7. Массивы и итераторы

Давайте напишем программу, которая просит нас ввести сколько угодно слов (по одному слову в строке до тех пор, пока мы не нажмём `Enter` на пустой строке) и которая затем повторяет нам эти слова в алфавитном порядке. Идёт?

Так... сначала мы — ээ... ну... хммм... Хорошо, мы могли бы — ээ... ну...

Вы знаете, я не думаю, что мы сможем это сделать. Нам нужен способ хранить неизвестное количество слов и как-то держать их все вместе, чтобы они не смешивались с другими переменными. Нам нужно поместить их все в какое-то подобие списка. Нам нужны массивы.

Массив — это просто список в вашем компьютере. Каждая ячейка в списке ведёт себя как переменная: вы можете посмотреть, на какой объект укзывает определённая ячейка, и вы можете сделать так, чтобы она указывала на другой объект. Давайте посмотрим на некоторые массивы:

```
[]                              #  пустой массив
[5]                             #  массив из одного числа
['Привет', 'До свидания']       #  массив из 2-х строк
flavor = 'ванильный'            #  Это не массив, конечно...
[89.9, flavor, [true, false]]   #  ...но это — массив.
```

Итак, сначала у нас есть пустой массив, затем массив, содержащий единственное число, затем массив, содержащий две строки. Потом у нас имеется простое присваивание; затем — массив, содержащий три объекта, последний из которых — это массив **`[true, false]`**. Помните, что переменные — это не объекты, поэтому наш последний массив в действительности указывает на число с плавающей точкой, на *строку* и на массив. Даже если бы мы сделали, чтобы переменная **`flavor`** указывала на что-нибудь другое, это не изменило бы этот массив.

Чтобы помочь нам с поиском конкретных объектов в массиве, каждой ячейке даётся номер — индекс. Программисты (и, по случайному совпадению, большинство математиков) начинают считать их с нуля, так что первая ячейка в массиве — это ячейка номер ноль. Вот как мы должны адресоваться к объектам в массиве:

```
names = ['Ада', 'Белль', 'Крис']
puts names
puts names[0]
puts names[1]
puts names[2]
puts names[3]  #  Это вне диапазона массива.
```

```
Ада
Белль
Крис
Ада
Белль
```

```
Крис
nil
```

Итак, мы видим, что **puts names** выводит каждое имя в массиве **names**. Затем мы используем **puts names[0]**, чтобы напечатать "первое" имя в массиве и **puts names[1]**, чтобы напечатать "второе"... Уверен, что это кажется запутанным, но вы *непременно* привыкните к этому. Вы просто должны действительно начать *думать*, что отсчёт начинается с нуля, и перестать использовать такие слова, как "первый" и "второй". Если вы идёте на обед из пяти блюд, не говорите о "первом" блюде; говорите о блюде ноль (а у себя в голове думайте о **course[0]**). На правой руке у вас пять пальцев с номерами 0, 1, 2, 3 и 4. Моя жена и я умеем жонглировать. Когда мы жонглируем шестью булавами, мы жонглируем булавами 0-5. Надеемся, что через несколько месяцев мы сможем жонглировать булавой номер 6 (и таким образом будем жонглировать между собой семью булавами). Вы поймёте, что привыкли к этому, когда начнёте использовать слово "нулевой". :-) Да, в самом деле есть такое слово: спросите у любого программиста или математика.

Наконец, мы попытались выполнить **puts names[3]**, просто чтобы посмотреть, что же получится. Вы ожидали ошибку? Иногда, когда вы задаёте вопрос, ваш вопрос не имеет смысла (по крайней мере для вашего компьютера); и тогда вам выдаётся ошибка. Иногда, однако, вы можете задать вопрос и ответ на него будет: *ничего*. Что в ячейке номер три? Ничего. Что в **names[3]**? **nil**: таким способом Ruby говорит "ничего". **nil** — это особенный объект, который обычно означает "никакой из объектов".

Если вся эта смешная нумерация ячеек массивов начинает вас "доставать", не бойтесь! Часто мы сможем обойтись совершенно без неё, используя различные методы массивов, как вот этот:

Метод each

each позволяет нам делать что-нибудь (всё, что мы хотим) с **каждым** объектом, на который указывает массив. Так, если мы хотим сказать что-либо приятное о языке в приведённном ниже массиве, мы сделаем так:

```
languages = ['английский', 'немецкий', 'Ruby']
languages.each do |lang|
  puts 'Мне нравится ' + lang + '!'
  puts 'А вам?'
end
puts 'А теперь давайте послушаем мнение о C++!'
puts '...'
```

```
Мне нравится английский!
А вам?
Мне нравится немецкий!
А вам?
Мне нравится Ruby!
А вам?
А теперь давайте послушаем мнение о C++!
```

```
...
```

Так что же сейчас произошло? Вот, мы смогли обойти каждый из объектов в массиве совсем без использования номеров, и это определённо приятно. В переводе на русский, приведённую выше программу можно прочитать примерно так: для каждого (**each**) объекта в **languages**, пусть переменная **lang** указывает на этот объект и затем выполнится (**do**) всё, что я скажу вам, пока вы не дойдёте до конца (**end**). (Чтобы вы знали: C++ это ещё один язык программирования. Его гораздо труднее изучить, чем Ruby; обычно, программа на C++ будет во много раз длиннее, чем программа на Ruby, которая делает то же самое.)
Вы, должно быть, думаете: "Это очень похоже на циклы, о которых мы узнали до этого". Ну да, похоже. Одно важное отличие заключается в том, что метод **each** — это просто-напросто метод. **while** и **end** (также как **do**, **if**, **else** и все другие **синие** слова) это не методы. Они являются основополагающей частью языка Ruby, как и операция = или скобки; они похожи на знаки пунктуации в английском [или русском — *Прим. перев.*] языке.

Но не **each**; **each** — это просто ещё один метод массива. Такие методы, как **each**, которые "ведут себя как" циклы, часто называют итераторами.

Нужно отметить ещё одну вещь об итераторах: за ними всегда следует **do**...**end**. Около **while** и **if** никогда не бывает **do**; мы используем **do** только с итераторами.

А вот другой маленький симпатичный итератор, но это не метод массива... Это метод целого числа!

```
3.times do
  puts 'Гип-гип-ура!'
end
```

```
Гип-гип-ура!
Гип-гип-ура!
Гип-гип-ура!
```

Другие методы массивов

Итак, мы изучили **each**, но у массивов есть много других методов... почти столько же много, как методов у строк! В сущности, некоторые из них (такие, как **length**, **reverse**, **+** и *****) работают так же, как и со строками, только они оперируют с ячейками массива, а не с буквами в строке. Другие, например, **last** и **join**, характерны только для массивов. Третьи, такие как **push** и **pop**, фактически изменяют массив. И так же, как с методами строк, вам не нужно помнить их все, покуда вы помните, где можно узнать о них (прямо здесь).

Сначала давайте взглянем на **to_s** и **join**. Метод **join** работает во многом схоже с методом **to_s**, за исключением того, что он добавляет строку между объектами массива. Давайте посмотрим:

```
foods = ['артишок', 'бриошь', 'карамель']
puts foods
puts
puts foods.to_s
```

```
puts
puts foods.join(', ')
puts
puts foods.join('  :)  ') + '  8)'
200.times do
  puts []
end
```

```
артишок
бриошь
карамель
артишокбриошькарамель
артишок, бриошь, карамель
артишок  :)  бриошь  :)  карамель  8)
```

Как видите, метод **puts** обращается с массивами не так, как с другими объектами: он просто вызывает **puts** для каждого из объектов в массиве. Вот почему вывод через **puts** пустого массива 200 раз ничего не делает: массив ни на что не указывает, так что нечего выводить с помощью **puts**. (Ничего не делать 200 раз — значит всё равно ничего не делать.) Попробуйте вывести методом **puts** массив, содержащий другие массивы: он делает то, что вы ожидали?

Кроме того, вы заметили, что я не употреблял в **puts** пустые строки, когда хотел вывести чистую строку? Получилось то же самое.

Теперь давайте посмотрим на **push**, **pop** и **last**. Методы **push** и **pop** в каком-то смысле противоположны, как **+** и **-**. **push** добавляет объект в конец вашего массива, а **pop** удаляет последний объект из массива (и сообщает вам, что это был за объект). **last** похож на **pop** в том, что он сообщает вам, что находится в конце массива, только он оставляет массив нетронутым. Повторю снова: **push** и **pop** *действительно изменяют массив*:

```
favorites = []
favorites.push 'капли дождя на розах'
favorites.push 'капли виски на котах'
puts favorites[0]
puts favorites.last
puts favorites.length
puts favorites.pop
puts favorites
puts favorites.length
```

```
капли дождя на розах
капли виски на котах
2
капли виски на котах
капли дождя на розах
1
```

Попробуйте ещё кое-что

• Напишите программу, о которой мы говорили в самом начале этой главы.

> ***Подсказка:*** *Есть прекрасный метод массива, который вернёт вам отсортированную версию массива:* **`sort`**. *Используйте его!*

• Попробуйте написать указанную программу *без* использования метода **`sort`**. Большая часть программирования - это преодоление сложностей, так что практикуйтесь чаще, насколько это возможно!

• Перепишите вашу программу "Содержание" (из главы о методах). Начните программу с массива, содержащего всю информацию для вашей таблицы с содержанием (названия глав, номера страниц и т. д.). Затем напечатайте информацию из массива в виде красиво отформатированного содержания.

До сих пор мы изучили довольно много разных методов. А сейчас пора научиться, как сделать свои собственные.