**Các kiểu dữ liệu: Numbers, Strings, Boolean**

Trong Ruby, thông tin (hoặc dữ liệu) của bạn có thể có nhiều loại khác nhau. Có ba **kiểu dữ liệu** trong Ruby mà chúng tôi quan tâm ngay bây giờ: **Numeric** (bất kỳ số nào), **Boolean** (có thể đúng hoặc sai) và **String** (các từ hoặc cụm từ như "I'm learning Ruby!").

**Biến**

Một trong những khái niệm cơ bản nhất trong lập trình máy tính là **biến** . Bạn có thể coi một biến là một từ hoặc tên chứa một giá trị duy nhất. Ví dụ: giả sử bạn cần số 25 từ ví dụ cuối cùng của chúng tôi, nhưng bạn sẽ không sử dụng nó ngay lập tức. Bạn có thể đặt một biến, chẳng hạn my\_num, để nắm bắt giá trị 25 và bám vào nó để sử dụng sau này, như sau:

my\_num = 25

**Toán học**

Ruby không giới hạn trong các biểu thức đơn giản của phép gán như my\_num = 100; nó cũng có thể làm tất cả các phép toán bạn đã học ở trường.

Có sáu toán tử số học mà chúng ta sẽ tập trung vào:

Phép cộng ( +)

Phép trừ ( -)

Phép nhân ( \*)

Bộ phận ( /)

Luỹ thừa ( \*\*)

Modulo ( %)

Những thứ duy nhất có thể trông kỳ lạ đối với bạn là lũy thừa và modulo. Luỹ thừa nâng một số (cơ số) lên lũy thừa của số kia (số mũ). Ví dụ, 2\*\*3là 8, từ 2\*\*3có nghĩa là "cho tôi 2 \* 2 \* 2" (2 nhân với nhau 3 lần). 3\*\*2là 9(3 \* 3), v.v.

Modulo trả về phần còn lại của phép chia. Ví dụ, 25 % 7sẽ là 4, vì 7 đi thành 25 ba lần với 4 còn lại.

**'puts' and 'print'**

Các printlệnh chỉ mất bất cứ điều gì bạn cho nó và in nó lên màn ảnh. puts(đối với “chuỗi đặt”) hơi khác: nó thêm một dòng mới (trống) sau thứ bạn muốn in. Bạn sử dụng chúng như thế này:

puts "What's up?"  
print "Oxnard Montalvo"

Không cần dấu ngoặc đơn hoặc dấu chấm phẩy!

**Mọi thứ trong Ruby là một đối tượng**

Bởi vì mọi thứ trong Ruby đều là một đối tượng (sẽ nói thêm về điều này sau), mọi thứ trong Ruby đều có một số khả năng tích hợp được gọi là **phương thức** . Bạn có thể coi các phương pháp là “kỹ năng” mà các đối tượng nhất định có. Ví dụ: chuỗi (từ hoặc cụm từ) có các phương thức tích hợp có thể cho bạn biết độ dài của chuỗi, đảo ngược chuỗi và hơn thế nữa.

**Phương thức '.length'**

Các phương thức được triệu hồi bằng cách sử dụng a .. Nếu bạn có một chuỗi "I love espresso"và lấy .lengthnó, Ruby sẽ trả về độ dài của chuỗi (nghĩa là số ký tự — chữ cái, số, khoảng trắng và ký hiệu). Kiểm tra nó ra:

**The '.reverse' Method**

Các .reversephương pháp được gọi là theo cùng một cách .length, nhưng thay vì hỏi Ruby để cho bạn biết bao lâu một chuỗi là, nó spits ra một phiên bản ngược của chuỗi bạn đã cho nó. Ví dụ,

"Eric".reverse

sẽ cho kết quả

"cirE"

**'.upcase' & '.downcase'**

Hãy thử một phương pháp nữa (er, *hai* phương pháp). Như bạn có thể đã đoán, các phương thức .upcasevà .downcasephương thức chuyển đổi một chuỗi thành TẤT CẢ TRƯỜNG HỢP LÊN LÊN hoặc tất cả các trường hợp viết thường, tương ứng.

**Multi-Line Comments**

You can write a comment that spans multiple lines by starting each line with a #, but there’s an easier way. If you start with =begin and end with =end, *everything* between those two expressions will be a comment. Take a look at this example:

**Getting Input**

variable\_name = gets.chomp

getslà phương thức Ruby nhận đầu vào từ người dùng. Khi nhận được đầu vào, Ruby sẽ tự động thêm một dòng trống (hoặc **dòng mới** ) sau mỗi bit đầu vào; chomploại bỏ dòng thừa đó. (Chương trình của bạn sẽ hoạt động tốt khi không có chomp, nhưng bạn sẽ có thêm dòng trống ở mọi nơi.)

**Printing the Output**

first\_name = "Kevin"  
puts "Your name is #{first\_name}!"

The code above will print Your name is Kevin!.

**Formatting with String Methods**

Tuyệt vời! Bây giờ chúng tôi đã có đầu ra của mình, nhưng như bạn có thể thấy, chúng tôi chưa sử dụng các phương thức chuỗi để viết hoa mọi thứ một cách chính xác.

print "This is my question?"  
answer = gets.chomp  
answer2 = answer.capitalize   
answer.capitalize!

1. Đầu tiên, chúng tôi giới thiệu một phương pháp mới capitalize, tại đây. Nó viết hoa chữ cái đầu tiên của một chuỗi và làm cho các chữ cái còn lại viết thường. Chúng tôi gán kết quả choanswer2
2. Dòng tiếp theo có thể trông hơi lạ, chúng tôi không gán kết quả của capitalizecho một biến. Thay vào đó, bạn có thể nhận thấy !ở cuối capitalize. Điều này sửa đổi giá trị chứa trong answerchính biến . Lần sau khi bạn sử dụng biến, answerbạn sẽ nhận được kết quảanswer.capitalize

**IF**

ifCâu lệnh của Ruby có một **biểu thức** , đây chỉ là một từ hoa mỹ để chỉ một thứ gì đó có giá trị đánh giá một trong hai truehoặc false. Nếu biểu thức đó là true, Ruby thực thi khối mã theo sau if. Nếu nó không đúng (nghĩa là false), Ruby sẽ không thực thi khối mã đó: nó sẽ bỏ qua nó và chuyển sang điều tiếp theo.

**Unless**

Đôi khi bạn muốn sử dụng luồng điều khiển để kiểm tra xem điều gì đó là *sai* , thay vì điều đó là đúng. Bạn có thể đảo ngược if/ else, nhưng Ruby sẽ làm cho bạn một điều tốt hơn: nó sẽ cho phép bạn sử dụng một unlesscâu lệnh.

Giả sử bạn không muốn ăn *trừ khi* đói. Đó là, trong khi bạn không đói, bạn viết chương trình, nhưng nếu bạn *đang* đói, bạn ăn. Bạn có thể viết chương trình đó bằng Ruby như thế này:

unless hungry  
  # Write some sweet programs  
else  
  # Have some noms  
end

**AND**

Các phép so sánh không phải là toán tử duy nhất có sẵn cho bạn trong Ruby. Bạn cũng có thể sử dụng **toán tử logic** hoặc **boolean** . Ruby có ba: và ( &&), hoặc ( ||), và không phải ( !). Toán tử boolean dẫn đến giá trị boolean: truehoặc false.

Toán tử boolean **và** , &&chỉ cho kết quả truekhi **cả hai** biểu thức ở hai bên của &&đều là true. Đây là cách &&hoạt động:

true && true # => true  
true && false # => false  
false && true # => false  
false && false # => false

**OR**

Ruby cũng có toán tử **hoặc** ( ||). Ruby ||được gọi là **bao hàm hoặc** vì nó đánh giá truethời điểm một hoặc biểu thức kia *hoặc cả hai* biểu thức đều đúng. Kiểm tra nó ra:

true || true # => true  
true || false # => true  
false || true # => true  
false || false # => false

**NOT**

Cuối cùng, Ruby có toán tử boolean **not** ( !). !tạo ra truecác giá trị falsevà ngược lại.

!true # => false  
!false # => true