1 Parts of speech

• ‘jump’
  – a verb? a noun?
• ‘you, jump!’
  – a noun, with a verb directed to it
• ‘you, say “yes, sir!”’
  – a noun (subject), verb, and another noun to modify the verb
  – we’re going to call that an argument

How about some rubyish examples?

• I.love(you)
• You.say(‘yes, sir!’)

And some real ruby?

• puts "hello"
  – puts is the method (verb), "hello" is the argument (adverb)

But this is a bit of a lie. This is shorthand for: Kernel.puts(“hello”)

wtf? Subject.verb(arguments)

More examples: - 1 + 1 - really, 1.+(1)

Recall that:
'hi'*5 => hihihihii

but

5*'hi' => TypeError: String can’t be coerced into Fixnum.

Let’s translate these to overly verbose Ruby:

• ‘hi’.*(5)
  – the ’hi’ does the doing (the multiplying), taking 5 as an argument
    * it interprets this as ‘make 5 copies of myself’

• 5.*(hi)
  – the 5 does the doing (the multiplying), taking ’hi’ as an argument
    * but it doesn’t know how to multiply itself by a string!

So the subject (in English) of the sentence does the doing. In Ruby, we’re going to mix this up and refer to the subject as the receiver because it is receiving a message telling it what to do.

2 What is ‘doing’?

Methods (functions, subroutines, procedures) apply to one subject (receiver). Try to avoid being confused by the fact that this receiver will often be something called an object, but this is not the same thing as the object (part of speech) in an English sentence. Methods take zero or more arguments (which are like adverbs)

s = Square.new
s.color(green)
s.draw

name = "Noah"
name.include?('o') => True
name.include?('x') => False

In Ruby, the receiver is the thing that does the doing. It gets a message telling it to hit the ball or print the answer. The receiver is responsible for implementing the method it’s asked to use; if it doesn’t know what to do, you’ll get an error.

Ruby has many hundreds of methods already. In the next couple of weeks you’ll learn how to create your own!
3 Method examples

• a = gets => (some result)
• a = Kernel.gets() => (some result)
  – takes no argument
  – if no argument, you may omit the ()
• puts "hi!" => (NO result, but performs output)
  – even with a single argument, you may omit the ()
• Kernel.puts("hi!") => same thing

So we have a subject (receiver), verb, some adverbs, all leading to some meaning (result). But sometimes there’s no meaning (puts), just an effect. Kind of like punching a wall.

• a = gets
• a = a.chomp
• a = gets.chomp
  – a = Kernel.gets().chomp()
    * this is kind of what the Java language is like
    * the . means apply the method to the right, to the result to the left
      · call Kernel.gets(), which gives us a string from the keyboard
      · take that resulting string, and call chomp() on it, which gives us a new string
      · assign that string to a

4 Conversion methods

Remember to_i, to_s, to_f? What do these do?

• 3.0.to_i => 3
  – 3.0 is the receiver, to_i is the method, 3 the result
  – why doesn’t it think .0 is a method? Ruby’s type inference is just a little bit smart.
• '3\n'.to_f => 3.0
  – 3\n (a String) is the receiver, to_f is the method, 3.0 the result

Why were we able to do puts a when a pointed to an Integer? - puts uses to_s implicitly - but only on the result of the argument - so you still need to_s in cases like puts ‘hi, ’ + name + ’, the answer is ’ + x.to_s - There is a nice shortcut for this - puts "hi, #{name}, the answer is #{x}" - the #{} calls to_s automatically - double quotes are required - double quotes let you do a few other things; we’ll get to that.

5 Naming variables

• Must start with a lower-case letter (a through z)
• Can contain the characters a through z, A through Z, digits 0 through 9, and - (dash, or minus).
• Be descriptive!
  – name
  – input_file
  – result
  – hypotenuse
• In well-known mathematical formulas, single letters are acceptable
  – c_squared = a**2 + b**2
  – e = m*c**2
  – force = m*a

6 Mad methods

6.1 String methods

upcase
downcase
length

6.2 Numeric methods

abs
round
floor
6.3 Math methods

Math::cos(x)
Math::sqrt(x)
Math::log(x)

6.4 Kernel methods

exit
puts
gets
rand
rand(100)
rand(2)