "colors", "red", "blue", "green" ]
s2 = [ "letters", "a", "b", "c" ]
s3 = "foo"
a  = [ s1, s2, s3 ]
a.assoc("letters")  #=> [ "letters", "a", "b", "c" ]
a.assoc("foo")      #=> nil
```

Для розумінь що відбуває, корисно глянути на метод rassoc

Array#at

```
array.at(index)   ->   obj  или  nil
```

Повертає елемент масиву array з індексом index. Негативна індексація має на увазі відлік з кінця масиву. повертає nil, якщо індекс виходить за межі припустимого дианазона.

```
a = [ "a", "b", "c", "d", "e" ]
a.at(0)     #=> "a"
a.at(-1)    #=> "e"
```

- Рекомендується також глянути на метод [] (aka «батарейка»), що має схожого функціонала
- Метод at працює ледве швидше методу [] за рахунок того, що не обробляє

нічого крім целочисленных індексів, тобто за рахунок ігнорування діапазонів

Array#clear

```
array.clear    ->  array
```

Видаляє всі елементи з масиву array.

```
a = [ "a", "b", "c", "d", "e" ]
a.clear    #=> [ ]
```

Array#collect

```
array.collect {|item| block }  -> an_array
array.map     {|item| block }  -> an_array
```

Виконує вираження block для кожного елемента масиву array. Створює новий масив, що складається зі значень, які отримані при обчисленні вираження block.

```
a = [ "a", "b", "c", "d" ]
a.collect {|x| x + "!" }   #=> ["a!", "b!", "c!", "d!"]
a                          #=> ["a", "b", "c", "d"]
```

- Є ітератори Enumerable#collect й Enumerable#map, які мають схожу функціональність
- Ітератори map й collect - те саме!

Array#collect!

```
array.collect! {|item| block }   ->   array
array.map!     {|item| block }   ->   array
```

Виконує вираження block для кожного елемента масиву array, обчислене вираження підставляється замість поточного елемента.

```
a = [ "a", "b", "c", "d" ]
a.collect! {|x| x + "!" }
a             #=>  [ "a!", "b!", "c!", "d!" ]
```

- Є ітератори Enumerable#collect й Enumerable#map, які мають схожу функціональність
- Ітератори map! і collect! - те саме!

Будьте уважні при використанні даного ітератора, тому що він змінює вихідний масив. Щоб виключити цю зміну використайте ітератори collect й map

.

Array#compact

```
array.compact     ->  an_array
```

Повертає копію масиву array з якого вилучені всі елементи nil.

```
[ "a", nil, "b", nil, "c", nil ].compact
                  #=> [ "a", "b", "c" ]
```

Array#compact!

```
array.compact!    ->   array  или  nil
```

Повертає копію масиву array з якого вилучені всі елементи nil.

```
[ "a", nil, "b", nil, "c" ].compact! #=> [ "a", "b", "c" ]
[ "a", "b", "c" ].compact!           #=> nil
```

Даний метод змінює вихідний масив. Методу compact ця «дурна звичка» не властива

Array#concat

```
array.concat(other_array)   ->  array
```

Додає до масиву array елементи масиву other_array.

```
[ "a", "b" ].concat( ["c", "d"] ) #=> [ "a", "b", "c", "d" ]
```

Array#delete

```
array.delete(obj)            -> obj или nil
array.delete(obj){ block }  -> obj или nil
```

Видаляє всі елементи масиву array, які рівні obj. Якщо нічого не було вилучено, то повертає nil. Якщо задано блок, те, у випадку відсутності елементів для видалення, повертає результат блоку.

```
a = [ "a", "b", "b", "b", "c" ]
a.delete("b")                   #=> "b"
a                               #=> ["a", "c"]
a.delete("z")                   #=> nil
a.delete("z") { "not found" }   #=> "not found"
```

Даний метод змінює вихідний масив. Будьте уважні! Щоб виключити подібного роду зміни використайте итератор reject

Array#delete_at

```
array.delete_at(index)  -> obj или nil
```

Видаляє елемент із масиву array, що має індекс index. Повертає значення вилученого елемента або nil, якщо index перебуває поза припустимим діапазоном.

```
a = %w( ant bat cat dog )
a.delete_at(2)    #=> "cat"
a                 #=> ["ant", "bat", "dog"]
a.delete_at(99)   #=> nil
```

Метод slice! має схожу функціональність

Будьте уважні при використанні даного методу, тому що він змінює вихідний масив. Щоб виключити цю зміну використайте метод slice

Array#delete_if

```
array.delete_if {|item| block }  -> array
```

Видаляє всі елементи масиву array для яких значення усередині блоку block дорівнює true.

```
a = [ "a", "b", "c" ]
a.delete_if {|x| x >= "b" }   #=> ["a"]
a                             #=> ["a"]
```

Корисно подивитися на наступні ітератори reject й reject!, які мають схожу функціональність

Даний итератор змінює значення вихідного масиву. Рекомендується використати итератор reject, що не має такої «дурної» звички, але назва якого гірше запам'ятовується

Array#each

```
array.each {|item| block }   ->   array
```

Виконує код в block для кожного елемента масиву array, передаючи в блок поточний елемент як параметр.

```
a = [ "a", "b", "c" ]
a.each {|x| print x, " -- " }
```

результат:

```
a -- b -- c --
```

- Даний ітератор повертає як результат вихідний масив. Це варто враховувати при побудові ланцюжка методів
- Варто використати даний метод тільки тоді, коли задачу неможливо вирішити

за допомогою інших ітераторів

Array#each_index

```
array.each_index {|index| block }  ->  array
```

Працює точно також, як й each, але передає в блок не поточний елемент, а індекс поточного елемента.

```
a = [ "a", "b", "c" ]
a.each_index {|x| print x, " -- " }
```

результат:

```
0 -- 1 -- 2 --
```

Array#empty?

```
array.empty?   -> true или false
```

Повертає true, якщо array не містить елементів.

```
[].empty?   #=> true
```

Array#eql?

```
array.eql?(other)  -> true или false
```

Повертає true, якщо array й other є тим самим об'єктом або обидва масиви мають однаковий уміст.

Array#fetch

```
array.fetch(index)                    -> obj
array.fetch(index, default )          -> obj
array.fetch(index) {|index| block }   -> obj
```

Спроба одержати елемент масиву array по індексі index. Якщо індекс виходить за межі масиву те, у першій формі виклику виникне помилка IndexError, у другій формі виклику буде повернуте значення default, і в третій формі виклику буде повернуте значення блоку block (у який передається запитуваний індекс як параметр). Негативна індексація має на увазі відлік з кінця масиву.

```
a = [ 11, 22, 33, 44 ]
a.fetch(1)               #=> 22
a.fetch(-1)              #=> 44
a.fetch(4, 'cat')        #=> "cat"
a.fetch(4) { |i| i*i }   #=> 16
```

Array#fill

```
array.fill(obj)                              -> array
array.fill(obj, start [, length])            -> array
array.fill(obj, range )                      -> array
array.fill {|index| block }                  -> array
array.fill(start [, length] ) {|index| block } -> array
array.fill(range) {|index| block }           -> array
```

Перші три форми виклику заміняють зазначені елементи масиву array на значення obj. Якщо значення start дорівнює nil, то це еквівалентно start=0. Якщо значення length дорівнює nil, то це еквівалентно lenght=array.lenght. Останні три форми виклику заповнюють масив значенням вираження в блоці block (у який передається індекс поточного замінного елемента).

```
a = [ "a", "b", "c", "d" ]
a.fill("x")              #=> ["x", "x", "x", "x"]
a.fill("z", 2, 2)        #=> ["x", "x", "z", "z"]
a.fill("y", 0..1)        #=> ["y", "y", "z", "z"]
a.fill {|i| i*i}         #=> [0, 1, 4, 9]
a.fill(-2) {|i| i*i*i}   #=> [0, 1, 8, 27]
```

Array#first

```
array.first    -> obj или nil
array.first(n) -> an_array
```

Повертає перший елемент або перші n елементів масиву array. Якщо масив порожній, то для першої форми виклику (без n) повертається nil, а для другий - порожній масив.

```
a = [ "q", "r", "s", "t" ]
a.first    #=> "q"
a.first(1) #=> ["q"]
a.first(3) #=> ["q", "r", "s"]
```

Array#flatten

```
array.flatten -> an_array
```

Перетворить багатомірний масив array в одномірний.

```
s = [ 1, 2, 3 ]           #=> [1, 2, 3]
t = [ 4, 5, 6, [7, 8] ]   #=> [4, 5, 6, [7, 8]]
a = [ s, t, 9, 10 ]       #=> [[1, 2, 3], [4, 5, 6, [7, 8]], 9, 10]
a.flatten                 #=> [1, 2, 3, 4, 5, 6, 7, 8, 9, 10
```

Array#flatten!

```
array.flatten! -> array или nil
```

Перетворить багатомірний масив array в одномірний. Повертає nil, якщо масив array споконвічно був одномірним.

```
a = [ 1, 2, [3, [4, 5] ] ]
a.flatten!   #=> [1, 2, 3, 4, 5]
a.flatten!   #=> nil
a            #=> [1, 2, 3, 4, 5]
```

Будьте уважні при використанні даного методу, тому що він змінює вихідний масив. Щоб виключити цю зміну використайте метод flatten

Array#frozen?

```
array.frozen?  -> true или false
```

Повертає true, якщо масив array заморожений (або тимчасово заморожений, поки йде сортування).

Array#hash

```
array.hash   -> fixnum
```

Обчислює хеш-код для масиву array. Два масиви з тим самим умістом будуть мати однаковий хеш-код (саме його використає eql?).

Array#include?

```
array.include?(obj)   -> true или false
```

Повертає true, якщо об'єкт obj є елементом масиву array (тобто якийсь із елементів масиву == obj). Інакше - false.

```
a = [ "a", "b", "c" ]
a.include?("b")   #=> true
a.include?("z")   #=> false
```

Array#index

```
array.index(obj)   ->  int или nil
```

Повертає індекс першого входження елемента в масив array, що == obj. Повертає nil, якщо такий елемент не був знайдений.

```
a = [ "a", "b", "c" ]
a.index("b")   #=> 1
a.index("z")   #=> nil
```

Раджу глянути на метод rindex, що не тільки схожий за назвою, але, у деяких випадках,

працює точно також, як й index

Array#indexes

```
array.indexes( i1, i2, ... iN )   -> an_array
array.indices( i1, i2, ... iN )   -> an_array
```

Використання даного методу різко засуджується Руби-співтовариством. Під час його використання інтерпретатор буде видавати попередження. Рекомендується використати метод values_at.

Array#indices

```
array.indexes( i1, i2, ... i )   -> an_array
array.indices( i1, i2, ... i )   -> an_array
```

Використання даного методу різко засуджується Руба-співтовариством. Під час його використання інтерпретатор буде видавати попередження. Рекомендується використати метод values_at.

Array#insert

```
array.insert(index, obj...)  -> array
```

Вставляє отримані значення перед елементом індексом index (може бути негативним).

```
a = %w{ a b c d }
a.insert(2, 99)         #=> ["a", "b", 99, "c", "d"]
a.insert(-2, 1, 2, 3)   #=> ["a", "b", 99, "c", 1, 2, 3, "d"]
```

Array#inspect

```
array.inspect  -> string
```

Створює «друковану» версію масиву array.

Array#join

```
array.join(sep=$,)    -> str
```

Повертає рядок, створений шляхом перетворення кожного елемента масиву в рядок,

розділених рядком sep.

```
[ "a", "b", "c" ].join        #=> "abc"
[ "a", "b", "c" ].join("-")   #=> "a-b-c"
```

Array#last

```
array.last     ->  obj або nil
array.last(n)  ->  an_array
```

Повертає останній елемент або останні n елементів масиву array. Якщо масив порожній, то для першої форми виклику (без n) повертається nil, а для другий - порожній масив.

```
[ "w", "x", "y", "z" ].last   #=> "z"
```

Array#length

```
array.length -> int
array.size   -> int
```

Повертає кількість елементів в array. Може бути нульовим (для порожнього масиву).

```
[ 1, 2, 3, 4, 5 ].length   #=> 5
```

Методи lenght й size — те саме!

Array#map

```
array.collect {|item| block }  -> an_array
array.map     {|item| block }  -> an_array
```

Виконує вираження block для кожного елемента масиву array. Створює новий масив, що складається зі значень, які отримані при обчисленні вираження block.

```
a = [ "a", "b", "c", "d" ]
a.map {|x| x + "!" }       #=> ["a!", "b!", "c!", "d!"]
a                          #=> ["a", "b", "c", "d"]
```

- Є итераторы Enumerable#collect й Enumerable#map, які мають схожу функціональність

- Итераторы map й collect - те саме!

Array#map!

```
array.collect! {|item| block }   ->   array
array.map!     {|item| block }   ->   array
```

Виконує вираження block для кожного елемента масиву array, обчислене вираження підставляється замість поточного елемента.

```
a = [ "a", "b", "c", "d" ]
a.map! {|x| x + "!" }
a             #=>  [ "a!", "b!", "c!", "d!" ]
```

- Є ітератори Enumerable#collect й Enumerable#map, які мають схожу функціональність

- Ітератори map! і collect! - те саме!

Будьте уважні при використанні даного ітератора, тому що він змінює вихідний масив. Щоб виключити цю зміну використайте ітератори collect й map

Array#nitems

```
array.nitems -> int
```

Повертає кількість елементів масиву array, значення яких не дорівнює nil. Може бути нульовим (для порожнього масиву або масиву, що заповнений nil.

```
[ 1, nil, 3, nil, 5 ].nitems   #=> 3
```

Array#pack

```
array.pack ( aTemplateString ) -> aBinaryString
```

Упаковує вміст масиву array у двійкову послідовність (у вигляді рядка) відповідно до опцій у рядку aTemplateString (див. таблицю нижче). Після опцій A, a й Z може йти число, що вказує на ширину результуючого поля. Для інших, число після опцій означає - кількість елементів масиву які необхідно обробити відповідно до зазначеної опції. Якщо в замість числа після директиви коштує символ * («зірочка»), то всі елементи, що залишилися, масиву необхідно обробити відповідно до цієї опції. Кожну з опцій s, S, i, I, l й L можна завершувати _ (знаком підкреслення) для того, щоб використати розмір для базової платформи; інакше, буде використаний платформонезависимый розмір. У рядку aTemplateString пробіли ігноруються.

```
a = [ "a", "b", "c" ]
n = [ 65, 66, 67 ]
a.pack("A3A3A3")   #=> "a  b  c  "
a.pack("a3a3a3")   #=> "a\000\000b\000\000c\000\000"
n.pack("ccc")      #=> "ABC"
```

Опції методу pack.

Опція	Опис
@	Переміститися на абсолютну позицію
A	Рядок ASCII (доповнена пробілами до зазначеної ширини)
a	Рядок ASCII (доповнена нульовими символами до зазначеної ширини)
B	Двійковий рядок (зростаюча двійкова послідовність)
b	Двійковий рядок (убутна двійкова послідовність)

C	Беззнаковий 8-бітний символ (char)
c	8-бітний символ (char)
D, d	Число із плаваючою крапкою подвійної точності, власний формат
E	Число із плаваючою крапкою подвійної точності, прямий порядок двійкової послідовності
e	Число із плаваючою крапкою одинарної точності, прямий порядок двійкової послідовності
F, f	Число із плаваючою крапкою одинарної точності, власний формат
G	Число із плаваючою крапкою подвійної точності, «мережна» (зворотна) двійкова послідовність
g	Число із плаваючою крапкою одинарної точності, «мережна» (зворотна) двійкова послідовність
H	Двійковий рядок (верхній напівбайт іде першим)
h	Двійковий рядок (нижній напівбайт іде першим)
I	Беззнакове ціле (unsigned integer)
i	Ціле (integer)
L	Велике беззнакове ціле (usigned long)
l	Велике ціле (long)
M	Рядок готова до печатки, закодована MIME (див. RFC2045)
m	Рядок закодована Base64 (див. RFC4648)
N	Велике ціле (long),«мережна» (зворотна) двійкова послідовність
n	Коротке ціле (short), «мережна» (зворотна) двійкова послідовність

P	Покажчик на структуру (рядок фіксованої млинці)
p	Покажчик на рядок (із завершальним нульовим символом)
Q, q	64-бітне число
S	Беззнакове коротке ціле (unsigned short)
s	Коротке ціле (short)
U	UTF-8
u	Рядок UU-кодована
V	Довге ціле (long), прямий порядок байт
v	Коротке ціле (short), прямий порядок байт
w	Із ціле
X	Символ повернення каретки
x	Нульовий символ
Z	Теж саме, що й директива а, крім те, що нульовий символ додається з * («зірочкою»)
Даний метод звичайно використається разом з методом String#unpack, що виконує зворотні дії	

Array#pop

```
array.pop  -> obj або nil
```

Видаляє останній елемент із масиву array і повертає його. Якщо на момент виклику масив був порожній, то повертає nil.

```
a = [ "a", "m", "z" ]
a.pop   #=> "z"
a       #=> ["a", "m"]
```

Даний метод змінює вихідний масив. Будьте уважні! Щоб виключити подібного роду зміни використайте метод [] («батарейка»)

Array#push

```
array.push(obj, ... )   -> array
```

Додати елемент - додає переданий об'єкт (або кілька об'єктів) у кінець масиву. Повертає масив з виконаним перетворенням.

```
a = [ "a", "b", "c" ]
a.push("d", "e", "f")    #=> ["a", "b", "c", "d", "e", "f"]
a                        #=> ["a", "b", "c", "d", "e", "f"]
```

Даний метод змінює вихідний масив. Будьте уважні! Щоб виключити подібного роду зміни використайте метод +

Array#rassoc

```
array.rassoc(key) -> an_array або nil
```

Шукає у двовимірному масиві масив, останній елемент якого дорівнює key (для порівняння використається метод ==). Повертає перший знайдений масив (тобто, асоційований масив) або nil, якщо такий знайти не вдалося

```
a = [ [ 1, "one"], [2, "two"], [3, "three"], ["ii", "two"] ]
a.rassoc("two")    #=> [2, "two"]
a.rassoc("four")   #=> nil
```

Для розуміння що відбуває, корисно глянути на метод assoc

Array#reject

```
array.reject {|item| block }  -> an_array
```

Повертає новий масив, що складається з елементів масиву array, для яких значення усередині блоку block дорівнює false (або nil).

```
a = [1,2,3,4,5]
a.reject{ |item| item > 2 }    #-> [1,2]
a                              #-> [1,2,3,4,5]
```

- Корисно глянути на ітератори delete_if й reject!, які відрізняється лише тим, що змінюють вихідний масив

- Даний метод реалізований на базі методу Enumerable#reject

Array#reject!

```
array.reject! {|item| block }  -> array або nil
```

Подібно итератору delete_if видаляє з масиву array елементи, для яких значення усередині блоку block дорівнює true, але повертає nil, якщо змін у масиві зроблено не було.

```
a = [1,2,3,4,5]
a.reject!{ |item| item > 2 }    #-> [1,2]
a                               #-> [1,2]
a.reject!{ |item| item > 2 }    #-> nil
```

Корисно подивитися опис итератора delete_if, що робить приблизно теж саме, але назва якого краще запам'ятовується

Даний метод змінює вихідний масив. Будьте уважні! Щоб виключити подібного роду зміни використайте метод [] («батарейка») або итератор reject

Array#replace

```
array.replace(other_array)  -> array
```

Заміняє вміст масиву array умістом масиву other_array.

```
a = [ "a", "b", "c", "d", "e" ]
a.replace([ "x", "y", "z" ])    #=> ["x", "y", "z"]
a                               #=> ["x", "y", "z"]
```

Даний метод працює як присвоювання, але може бути використаний при побудові ланцюжка методів (яку присвоювання звичайно розриває)

Будьте уважні при використанні даного методу, тому що він змінює вихідний масив

.

Array#reverse

```
array.reverse -> an_array
```

Повертає новий масив, у якому елементи масиву array коштують у зворотному порядку.

```
[ "a", "b", "c" ].reverse   #=> ["c", "b", "a"]
[ 1 ].reverse               #=> [1]
```

Array#reverse!

```
array.reverse!   -> array
```

Змінює порядок елементів у масиві array на зворотний.

```
a = [ "a", "b", "c" ]
```

```
a.reverse!        #=> ["c", "b", "a"]
a                 #=> ["c", "b", "a"]
```

Будьте уважні при використанні даного методу, тому що він змінює вихідний масив. Щоб виключити цю зміну використайте метод reverse

.

Array#reverse_each

```
array.reverse_each {|item| block }
```

Працює точно також, як й итератор each, але елементи масиву array обробляються у зворотному порядку.

```
a = [ "a", "b", "c" ]
a.reverse_each {|x| print x, " " }
```

результат:

c b a

Досягти точно такого ж результату можна от так: array.reverse.each {|item| block }

Array#rindex

```
array.rindex(obj)    ->  int або nil
```

Повертає індекс останнього входження елемента в масив array, що == obj. Повертає nil, якщо такий елемент не був знайдений Простою мовою - знаходить obj з кінця.

```
a = [ "a", "b", "b", "b", "c" ]
a.rindex("b")   #=> 3
a.rindex("z")   #=> nil
```

Раджу глянути на метод index, що не тільки схожий за назвою, але, у деяких випадках, працює точно також, як й rindex

Array#select

```
array.select {|item| block } -> an_array
```

Виконує вираження в block для кожного елемента масиву array і повертає масив, що складається з елементів, для яких вираження в block приймає значення true.

```
a = %w{ a b c d e f }
a.select {|v| v =~ /[aeiou]/}   #=> ["a", "e"]
```

Є ітератори Enumerable#select й #Enumerable#find_all, які мають схожу функціональність

Array#shift

```
array.shift   ->   obj or nil
```

Видаляє перший елемент із масиву array і повертає його (як би «зрушує» масив уліво на один елемент). Якщо на момент виклику масив був порожній, то повертає nil.

```
args = [ "-m", "-q", "filename" ]
args.shift   #=> "-m"
args         #=> ["-q", "filename"]
```

Даний метод звичайно використають разом з методами push, pop й unshift

Даний метод змінює вихідний масив. Будьте уважні! Щоб виключити подібного роду зміни використайте метод [] («батарейка»)

Array#size

```
array.length -> int
array.size   -> int
```

Повертає кількість елементів в array. Може бути нульовим (для порожнього масиву).

```
[ 1, 2, 3, 4, 5 ].size   #=> 5
```

Методи lenght й size — те саме!

Array#slice

```
array[index]                -> obj      or nil
array[start, length]        -> an_array or nil
array[range]                -> an_array or nil
array.slice(index)          -> obj      or nil
array.slice(start, length)  -> an_array or nil
array.slice(range)          -> an_array or nil
```

Одержання значення елемента - повертає елемент із індексом index, або подмассив довжини length, починаючи з індексу start, або подмассив, що розташовується в діапазоні range. Негативна індексація припускає звіт з кінця масиву (по індексі -1 розташовується останній елемент). Повертає nil, якщо індекс виходить за діапазон припустимих значень.

```
a = [ "a", "b", "c", "d", "e" ]
a.slice(2) +  a.slice(0) + a.slice[1]    #=> "cab"
a.slice[6]                               #=> nil
a.slice(1, 2)                            #=> [ "b", "c" ]
a.slice(1..3)                            #=> [ "b", "c", "d" ]
a.slice(4..7)                            #=> [ "e" ]
a.slice(6..10)                           #=> nil
a.slice(-3, 3)                           #=> [ "c", "d", "e" ]
```

спеціальні випадки

```
a.slice(5)                               #=> nil
a.slice(5, 1)                            #=> []
```

```
a.slice(5..10)                          #=> []
```

- Методи slice й [] - одне й теж!

- Існує метод at, що володіє схожої, але меншою функціональністю

Array#slice!

```
array.slice!(index)         -> obj або nil
array.slice!(start, length) -> sub_array або nil
array.slice!(range)         -> sub_array або nil
```

Видаляє елемент або елементи масиву array по індексі (іноді із тривалістю lenght) або діапазону. Повертає видаля об'єкт, подмассив або nil (якщо зазначений індекс виходить за межу припустимих значень).

```
def slice!(*args)
  result = self[*args]
  self[*args] = nil
  result
end

a = [ "a", "b", "c" ]
a.slice!(1)     #=> "b"
a               #=> ["a", "c"]
a.slice!(-1)    #=> "c"
a               #=> ["a"]
a.slice!(100)   #=> nil
a               #=> ["a"]
```

Приватними реалізаціями даного методу є методи pop (array.slice!(-1)) і shift (array.slice!(0))

Даний метод змінює вихідний масив. Будьте уважні! Щоб виключити подібного роду зміни використайте методи [] («батарейка»), values_at або итератор reject

Array#sort

```
array.sort                   -> an_array
array.sort {| a,b | block }  -> an_array
```

Повертає новий масив, що отриманий шляхом сортування масиву array (по зростанню). Елементи масиву повинні допускати порівняння за допомогою методу <=>, якщо це не забезпечується, то можна використати необов'язковий блок. Блок призначений для порівняння параметрів a й b, повинен повертати значення -1, 0, або +1.

```
a = [ "d", "a", "e", "c", "b" ]
a.sort                    #=> ["a", "b", "c", "d", "e"]
a.sort {|x,y| y <=> x }   #=> ["e", "d", "c", "b", "a"]
```

Більше цікава й частіше використовувана форма сортування виробляється за допомогою итератора Enumerable#sort_by

Array#sort!

```
array.sort!                      -> array
array.sort! {| a,b | block }  -> array
```

Сортує масив array (по зростанню). Елементи масиву повинні допускати порівняння за допомогою методу <=>, якщо це не забезпечується, то можна використати необов'язковий блок. Блок призначений для порівняння параметрів a й b, повинен повертати значення -1, 0, або +1.

```
a = [ "d", "a", "e", "c", "b" ]
a.sort                     #=> ["a", "b", "c", "d", "e"]
a.sort {|x,y| y <=> x }    #=> ["e", "d", "c", "b", "a"]
```

Більше цікава й частіше використовувана форма сортування виробляється за допомогою методу Enumerable#sort_by

Будьте уважні при використанні даного итератора, тому що він змінює вихідний масив. Щоб виключити цю зміну використайте итераторы sort й Enumerable#sort_by

.

Array#to_a

```
array.to_a     -> array
```

Повертає покажчик на self. Якщо викликається від нащадка класу Array, то конвертує зазначений об'єкт у масив (об'єкт класу Array).

Array#to_ary

```
array.to_ary -> array
```

Повертає покажчик на self. По суті, нічого не робить.

Array#to_s

```
array.to_s -> string
```

Повертає результат self.join.

```
[ "a", "e", "i", "o" ].to_s   #=> "aeio"
```

Array#transpose

```
array.transpose -> an_array
```

Має на увазі, що array є масивом масивів (тобто з розмірністю більше, ніж один) і міняє місцями стовпці з рядками (транспонує).

```
a = [[1,2], [3,4], [5,6]]
a.transpose   #=> [[1, 3, 5], [2, 4, 6]]
```

Array#uniq

```
array.uniq   -> an_array
```

Повертає новий масив, що отриманий з масиву array шляхом видалення всіх повторюваних елементів (тобто залишаються тільки «uniq'альные елементи»).

```
a = [ "a", "a", "b", "b", "c" ]
a.uniq   #=> ["a", "b", "c"]
```

Array#uniq!

```
array.uniq! -> array або nil
```

Видаляє з масиву array всі повторювані елементи (тобто залишаються тільки «uniq'альные елементи»). Повертає nil, якщо повторювані елементи в масиві array були відсутні.

```
a = [ "a", "a", "b", "b", "c" ]
a.uniq!   #=> ["a", "b", "c"]
b = [ "a", "b", "c" ]
b.uniq!   #=> nil
```

Даний метод змінює вихідний масив. Будьте уважні! Щоб виключити подібного роду зміни використайте метод uniq

Array#unshift

```
array.unshift(obj, ...)  -> array
```

Додає елементи в початок масиву array (зі зрушенням вправо вже існуючих).

```
a = [ "b", "c", "d" ]
a.unshift("a")   #=> ["a", "b", "c", "d"]
a.unshift(1, 2)  #=> [ 1, 2, "a", "b", "c", "d"]
a                #=> [ 1, 2, "a", "b", "c", "d"]
```

Даний метод змінює вихідний масив. Будьте уважні! Щоб виключити подібного роду зміни використайте метод +

Array#values_at

```
array.values_at(selector,... )  -> an_array
```

Повертає масив, що складається з елементів масиву array, які відповідають переданим

селекторам. Селектори можуть бути цілими числами (індексами) або целочисленными діапазонами.

```
a = %w{ a b c d e f }
a.values_at(1, 3, 5)
a.values_at(1, 3, 5, 7)
a.values_at(-1, -3, -5, -7)
a.values_at(1..3, 2...5)
```

Для подібних цілей також використаються методи select й [] («батарейка»). От тільки в «батарейці» не можна вказувати кілька непослідовних індексів або діапазонів, а select відбирає елементи за умовою (а не по індексі)

Array#zip

```
array.zip(arg, ...)                      -> an_array
array.zip(arg, ...) {| arr | block }  -> nil
```

Перетворить аргументи в масиви (за допомогою методу to_a). Поєднує кожен елемент масиву array з відповідним масивом, переданим як аргумент. У результаті цього створюється двовимірний масив з array.size рядками й з n стовпцями, де n - максимальна з довжин аргументів-масивів. Якщо довжина якого або з аргументів-масивів менше n, то він розширюється до довжини n (доповнюється nil). Якщо задано блок, то одержувані масиви передаються в блок як параметр, інакше - повертається двовимірний масив.

```
a = [ 4, 5, 6 ]
b = [ 7, 8, 9 ]

 [1,2,3].zip(a, b)      #=> [[1, 4, 7], [2, 5, 8], [3, 6, 9]]
 [1,2].zip(a,b)         #=> [[1, 4, 7], [2, 5, 8]]
a.zip([1,2],[8])        #=> [[4,1,8], [5,2,nil], [6,nil,nil]]
```

Звичайно використається разом з методом assoc, тобто zip підготовляє для нього масив, по якому буде здійснюватися асоціація

Клас Hash

Хеш - колекція пар джерело-значення. Хеш подібний до класу Array, за винятком того, що індексація здійснюється через ключі (об'єкти будь-якого типу), а не через целочисленные індекси. Послідовність перерахування пара джерел-значень хеша може виявитися довільної, і звичайно не збігається з тієї, у якій ви заповнювали хеш. При звертанні до хешу по ключі, якого не існує, повертається значення по-умовчанню. Споконвічно, цим значенням є nil.

Домішки

`Enumerable (all?, any?, collect, detect, each_cons, each_slice, each_with_index, entries, enum_cons, enum_slice, enum_with_index, find, find_all, grep, group_by, include?, index_by, inject, map, max, member?, min, partition, reject, select, sort, sort_by, sum, to_a, to_set, zip)`

Методи класу

`[], new`

Методи об'єкта

```
[]=, [], ==, clear, default=, default_proc, default, delete_if, delete,
each_key, each_pair, each_value, each, empty?, fetch, has_key?, has_value?,
include?, indexes, index, indices, inspect, invert, key?, keys, length, member?,
merge!, merge, rehash, reject!, reject, replace, select, shift, size, sort,
store, to_a, to_hash, to_s, update, value?, values_at, values
```

Hash::[]

```
Hash[ [key =>|, value]* ]   #-> hash
```

Створює новий хеш, заповнений заданими об'єктами. Еквівалентно литералу { key, value, ... }.

```
Hash["a", 100, "b", 200]       #-> {"a"=>100, "b"=>200}
Hash["a" => 100, "b" => 200]   #-> {"a"=>100, "b"=>200}
{ "a" => 100, "b" => 200 }     #-> {"a"=>100, "b"=>200}
```

Ключі й значення складаються в парах, тому потрібне парне число аргументів

Hash::new

```
Hash.new                          #-> hash
Hash.new(obj)                     #-> aHash
Hash.new {|hash, key| block }     #-> aHash
```

Повертає новий хеш. При наступному звертанні до хешу по ключі, якого не існує в цьому хеше, що повертає значення залежить від форми виклику методу new. У першій формі виклику повернеться значення nil. Якщо зазначено об'єкт obj, то цей єдиний об'єкт буде використатися для всіх значень по-умовчанню. Якщо зазначено блок, тоді значення по-умовчанню обчислюється в даному блоці, якому передаються хеш (поточний) і ключ. У блоці можна записати значення в хеш, якщо це необхідно.

```
h = Hash.new("Go Fish")
h["a"] = 100
h["b"] = 200
h["a"]           #-> 100
h["c"]           #-> "Go Fish"
# Змінюється єдиний об'єкт по-умовчанню
h["c"].upcase!   #-> "GO FISH"
h["d"]           #-> "GO FISH"
h.keys           #-> ["a", "b"]

# Створюється новий об'єкт за замовчуванням щораз
h = Hash.new { |hash, key| hash[key] = "Go Fish: #{key}" }
h["c"]           #-> "Go Fish: c"
h["c"].upcase!   #-> "GO FISH: C"
h["d"]           #-> "Go Fish: d"
h.keys           #-> ["c", "d"]
```

Hash#==

```
hsh == other_hash    #-> true або false
```

Рівність - два хеша вважаються рівними, якщо вони містять однакове число ключів, і якщо кожна пара джерело-значення еквівалентн (відповідно до методу Object#==) відповідним елементам в іншому хеше.

```
h1 = { "a" => 1, "c" => 2 }
h2 = { 7 => 35, "c" => 2, "a" => 1 }
h3 = { "a" => 1, "c" => 2, 7 => 35 }
h4 = { "a" => 1, "d" => 2, "f" => 35 }
h1 == h2   #-> false
h2 == h3   #-> true
h3 == h4   #-> false
```

Hash#[]

```
hsh[key]    #->  value
```

Одержання елемента - повертає значення відповідному ключу key. Якщо ключа не існує, то повертається значення по-умовчанню (див. Hash::new).

```
h = { "a" => 100, "b" => 200 }
h["a"]   #-> 100
h["c"]   #-> nil
```

Hash#[]=

```
hsh[key] = value        #-> value
```

Присвоювання - асоціює значення value із ключем key.

```
h = { "a" => 100, "b" => 200 }
h["a"] = 9
h["c"] = 4
h   #-> {"a"=>9, "b"=>200, "c"=>4}
```

Даний метод і метод store — абсолютно ідентичні, тобто є іменами того самого методу

Hash#clear

```
hsh.clear #-> hsh
```

Видаляє все пари значення^-значення-джерело-значення з хеша hsh.

```
h = { "a" => 100, "b" => 200 }   #-> {"a"=>100, "b"=>200}
h.clear                          #-> {}
```

Hash#default

```
hsh.default(key=nil)   #-> obj
```

Повертає значення по-умовчанню, тобто значення, що буде повертати hsh[key], якщо ключа key не існує в хеше hsh.

```
h = Hash.new                        #-> {}
```

```
h.default                                   #-> nil
h.default(2)                                #-> nil

h = Hash.new("cat")                         #-> {}
h.default                                   #-> "cat"
h.default(2)                                #-> "cat"

h = Hash.new {|h,k| h[k] = k.to_i*10}    #-> {}
h.default                                   #-> 0
h.default(2)                                #-> 20
```

Корисно глянути також на методи Hash::new й Hash#default=

Hash#default=

```
hsh.default = obj      #-> hsh
```

Установлює значення по-умовчанню, тобто значення, що повертає hsh[key], якщо ключа key не існує в хеше hsh.

h = { "a" => 100, "b" => 200 }

```
h.default = "Go fish"
h["a"]     #-> 100
h["z"]     #-> "Go fish"

# Це не зробить того, на що ви надеятесь...
h.default = proc do |hash, key|
   hash[key] = key + key
end
h[2]       #-> #<Proc:0x401b3948@-:6>
h["cat"]   #-> #<Proc:0x401b3948@-:6>
```

Таким способом як значення по-умовчанню не можна встановити Proc, що буде виконуватися при кожному обігу по ключі

Hash#default_proc

```
hsh.default_proc  #-> anObject
```

Якщо метод Hash::new був викликаний із блоком, то повертає блок, інакше повертає nil.

```
h = Hash.new {|h,k| h[k] = k*k }   #-> {}
p = h.default_proc                 #-> #<Proc:0x401b3d08@-:1>
a = []                             #-> []
p.call(a, 2)
a                                  #-> [nil, nil, 4]
```

Hash#delete

```
hsh.delete(key)                    #-> value
hsh.delete(key) {| key | block }   #-> value
```

Видаляє пару значення^-значення-джерело-значення з хеша hsh, що відповідає ключу key.

Повертається значення, що відповідає ключу. Якщо ключ не був знайдений, тоді повертається "значення по-умовчанню". Якщо використається конструкція із блоком і ключ не був знайдений, то повертається результат виконання блоку block, якому передається ключ key.

```
h = { "a" => 100, "b" => 200 }
h.delete("a")                                #-> 100
h.delete("z")                                #-> nil
h.delete("z") { |el| "#{el} не знайдений" }   #-> "z не знайдений"
```

Hash#delete_if

```
hsh.delete_if {| key, value | block }  #-> hsh
```

Видаляє все пари значення^-значення-джерело-значення з хеша hsh для яких блок block обчислює значення true.

```
h = { "a" => 100, "b" => 200, "c" => 300 }
h.delete_if {|key, value| key >= "b" }   #-> {"a"=>100}
```

Hash#each

```
hsh.each {| key, value | block }   #-> hsh
```

Викликає блок для кожен пари значення^-значення-джерело-значення хеша hsh і передає в блок поточну пару значення^-значення-джерело-значення у вигляді масиву із двох елементів. У виді семантичних особливостей параметрів блоку, ці параметри можуть бути представлені не масивом, а двома елементами з різними іменами. Ще існує метод each_pair, що ледве більше ефективний для блоків із двома параметрами.

```
h = { "a" => 100, "b" => 200 }
h.each {|key, value| puts "#{key} => #{value}" }
результат:
 a => 100
 b => 200
```

Hash#each_key

```
hsh.each_key {| key | block } #-> hsh
```

Виконує блок block для кожного ключа в хеше hsh, передаючи в блок ключ key як параметр.

```
h = { "a" => 100, "b" => 200 }
h.each_key {|key| puts key }
```

результат:

a

b

Hash#each_pair

```
hsh.each_pair {| key_value_array | block } #-> hsh
```

Виконує блок block для кожного ключа в хеше hsh, передаючи в блок ключ і значення як параметри.

```
h = { "a" => 100, "b" => 200 }
h.each_pair {|key, value| puts "#{key} => #{value}" }
```

результат:

```
a => 100
b => 200
```

Hash#each_value

```
hsh.each_value {| value | block } #-> hsh
```

Виконує блок block для кожного ключа в хеше hsh, передаючи в блок значення value, що відповідає ключу, як параметр.

```
h = { "a" => 100, "b" => 200 }
h.each_value {|value| puts value }
```

результат:

100

200

Hash#empty?

```
hsh.empty?    #-> true або false
```

Повертає true, якщо хеш hsh не містить пара значення^-значення-джерело-значення зовсім.

{}.empty?   #-> true

Hash#fetch

```
hsh.fetch(key [, default] )       #-> obj
hsh.fetch(key) {| key | block }   #-> obj
```

Повертає значення, що відповідає ключу key. Якщо ключ не був знайдений, тоді є кілька ситуацій: Без інших аргументів буде підніматися виключення IndexError; Якщо задано параметр default, тоді він і буде повернутий; Якщо конструкція визначена із блоком, тоді буде виконуватися блок, якому як аргумент буде переданий ключ.

```
h = { "a" => 100, "b" => 200 }
h.fetch("a")                            #-> 100
h.fetch("z", "go fish")                 #-> "go fish"
h.fetch("z") { |el| "go fish, #{el}"}   #-> "go fish, z"
```

Наступний приклад показує, що якщо ключ не знайдений і значення по-умовчанню не поставляється, то піднімається виключення.

```
h = { "a" => 100, "b" => 200 }
h.fetch("z")
```

результат:

```
prog.rb:2:in 'fetch': key not found (IndexError)
 from prog.rb:2
```

Hash#has_key?

```
hsh.has_key?(key)    #-> true або false
```

Повертає true, якщо заданий ключ перебуває в хеше hsh.

```
h = { "a" => 100, "b" => 200 }
h.has_key?("a")   #-> true
h.has_key?("z")   #-> false
```

Методи has_key?, include?, key? і member? — абсолютно ідентичні, тобто є іменами того самого методу

Hash#has_value?

```
hsh.has_value?(value)    #-> true або false
```

Повертає true, якщо задане значення належить деякому ключу в хеше hsh.

```
h = { "a" => 100, "b" => 200 }
h.has_value?(100)   #-> true
h.has_value?(999)   #-> false
```

Методи has_value? і value? — абсолютно ідентичні, тобто є іменами того самого методу

Hash#include?

```
hsh.include?(key)    #-> true або false
```

Повертає true, якщо заданий ключ перебуває в хеше hsh.

```
h = { "a" => 100, "b" => 200 }
h.include?("a")   #-> true
h.include?("z")   #-> false
```

Методи include?, has_key?, key? і member? — абсолютно ідентичні, тобто є іменами того самого методу

Hash#index

```
hsh.index(value)    #-> key
```

Повертає ключ для заданого значення. Якщо значення не знайдене, повертає nil.

```
h = { "a" => 100, "b" => 200 }
h.index(200)   #-> "b"
h.index(999)   #-> nil
```

Hash#indexes

```
hsh.indexes(key, ...)    #-> array
```

Даний метод є «застарілої» і його використання засуджується розроблювачами. У наступних версіях мови він буде вилучена й ваша програма перестане працювати. Використайте метод select, що має схожий функціонал, але не є «застарілої»

Методи indexes й indices — абсолютно ідентичні, тобто є іменами того самого методу

Hash#indices

```
hsh.indices(key, ...)     #-> array
```

Даний метод є «застарілої» і його використання засуджується розроблювачами. У наступних версіях мови він буде вилучена й ваша програма перестане працювати. Використайте метод select, що має схожий функціонал, але не є «застарілої»

Методи indices й indexes — абсолютно ідентичні, тобто є іменами того самого методу

Hash#inspect

```
hsh.inspect  #-> string
```

Повертає вміст хеша у вигляді рядка.

Hash#invert

```
hsh.invert #-> aHash
```

Повертає новий хеш, створений шляхом використання значень хеша hsh як ключі, а ключів як значення.

```
h = { "n" => 100, "m" => 100, "y" => 300, "d" => 200, "a" => 0 }
h.invert   #-> {0=>"a", 100=>"n", 200=>"d", 300=>"y"}
```

Hash#key?

```
hsh.key?(key)     #-> true або false
```

Повертає true, якщо заданий ключ перебуває в хеше hsh.

```
h = { "a" => 100, "b" => 200 }
h.key?("a")    #-> true
h.key?("z")    #-> false
```

Методи key?, has_key?, include? і member? — абсолютно ідентичні, тобто є іменами

того самого методу

Hash#keys

```
hsh.keys    #-> array
```

Повертає новий масив, що складається із ключів даного хеша.

```
h = { "a" => 100, "b" => 200, "c" => 300, "d" => 400 }
h.keys   #-> ["a", "b", "c", "d"]
```

Корисно подивитися на метод values, що має схожу функціональність

Hash#length

```
hsh.length    #->  fixnum
```

Повертає кількість пар значення^-значення-джерело-значення в даному хеше.

```
h = { "d" => 100, "a" => 200, "v" => 300, "e" => 400 }
h.length        #-> 4
h.delete("a")   #-> 200
h.length        #-> 3
```

Методи length й size — абсолютно ідентичні, тобто є іменами того самого методу

Hash#member?

```
hsh.member?(key)    #-> true або false
```

Повертає true, якщо заданий ключ перебуває в хеше hsh.

```
h = { "a" => 100, "b" => 200 }
h.member?("a")   #-> true
h.member?("z")   #-> false
```

Методи member?, has_key?, include? і key? — абсолютно ідентичні, тобто є іменами того самого методу

Hash#merge

```
hsh.merge(other_hash)                                #-> a_hash
hsh.merge(other_hash){|key, oldval, newval| block}   #-> a_hash
```

Повертає новий хеш, що складається із умісту хешей other_hash й hsh. Якщо в результаті злиття виявляться однакові ключі, то для нього буде записане значення з хеша other_hash (якщо задано блок, то буде записане значення, що вийде в результаті виконання блоку).

```
h1 = { "a" => 100, "b" => 200 }
h2 = { "b" => 254, "c" => 300 }
```

```
h1.merge(h2)   #=> {"a"=>100, "b"=>254, "c"=>300}
h1             #=> {"a"=>100, "b"=>200}
```

Hash#merge!

```
hsh.merge!(other_hash)                                 #-> hsh
hsh.update(other_hash)                                 #-> hsh
hsh.merge!(other_hash){|key, oldval, newval| block}    #-> hsh
hsh.update(other_hash){|key, oldval, newval| block}    #-> hsh
```

Додає вміст хеша other_hash до хешу hsh. Якщо виявляться дублюючі ключі, то значення для нього буде взяте з other_hash (або отримано в результаті виконання блоку, якщо він заданий).

```
h1 = { "a" => 100, "b" => 200 }
h2 = { "b" => 254, "c" => 300 }
h1.merge!(h2)   #=> {"a"=>100, "b"=>254, "c"=>300}
```

Методи megre! і update — абсолютно ідентичні, тобто є іменами того самого методу

Даний метод є «небезпечним», тому що змінює вихідний об'єкт. Замість нього рекомендується використати метод merge, що не має даного побічного ефекту

Hash#rehash

```
hsh.rehash   #-> hsh
```

Якщо ключами хеша є змінні, то може виникнути ситуація, коли їхнє значення міняється. Щоб мати доступ до асоційованого з ними даним потрібно викликати даний метод, щоб він привів ключі у відповідність із новим значеним змінних. Якщо метод викликається, у той час як итератор обходить цей самий хеш, то буде порушена помилка виду IndexError.

```
a = [ "a", "b" ]
c = [ "c", "d" ]
h = { a => 100, c => 300 }
h[a]       #-> 100
a[0] = "z"
h[a]       #-> nil
h.rehash   #-> {["z", "b"]=>100, ["c", "d"]=>300}
h[a]       #-> 100
```

Hash#reject

```
hsh.reject {| key, value | block }  #-> a_hash
```

Те ж саме, що й метод delete_if, але обробляє (і повертає) копію хеша hsh. По суті, даний метод еквівалентний hsh.dup.delete_if.

Hash#reject!

```
hsh.reject! {| key, value | block }  #-> hsh або nil
```

Еквівалентно delete_if, але повертає nil, якщо хеш не був змінений у результаті роботи даного методу.

Hash#replace

```
hsh.replace(other_hash) #-> hsh
```

Заміняє вміст хеша hsh на вміст хеша other_hash.

```
h = { "a" => 100, "b" => 200 }
h.replace({ "c" => 300, "d" => 400 })   #-> {"c"=>300, "d"=>400}
```

Hash#select

hsh.select {|key, value| block}   #-> array

Повертає новий масив, що складається з пар [ключ, значення], для яких блок обчислює значення true.

h = { "a" => 100, "b" => 200, "c" => 300 }

h.select {|k,v| k > "a"} #-> [["b", 200], ["c", 300]]

h.select {|k,v| v < 200} #-> [["a", 100]]

Корисно подивитися на метод values_at, що має схожу функціональність

Hash#shift

hsh.shift #-> anArray або obj

Видаляє пару значення^-значення-джерело-значення з хеша hsh і повертає цю пару у вигляді масиву [key, value]. Якщо хеш порожній, то повертає значення по-умовчанню.

h = { 1 => "a", 2 => "b", 3 => "c" }

h.shift   #-> [1, "a"]

h   #-> {2=>"b", 3=>"c"}

Hash#size

hsh.size   #-> fixnum

Повертає кількість пар значення^-значення-джерело-значення в даному хеше.

h = { "d" => 100, "a" => 200, "v" => 300, "e" => 400 }

h.size   #-> 4

h.delete("a")   #-> 200

h.size   #-> 3

Методи size й length — абсолютно ідентичні, тобто є іменами того самого методу

Hash#sort

```
hsh.sort                        #-> array
hsh.sort {| a, b | block }  #-> array
```

Перетворить хеш hsh у масив масивів - [key, value] і сортує його, використовуючи Array#sort.

```
h = { "a" => 20, "b" => 30, "c" => 10  }
h.sort                          #-> [["a", 20], ["b", 30], ["c", 10]]
h.sort {|a,b| a[1]<=>b[1]}    #-> [["c", 10], ["a", 20], ["b", 30]]
```

Hash#store

```
hsh.store(key, value)   #-> value
```

Присвоювання - асоціює значення value із ключем key. Ідентичний методу []=.

```
h = { "a" => 100, "b" => 200 }
h.store("a", 9)
h.store("c", 4)
h   #-> {"a"=>9, "b"=>200, "c"=>4}
```

Hash#to_a

```
hsh.to_a #-> array
```

Конвертує хеш hsh у масив, що складається з масивів [key, value].

```
h = { "c" => 300, "a" => 100, "d" => 400, "c" => 300  }
h.to_a   #-> [["a", 100], ["c", 300], ["d", 400]]
```

Hash#to_hash

```
hsh.to_hash   #-> hsh
```

Повертає hsh.

Hash#to_s

```
hsh.to_s   #-> string
```

Перетворить хеш hsh у рядок шляхом перетворення хеша в масив масивів [key, value], і перетворення цього масиву в рядок, використовуючи Array#join зі стандартним роздільником.

```
h = { "c" => 300, "a" => 100, "d" => 400 }
```

```
h.to_s   #-> "a100c300d400"
```

Hash#update

```
hsh.merge!(other_hash)                                  #-> hsh
hsh.update(other_hash)                                  #-> hsh
hsh.merge!(other_hash){|key, oldval, newval| block}    #-> hsh
hsh.update(other_hash){|key, oldval, newval| block}    #-> hsh
```

Додає вміст хеша other_hash до хешу hsh. Якщо виявляться дублюючі ключі, то значення для нього буде взяте з other_hash (або отримано в результаті виконання блоку, якщо він заданий).

```
h1 = { "a" => 100, "b" => 200 }
h2 = { "b" => 254, "c" => 300 }
h1.update(h2)   #=> {"a"=>100, "b"=>254, "c"=>300}
```

Методи megre! і update — абсолютно ідентичні, тобто є іменами того самого методу

Даний метод є «небезпечним», тому що змінює вихідний об'єкт. Замість нього рекомендується використати метод merge, що не має даного побічного ефекту

Hash#value?

```
hsh.value?(value)    #-> true або false
```

Повертає true, якщо задане значення належить деякому ключу в хеше hsh.

```
h = { "a" => 100, "b" => 200 }
h.value?(100)   #-> true
h.value?(999)   #-> false
```

Методи value? і has_value? — абсолютно ідентичні, тобто є іменами того самого методу

Hash#values

```
hsh.values    #-> array
```

Повертає новий масив, що складається зі значень даного хеша.

```
h = { "a" => 100, "b" => 200, "c" => 300 }
h.values   #=> [100, 200, 300]
```

Корисно подивитися на метод keys, що має схожу функціональність

Hash#values_at

```
hsh.values_at(key, ...)   #-> array
```

Повертає масив утримуючого значення, що відповідають заданим ключам. (Див. Hash.select).

```
h = { "cat" => "feline", "dog" => "canine", "cow" => "bovine" }
```

```
h.values_at("cow", "cat")  #-> ["bovine", "feline"]
```

Клас Matrix

Клас Matrix забезпечує роботу з математичними матрицями. Зокрема, реалізує методи для створення матриць спеціального виду (нульового, одиничного, діагонального, сингулярні, вектора), арифметичні й алгебраїчні операції над ними, а також обчислення їхніх математичних властивостей (слід, ранг, детермінант, зворотну матрицю).

- Хоча матриці теоретично повинні бути прямокутними, то клас цього не відслідковує

- Детермінант може бути невірно обчислений, якщо не підключена бібліотека mathn (командою require 'mathn'). Ця проблема може бути виправлена в майбутньому

Каталог методів

Створення матриці:

- Matrix[*rows]
- Matrix.[](*rows)
- Matrix.rows(rows, copy = true)
- Matrix.columns(columns)
- Matrix.diagonal(*values)
- Matrix.scalar(n, value)
- Matrix.scalar(n, value)
- Matrix.identity(n)
- Matrix.unit(n)
- Matrix.I(n)
- Matrix.zero(n)
- Matrix.row_vector(row)
- Matrix.column_vector(column)

Доступ до елементів/стовпцям/рядкам/подматрицам матриці:

- [](i, j)
- #row_size
- #column_size
- #row(i)
- #column(j)
- #collect
- #map
- #minor(*param)

Властивості матриці:

- #regular?
- #singular?
- #square?

Матрична арифметика:

- *(m)
- +(m)
- -(m)
- #/(m)
- #inverse
- #inv
- **

Матричні функції:

- #determinant
- #det
- #rank
- #trace
- #tr
- #transpose
- #t

Перетворення в інші типи даних:

- #coerce(other)
- #row_vectors
- #column_vectors
- #to_a

Строкові подання:

- #to_s
- #inspect

Домішки

`ExceptionForMatrix ()`

Методи класу

`[], column_vector, columns, diagonal, identity, row_vector, rows, scalar, zero`

Методи об'єкта

```
[], **, *, +, -, /, ==, clone, coerce, collect, column_size, column_vectors,
column, compare_by_row_vectors, determinant, det, eql?, hash, inspect,
inverse_from, inverse, inv, map, minor, rank, regular?, row_size, row_vectors,
row, singular?, square?, to_a, to_s, trace, transpose, tr, t
```

Matrix::[]

```
Matrix::[](*rows)     #-> matrix
```

Створює матрицю, де кожний з переданих аргументів *rows інтерпретується як рядок (у змісті "ряд").

```
Matrix[[25, 93], [-1, 66]]   #-> Matrix[[25, 93], [-1, 66]]
```

Matrix::column_vector

```
Matrix::column_vector(column)   #-> matrix
```

Створює матрицю, що складається з одного стовпця, значення якого беруться з параметра column.

```
Matrix.column_vector([4, 5, 6])   #-> Matrix[[4], [5], [6]]
```

Matrix::columns

```
Matrix::columns(columns)
```

Створює матрицю, де кожен подмассив масиву columns інтерпретується як стовпець.

```
Matrix.columns([[25, 93], [-1, 66]])   #-> Matrix[[25, -1], [93, 66]]
```

Matrix::diagonal

Matrix::diagonal(*values)   #-> matrix

Створює матрицю, у якій передані аргументи *values розташовані по головній діагоналі.

```
Matrix.diagonal(9, 5, -3)   #-> Matrix[[9, 0, 0], [0, 5, 0], [0, 0, -3]]
 Matrix::identity
```

```
Matrix::identity(n)   #-> matrix
```

Створює одиничну матрицю розмірності n.

```
Matrix.identity(2)   #-> Matrix[[1, 0], [0, 1]]
```

Matrix::row_vector

```
Matrix::row_vector(row)   #-> matrix
```

Створює однорядкову матрицю, значення якої беруться з row.

```
Matrix.row_vector([4, 5, 6])   #-> Matrix[[4, 5, 6]]
```

Matrix::rows

```
Matrix::rows(rows, copy = true)   #-> matrix
```

Створює матрицю на підставі даних двовимірного масиву rows кожен подмассив якого інтерпретується як рядок матриці. Якщо опциональный параметр copy дорівнює false, то переданий масив rows буде використаний усередині стурктуры матриці без попереднього копіювання.

```
Matrix.rows([[25, 93], [-1, 66]])   #-> Matrix[[25, 93], [-1, 66]]
```

Matrix::scalar

```
Matrix::scalar(n, value)   #-> matrix
```

Створює матрицю розміром n на n, де кожен елемент головної діагоналі дорівнює value.
```
Matrix.scalar(2, 5)   #-> Matrix[[5, 0], [0, 5]]
```

Matrix::zero

```
Matrix::zero(n)   #-> matrix
```

Створює нульову матрицю розмірністю n на n.
```
Matrix.zero(2)   #-> Matrix[[0, 0], [0, 0]]
```

Matrix#*

```
m * other   #-> matrix
```

Матричне множення.
```
Matrix[[2, 4], [6, 8]] * Matrix.identity(2)   #-> Matrix[[2, 4], [6, 8]]
```

Matrix#**

```
m ** n   #-> matrix
```

Матричне піднесення в ступінь. Еквівалентно перемножуванню матриці саму на себе n-раз.
```
Matrix[[7,6], [3,9]] ** 2   #-> Matrix[[67, 96], [48, 99]]
```

Працює тільки для целочисленних ступенів n, тобто n повинен бути підкласом Integer

Matrix#+

```
m + other   #-> matrix
```

Матричне додавання.
```
Matrix.scalar(2, 5) + Matrix[[1, 0], [-4, 7]]   #-> Matrix[[6, 0], [-4, 12]]
```

Matrix#-

```
m - other   #-> matrix
```

Матричне вирахування.
```
Matrix[[1, 5], [4, 2]] - Matrix[[9, 3], [-4, 1]]   #-> Matrix[[-8, 2], [8, 1]]
```

Matrix#/

```
m / other   #-> matrix
```

Матричне ділення (тобто множення на зворотну матрицю).

```
Matrix[[7, 6], [3, 9]] / Matrix[[2, 9], [3, 1]]   #-> Matrix[[-7, 1], [-3, -6]]
```

Matrix#==

```
m == other   #-> matrix
```

Повертає true, якщо в матрицях m й other соотвествующие елементи рівні.

```
Matrix[[7,6], [3,9]] == Matrix[[7,6], [3,9]]      #-> true
Matrix[[14,12], [6,18]] == Matr ix[[7,6], [3,9]]   #-> false
```

Методи == й eql? — абсолютно ідентичні, тобто є іменами того самого методу

Matrix#[]

```
m[i,j]   #-> numeric
```

Повертає елемент матриці з індексами (i, j), де i - номер рядка, а j - номер стовпця.

Matrix#clone

```
m.clone   #-> matrix
```

Повертає дублікат матриці, такий, що кожний з елементів матриці також дублюється.

Matrix#coerce

```
m.coerce(other)   #-> array
```

Повертає масив, у якому other перетворений до класу Matrix::Scalar, а другий елемент - вихідна матриця m.

```
Matrix[[14,12], [6,18]].coerce 5
   #-> [#<Matrix::Scalar:0xc6d7048 @value=5>, Matrix[[14, 12], [6, 18]]]
Matrix[[14,12], [6,18]].coerce 12.2
   #-> [#<Matrix::Scalar:0xc6c9664 @value=12.2>, Matrix[[14, 12], [6, 18]]]
```

Matrix#collect

```
m.collect{|e| ... }   #-> matrix
```

Повертає матрицю, у якій всі елементи перетворені за правилом, описаному в блоці даного итератора.

```
Matrix[ [1,2], [3,4] ].collect { |i| i**2 }   #-> Matrix[[1, 4], [9, 16]]
```

Методи map й collect — абсолютно ідентичні, тобто є іменами того самого методу

Matrix#column

```
m.column(j)              #-> vector
m.column(j){|e| ... }    #-> 0
```

Повертає стовпець j матриці m у вигляді об'єкта класу Vector (нумерація стовпців починається з 0, як у масивах). Якщо використається виклик методу із блоком, то метод викликає блок для кожного елемента отриманого вектора й повертає 0 (що кричущо невірно й повинне бути виправлене).

Корисно подивитися на метод row, що має схожу функціональність

Matrix#column_size

```
m.column_size   #-> integer
```

Повертає число стовпців матриці m. Помітимо, що метод можна викликати для матриці з нерівною кількістю стовпців (тобто Matrix[[1,2,3], [4,5]]), але це буде математично невірно. Метод повертає кількість стовпців першого рядка (тобто з індексом 0) як результат.

```
Matrix[[1, 2, 3], [4, 5]].column_size   #-> 3
Matrix[[1, 2], [3, 4, 5]].column_size   #-> 2
```

Корисно подивитися на метод row_size, що має схожу функціональність

Matrix#column_vectors

```
m.column_vectors   #-> array
```

Повертає масив стовпців матриці у вигляді векторів.

Корисно подивитися на метод row_vectors, що має схожу функціональність

Matrix#compare_by_row_vectors

```
m.compare_by_row_vectors(rows)   #-> true або false
```

Використається усередині бібліотеки mathn.rb для виконання порівняння матриць (у методах == й eql?). Параметр rows -і це двовимірний масив елементів (який можна одержати за допомогою методу to_a).

```
m = Matrix[ [1,2], [3,4,5] ]
```

```
m.compare_by_row_vectors( [[25, 93], [-1, 66]] )    #-> false
m.compare_by_row_vectors( m.to_a )                  #-> true
m.compare_by_row_vectors( [[1, 2], [3, 4, 5]] )     #-> true
```

Автор методу, так і не зміг вирішити потрібно відкривати його для публічного чи користування ні

Matrix#det

```
m.det   #-> integer
```

Методи det й determinant — абсолютно ідентичні, тобто є іменами того самого методу

Matrix#determinant

```
m.determinant   #-> integer
```

Повертає детермінант матриці m. Якщо матриця m не є квадратної, то повертає 0.

```
Matrix[[7,6], [3,9]].determinant   #-> 63
```

Методи det й determinant — абсолютно ідентичні, тобто є іменами того самого методу

Matrix#eql?

```
m.eql?(other)   #-> true або false
```

Повертає true, якщо в матрицях m й other соотвествующие елементи рівні.

Matrix#hash

```
m.hash   #-> fixnum
```

Повертає контрольну суму для матриці m.

Matrix#inspect

```
m.inspect   #-> string
```

Перевизначає метод Object#inspect.

Matrix#inv

```
m.inv   #-> matrix
```

Методи inv й inverse — абсолютно ідентичні, тобто є іменами того самого методу

Matrix#inverse

```
m.inverse   #-> matrix
```

Повертає зворотну матрицю для матриці m.

```
Matrix[[1, 2], [2, 1]].inverse   #-> Matrix[[-1, 1], [0, -1]]
```

Методи inv — абсолютно ідентичні, тобто є іменами того самого методу

Matrix#inverse_from

```
m.inverse_from(src)   #-> matrix
```

Використається усередині бібліотеки matrix.rb для обчислення зворотної матриці. Але навіщо використати саме його, якщо є inv, що набагато зручніше (і коротше)?

```
m = Matrix[[1, 2], [2, 1]]
m.inv                                #-> Matrix[[-1, 1], [0, -1]]
Matrix.I(m.row_size).inverse_from(m)   #-> Matrix[[-1, 1], [0, -1]]
```

Корисно подивитися на методи inv й inverse, які мають схожу функціональність

Автор методу, так і не зміг вирішити потрібно відкривати його для публічного чи користування ні

Matrix#map

```
m.map{|e| ... }   #-> matrix
```

Методи map й collect — абсолютно ідентичні, тобто є іменами того самого методу

Matrix#minor

```
m.minor(*param)   #-> matrix
```

Повертає подматрицу матриці m, де координати задаються в такий спосіб:

- start_row, nrows, start_col, ncols (координата й відхилення по кожному вимірі)
- col_range, row_range (через діапазони)

```
Matrix.diagonal(9, 5, -3).minor(0..1, 0..2) #-> Matrix[[9, 0, 0], [0, 5, 0]]
```

Matrix#rank

```
m.rank   #-> integer
```

Повертає ранг матриці m. Отерегайтесь використати для матриць із дійсними числами, тому що через погрішність обчислень ранг може бути обчислений невірно. Краще використайте раціональні числа, якщо це можливо.

```
Matrix[[7,6], [3,9]].rank   #-> 2
```

Matrix#regular?

```
m.regular?   #-> true або false
```

Повертає true, якщо m є невырожденной матрицею.

```
Matrix[[1]].regular?           #-> true
Matrix[[1,2]].regular?         #-> false
Matrix[[1,2],[3,4]].regular?  #-> true
Matrix[[1,2],[3]].regular?    #-> NoMethodError
```

Корисно подивитися на метод singular?, що має схожу функціональність

Даний метод не працює з матрицями в які рядка мають різну довжину

Matrix#row

```
m.row(i)              #-> vector
m.row(i){|e| ... }    #-> 0
```

Повертає рядок i матриці m у вигляді об'єкта класу Vector (нумерація рядків починається з 0, як у масивах). Якщо використається виклик методу із блоком, то метод викликає блок для кожного елемента отриманого вектора й повертає 0 (що кричущо невірно й повинне бути виправлене).

Корисно подивитися на метод column, що має схожу функціональність

Matrix#row_size

```
m.row_size   #-> integer
```

Повертає кількість рядків матриці m.

Корисно подивитися на метод column_size, що має схожу функціональність

Matrix#row_vectors

```
m.row_vectors   #-> array
```

Повертає масив рядків матриці у вигляді векторів.

Корисно подивитися на метод column_vectors, що має схожу функціональність

Matrix#singular?

```
m.singular?   #-> true або false
```

Повертає true, якщо m є невырожденной матрицею.

```
Matrix[[1]].singular?           #-> false
Matrix[[1,2]].singular?         #-> true
Matrix[[1,2],[3,4]].singular?   #-> false
Matrix[[1,2],[3]].singular?     #-> NoMethodError
```

Корисно подивитися на метод regular?, що має схожу функціональність

Даний метод не працює з матрицями в які рядка мають різну довжину

Matrix#square?

```
m.square?   #-> true або false
```

Повертає true, якщо m є квадратною матрицею (тобто число рядків у якій рівняється кількості стовпців).

```
Matrix[[1]].square?           #-> true
Matrix[[1,2]].square?         #-> false
Matrix[[1,2],[3,4]].square?   #-> true
Matrix[[1,2],[3]].square?     #-> true
```

Даний метод видає помилкові дані для матриць, які маю рядка різної довжини. Це зв'язано зі специфікою роботи методу column_size

Matrix#t

```
m.t   #-> matrix
```

Методи t й transpose — абсолютно ідентичні, тобто є іменами того самого методу

Matrix#to_a

```
m.to_a   #-> array
```

Повертає двовимірний масив, що складається з елементів матриці m.

Самий корисний метод! Завдяки йому можна в будь-який момент перетворити матрицю до масиву й скористатися його великою бібліотекою методів

Matrix#to_s

```
m.to_s   #-> string
```

Перевизначає метод Object#to_s.

Matrix#tr

```
m.tr   #-> integer
```

Методи tr й trace — абсолютно ідентичні, тобто є іменами того самого методу

Matrix#trace

```
m.trace   #-> integer
```

Повертає суму діагональних елементів матриці m (так званий слід матриці).

```
Matrix[[7,6], [3,9]].trace   #-> 16
```

Методи tr й trace — абсолютно ідентичні, тобто є іменами того самого методу

Matrix#transpose

```
m.transpose   #-> matrix
```

Повертає результат транспонування матриці m.

```
m = Matrix[[1,2], [3,4], [5,6]]
m.transpose   #-> Matrix[[1, 3, 5], [2, 4, 6]]
```

Методи t й transpose — абсолютно ідентичні, тобто є іменами того самого методу

Клас Proc

Об'єкти Proc є блоками коду, які пов'язані з локальними змінними. Блок коду може бути виконаний в іншому контексті.

```
def gen_times(factor)
  return Proc.new {|n| n*factor }
end

times3 = gen_times(3)
times5 = gen_times(5)
```

```
times3.call(12)              #-> 36
times5.call(5)               #-> 25
times3.call(times5.call(4))  #-> 60
```

Методи класу

`new`

Методи об'єкта

`[], ==, arity, binding, call, clone, dup, to_proc, to_s`

Proc::new

```
Proc.new {|...| block } #-> a_proc
Proc.new                #-> a_proc
```

Створює новий об'єкт класу Proc і запам'ятовує в ньому поточний контекст. Proc::new може бути викликаний без блоку тільки в межах методу до якого причеплений блок (під час виклику). У цьому випадку блок буде перетворений в об'єкт класу Proc.

```
def proc_from
  Proc.new
end
proc = proc_from { "hello" }
proc.call   #-> "hello"
```

Proc#==

```
prc == other_proc   #->  true або false
```

Повертає true, якщо prc й other_proc -і- той самий об'єкт, або якщо обидва блоки мають однакове тіло.

Proc#[]

```
prc.call(params,...)   #-> obj
prc[params,...]        #-> obj
```

Виконує блок, привласнюючи параметрам блоку значення params й інших змінних, позначених троеточием. Видає попередження, якщо блок очікує лише одне значення, а йому передається більше (проте, він перетворить список параметрів у масив і спробує виконати блок). Для блоків, створюваних з використанням Kernel.proc, генерується помилка якщо число параметрів переданих у блок перевищує число параметрів оголошених під час його створення. Для блоків, створених за допомогою Proc.new, додаткові параметри просто відкидаються. Повертає значення останнього обчисленого вираження в блоці. Дивися ще Proc#yield.

```
a_proc = Proc.new {|a, *b| b.collect {|i| i*a }}
a_proc.call(9, 1, 2, 3)   #-> [9, 18, 27]
a_proc[9, 1, 2, 3]        #-> [9, 18, 27]
a_proc = Proc.new {|a,b| a}
a_proc.call(1,2,3)
```

результат:

```
prog.rb:5: wrong number of arguments (3 for 2) (ArgumentError)
 from prog.rb:4:in 'call'
 from prog.rb:5
```

(ще відомий как.call)

Proc#arity

```
prc.arity   #-> fixnum
```

Повертає кількість аргументів, які можуть бути сприйняті блоком. Якщо блок оголошений без вказівки аргументів, то повертає 0. Якщо число агрументов точно дорівнює n, то повертає n. Якщо блок має оціональний аргумент, то повертає -n-1, де n -i- кількість обов'язкових аргументів. Блок proc без аргументів звичайно містить || замість аргументів.

```
Proc.new {}.arity           #->  0
Proc.new {||}.arity         #->  0
Proc.new {|a|}.arity        #->  1
Proc.new {|a,b|}.arity      #->  2
Proc.new {|a,b,c|}.arity    #->  3
Proc.new {|*a|}.arity       #-> -1
Proc.new {|a,*b|}.arity     #-> -2
```

Proc#binding

```
prc.binding    #-> binding
```

Повертає об'єкт класу Binding асоційований з prc. Наприклад, Kernel#eval приймає об'єкти Proc або Binding як другий аргумент.

```
def fred(param)
  proc {}
end

b = fred(99)
eval("param", b.binding)   #-> 99
eval("param", b)           #-> 99
```

Proc#call

```
prc.call(params,...)   #-> obj
prc[params,...]        #-> obj
```

Виконує блок, привласнюючи параметрам блоку значення params й інших змінних, позначених троеточием. Видає попередження, якщо блок очікує лише одне значення, а йому передається більше (проте, він перетворить список параметрів у масив і спробує виконати блок). Для блоків, створюваних з використанням Kernel.proc, генерується помилка якщо число параметрів переданих у блок перевищує число параметрів оголошених під час його створення. Для блоків, створених за допомогою Proc.new, додаткові параметри просто відкидаються. Повертає значення останнього обчисленого вираження в блоці. Дивися ще Proc#yield.

```
a_proc = Proc.new {|a, *b| b.collect {|i| i*a }}
a_proc.call(9, 1, 2, 3)   #-> [9, 18, 27]
```

```
a_proc[9, 1, 2, 3]        #-> [9, 18, 27]
a_proc = Proc.new {|a,b| a}
a_proc.call(1,2,3)
результат:
prog.rb:5: wrong number of arguments (3 for 2) (ArgumentError)
 from prog.rb:4:in 'call'
 from prog.rb:5
```

(ще відомий як [])

Proc#clone

```
prc.clone   #-> other_proc
```

Створює копію блоку prc.

Proc#dup

```
prc.dup   #-> other_proc
```

Створює копію блоку prc.

Proc#to_proc

```
prc.to_proc   #-> prc
```

Частина сооглашения про преобразованиии інших об'єктів в об'єкти класу Proc. Усередині класу Proc він просто повертає сам себе.

Proc#to_s

```
prc.to_s   #-> string
```

Повертає рядок з унікальним ідентифікатором для prc, разом з покажчиком на місце, де блок був оголошений.

Клас Range

Об'єкти класу Range являють собою інтервал -і- множина значень між початком і кінцем інтервалу. Інтервали можу бути створені з використанням литералов s..e й s...e, або за допомогою методу Range::new. Інтервали, створені за допомогою .., ідуть із початку по кінець включно. Навпроти, інтервали, які створені за допомогою ... виключають останнє значення. Коли інтервали використаються в итераторах, вони повертають кожне значення із заданого діапазону.

```
(-1..-5).to_a      #-> []
(-5..-1).to_a      #-> [-5, -4, -3, -2, -1]
('a'..'e').to_a    #-> ["a", "b", "c", "d", "e"]
('a'...'e').to_a   #-> ["a", "b", "c", "d"]
```

Інтервали можуть бути створені з використанням об'єктів будь-якого типу, за умови, що вони порівнянні за допомогою оператора <=> і містять метод succ, що повертає наступний об'єкт послідовності.

```
class Xs                # represent a string of 'x's
  include Comparable
  attr :length
  def initialize(n)
    @length = n
  end
  def succ
    Xs.new(@length + 1)
  end
  def <=>(other)
    @length <=> other.length
  end
  def to_s
    sprintf "%2d #{inspect}", @length
  end
  def inspect
    'x' * @length
  end
end

r = Xs.new(3)..Xs.new(6)    #-> xxx..xxxxxx
r.to_a                      #-> [xxx, xxxx, xxxxx, xxxxxx]
r.member?(Xs.new(5))        #-> true
```

У попередньому прикладі, клас Xs підключає домішка Comparable. Тому метод Enumerable#member? має можливість використати оператор == для перевірки еквівалентності. Підключена домішка Comparable реалізує оператор==, але для його коректної роботи в класі Xs повинен бути реалізований оператор <=>.

Домішки

```
Enumerable (all?, any?, collect, detect, each_cons, each_slice, each_with_index,
entries, enum_cons, enum_slice, enum_with_index, find, find_all, grep, group_by,
include?, index_by, inject, map, max, member?, min, partition, reject, select,
sort, sort_by, sum, to_a, to_set, zip)
```

Методи класу

```
new
```

Методи об'єкта

```
===, ==, begin, each, end, eql?, exclude_end?, first, hash, include?, inspect,
last, member?, step, to_s
```

Range::new

```
Range.new(start, end, exclusive=false)    #-> range
```

Створює інтервал, використовуючи start й end. Якщо третій параметр пропущений або дорівнює false, то range буде включати кінцевий об'єкт (рівний end); інакше він буде виключений.

Range#==

```
rng == obj    #-> true or false
```

Повертає true тільки якщо obj є інтервалом, початковий і кінцевий об'єкт якого еквівалентні соотвествующим параметрам rng (рівняється за допомогою ==), і результат методу #exclude_end? такий же, як й в rng.

```
(0..2) == (0..2)            #-> true
(0..2) == Range.new(0,2)    #-> true
(0..2) == (0...2)           #-> false
```

Range#===

```
rng === obj        #->  true або false
rng.member?(val)   #->  true або false
rng.include?(val)  #->  true або false
```

Повертає true якщо obj є елементом rng, false інакше. Оператор порівняння === використається для зіставлення в конструкції case.

```
case 79
   when 1..50   then   print "низько\n"
   when 51..75  then   print "середньо\n"
   when 76..100 then   print "високо\n"
end
```

результат:

високо

(ще відомий як member?, include?)

Range#begin

```
rng.first    #-> obj
rng.begin    #-> obj
```

Повертає перший об'єкт із інтервалу rng.

(ще відомий як first)

Range#each

```
rng.each {| i | block } #-> rng
```

Перебирає елементи rng, які передаються кожну ітерацію усередину блоку. Ви можете здійснити перебір тільки в тому випадку, якщо початковий об'єкт підтримує метод succ (у такий спосіб перебір елементів з інтервалу, що складає з об'єктів Float неможливий).

```
(10..15).each do |n|
   print n, ' '
end
```

результат:

10 11 12 13 14 15

Range#end

```
rng.end    #-> obj
rng.last   #-> obj
```

Повертає елемент, що в rng оголошений як останній.

```
(1..10).end    #-> 10
(1...10).end   #-> 10
```

(ще відомий як last)

Range#eql?

```
rng.eql?(obj)    #-> true або false
```

Повертає true тільки якщо obj є інтервалом, початковий і кінцевий об'єкт якого еквівалентні соотвествующим параметрам rng (рівняється за допомогою #eql?), і результат методу #exclude_end? такий же, як й в rng.

```
(0..2) == (0..2)            #-> true
(0..2) == Range.new(0,2)    #-> true
(0..2) == (0...2)           #-> false
```

Range#exclude_end?

```
rng.exclude_end?    #-> true або false
```

Повертає true якщо rng не включає останній елемент (створений за допомогою ...).

Range#first

```
rng.first    #-> obj
rng.begin    #-> obj
```

Повертає перший об'єкт із інтервалу rng.

(ще відомий як begin)

Range#hash

```
rng.hash    #-> fixnum
```

Повертає контрольну суму, що для двох діапазонів з однаковими початковими й кінцевими об'єктами, а також з однаковими значеннями методу #exclude_end? -i- буде збігатися.

Range#include?

```
rng === obj        #->  true або false
rng.member?(val)   #->  true або false
rng.include?(val)  #->  true або false
```

Повертає true якщо obj є елементом rng, false інакше. Оператор порівняння === використається для зіставлення в конструкції case.

```
case 79
   when 1..50   then   print "низько\n"
   when 51..75  then   print "середньо\n"
   when 76..100 then   print "високо\n"
end
```

результат:

високо

(ще відомий як ===, member?)

Range#inspect

```
rng.inspect  #-> string
```

Перетворить инретвал rng у друковану форму (використає метод inspect для перетворення початкового й кінцевого об'єктів).

Range#last

```
rng.end    #-> obj
rng.last   #-> obj
```

Повертає елемент, що в rng оголошений як останній.

```
(1..10).end    #-> 10
(1...10).end   #-> 10
```

(ще відомий як end)

Range#member?

```
rng === obj        #->  true або false
rng.member?(val)   #->  true або false
rng.include?(val)  #->  true або false
```

Повертає true якщо obj є елементом rng, false інакше. Оператор порівняння === використається для зіставлення в конструкції case.

```
case 79
   when 1..50   then   print "низько\n"
   when 51..75  then   print "середньо\n"
   when 76..100 then   print "високо\n"
end
```

результат:

високо

(ще відомий як ===, include?)

Range#step

```
rng.step(n=1) {| obj | block }    #-> rng
```

Перебирає елементи з діапазону rng, передаючи кожен n-ий елемент у блок. Якщо діапазон складається із цілих чисел або рядків, то елементи обчислюються целочисленным діленням. Інакше step використає метод succ для перебору элментов. Наступний код використає клас Xs, що використався нами раніше (див. Range).

```
range = Xs.new(1)..Xs.new(10)
range.step(2) {|x| puts x}
range.step(3) {|x| puts x}
результат:
1 x
3 xxx
5 xxxxx
7 xxxxxxx
9 xxxxxxxxx
1 x
4 xxxx
7 xxxxxxx
10 xxxxxxxxxx
```

Range#to_s

```
rng.to_s   #-> string
```

Перетворить інтервал rng у друковану форму.

Клас String

Рядка зберігають і маніпулюють довільними послідовностями байт (звичайно це символи). Рядки можуть бути створені за допомогою методу String::new або як литералы. Для запобігання трудноуловимых помилок у коді, необхідно знати методи, які міняють вихідний рядок (а не створюють нову). Звичайно імена цих методів закінчуються на !. Якщо ім'я методу не закінчується на !, те даний метод створює новий рядок, а не модифікує вихідну. Правда, як й у будь-якого правила, є виключення. Наприклад, метод String#[]=.

Домішки

```
Comparable (<, <=, ==, >, >=, between?),
Enumerable (all?, any?, collect, detect, each_cons, each_slice, each_with_index,
entries, enum_cons, enum_slice, enum_with_index, find, find_all, grep, group_by,
include?, index_by, inject, map, max, member?, min, partition, reject, select,
sort, sort_by, sum, to_a, to_set, zip)
```

Константи

```
DeletePatternCache, HashCache, PATTERN_EUC, PATTERN_SJIS, PATTERN_UTF8, RE_EUC,
RE_SJIS, RE_UTF8, SUCC, SqueezePatternCache, TrPatternCache
```

Методи класу

```
new
```

Методи об'єкта

```
[]=, [], %, *, +, <<, <=>, ==, =~, capitalize!, capitalize, casecmp, center,
chomp!, chomp, chop!, chop, concat, count, crypt, delete!, delete, downcase!,
downcase, dump, each_byte, each_char, each_line, each, empty?, eql?, gsub!,
gsub, hash, hex, include?, index, insert, inspect, intern, length, ljust,
lstrip!, lstrip, match, next!, next, nstrip, oct, replace, reverse!, reverse,
rindex, rjust, rstrip!, rstrip, scanf, scan, size, slice!, slice, split,
squeeze!, squeeze, strip!, strip, sub!, sub, succ!, succ, sum, swapcase!,
swapcase, to_blob, to_f, to_i, to_str, to_sym, to_s, tr!, tr_s!, tr_s, tr,
unpack, upcase!, upcase, upto
```

String::new

```
String.new(str="")   #-> new_str
```

Повертає новий рядок, що містить копію рядка str, переданої як параметр.

String#%

```
str % arg   #-> new_str
```

Форматування рядка. У керуючий рядок str замість специфікаторів перетворення підставляються дані з arg. Якщо в рядку str зазначено кілька специфікаторів перетворення, то arg повинен бути масивом.

```
" %05d" % 123                     #-> "00123"
" %-5s: %08x" % [ "ID", self.id ]   #-> "ID   : 200e14d6"
```

Опис формату керуючого рядка й специфікаторів можна подивитися в описі методу Kernel::sprintf

String#*

```
str * integer  #-> new_str
```

Повторення - повертає новий рядок, що складається з integer-копій рядка str.

```
"Ho! " * 3   #-> "Ho! Ho! Ho! "
```

String#+

```
str + other_str   #-> new_str
```

Зчеплення - повертає новий рядок, що складається з рядків str й other_str.

```
"Hello from " + self.to_s   #-> "Hello from main"
```

Корисно подивитися на методи << й concat, які мають схожу функціональність

String#<<

```
str << fixnum         #-> str
str.concat(fixnum)    #-> str
str << obj            #-> str
str.concat(obj)       #-> str
```

Додавання - приєднує аргумент до str. Якщо аргумент типу Fixnum у діапазоні від 0 до 255, то він перед приєднанням конвертується в символ.

```
a = "hello ";
a << "world"   #-> "hello world"
a << 33        #-> "hello world!"
```

Методи << й concat — абсолютно ідентичні, тобто є іменами того самого методу

String#<=>

```
str <=> other_str   #-> -1, 0, +1
```

Порівняння - повертає -1, якщо other_str менше str; повертає 0, якщо other_str й str рівні; повертає +1, якщо other_str більше str. Якщо рядки розрізняються по довжині, але еквівалентні на довжині самої короткої із двох, то більшої вважається та, котра длиннее. Якщо змінна $= дорівнює false, то порівняння базується на порівнянні двійкових значень кожного символу в рядку. У старих версіях Рубай, зміною $= можна було домогтися регистронезависимого порівняння, але тепер цю функцію виконує метод casecmp, що і рекомендується використати для цих цілей.

```
"abcdef" <=> "abcde"     #-> 1
"abcdef" <=> "abcdef"    #-> 0
"abcdef" <=> "abcdefg"   #-> -1
"abcdef" <=> "ABCDEF"    #-> 1
```

Корисно подивитися на метод casecmp, що має схожу функціональність

Метод <=> є основою для таких методів, як: <, <=, >, >= й between?, які підключаються разом з модулем Comparable

String#==

```
str == obj   #-> true або false
```

Еквівалентність - якщо рядка str й obj не збігаються, то повертається false. Повертається true тільки у випадку, коли код str <=> obj повертає 0.

String#=~

```
str =~ obj   #-> fixnum або nil
```

Зіставлення із шаблоном - якщо obj є правилом, то він використається як шаблон для зіставлення з str і повертає позицію з якої знайдене соспоставление або nil, якщо такого зіставлення знайти не вдалося. Інакше, відбувається виклик методу obj.=~, якому передається str як аргумент. За замовчуванням, метод =~ у класі Object повертає false.

```
"cat o' 9 tails" =~ /\d/   #-> 7
"cat o' 9 tails" =~ 9      #-> false
```

Корисно подивитися на метод match, що має схожу функціональність

String#[]

```
str[index]                  #-> fixnum або nil
str[start, length]          #-> new_str або nil
str[range]                  #-> new_str або nil
str[regexp]                 #-> new_str або nil
str[regexp, index]          #-> new_str або nil
str[other_str]              #-> new_str або nil
str.slice(index)            #-> fixnum або nil
str.slice(start, length)    #-> new_str або nil
str.slice(range)            #-> new_str або nil
str.slice(regexp)           #-> new_str або nil
str.slice(regexp, index)    #-> new_str або nil
str.slice(other_str)        #-> new_str або nil
```

Одержання подстроки або символу - повертає код символу з індексом index, або подстроку довжини length, починаючи з індексу start, або подстроку, що розташовується в діапазоні range. У всіх цих варіантах виклику негативна індексація має на увазі відлік з кінця рядка str. Повертається nil, якщо індекс index виходить за межі припустимого діапазону, розмір length запитуваної подстроки негативний або початок діапазону range попадає після кінця рядка str. Якщо передається правило regexp, то повертається фрагмент рядка str, що задовольняє правилу regexp (або nil, якщо такого збігу знайти не вдалося). Якщо задано додатковий параметр index, то повертається вміст угруповання правила з номером index (якщо index дорівнює 0, то повертається фрагмент рядка, що задовольняє правилу regexp). Якщо як аргумент передається рядок other_str, то повертається рядок other_str, якщо вона є подстрокой рядка str (або nil, інакше).

```
a = "hello there"
a[1]                    #-> 101
a[1,3]                  #-> "ell"
a[1..3]                 #-> "ell"
a[-3,2]                 #-> "er"
a[-4..-2]               #-> "her"
a[12..-1]               #-> nil
a[-2..-4]               #-> ""
a[/[aeiou](.)\1/]       #-> "ell"
a[/[aeiou](.)\1/, 0]    #-> "ell"
a[/[aeiou](.)\1/, 1]    #-> "l"
a[/[aeiou](.)\1/, 2]    #-> nil
a["lo"]                 #-> "lo"
a["bye"]                #-> nil
```

Методи slice й [] («батарейка») — абсолютно ідентичні!

String#[]=

```
str[index] = fixnum
str[index] = new_str
str[start, length] = new_str
str[range] = new_str
str[regexp] = new_str
str[regexp, index] = new_str
str[other_str] = new_str
```

Присвоювання значення елементу - заміняє частина або весь уміст рядка str. Усередині квадратних дужок використаються ті ж самі параметри, що й в [] («батарейка»). Якщо замінний рядок не збігається по розмірі із що заміняє, то відбувається соотвествующая адаптація. Якщо правило або рядок або індекс не дозволяють визначити позицію для заміни, то виникає помилка IndexError. Якщо використається форма виклику з regexp й index, то відбувається заміна подстроки, що збігається угрупованням в regexp з номером index. Форми виклику, яким передається index або start можуть викликати помилку IndexError, якщо їхнє значення виходить за межі індексації рядка; форма виклику з range може викликати помилку RangeError, а форма виклику з regexp або other_str тихо реагують на можливі помилки (відсутність збіг із правилом або рядком).

String#capitalize

```
str.capitalize   #-> new_str
```

Повертає копію рядка str у якій перший символ перетвориться у верхній регістр, а інші - у нижній.

```
"hello".capitalize    #-> "Hello"
"HELLO".capitalize    #-> "Hello"
"123ABC".capitalize   #-> "123abc"
```

Корисно подивитися на методи upcase, downcase й swapcase, які мають схожу функціональність

У всіх символів, крім латиниці, на ці зміни стійкий імунітет. Це не виходить, що буде виникати помилка. Просто, вони не будуть перетворені за вищеописаним правилом

String#capitalize!

```
str.capitalize!   #-> str або nil
```

Модифікує рядок str за правилом: перший символ перетвориться у верхній регістр, а інші - у нижній. Повертає nil, якщо зміни не потрібні.

```
a = "hello"
a.capitalize!   #-> "Hello"
a               #-> "Hello"
a.capitalize!   #-> nil
```

Корисно подивитися на методи upcase!, downcase! і swapcase!, які мають схожу

функціональність

Даний метод є «небезпечним», тому що змінює вихідний об'єкт. Замість нього рекомендується використати метод capitalize, що не має даного побічного ефекту

У всіх символів, крім латиниці на ці зміни стійкий імунітет. Це не виходить, що буде виникати помилка. Просто, вони не будуть перетворені за вищеописаним правилом

String#casecmp

```
str.casecmp(other_str)   #-> -1, 0, +1
```

Регистронезависимая версія методу <=>.

```
"abcdef".casecmp("abcde")     #-> 1
"aBcDe".casecmp("abcdef")    #-> 0
"abcdef".casecmp("abcdefg")   #-> -1
"abcdef".casecmp("ABCDEF")    #-> 0
```

String#center

```
str.center(integer, padstr)   #-> new_str
```

Якщо integer більше, ніж str.length, те повертає новий рядок, довжина якої дорівнює integer, рядок str розташовується посередине, оббита символами рядка padstr; інакше, повертає str.

```
"hello".center(4)         #-> "hello"
"hello".center(20)        #-> "       hello        "
"hello".center(20, '123') #-> "1231231hello12312312"
```

Корисно подивитися на методи ljust й rjust, які мають схожу функціональність

String#chomp

```
str.chomp(separator=$/)   #-> new_str
```

Повертає новий рядок, що є копією рядка str у якій вилучений останній символ separator. Якщо системна змінна $/ не була змінена й не передається параметр separator, то метод chomp видалить завершальні символи рядка (такі як \n, \r й \r\n).

```
"hello".chomp            #-> "hello"
"hello\n".chomp          #-> "hello"
"hello\r\n".chomp        #-> "hello"
"hello\n\r".chomp        #-> "hello\n"
"hello\r".chomp          #-> "hello"
"hello \n there".chomp   #-> "hello \n there"
"hello".chomp("llo")     #-> "he"
```

Корисно подивитися на метод chop, що має схожу функціональність

String#chomp!

```
str.chomp!(separator=$/)    #-> str або nil
```

Модифікує рядок str по алгоритму, описаному в методі chomp. Повертає оброблений рядок str або nil, якщо зміни не потрібні.

```
"hello".chomp!              #-> nil
"hello\n".chomp!            #-> "hello"
"hello".chomp!("llo")       #-> "he"
```

Корисно подивитися на метод chop!, що має схожу функціональність

Даний метод є «небезпечним», тому що змінює вихідний об'єкт. Замість нього рекомендується використати метод chomp, що не має даного побічного ефекту

String#chop

```
str.chop   #-> new_str
```

Повертає копію рядка str з якої вилучений останній символ. Якщо рядок закінчується комбінацією символів \r\n, то буде вилучена вся комбінація. Виклик методу chop від порожнього рядка повертає порожній рядок.

```
"string\r\n".chop   #-> "string"
"string\n\r".chop   #-> "string\n"
"".chop             #-> ""
"string".chop       #-> "strin"
"x".chop.chop       #-> ""
```

Як альтернатива методу chop краще використати метод chomp, що видаляє тільки символи перекладу рядка, а не просто останній символ. Це дозволить уникнути багатьох трудновыявляемых помилок у майбутньому

String#chop!

```
str.chop!   #-> new_str або nil
```

Модифікує рядок str по алгоритму, описаному в методі chop. Повертає оброблений рядок str або nil, якщо зміни не потрібні (наприклад, якщо метод chop! був викликаний від порожнього рядка).

```
"string\r\n".chop!   #-> "string"
"string\n\r".chop!   #-> "string\n"
"".chop!             #-> nil
"string".chop!       #-> "strin"
"x".chop.chop!       #-> ""
```

Як альтернатива методу chop! краще використати метод chomp!, що видаляє тільки символи перекладу рядка, а не просто останній символ. Це дозволить уникнути багатьох трудновыявляемых помилок у майбутньому

Даний метод є «небезпечним», тому що змінює вихідний об'єкт. Замість нього рекомендується використати метод chop, що не має даного побічного ефекту

String#concat

```
str << fixnum         #-> str
str.concat(fixnum)    #-> str
str << obj            #-> str
str.concat(obj)       #-> str
```

Додавання - приєднує аргумент до str. Якщо аргумент типу Fixnum у діапазоні від 0 до 255, то він перед приєднанням конвертується в символ.

```
a = "hello ";
a.concat("world")   #-> "hello world"
a.concat(33)        #-> "hello world!"
```

Методи << й concat — абсолютно ідентичні, тобто є іменами того самого методу

String#count

```
str.count(other_str*)   #-> fixnum
```

Кожен параметр other_str перетвориться в множину символів. Метод підраховує кількість символів str, які належать цій множині. За допомогою символу "галочка" (^) задаються виключення із множини. Вираження типу c1-c2 задають множина символів, які розташовуються між символами c1 й c2.

```
a = "hello world"
a.count "lo"            #-> 5
a.count "lo", "o"       #-> 2
a.count "hello", "^l"   #-> 4
a.count "ej-m"          #-> 4
```

String#crypt

```
str.crypt(salt_str)   #-> new_str
```

Застосовує однопрохідне криптографічне хеширование до рядка str за допомогою функції crypt зі стандартної бібліотеки мови Си. Аргументом методу є рядок salt_str, що містить "шумові" символи. Вона повинна бути длиннее двох символів, кожний з яких повинен належати множині [a-zA-Z0-9./].

String#delete

```
str.delete(other_str*)   #-> new_str
```

Повертає копію рядка str, у якій вилучені всі символи, передані параметром.

```
"hello".delete "l","lo"        #-> "heo"
"hello".delete "lo"            #-> "he"
"hello".delete "aeiou", "^e"   #-> "hell"
"hello".delete "ej-m"          #-> "ho"
```

Використає ті ж самі правила створення множин символів, що й у методі count

String#delete!

```
str.delete!(other_str*)   #-> str або nil
```

Видаляє з рядка str всі символи, передані параметром. Повертає модифікований рядок str або nil, якщо рядок str не була модифікована.

```
str="hello"
str.delete! "l","lo"        #-> "heo"
str                         #-> "heo"
str.delete! "l","lo"        #-> nil
```

Використає ті ж самі правила створення множин символів, що й у методі count

Даний метод є «небезпечним», тому що змінює вихідний об'єкт. Замість нього рекомендується використати метод delete, що не має даного побічного ефекту

String#downcase

```
str.downcase   #-> new_str
```

Повертає копію рядка str у якій всі символи вернего регістра замінені на відповідні символи нижнього.

```
"hEll".downcase   #-> "hello"
```

Корисно подивитися на методи capitalize, upcase й swapcase, які мають схожу функціональність

У всіх символів, крім латиниці на ці зміни стійкий імунітет. Це не виходить, що відбудеться помилка. Просто, вони не будуть перетворені за вищеописаним правилом

String#downcase!

```
str.downcase!   #-> str або nil
```

Модифікує рядок str за правилом: всі символи верхнього регістра перетворюють у відповідні символи нижнього. Повертає nil, якщо зміни не потрібні.

```
str="hEll"
str.downcase! #-> "hello"
str           #-> "hello"
str.downcase! #-> nil
```

Корисно подивитися методи capitalize!, upcase! і swapcase!, які також роблять перетворення регістра

- Даний метод змінює вихідний рядок. Будьте уважні! Щоб виключити подібного роду зміни використайте метод downcase
- У всіх символів, крім латиниці на ці зміни стійкий імунітет. Це не виходить, що буде виникати помилка... просто вони не будуть перетворені за вищеописаним правилом

String#dump

```
str.dump   #-> new_str
```

Створює версію рядка str у якій всі недруковані символи замінені на \nnn нотацію й всі специциальные символи екрановані.

```
"Hello world!\n".dump   #-> "\"Hello world!\\n\""
```

String#each

```
str.each(separator=$/) {|substr| block }        #-> str
str.each_line(separator=$/) {|substr| block }   #-> str
```

Розбиває рядок str використовуючи значення параметра separator (за замовчуванням separator=$/), передаючи кожну з отриманих подстрок як параметр блоку. Якщо як параметр separator передається порожній рядок, то рядок буде ділиться по символі \n, крім випадку, коли кілька символів \n ідуть підряд (всі символи \n будуть зараховуватися як один).

```
print "Example one\n"
"hello\nworld".each {|s| p s}
print "Example two\n"
"hello\nworld".each('l') {|s| p s}
print "Example three\n"
"hello\n\n\nworld".each('') {|s| p s}
```

результат:

```
 Example one
 "hello\n"
 "world"
 Example two
 "hel"
 "l"
 "o\nworl"
 "d"
 Example three
 "hello\n\n\n"
 "world"
```

- Зверніть увагу, що роздільник separator не віддаляється (на відміну від результату роботи методу split), а є присутнім у результаті
- Як результат итератор повертає вихідний рядок str

- Итераторы each й each_line - абсолютно ідентичні!

String#each_byte

```
str.each_byte {|fixnum| block }    #-> str
```

Передає кожен байт рядка str у блок.

```
"hello".each_byte {|c| print c, ' ' }
```

результат:

```
104 101 108 108 111
```

Як результат итератор повертає вихідний рядок str

String#each_char

```
str.each_char {|char| block }    #-> str
```

Передає кожен символ рядка str у блок.

```
"hello".each_char  {|c| print c, ' ' }
```

результат:

```
h e l l o
```

Як результат итератор повертає вихідний рядок str

String#each_line

```
str.each(separator=$/) {|substr| block }        #-> str
str.each_line(separator=$/) {|substr| block }   #-> str
```

Розбиває рядок str використовуючи значення параметра separator (за замовчуванням separator=$/), передаючи кожну з отриманих подстрок як параметр блоку. Якщо як параметр separator передається порожній рядок, то рядок буде ділиться по символі \n, крім випадку, коли кілька символів \n ідуть підряд (всі символи \n будуть зараховуватися як один).

```
print "Example one\n"
"hello\nworld".each_line{|s| p s}
print "Example two\n"
"hello\nworld".each_line('l'){|s| p s}
print "Example three\n"
"hello\n\n\nworld".each_line(''){|s| p s}
```

результат:

```
Example one
"hello\n"
"world"
Example two
"hel"
"l"
"o\nworl"
"d"
```

```
Example three
"hello\n\n\n"
"world"
```

- Зверніть увагу, що роздільник separator не віддаляється (на відміну від результату роботи методу split), а є присутнім у результаті
- Як результат итератор повертає вихідний рядок str
- Итераторы each й each_line - абсолютно ідентичні!

String#empty?

```
str.empty?   #-> true або false
```

Повертає true якщо рядок str має нульову довжину (тобто, якщо рядок str - порожня).

```
"hello".empty?   #-> false
"".empty?        #-> true
```

String#eql?

```
str.eql?(other_str)   #-> true або false
```

Два рядки називаються еквівалентними, якщо вони мають однаковий уміст і довжину.

String#gsub

```
str.gsub(pattern, replacement)       #-> new_str
str.gsub(pattern) {|match| block }   #-> new_str
```

Повертає копію рядка str, де всі збіги із шаблоном (або рядком) pattern замінені на рядок replacement або результат виконання блоку (якому параметром передається результат збігу).

У результаті роботи методу, знайдене збіг із шаблоном pattern записується в спеціальну змінну $& (спадщина мови Perl). У рядку replacement можливе використання послідовностей виду \1, \2 і так далі до \9, які є посиланнями на збіги з угрупованнями (номер угруповання вважається ліворуч праворуч). Усередині блоку на угруповання можна посилатися за допомогою спеціальних змінних виду $1, $2 і так далі до $9. Також, вираженню в блоці доступні спеціальні змінні $', $& й $', які дозволяють одержати доступ до подстроке до збігу, збігу й подстроке після збігу, соотвественно.

```
"hello".gsub(/[aeiou]/, '*')              #-> "h*ll*"
"hello".gsub(/([aeiou])/, '<\1>')         #-> "h<e>ll<o>"
"hello".gsub(/./) {|s| s[0].to_s + ' '}   #-> "104 101 108 108 111 "
```

Корисно подивитися на метод sub, що має схожу функціональність

String#gsub!

```
str.gsub!(pattern, replacement)        #-> str або nil
str.gsub!(pattern) {|match| block }    #-> str або nil
```

Виконує заміни подібно gsub, але змінює вихідний рядок str. Повертає результат замін або nil, якщо заміни неможливі.

Корисно подивитися на метод sub!, що має схожу функціональність

Даний метод є «небезпечним», тому що змінює вихідний об'єкт. Замість нього рекомендується використати метод gsub, що не має даного побічного ефекту

String#hash

```
str.hash   #-> fixnum
```

Обчислює хеш-код для рядка str. Два рядки з тим самим умістом будуть мати однаковий хеш-код (саме його використає метод eql?).

```
"rubynovich".hash   #-> 958124529
```

String#hex

```
str.hex   #-> integer
```

Трактує рядок str як рядок шестнадцатиричных цифр (з необов'язковою вказівкою знака або необятельным префіксом 0x) і повертає соотвествующее число. Якщо перетворення не вдається, то повертає нуль.

```
"0x0a".hex     #-> 10
"-1234".hex    #-> -4660
"0".hex        #-> 0
"wombat".hex   #-> 0
```

- Замість даного методу рекомендується використати метод to_i, що є універсальною заміною методів hex й oct
- Корисно глянути на метод oct, що має схожий функціонал

String#include?

```
str.include? other_str   #-> true або false
str.include? fixnum      #-> true або false
```

Повертає true, якщо рядок str містить передані параметром рядок other_str або символ fixnum.

```
"hello".include? "lo"   #-> true
"hello".include? "ol"   #-> false
"hello".include? ?h     #-> true
```

String#index

```
str.index(substring [, offset])   #-> fixnum або nil
str.index(fixnum [, offset])      #-> fixnum або nil
str.index(regexp [, offset])      #-> fixnum або nil
```

Повертає індекс першого входження переданої параметром подстроки (substring), символу (fixnum) або збігу із правилом (regexp) у рядку str. Повертає nil, якщо входження немає. Як другий параметр передається індекс (offset), з якого варто починати пошук входжень.

```
"hello".index('e')             #-> 1
"hello".index('lo')            #-> 3
"hello".index('a')             #-> nil
"hello".index(101)             #-> 1
"hello".index(/[aeiou]/, -3)   #-> 4
```

Корисно також глянути на метод rindex, що має подібний функціонал

String#insert

```
str.insert(index, other_str)   #-> str
```

Вставляє рядок other_str після зазначеного індексу index у рядок str. Негативна індексація має на увазі нумерацію з кінця рядка.

```
"abcd".insert(0, 'X')    #-> "Xabcd"
"abcd".insert(3, 'X')    #-> "abcXd"
"abcd".insert(4, 'X')    #-> "abcd"
"abcd".insert(-3, 'X')   #-> "abXcd"
"abcd".insert(-1, 'X')   #-> "abcd"
```

String#inspect

```
str.inspect   #-> string
```

Повертає друковану версію рядка str, у якій всі спеціальні символи екрановані.

```
str = "hello"
str[3] = 8
str.inspect       #-> "hel\010o"
```

String#intern

```
str.intern   #-> symbol
str.to_sym   #-> symbol
```

Повертає об'єкт класу Symbol, що соотвествует рядку str.

```
"Koala".intern         #-> :Koala
s = 'cat'.intern       #-> :cat
s == :cat              #-> true
```

```
s = '@cat'.intern       #-> :@cat
s == :@cat              #-> true
```

Цей метод може бути використаний для створення символів, які не можуть бути створені в :xxx нотації.

```
'cat and dog'.intern   #-> :"cat and dog"
```

- Корисно також глянути на метод Symbol#id2name, що здійснює зворотне перетворення

- Методи intern й to_sym - абсолютно ідентичні, але використання методу to_sym переважніше

String#length

```
str.length   #-> integer
str.size     #-> integer
```

Повертає розмір рядка str.

```
string = "54321"
string.length   #-> 5
```

Методи size й length — абсолютно ідентичні, тобто є іменами того самого методу

String#ljust

```
str.ljust(integer, padstr=' ')   #-> new_str
```

Якщо параметр integer більше, ніж довжина рядка str, то повертається новий рядок довжини integer з рядком str, що вирівняна по лівому краї, а виникле порожнє місце заповнюється символами padstr; інакше, повертається рядок str.

```
"hello".ljust(4)            #-> "hello"
"hello".ljust(20)           #-> "hello               "
"hello".ljust(20, '1234')   #-> "hello123412341234123"
```

Корисно подивитися на методи rjust й center, які мають схожу функціональність

String#lstrip

```
str.lstrip   #-> new_str
```

Повертає копію рядка str, у якій вилучені всі провідні пробельные символи.

```
"  hello  ".lstrip   #-> "hello  "
"hello".lstrip       #-> "hello"
```

Корисно подивитися на методи rstrip й strip, які мають схожу функціональність

String#lstrip!

```
str.lstrip!   #-> self або nil
```

Видаляє провідні пробіли з рядка str. Повертає nil, якщо в результаті роботи методу рядок str залишилася незмінною.

```
"  hello  ".lstrip   #-> "hello  "
"hello".lstrip!      #-> nil
```

Корисно подивитися на методи rstrip! і strip!, які мають схожу функціональність

Даний метод є «небезпечним», тому що змінює вихідний об'єкт. Замість нього рекомендується використати метод lstrip, що не має даного побічного ефекту

String#match

```
str.match(pattern)   #-> matchdata або nil
```

Перетворить параметр pattern у правило (якщо він таким не був) і здійснює зіставлення цього правила з рядком str.

```
'hello'.match('(.)\1')       #-> #<MatchData:0x401b3d30>
'hello'.match('(.)\1')[0]    #-> "ll"
'hello'.match(/(.)\1/)[0]    #-> "ll"
'hello'.match('xx')          #-> nil
```

Даний метод практично не використається розроблювачами. Замість нього рекомендується використати метод [] («батарейка»), що має схожим, але більше розгорнутим функціоналом

String#next

```
str.succ   #-> new_str
str.next   #-> new_str
```

Розглядає рядок str як елемент символьної послідовності й повертає наступний за рядком str елемент. Наступний елемент обчислюється збільшенням коду крайнього правого елемента рядка str на одиницю. У результаті збільшення символу, що є цифрою - вийде цифра, а для символу, що є буквою - буква. Збільшення інших символів відбувається з використанням базової символьної впорядкованої послідовності.

Якщо в результаті збільшення виникає необхідність «переносу», символ збільшує левее в даний момент - теж збільшується. Цей процес повторюється для всіх необхідних «переносів». Якщо необхідно, то до рядка str буде доданий додатковий символ.

```
"abcd".next         #-> "abce"
"THX1138".next      #-> "THX1139"
"<<koala>>".next    #-> "<<koalb>>"
"1999zzz".next      #-> "2000aaa"
"ZZZ9999".next      #-> "AAAA0000"
"***".next          #-> "**+"
```

```
"zzz".next          #-> "aaaa"
```

Методи next й succ — абсолютно ідентичні, тобто є іменами того самого методу

String#next!

```
str.succ!   #-> str
str.next!   #-> str
```

Еквівалентний методу next, але міняє рядок str на результат своєї роботи.

Методи next! і succ! — абсолютно ідентичні, тобто є іменами того самого методу

Даний метод є «небезпечним», тому що змінює вихідний об'єкт. Замість нього рекомендується використати метод next, що не має даного побічного ефекту

String#oct

```
str.oct   #-> integer
```

Трактує рядок str як рядок восьмеричних цифр (з необов'язковою вказівкою знака або необятельным префіксом 0) і повертає соотвествующее число. Якщо перетворення не вдається, то повертає нуль.

```
"123".oct       #-> 83
"-377".oct      #-> -255
"bad".oct       #-> 0
"0377bad".oct   #-> 255
```

- Замість даного методу рекомендується використати метод to_i, що є універсальною заміною методів hex й oct

- Корисно глянути на метод hex, що має схожий функціонал

String#replace

```
str.replace(other_str)   #-> str
```

Заміняє вміст рядка str на значення параметра other_str.

```
s = "hello"          #-> "hello"
s.replace "world"    #-> "world"
```

String#reverse

```
str.reverse   #-> new_str
```

Повертає новий рядок, у якій символи рядка str переставлені у зворотному порядку.

```
"stressed".reverse   #-> "desserts"
```

String#reverse!

```
str.reverse!   #-> str
```

Змінює порядок символів рядка str на зворотний.

```
str="stressed"
str.reverse!   #-> "desserts"
str            #-> "desserts"
```

Зверніть увагу, що ні за яких умов метод reverse! не повертає nil

Даний метод є «небезпечним», тому що змінює вихідний об'єкт. Замість нього рекомендується використати метод reverse, що не має даного побічного ефекту

String#rindex

```
str.rindex(substring [, fixnum])   #-> fixnum або nil
str.rindex(fixnum [, fixnum])      #-> fixnum або nil
str.rindex(regexp [, fixnum])      #-> fixnum або nil
```

Повертає індекс останнього входження переданої параметром подстроки (substring), символу (fixnum) або збігу із правилом (regexp) у рядку str. Повертає nil, якщо входження немає. Як другий параметр передається індекс (fixnum), на якому варто закінчити пошук.

```
"hello".rindex('e')             #-> 1
"hello".rindex('l')             #-> 3
"hello".rindex('a')             #-> nil
"hello".rindex(101)             #-> 1
"hello".rindex(/[aeiou]/, -2)   #-> 1
```

Корисно подивитися на метод index, що має схожу функціональність

String#rjust

```
str.rjust(integer, padstr=' ')   #-> new_str
```

Якщо параметр integer більше, ніж довжина рядка str, то метод повертає новий рядок довжини integer з рядком str вирівняної по правому краї, а виникле порожнє місце заповнюється символами padstr; інакше, повертається рядок str.

```
"hello".rjust(4)            #-> "hello"
"hello".rjust(20)           #-> "               hello"
"hello".rjust(20, '1234')   #-> "123412341234123hello"
```

Корисно подивитися на методи ljust й center, які мають схожу функціональність

String#rstrip

```
str.rstrip   #-> new_str
```

Повертає копію рядка str, у якій вилучені всі замикаючі пробельные символи.

```
"  hello  ".rstrip   #-> "  hello"
"hello".rstrip       #-> "hello"
```

Варто звернути увагу на методи lstrip й strip, які мають схожу функціональність

String#rstrip!

```
str.rstrip!   #-> self або nil
```

Видаляє замикаючі пробельные символи з рядка str. Повертає nil, якщо в результаті роботи методу ніяких змін зроблено не було.

```
"  hello  ".rstrip   #-> "  hello"
"hello".rstrip!      #-> nil
```

Корисно подивитися опис методів lstrip! і strip!, які мають схожу функціональність

String#scan

```
str.scan(pattern)                          #-> array
str.scan(pattern) {|match, ...| block }   #-> str
```

Обидві форми виклику послідовно переглядають str на предмет збігу із шаблоном (який може бути як рядком, так і правилом. Кожен збіг записується в масив (для першої форми виклику) або передається в блок (для другої форми виклику). Якщо шаблон (який є правилом) не містить угруповань, то кожен збіг записується в окремий елемент (для першої форми виклику) або передається параметром блоку (для другої форми виклику). Також можливе використання спеціальної змінної $&, що містить результат останнього збігу (актуально при використанні другої форми виклику). Якщо шаблон містить угруповання, то кожен збіг розбивається на збіги угрупованням (виходить двовимірний масив).

```
a = "cruel world"
a.scan(/\w+/)        #-> ["cruel", "world"]
a.scan(/.../)        #-> ["cru", "el ", "wor"]
a.scan(/(...)/)      #-> [["cru"], ["el "], ["wor"]]
a.scan(/(..)(..)/)   #-> [["cr", "ue"], ["l ", "wo"]]
```

…і для другої форми виклику:

```
a.scan(/\w+/) {|w| print "<<#{w}>> " }
print "\n"
a.scan(/(.)(.)/) {|a,b| print b, a }
print "\n"
```

результат:

```
 <<cruel>> <<world>>
 rceu lowlr
```

Корисно подивитися на метод split, що має схожу функціональність

String#size

```
str.length   #-> integer
str.size     #-> integer
```

Повертає розмір рядка str.

```
string = "54321"
string.size   #-> 5
```

Методи size й length — абсолютно ідентичні, тобто є іменами того самого методу

String#slice

```
str[index]                  #-> fixnum або nil
str[start, length]          #-> new_str або nil
str[range]                  #-> new_str або nil
str[regexp]                 #-> new_str або nil
str[regexp, index]          #-> new_str або nil
str[other_str]              #-> new_str або nil
str.slice(index)            #-> fixnum або nil
str.slice(start, length)    #-> new_str або nil
str.slice(range)            #-> new_str або nil
str.slice(regexp)           #-> new_str або nil
str.slice(regexp, index)    #-> new_str або nil
str.slice(other_str)        #-> new_str або nil
```

Одержання подстроки або символу - повертає код символу з індексом index, або подстроку довжини length, починаючи з індексу start, або подстроку, що розташовується в діапазоні range. У всіх цих варіантах виклику негативна індексація має на увазі відлік з кінця рядка str. Повертається nil, якщо індекс index виходить за межі припустимого діапазону, розмір length запитуваної подстроки негативний або початок діапазону range попадає після кінця рядка str. Якщо передається правило regexp, то повертається фрагмент рядка str, що задовольняє правилу regexp (або nil, якщо такого збігу знайти не вдалося). Якщо задано додатковий параметр index, то повертається вміст угруповання правила з номером index (якщо index дорівнює 0, то повертається фрагмент рядка, що задовольняє правилу regexp). Якщо як аргумент передається рядок other_str, то повертається рядок other_str, якщо вона є подстрокой рядка str (або nil, інакше).

```
a = "hello there"
a.slice(1)                    #-> 101
a.slice(1,3)                  #-> "ell"
a.slice(1..3)                 #-> "ell"
a.slice(-3,2)                 #-> "er"
a.slice(-4..-2)               #-> "her"
a.slice(12..-1)               #-> nil
a.slice(-2..-4)               #-> ""
a.slice(/[aeiou](.)\1/)       #-> "ell"
a.slice(/[aeiou](.)\1/, 0)    #-> "ell"
a.slice(/[aeiou](.)\1/, 1)    #-> "l"
a.slice(/[aeiou](.)\1/, 2)    #-> nil
a.slice("lo")                 #-> "lo"
a.slice("bye")                #-> nil
```

Методи slice й [] («батарейка») — абсолютно ідентичні!

String#slice!

```
str.slice!(fixnum)            #-> fixnum або nil
str.slice!(fixnum, fixnum)    #-> new_str або nil
str.slice!(range)             #-> new_str або nil
str.slice!(regexp)            #-> new_str або nil
str.slice!(other_str)         #-> new_str або nil
```

Видаляє певний шматок тексту з рядка str і повертає цей шматок як результат. Якщо як параметр передається число (Fixnum), то можливо виникнення помилки типу IndexError, коли значення цього числа перебуває поза припустимим діапазоном. Якщо як параметр передається діапазон (Range), то можливо виникнення помилки типу RangeError, коли діапазон виходить за рамки припустимих значень. У випадку з параметрами типу Regexp й String метод мовчачи реагує на неприпустимі значення.

```
string = "this is a string"
string.slice!(2)        #-> 105
string.slice!(3..6)     #-> " is "
string.slice!(/s.*t/)   #-> "sa st"
string.slice!("r")      #-> "r"
string                  #-> "thing"
```

Даний метод змінює вихідний рядок. Будьте уважні! Щоб виключити подібного роду зміни використайте метод slice або [] («батарейка»)

> Корисно подивитися на методи slice й [] («батарейка»), які мають подібну функціональність

String#split

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

Ділить рядок str на подстроки по роздільнику pattern (який може бути як правилом, так і рядком). Якщо роздільник pattern не зазначений, то ділення відбувається по пробельному символі (якщо інше не привласнене спеціальної змінної $;). У результаті ділення повертається масив, що містить фрагменти рядка str (сам роздільник у результат не входить).

Якщо роздільник pattern є правилом, то ділення виробляється по подстрокам, що підходить під дане правило. Якщо pattern - рядок, то ділення виробляється по подстрокам, які збігаються з роздільником.

Якщо задано необов'язковий параметр limit, то результуючий масив буде мати кількість фрагментів рядка str рівне limit. Останній елемент буде містити остача, що, можливо, ще можна поділити (тобто в рядку є ще роздільники).

```
" now's  the time".split        #-> ["now's", "the", "time"]
" now's  the time".split(' ')   #-> ["now's", "the", "time"]
" now's  the time".split(/ /)   #-> ["", "now's", "", "the", "time"]
"1, 2.34,56, 7".split(%r{,\s*}) #-> ["1", "2.34", "56", "7"]
"hello".split(//)               #-> ["h", "e", "l", "l", "o"]
```

```
"hello".split(//, 3)            #-> ["h", "e", "llo"]
"hi mom".split(%r{\s*})         #-> ["h", "i", "m", "o", "m"]

"mellow yellow".split("ello")   #-> ["m", "w y", "w"]
"1,2,,3,4,,".split(',')         #-> ["1", "2", "", "3", "4"]
"1,2,,3,4,,".split(',', 4)      #-> ["1", "2", "", "3,4,,"]
"1,2,,3,4,,".split(',', -4)     #-> ["1", "2", "", "3", "4", "", ""]
```

Корисно подивитися на метод scan, що має схожу функціональність

String#squeeze

```
str.squeeze(del=nil)    #-> new_str
```

Створює множина символів з рядка (або рядків) del, переданих як параметр. Повертає новий рядок, що де йдуть підряд однакові символи, які належать множині (переданому як параметр), заміняються на один такий символ. Якщо метод викликаний без параметра, то всі идущие підряд повторювані символи будуть замінені на соотвествующий одиничний.

```
"yellow moon".squeeze                  #-> "yelow mon"
"  now   is  the".squeeze(" ")         #-> " now is the"
"putters shoot balls".squeeze("m-z")   #-> "puters shot balls"
```

Використає ті ж самі правила створення множин символів, що й у методі count

String#squeeze!

```
str.squeeze!(del=nil)    #-> new_str
```

Працює точно також, як і метод squeeze, але результат своєї роботи записує у вихідний рядок.

```
str = "yellow moon"
str.squeeze! #-> "yelow mon"
str          #-> "yelow mon
```

Даний метод є «небезпечним», тому що змінює вихідний об'єкт. Замість нього рекомендується використати метод squeeze, що не має даного побічного ефекту

String#strip

```
str.strip   #-> new_str
```

Повертає копію рядка str у якій вилучені провідні й замикаючі пробельные символи.

```
"    hello    ".strip   #-> "hello"
"\tgoodbye\r\n".strip   #-> "goodbye"
```

Корисно подивитися на методи lstrip й rstrip, які мають схожу функціональність

String#strip!

```
str.strip!   #-> str або nil
```

Видаляє провідні й замикаючі пробельные символи з рядка str. Повертає nil, якщо рядок str таких не містить.

```
str = "    hello    "
str.strip!         #-> "hello"
str                #-> "hello"
"goodbye".strip    #-> nil
```

Корисно подивитися на методи lstrip! і rstrip!, які мають схожу функціональність

Даний метод є «небезпечним», тому що змінює вихідний об'єкт. Замість нього рекомендується використати метод strip, що не має даного побічного ефекту

String#sub

```
str.sub(pattern, replacement)         #-> new_str
str.sub(pattern) {|match| block }     #-> new_str
```

Повертає копію рядка str, де перший збіг із шаблоном (або рядком) pattern замінено на рядок replacement або результат виконання блоку (якому передається параметром результат збігу).

У результаті роботи методу, знайдене збіг із шаблоном pattern записується в спеціальну змінну $& (спадщина мови Perl). У рядку replacement можливе використання послідовностей виду \1, \2 і так далі до \9, які є посиланнями на збіги з угрупованнями (номер угруповання вважається ліворуч праворуч). Усередині блоку на угруповання можна посилатися за допомогою спеціальних змінних виду $1, $2 і так далі до $9. Також, вираженню в блоці доступні спеціальні змінні $', $& й $', які дозволяють одержати доступ до подстроке до збігу, збігу й подстроке після збігу, соотвественно.

```
"hello".sub(/[aeiou]/, '*')              #-> "h*llo"
"hello".sub(/([aeiou])/, '<\1>')         #-> "h<e>llo"
"hello".sub(/./) {|s| s[0].to_s + ' ' }  #-> "104 ello"
```

Корисно подивитися на метод gsub, що має схожу функціональність

String#sub!

```
str.sub!(pattern, replacement)          #-> str або nil
str.sub!(pattern) {|match| block }      #-> str або nil
```

Виконує заміну подібно sub, але змінює вихідний рядок str. Повертає результат заміни або nil, якщо заміна неможлива.

Корисно подивитися на метод gsub!, що має схожу функціональність

Даний метод є «небезпечним», тому що змінює вихідний об'єкт. Замість нього рекомендується використати метод sub, що не має даного побічного ефекту

String#succ

```
str.succ   #-> new_str
str.next   #-> new_str
```

Розглядає рядок str як елемент символьної послідовності й повертає наступний за рядком str елемент. Наступний елемент обчислюється збільшенням коду крайнього правого елемента рядка str на одиницю. У результаті збільшення символу, що є цифрою - вийде цифра, а для символу, що є буквою - буква. Збільшення інших символів відбувається з використанням базової символьної впорядкованої послідовності. Якщо в результаті збільшення виникає необхідність «переносу», символ збільшує левее в даний момент - теж збільшується. Цей процес повторюється для всіх необхідних «переносів». Якщо необхідно, то до рядка str буде доданий додатковий символ.

```
"abcd".succ         #-> "abce"
"THX1138".succ      #-> "THX1139"
"<<koala>>".succ    #-> "<<koalb>>"
"1999zzz".succ      #-> "2000aaa"
"ZZZ9999".succ      #-> "AAAA0000"
"***".succ          #-> "**+"
"zzz".succ          #-> "aaaa"
```

Методи succ й next — абсолютно ідентичні, тобто є іменами того самого методу

String#succ!

```
str.succ!   #-> str
str.next!   #-> str
```

Еквівалентний методу succ, але міняє рядок str на результат своєї роботи.

Методи succ! і next! — абсолютно ідентичні, тобто є іменами того самого методу

Даний метод є «небезпечним», тому що змінює вихідний об'єкт. Замість нього рекомендується використати метод succ, що не має даного побічного ефекту

String#sum

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

Повертає просту n-бітну контрольну суму символів рядка str, де n опциональный целочисленный (Fixnum) параметр, по-умовчанню рівний 16. Результатом -і це просте підсумовування двійкових значень кожного символу рядка str по модулі 2n - 1.

Використання даного методу для підрахунку контрольної суми -і не сама вдала ідея. Рекомендується використати алгоритми MD5 або SHA1, які описані в стандартних бібліотеках md5 й sha1

String#swapcase

```
str.swapcase    #-> new_str
```

Повертає копію рядка str у якій всі символи нижнього регістра замінені на відповідні символи верхнього й всі символи верхнього регістра замінені на відповідні символи нижнього.

```
"Hello".swapcase           #-> "hELLO"
"cYbEr_PuNk11".swapcase    #-> "CyBe_pUn11"
```

- У перекладі з англійського, «swap» - переставляти місцями

- Корисно подивитися методи capitalize, downcase й upcase, які також роблять перетворення регістра

У всіх символів, крім латиниці на ці зміни стійкий імунітет. Це не виходить, що буде виникати помилка... просто вони не будуть перетворені за вищеописаним правилом

String#swapcase!

```
str.swapcase!    #-> str або nil
```

Модифікує рядок str за правилом: всі символи нижнього регістра замінені на відповідні символи верхнього й всі символи верхнього регістра замінені на відповідні символи нижнього. Повертає nil, якщо зміни не потрібні.

```
str="Hello"
str.swapcase!          #-> "hELLO"
str                    #-> "hELLO"
str.swapcase!          #-> "Hello"
"12345".swapcase!      #-> nil
```

- У перекладі з англійського, «swap» - переставляти місцями

- Корисно подивитися методи capitalize!, downcase! і upcase!, які також роблять перетворення регістра

- Даний метод змінює вихідний рядок. Будьте уважні! Щоб виключити подібного роду зміни використайте метод swapcase
- У всіх символів, крім латиниці на ці зміни стійкий імунітет. Це не виходить, що буде виникати помилка... просто вони не будуть перетворені за вищеописаним правилом

String#to_f

```
str.to_f    #-> float
```

Повертає результат інтерпретації рядка str, як дробового числа. Символи після останнього числового - ігноруються. Якщо рядок str не є дробовим числом, то повертається 0.0.

```
"123.45e1".to_f        #-> 1234.5
"45.67 degrees".to_f   #-> 45.67
"thx1138".to_f         #-> 0.0
```

Корисно подивитися на метод to_i, що має схожу функціональність

Даний метод ставиться до загону «тихих» методів, тобто не помилок, що породжують, у результаті своєї роботи

String#to_i

```
str.to_i(base=10)   #-> integer
```

Повертає як результат інтерпретацію символів рядка str, як цілого числа з підставою base (2, 8, 10, 16 або будь-якого іншого). Символи після останнього числового - ігноруються. Якщо рядок str не є числом, то повертається 0.

```
"12345".to_i             #-> 12345
"99 red balloons".to_i   #-> 99
"0a".to_i                #-> 0
"0a".to_i(16)            #-> 10
"hello".to_i             #-> 0
"1100101".to_i(2)        #-> 101
"1100101".to_i(8)        #-> 294977
"1100101".to_i(10)       #-> 1100101
"1100101".to_i(16)       #-> 17826049
```

Даний метод ставиться до загону «тихих» методів, тобто не помилок, що породжують, у результаті своєї роботи

String#to_s

```
str.to_s     #-> str
str.to_str   #-> str
```

Повертає рядок str як результат. Ніяких перетворень не виробляється.

Методи to_s й to_str — абсолютно ідентичні, але використання методу to_s переважніше, тому що його назва коротше

String#to_str

```
str.to_s     #-> str
str.to_str   #-> str
```

Повертає рядок str як результат. Ніяких перетворень не виробляється.

Методи to_s й to_str — абсолютно ідентичні, але використання методу to_s

переважніше, тому що його назва коротше

String#to_sym

```
str.intern   #-> symbol
str.to_sym   #-> symbol
```

Повертає об'єкт класу Symbol, що соотвествует рядку str.

```
"Koala".to_sym         #-> :Koala
s = 'cat'.to_sym       #-> :cat
s == :cat              #-> true
s = '@cat'.to_sym      #-> :@cat
s == :@cat             #-> true
```

Цей метод може бути використаний для створення символів, які не можуть бути створені в :xxx нотації.

```
'cat and dog'.to_sym   #-> :"cat and dog"
```

- Корисно також глянути на метод Symbol#id2name, що здійснює зворотне перетворення
- Методи intern й to_sym - абсолютно ідентичні, але використання методу to_sym переважніше

String#tr

```
str.tr(from, to)  #-> new_str
```

Повертає копію рядка str у якій символи з рядка from замінені на відповідні символи з рядка to. Якщо рядок to коротше, ніж from, то рядок доповнюється своїм останнім символом до довжини рядка from. Обоє строкових параметра (from й to) можуть використати нотацію c1-c2, що розвертається в послідовність символів у діапазоні від з1 до з2. Якщо першим символом у строковому параметрі вказати символ ^, то це буде означати всю множину символів, крім зазначених.

```
"hello".tr('aeiou', '*')    #-> "h*ll*"
"hello".tr('^aeiou', '*')   #-> "*e**o"
"hello".tr('el', 'ip')      #-> "hippo"
"hello".tr('a-y', 'b-z')    #-> "ifmmp"
```

Корисно подивитися на метод tr_s, що має схожу функціональність

String#tr!

```
str.tr!(from, to)  #-> new_str або nil
```

Перетворить рядок str, використовуючи алгоритм, описаний для методу tr. Повертає результат перетворення або nil, якщо в результаті роботи методу змін зроблено не було.

Корисно подивитися на метод tr_s!, що має схожу функціональність

Даний метод є «небезпечним», тому що змінює вихідний об'єкт. Замість нього рекомендується використати метод tr, що не має даного побічного ефекту

String#tr_s

```
str.tr_s(from, to)   #-> new_str
```

Створює копію рядка str, що перетворена по алгоритму, описаному в методі tr, але з попереднім видаленням дублікатів символів, які описані в рядку from.

```
"hello".tr_s('l', 'r')      #-> "hero"
"hello".tr_s('el', '*')     #-> "h*o"
"hello".tr_s('el', 'hx')    #-> "hhxo"
```

Корисно подивитися на метод tr, що має схожу функціональність

String#tr_s!

```
str.tr_s!(from, to)  #-> new_str або nil
```

Перетворить рядок str по алгоритму описаному в методі tr_s. Повертає результат перетворення або nil, якщо в результаті роботи методу перетворень зроблено не було.

Корисно подивитися на метод tr!, що має схожу функціональність

Даний метод є «небезпечним», тому що змінює вихідний об'єкт. Замість нього рекомендується використати метод tr_s, що не має даного побічного ефекту

String#unpack

```
str.unpack(format)   #-> anArray
```

Декодирует рядок str (яка може містити двійкові дані) відповідно до опцій, заданими в рядку format і повертає масив з декодованими даними. Рядок format складається з односимвольних директив (див. таблицю нижче). За кожною директивою може випливати число, що описує кількість повторень цієї опції. Символ «зірочка» (*) застосовує опцію до всіх елементів строки (як би нескінченна кількість повторень опції). Після директив s, S, i, I, l й L може випливати символ підкреслення(_), що дозволяє використати розміри типів (кількість біт), які специфічні для даної платформи; інакше, будуть використатися платформо-независимые розміри типів. Пробельные сивмолы в рядку format ігноруються.

```
"abc \0\0abc \0\0".unpack('A6Z6')   #-> ["abc", "abc "]
"abc \0\0".unpack('a3a3')           #-> ["abc", " \000\000"]
"abc \0abc \0".unpack('Z*Z*')       #-> ["abc ", "abc "]
"aa".unpack('b8B8')                 #-> ["10000110", "01100001"]
"aaa".unpack('h2H2c')               #-> ["16", "61", 97]
```

```
"\xfe\xff\xfe\xff".unpack('s')       #-> [-2, 65534]
"now=20is".unpack('M*')               #-> ["now is"]
"whole".unpack('xax2a2a1a2a')     #-> ["h", "e", "l", "l", "o"]
```

Опція	Тип	Опис
A	String	рядок з вилученими замикаючими пробельными символами
a	String	рядок
B	String	витягти біти з кожного символу (старший двійковий розряд іде першим)
b	String	витягти біти з кожного символу (молодший двійковий розряд іде першим)
C	Fixnum	витягти символ, як беззнакове ціле (unsigned int)
c	Fixnum	витягти символ, як ціле (int)
d, D	Float	інтерпретувати sizeof(double) символів як дробове подвійної точності (double), власний формат («нативный»)
E	Float	інтерпретувати sizeof(double) символів як дробове подвійної точності (double), прямий порядок байт
e	Float	інтерпретувати sizeof(float) символів як дробове (float), прямий порядок байт
f, F	Float	інтерпретувати sizeof(float) символів як дробове (float), власний формат («нативный»)
G	Float	інтерпретувати sizeof(double) символів як дробове з подвійною точністю (double), «мережний» порядок байт
g	Float	інтерпретувати sizeof(float) символів як дробове (float), «мережний» порядок байт
H	String	витягти шестнадцатиричные напівбайти з кожного символу (старший напівбайт іде першим)
h	String	витягти шестнадцатиричные напівбайти з кожного символу (молодший напівбайт іде першим)
I	Integer	інтерпретувати sizeof(int) (може відрізнятися, якщо використається _) наступних символів як беззнакове ціле (unsigned int), власний формат («нативный»)
i	Integer	інтерпретувати sizeof(int) (може відрізнятися, якщо використається _) наступних символів як ціле (int), власний формат («нативный»)
L	Integer	інтерпретувати чотири (може відрізнятися, якщо використається _) наступного символу як беззнакове довге ціле (unsigned long int), власний формат («нативный»)
l	Integer	інтерпретувати чотири (може відрізнятися, якщо використається _) наступного символу як довге ціле (long int), власний формат («нативный»)
M	String	рядок готова до печатки, закодована MIME (див. RFC2045)
m	String	рядок закодована Base64 (див. RFC4648)

N	Integer	інтерпретувати чотири символи як беззнакове довге ціле (unsigned long int), «мережний» порядок байт
n	Fixnum	інтерпретувати два символи як беззнакове коротке ціле (unsigned short int), «мережний» порядок байт
P	String	інтерпретувати sizeof(char *) символів як покажчик і повернути \emph{len} символів із зазначеної області
p	String	інтерпретувати sizeof(char *) символів як покажчик на рядок із завершальним нулем
Q	Integer	інтерпретувати 8 символів як беззнакове 64-бітне ціле
q	Integer	інтерпретувати 8 символів як 64-бітне ціле
S	Fixnum	інтерпретувати дві (кількості може відрізнятися, якщо використається _) наступного символу як беззнакове коротке ціле (unsigned short int), власний формат («нативный»)
s	Fixnum	інтерпретувати дві (кількості може відрізнятися, якщо використається _) наступного символу як коротке ціле (short int), власний формат («нативный»)
U	Integer	UTF-8 символи як беззнакове ціле
u	String	рядок UU-кодована
V	Fixnum	інтерпретувати чотири символи як беззнакове довге ціле (unsigned long int), прямий порядок байт
v	Fixnum	інтерпретувати два символи як беззнакове коротке ціле (unsigned short int), прямий порядок байт
w	Integer	Із ціле (див. Array.pack)
X	—	перемістити покажчик на один символ назад
x	—	перемістити покажчик на один символ уперед
Z	String	рядок з вилученими замикаючими нульовими символами (null) до першого null з *
@	—	перемістити покажчик на абсолютну позицію
Корисно подивитися на метод pack, що має схожу функціональність		

String#upcase

```
str.upcase   #-> new_str
```

Повертає копію рядка str у якій всі символи нижнього регістра замінені на відповідні символи верхнього.

```
"hEll".upcase    #-> "HELLO"
```

Корисно подивитися на методи capitalize, downcase й swapcase, які мають схожу функціональність

У всіх символів, крім латиниці на ці зміни стійкий імунітет. Це не виходить, що виникне

помилка. Просто, вони не будуть перетворені за вищеописаним правилом

String#upcase!

str.upcase!   #-> str або nil

Модифікує рядок str за правилом: всі символи нижнього регістра перетворюють у відповідні символи верхнього. Повертає nil, якщо зміни не потрібні.

```
str="hEll"
str.upcase!     #-> "HELLO"
str             #-> "HELLO"
str.upcase!     #-> nil
```

Корисно подивитися на методи capitalize!, downcase! і swapcase!, які мають схожу функціональність

Даний метод є «небезпечним», тому що змінює вихідний об'єкт. Замість нього рекомендується використати метод upcase, що не має даного побічного ефекту

У всіх символів, крім латиниці на ці зміни стійкий імунітет. Це не виходить, що виникне помилка. Просто, вони не будуть перетворені за вищеописаним правилом

String#upto

```
str.upto(other_str) {|s| block }   #-> str
```

Даний итератор проходить всі значення між str й other_str включно, передаючи їх у блок як параметр.

```
"a8".upto("b6") {|s| print s, ' ' }
for s in "a8".."b6"
  print s, ' '
end
```

результат:

```
 a8 a9 b0 b1 b2 b3 b4 b5 b6
 a8 a9 b0 b1 b2 b3 b4 b5 b6
```

- Для пошуку наступного значення послідовності використається метод succ
- Як результат повертає рядок str (від якої був викликаний)

Клас Struct

Клас Struct реалізує зручний спосіб об'єднання набору атрибутів, використовуючи методи доступу, без явного створення класу. Клас Struct створює специфічні класи, які містять тільки набір змінних і методи доступу до них. У наступному прикладі ми створимо клас Customer і продемонструємо приклад використання об'єкта цього класу. У наступному описі, параметр symbol може бути рядком або екземпляром класу Symbol (наприклад, :name).

Домішки

```
Enumerable (all?, any?, collect, detect, each_cons, each_slice, each_with_index,
entries, enum_cons, enum_slice, enum_with_index, find, find_all, grep, group_by,
```

```
include?, index_by, inject, map, max, member?, min, partition, reject, select,
sort, sort_by, sum, to_a, to_set, zip)
```

Методи класу

```
new, new
```

Методи об'єкта

```
[]=, [], ==, each_pair, each, eql?, hash, inspect, length, members, select,
size, to_a, to_s, values_at, values
```

Struct::new

```
Struct.new( string, symbols* )     #-> StructClass
StructClass.new(arg, ...)          #-> obj
StructClass[arg, ...]              #-> obj
```

Створення нового класу з ім'ям string, що містить методи доступу до атрибутів. Якщо ім'я string пропущене, то буде створена анонімна структура. Інакше, ім'я цієї структури буде виглядати як константа класу Struct. Урахуйте, що ім'я структури повинне бути унікально й починатися з великої букви. Struct::new повертає об'єкт класу Class, що може бути використаний для створення специфічних екземплярів нової структури. Число дійсних параметрів повинне бути менше або дорівнює числу атрибутів, оголошених для цієї структури.

Створення структури з ім'ям усередині класу Struct

```
Struct.new("Customer", :name, :address)    #-> Struct::Customer
Struct::Customer.new("Dave", "123 Main")   #-> #<Struct::Customer name="Dave",
address="123 Main">
```

Ім'я структури залежить від імені константи

```
Customer = Struct.new(:name, :address)     #-> Customer
Customer.new("Dave", "123 Main")           #-> #<Customer name="Dave",
address="123 Main">
```

Struct#==

```
struct == other_struct     #-> true або false
```

Еквівалентність --- повертає true якщо other_struct дорівнює : вони повинні бути об'єктами того самого класу, створеного за допомогою Struct::new, і значення внутрішніх змінних повинні збігатися (відповідно до методу Object#==).

```
Customer = Struct.new(:name, :address, :zip)
joe   = Customer.new("Joe Smith", "123 Maple, Anytown NC", 12345)
joejr = Customer.new("Joe Smith", "123 Maple, Anytown NC", 12345)
jane  = Customer.new("Jane Doe", "456 Elm, Anytown NC", 12345)
joe == joejr   #-> true
joe == jane    #-> false
```

Struct#[]

```
struct[symbol]    #-> anObject
struct[fixnum]    #-> anObject
```

Значення атрибута можна одержати, якщо передати як параметр ім'я атрибута (у вигляді рядка або символу) або числовий індекс в інтервалі (0..розмір-1). Виникне помилка NameError, якщо ім'я атрибута відсутній або IndexError, якщо числовий індекс буде лежати поза припустимим інтервалом.

```
Customer = Struct.new(:name, :address, :zip)
joe = Customer.new("Joe Smith", "123 Maple, Anytown NC", 12345)

joe["name"]   #-> "Joe Smith"
joe[:name]    #-> "Joe Smith"
joe[0]        #-> "Joe Smith"
```

Struct#[]=

```
struct[symbol] = obj    #-> obj
struct[fixnum] = obj    #-> obj
```

Змінити значення атрибута можна передавши як параметр ім'я атрибута (у вигляді рядка або символу) або целочисленный індекс в інтервалі (0..розмір-1), а також об'єкт obj, якому треба привласнити атрибуту. Виникне помилка NameError, якщо імені атрибута не існує або IndexError, якщо целочисленный індекс буде лежати поза діапазоном.

```
Customer = Struct.new(:name, :address, :zip)
joe = Customer.new("Joe Smith", "123 Maple, Anytown NC", 12345)

joe["name"] = "Luke"
joe[:zip]   = "90210"

joe.name   #-> "Luke"
joe.zip    #-> "90210"
```

Struct#each

```
struct.each {|obj| block }  #-> struct
```

Виконує block по разі на кожен атрибут, передаючи в блок його значення.

```
Customer = Struct.new(:name, :address, :zip)
joe = Customer.new("Joe Smith", "123 Maple, Anytown NC", 12345)
joe.each {|x| puts(x) }
```

результат:

```
Joe Smith
123 Maple, Anytown NC
12345
```

Struct#each_pair

```
struct.each_pair {|sym, obj| block }     #-> struct
```

Виконує block по разі на кожен атрибут, передаючи в блок ім'я (як символ) і значення параметра.

```
Customer = Struct.new(:name, :address, :zip)
joe = Customer.new("Joe Smith", "123 Maple, Anytown NC", 12345)
joe.each_pair {|name, value| puts("#{name} => #{value}") }
```
результат:
```
name => Joe Smith
address => 123 Maple, Anytown NC
zip => 12345
```

Struct#eql?

```
struct.eql?(other)   #-> true або false
```

Дві структури еквівалентні, якщо вони є тим самим об'єктом або якщо всі їхні атрибути еквіваленти (використається метод eql?).

Struct#hash

```
struct.hash   #-> fixnum
```

Повертає контрольну суму, заснована на контрольних сумах значень атрибутів.

Struct#inspect

```
struct.to_s      #-> string
struct.inspect   #-> string
```

Демонструє вміст структури у вигляді рядка.

(ще відомий як to_s)

Struct#length

```
struct.length    #-> fixnum
struct.size      #-> fixnum
```

Повертає кількість атрибутів структури.
```
Customer = Struct.new(:name, :address, :zip)
joe = Customer.new("Joe Smith", "123 Maple, Anytown NC", 12345)
joe.length   #-> 3
```
(ще відомий як size)

Struct#members

```
struct.members    #-> array
```

Повертає масив рядків, що складає з імен атрибутів структури.
```
Customer = Struct.new(:name, :address, :zip)
```

```
joe = Customer.new("Joe Smith", "123 Maple, Anytown NC", 12345)
joe.members   #-> ["name", "address", "zip"]
```

Struct#select

```
struct.select {|i| block }    #-> array
```

Виконує блок для кожного значення атрибута в структурі, повертає масив утримуючі елементи для яких блок повернув значення true (еквівалент методу Enumerable#select).

```
Lots = Struct.new(:a, :b, :c, :d, :e, :f)
l = Lots.new(11, 22, 33, 44, 55, 66)
l.select {|v| (v % 2).zero? }   #-> [22, 44, 66]
```

Struct#size

```
struct.length    #-> fixnum
struct.size      #-> fixnum
```

Повертає кількість атрибутів структури.

```
Customer = Struct.new(:name, :address, :zip)
joe = Customer.new("Joe Smith", "123 Maple, Anytown NC", 12345)
joe.length   #-> 3
```

(ще відомий як length)

Struct#to_a

```
struct.to_a     #-> array
struct.values   #-> array
```

Повертає масив значень атрибутів структури.

```
Customer = Struct.new(:name, :address, :zip)
joe = Customer.new("Joe Smith", "123 Maple, Anytown NC", 12345)
joe.to_a[1]   #-> "123 Maple, Anytown NC"
```

(ще відомий як values)

Struct#to_s

```
struct.to_s      #-> string
struct.inspect   #-> string
```

Демонструє вміст структури у вигляді рядка.

(ще відомий як inspect)

Struct#values

struct.to_a   #-> array

```
struct.values   #-> array
```

Повертає масив значень атрибутів структури.

```
Customer = Struct.new(:name, :address, :zip)
joe = Customer.new("Joe Smith", "123 Maple, Anytown NC", 12345)
joe.to_a[1]   #-> "123 Maple, Anytown NC"
```

(ще відомий як to_a)

Struct#values_at

```
struct.values_at(selector,... )  #-> an_array
```

Повертає масив, що складається зі значень елементів, індекси яких передані як параметри. Індекси можуть бути цілим числом або целочисленным інтервалом. Дивися також select.

```
a = %w{ a b c d e f }
a.values_at(1, 3, 5)
a.values_at(1, 3, 5, 7)
a.values_at(-1, -3, -5, -7)
a.values_at(1..3, 2...5)
```

Клас GC

Модуль GC забезпечує Руба-інтерфейс, що дозволяє управляти механізмом зборки сміття. Деякі з методів також доступні через модуль ObjectSpace.

Методи класу

```
disable, enable, start
```

Методи об'єкта

```
garbage_collect
```

GC::disable

```
GC.disable    #-> true або false
```

Відключає зборку сміття, повертає true якщо зборка сміття вже була відключена.

```
GC.disable   #-> false
GC.disable   #-> true
```

GC::enable

```
GC.enable    #-> true або false
```

Включає зборку сміття, повертає true якщо зборка сміття була попередньо відключена.

```
GC.disable   #-> false
GC.enable    #-> true
GC.enable    #-> false
```

GC::start

```
GC.start                        #-> nil
gc.garbage_collect              #-> nil
ObjectSpace.garbage_collect     #-> nil
```

Починає зборку сміття, поки не відключена вручну.

GC#garbage_collect


```
GC.start                        #-> nil
gc.garbage_collect              #-> nil
ObjectSpace.garbage_collect     #-> nil
```

Починає зборку сміття, поки не відключена вручну.

ПРИКЛАД ОФОРМЛЕННЯ ПРОТОКОЛУ З ЛАБОРАТОРНОЇ РОБОТИ

Національна академія управління

Кафедра інтелектуальних технологій

Об'єктно-орієнтоване програмування

Лабораторна робота №1

Студентов Студент Студентович

Група № номер

Варіант № номер

Завдання 1.1.

1.

Постановка задачі

………….

Алгоритм рішення

………....

Реалізація мовою Ruby

…………

Скріншот результату роботи програми

2.

Постановка задачі

………….

Алгоритм рішення

………....

Реалізація мовою Ruby

…………

Скріншот результату роботи програми

Завдання 1.2.

1.

Постановка задачі

………….

Алгоритм рішення

………....

Реалізація мовою Ruby

…………

Скріншот результату роботи програми

2.

Постановка задачі

………….

Алгоритм рішення

………....

Реалізація мовою Ruby

…………

Скріншот результату роботи програми

Індивідуальне завдання – варіант № …

Завдання 1

1.

Постановка задачі

………….

Алгоритм рішення

……….…

Реалізація мовою Ruby

…………

Скріншот результату роботи програми

2.

Постановка задачі

………….

Алгоритм рішення

……….…

Реалізація мовою Ruby

…………

Скріншот результату роботи програми

Завдання 2

1.

Постановка задачі

………….

Алгоритм рішення

……….…

Реалізація мовою Ruby

…………

Скріншот результату роботи програми

2.

Постановка задачі

………….

Алгоритм рішення

……….…

Реалізація мовою Ruby

…………

Скріншот результату роботи програми

Завдання 3

Постановка задачі

………….

Алгоритм рішення

……….…

Реалізація мовою Ruby

…………

Скріншот результату роботи програми