**Phương pháp .split**

Tiếp theo, chúng tôi sẽ muốn chia đầu vào của người dùng thành các từ riêng lẻ.

Ruby có một phương thức tích hợp cho cái này được gọi là .split; nó nhận trong một chuỗi và trả về một mảng. Nếu chúng ta chuyển nó một bit văn bản trong dấu ngoặc đơn, .splitsẽ chia chuỗi ở bất cứ nơi nào nó nhìn thấy bit văn bản đó, được gọi là *dấu phân cách* . Ví dụ,

text.split(",")

**Arrays**

Trước đó, chúng ta đã thấy rằng một mảng có thể được sử dụng để lưu trữ danh sách các giá trị trong một biến duy nhất. Bạn có thể nhồi bất kỳ số lượng số nào vào đó, bạn có thể lặp lại các số và chúng không nhất thiết phải theo thứ tự số!

**Access by Index**

Đây là một số điều thú vị về mảng: mỗi phần tử trong mảng có cái được gọi là index . Phần tử đầu tiên ở index 0, phần tử tiếp theo là index 1, phần tử sau là index 2, v.v. Chúng ta có thể truy cập trực tiếp các phần tử của mảng thông qua các số này bằng cách sử dụng dấu ngoặc vuông, như sau:

array = [5, 7, 9, 2, 0]  
array[2]  
# returns "9", since "9"  
# is at index 2

Sơ đồ dưới đây cho thấy cách các chỉ số này hoạt động cho mảng mẫu của chúng tôi , [5, 7, 9, 2, 0]. Phần tử đầu tiên có index 0, phần tử tiếp theo có 1, phần tử tiếp theo có 2, v.v.

        +---+---+---+---+---+  
array   | 5 | 7 | 9 | 2 | 0 |  
        +---+---+---+---+---+  
index     0   1   2   3   4

(Đây là một chút đơn giản hóa quá mức, nhưng nó sẽ có được ý tưởng từ bây giờ).

Chúng ta có thể truy cập vào iyếu tố thứ của một mảng được gọi arraybằng cách đặt các chỉ số trong ngoặc vuông, như vậy: array[i]. array[0]nhận phần tử đầu tiên, array[1]nhận phần tử thứ hai, v.v. Đây được gọi là truy cập theo index .

**Mảng không phải số**

Đây là điều mà bạn có thể chưa biết: bạn có thể tạo một mảng của bất kỳ bộ sưu tập các đối tượng Ruby nào. Bạn có thể tạo một mảng boolean! Một mảng các chuỗi! Khả năng là gần như vô tận

**Arrays of Arrays**

Bạn có thể tự hỏi mình: "Nếu tôi có thể đặt bất cứ thứ gì vào một mảng, tôi có thể tạo một mảng của mảng không?" Câu trả lời là: có!

Kiểm tra mảng mảng mà chúng tôi có trong trình chỉnh sửa. Mảng của mảng được gọi là mảng nhiều chiều , vì hành động thêm nhiều mảng hơn sẽ mở rộng mảng ra khỏi hình dạng giống như chuỗi của nó. Ví dụ, mảng trong trình soạn thảo là một mảng hai chiều.

ulti\_d\_array = [[0,0,0,0],[0,0,0,0],[0,0,0,0],[0,0,0,0]]

multi\_d\_array.each { |x| puts "#{x}\n" }

**Hashes**

Chúng ta biết rằng mảng được lập chỉ mục bằng các số bắt đầu bằng 0 và tăng dần độ dài của mảng trừ đi một. (Hãy nghĩ về nó: một mảng có bốn phần tử có các chỉ số 0, 1, 2 và 3)

Nhưng nếu chúng ta muốn sử dụng các chỉ số không theo thứ tự từ 0 đến cuối mảng thì sao? Điều gì sẽ xảy ra nếu chúng ta không muốn sử dụng số làm chỉ số? Chúng ta sẽ cần một cấu trúc mảng mới được gọi là hàm băm .

Hàm băm giống như các đối tượng JavaScript hoặc từ điển Python. Nếu bạn chưa học các ngôn ngữ đó, tất cả những gì bạn cần biết rằng hàm băm là một tập hợp các cặp khóa-giá trị . Cú pháp băm trông như thế này:

hash = {  
  key1 => value1,  
  key2 => value2,  
  key3 => value3  
}

Giá trị được gán cho các khóa bằng cách sử dụng =>. Bạn có thể sử dụng bất kỳ đối tượng Ruby nào cho một khóa hoặc giá trị.

**Sử dụng Hash.new**

Những gì chúng tôi vừa cho bạn thấy là *ký hiệu theo nghĩa đen* . Chúng tôi gọi nó như vậy vì bạn mô tả theo nghĩa đen những gì bạn muốn trong hàm băm: bạn đặt tên cho nó và bạn đặt nó bằng một hoặc nhiều key => valuecặp bên trong dấu ngoặc nhọn.

Bạn cũng có thể tạo một hàm băm bằng cách sử dụng Hash.new, như sau:

my\_hash = Hash.new

Đặt một biến bằng để Hash.newtạo một băm mới, trống; nó giống như việc đặt biến bằng dấu ngoặc nhọn rỗng ( {}).

**Thêm vào một băm**

Chúng ta có thể thêm vào hàm băm theo hai cách: nếu chúng ta tạo nó bằng cách sử dụng ký hiệu theo nghĩa đen, chúng ta có thể chỉ cần thêm một cặp khóa-giá trị mới trực tiếp vào giữa các dấu ngoặc nhọn. Nếu chúng ta đã sử dụng Hash.new, chúng ta có thể thêm vào hàm băm bằng cách sử dụng ký hiệu ngoặc:

pets = Hash.new  
pets["Stevie"] = "cat"  
# Adds the key "Stevie" with the  
# value "cat" to the hash

**Truy cập giá trị băm**

Bạn có thể truy cập các giá trị trong một hàm băm giống như một mảng.

pets = {  
  "Stevie" => "cat",  
  "Bowser" => "hamster",  
  "Kevin Sorbo" => "fish"  
}  
   
puts pets["Stevie"]  
# will print "cat"

1. Trong ví dụ trên, chúng ta tạo một hàm băm được gọi pets.
2. Sau đó, chúng tôi in catbằng cách truy cập vào khóa "Stevie"trong petshàm băm.

**Iterating Over Arrays**

Lặp lại các mảng dễ dàng hơn vẻ ngoài.

numbers = [1, 2, 3, 4, 5]  
numbers.each { |element| puts element }

1. Trong ví dụ trên, chúng ta tạo một mảng numberscó 5 phần tử.
2. Sau đó, chúng tôi nói, "Lấy mảng này và đối với mỗi phần tử, in nó ra bảng điều khiển." Như thường lệ, chúng ta có thể sử dụng bất kỳ tên trình giữ chỗ nào cho bit giữa hai | |ký tự.

**Iterating Over Multidimensional Arrays**

Bây giờ chúng ta hãy xem cách lặp qua một mảng đa chiều.

Chúng tôi đã tạo một mảng 2-D, s(cho "bánh mì"). Chúng tôi muốn lặp lại theo scách không in ra từng phần tử dưới dạng một mảng, chẳng hạn như ["ham", "swiss"], mà là từng phần tử trong mỗi mảng con, vì vậy chúng tôi nhận được danh sách tất cả các loại thịt và pho mát bên trong s.

Nếu chúng tôi chỉ muốn truy cập "swiss", chúng tôi có thể nhập

s[0][1]

Có nghĩa là, “mang cho tôi phần tử thứ hai của phần tử đầu tiên,” là “swiss”. Nếu chúng ta lặp qua một mảng thông thường bằng cách sử dụng

array.each { |element| action }

**Iterating Over Hashes**

**Lặp lại các bước**

Khi lặp qua các hàm băm, chúng ta cần hai biến giữ chỗ để đại diện cho mỗi cặp khóa / giá trị.

restaurant\_menu = {  
  "noodles" => 4,  
  "soup" => 3,  
  "salad" => 2  
}  
   
restaurant\_menu.each do |item, price|  
  puts "#{item}: #{price}"  
end

1. Trong ví dụ trên, chúng tôi tạo một hàm băm mới được gọi là restaurant\_menu.
2. Sau đó, chúng ta lặp qua restaurant\_menuhàm băm và gán khóa itemvà giá trị pricecho mỗi lần lặp.
3. Cuối cùng, chúng tôi putsra:

noodles: 4  
soup: 3  
salad: 2

**Mảng đa chiều**

array = [[thing, thing], [thing, thing]]