**Method Syntax**

1. Các *tiêu đề* , trong đó bao gồm các deftừ khóa, tên của phương pháp này, và bất kỳ *đối số* phương pháp mất. (Chúng ta sẽ đi đến các đối số trong phần tiếp theo)
2. Phần *thân* , là khối mã mô tả các thủ tục mà phương thức thực hiện. Cơ thể được thụt vào hai khoảng trống theo quy ước (như với for, if, elsif, và elsebáo cáo)
3. Phương thức kết thúc bằng endtừ khóa.

Đây là một ví dụ về cú pháp sẽ trông như thế nào đối với một hàm đơn giản welcome, chỉ in ra “Chào mừng bạn đến với Ruby!” vào bảng điều khiển:

def welcome  
  puts "Welcome to Ruby!"  
end

**Tham số và đối số**

Nếu một phương thức nhận các đối số, chúng tôi nói rằng nó *chấp nhận* hoặc *mong đợi* các đối số đó. Chúng ta có thể xác định một hàm square, như sau:

def square(n)  
  puts n \*\* 2  
end

và gọi nó như thế này:

square(12)  
# ==> prints "144"

Đối *số* là đoạn mã bạn thực sự đặt giữa các dấu ngoặc đơn của phương thức khi bạn gọi nó và *tham số* là tên bạn đặt giữa các dấu ngoặc đơn của phương thức khi bạn *định nghĩa* nó. Ví dụ, khi chúng ta định nghĩa squareở trên, chúng ta đã cho nó tham số n(cho “number”) và chuyển nó vào đối số 12khi chúng ta gọi nó.

Bạn có thể coi các tham số như trình giữ chỗ mà định nghĩa phương thức cung cấp cho các đối số vì nó không biết trước chính xác đối số mà nó sẽ nhận.

Dấu ngoặc đơn thường là tùy chọn trong Ruby, nhưng bạn nên đặt các tham số và đối số của bạn trong dấu ngoặc đơn để dễ đọc.

**Splat!**

Nói về việc không biết điều gì sẽ xảy ra: các phương pháp của bạn không chỉ không biết chúng sẽ đi trước những đối số nào mà đôi khi, họ thậm chí còn không biết sẽ có bao nhiêu đối số.

Hãy nói rằng bạn có một phương pháp, friendrằng putslập luận mà nó nhận được từ người sử dụng. Nó có thể trông giống như sau:

def friend(name):  
  puts "My friend is " + name + "."  
end

Điều này thật tuyệt vời cho chỉ một người bạn, nhưng nếu bạn muốn in ra tất cả bạn bè của người dùng mà không biết người dùng sẽ đặt trước bao nhiêu tên bạn bè?

Giải pháp: *đối số splat* . Các đối số splat là các đối số đứng trước a \*, điều này cho chương trình biết rằng phương thức có thể nhận một hoặc nhiều đối số.

**Return**

Đôi khi chúng ta không chỉ muốn một phương thức in một thứ gì đó ra bảng điều khiển, mà chúng ta thực sự muốn phương thức đó chuyển cho chúng ta (hoặc một phương thức khác!) Trả lại một giá trị. Đối với điều đó, chúng tôi sử dụng return.

def double(n)  
  return n \* 2  
end  
   
output = double(6)  
output += 2  
puts output

1. Trong ví dụ trên, chúng ta định nghĩa một phương thức mới được gọi là phương thức doublechấp nhận một đối số được gọi n.
2. Bên trong phương thức, chúng tôi trả về hai lần n.
3. Sau đó, chúng tôi gọi doublephương thức mới của mình với một đối số là 6và lưu trữ kết quả 12, trong output.
4. Sau đó, chúng tôi tăng outputlên 14 và in nó ra bảng điều khiển.

**Các khối giống như các phương pháp không tên**

Hầu hết các phương thức mà bạn đã làm việc đều có tên được xác định mà bạn hoặc người khác đã đặt cho chúng ( *ví dụ:* [array] .sort (), “string” .downcase (), v.v.). Bạn có thể nghĩ về các khối như một cách tạo ra các phương thức không có tên. (Các hàm này tương tự như các hàm ẩn danh trong JavaScript hoặc lambdas trong Python.)

Các khối có thể được xác định bằng từ khóa dovà endhoặc bằng dấu ngoặc nhọn ( {}).

1.times do

  puts "I'm a code block!"

end

1.times { puts "As am I!" }

**Cách khối khác với các phương pháp**

Tuy nhiên, có một số khác biệt giữa các khối và phương thức.

Kiểm tra mã trong trình chỉnh sửa. Các capitalizephương pháp tận một từ, và chúng ta có thể tiếp tục gọi capitalizephương pháp theo tên. Chúng tôi có thể capitalize("matz"), capitalize("eduardo")hoặc bất kỳ chuỗi nào chúng tôi thích đối với nội dung trái tim của chúng tôi.

Tuy nhiên, khối mà chúng ta xác định (sau .each) sẽ chỉ được gọi *một lần* và trong ngữ cảnh của mảng mà chúng ta đang lặp lại. Nó xuất hiện chỉ đủ lâu để làm một số công việc each, sau đó biến mất vào màn đêm.

# method that capitalizes a word

def capitalize(string)

  puts "#{string[0].upcase}#{string[1..-1]}"

end

capitalize("ryan") # prints "Ryan"

capitalize("jane") # prints "Jane"

# block that capitalizes each string in the array

["ryan", "jane"].each {|string| puts "#{string[0].upcase}#{string[1..-1]}"} # prints "Ryan", then "Jane"

**Using Code Blocks**  
Một phương thức có thể nhận một khối làm tham số. Đó là những gì .eachđã làm trong suốt thời gian qua: lấy một khối làm tham số và thực hiện các công việc với nó! Bạn chỉ không nhận thấy vì chúng tôi không sử dụng dấu ngoặc đơn tùy chọn. Chúng tôi đang lén lút.

Truyền một khối cho một phương thức là một cách tuyệt vời để *trừu tượng hóa* các tác vụ nhất định khỏi phương thức và xác định các tác vụ đó khi chúng ta gọi phương thức. Trừu tượng là một ý tưởng quan trọng trong khoa học máy tính và bạn có thể nghĩ nó có nghĩa là “làm cho một cái gì đó đơn giản hơn”. Hãy tưởng tượng nếu khi bạn muốn đi săn nhà, bạn phải nói, "Em yêu, chúng ta hãy xem xét cấu hình của bê tông, ván ép và vách ngăn bằng nhựa vinyl." Điều đó thật điên rồ! Cũng giống như việc nói “house” đơn giản hóa việc liệt kê các thành phần của nó, việc sử dụng một khối để xác định nhiệm vụ bạn muốn phương thức (như .each) thực hiện sẽ đơn giản hóa nhiệm vụ đang thực hiện.

**Sorting**

Sắp xếp mảng là một vấn đề rất phổ biến trong khoa học máy tính, và được nghiên cứu kỹ lưỡng vì lý do đó. Có nhiều *thuật toán* — trình tự các bước được xác định rõ ràng — mỗi *thuật toán* đều có những đánh đổi và lợi thế riêng.

Thuật toán sắp xếp có thể là một giới thiệu tuyệt vời về khoa học máy tính như một ngành lý thuyết, nhưng bây giờ chúng ta sẽ tập trung vào cách sử dụng thư viện sắp xếp tích hợp của Ruby (thực hiện một vài thuật toán này).

Trong Ruby, có hai phương pháp sắp xếp, .sorthoặc sort!. Phương thức đầu tiên .sort, chỉ trả về một mảng đã sắp xếp trong khi vẫn để nguyên mảng ban đầu. Phương thức thứ hai .sort!, sửa đổi mảng thực tế.

**Foundations**

Nếu chúng tôi giao cho bạn năm cuốn sách và yêu cầu bạn sắp xếp chúng, sắp xếp theo tên sách, trên giá, bạn sẽ làm như thế nào?

Hầu hết các thuật toán sắp xếp đều giả định rằng chúng ta đang sắp xếp một mảng các mục, liên quan đến việc so sánh hai mục bất kỳ trong mảng và quyết định cái nào nên đến trước.

Đối với ví dụ về sách của chúng tôi, nếu đối với bất kỳ cặp sách nào, chúng tôi luôn chọn cuốn sách có tên đầu tiên trong bảng chữ cái, chúng tôi có thể đưa ra chiến lược sắp xếp. Các “chiến lược” này là các thuật toán sắp xếp được đề cập trong ví dụ trước. Công việc của chúng ta là quyết định cách so sánh hai mục trong danh sách và để Ruby quyết định sử dụng chiến lược nào.

**Toán tử so sánh kết hợp**

Chúng ta cũng có thể sử dụng một toán tử mới được gọi là *toán tử so sánh kết hợp* để so sánh hai đối tượng Ruby. Các toán tử so sánh kết hợp trông như thế này: <=>. Nó trả về 0nếu *toán hạng* đầu tiên (mục được so sánh) bằng thứ hai, 1nếu toán hạng đầu tiên lớn hơn toán hạng thứ hai và -1nếu toán hạng đầu tiên nhỏ hơn toán hạng thứ hai.

Một khối đó được truyền vào sortphương pháp phải trả lại một trong hai 1, 0hoặc -1. Nó sẽ trả về -1 nếu tham số khối đầu tiên phải đến trước tham số thứ hai, 1nếu ngược lại và 0nếu chúng có trọng số bằng nhau, nghĩa là cái này không đến trước cái kia ( *tức là* nếu hai giá trị bằng nhau)

book\_1 = "A Wrinkle in Time"

book\_2 = "A Brief History of Time"

book\_1 <=> book\_2