Ruby Expressions & Operators

CSCI400

29 August 2017

Color Key

- Clickable URL link
- Write down an answer to this for class participation
- Just a comment – don't confuse with yellow

Method Invocation

- `puts "yes"`
  - Global method, provided by `Kernel`
- `Math.sqrt 2` test
  - `Math` object, method `sqrt`
- `message.length`
  - Object `message`, method `length`, no args

Compare to other languages

Method Invocation

- `a.each { |x| puts x }`
  - Object `a`, method `each`, block arg
- `a[0] = a[1]`
  - `a.[](0)`, object `a`, method `[]=`, arg `0`
  - `a.[](1)`, object `a`, method `[]`, arg `1`
- `x + y`
  - `x.+(y)`

Compare to other languages

Assignment

- `lvalue = rvalue`
  - `lvalue` must be address, `rvalue` must be value

```
x = 1
x += 1 # 'abbreviated assignment pseudooperators'
x, y, z = 1, 2, 3 # parallel assignment
x = y = 0 # chained assignment, right-associative
MAX = 5; MAX = 6 # changing a 'constant' -- not
                 # really, but Ruby gives warning
```

||=

- Goal: assign array if it exists, otherwise assign a new one
  - `results = results || []`
- `results ||= []`
  - How does this work? (Recall: Short-circuit evaluation)
  - `lvalue` must be `nil` or `false`
- Example of a Ruby *idiom*
  - Programming idiom: means of expressing recurring construct or simple task. Important for fluency in a language.

More Parallel Assignment

```ruby
x, y = y, x       # swap
x = 1, 2, 3       # x => [1,2,3]
x, y = [1, 2]     # same as x = 1; y = 2
a, b, c = 1, 2    # a = 1; b = 2; c = nil
x, *y = 1, 2, 3 # x = 1, y = [2, 3]
*x, y = 1, 2, 3 # x = [1, 2]; y = 3
```

- * is called 'splat'; can only have one per assignment

Operators

- Typical associativity rules
  - Left $\rightarrow$ right, except assignment and `**`
  - Sometimes unary operators associate right $\rightarrow$ left
- Typical precedence
  - Parentheses
  - Unary operators
  - `**` (if language supports it)
  - `*`, `/`, `%`
  - `+`, `-`
- APL is different: all ops have equal precedence and associate right $\rightarrow$ left (can override these rules with parenthesis). Would this be more or less confusing?

Ambiguity

- Local variables and method names start with lowercase letter
- So...how does Ruby distinguish between the two?
- If prior assignment $\rightarrow$ variable. Otherwise $\rightarrow$ method invocation.

Conditional Expressions

- C, C++, Perl, Javascript, Ruby all have these
- Often use *ternary operator* (`?:`)

Example:

```
average = (count == 0) ? 0 : sum / count
# above is the same as:
if count == 0
    average = 0
else
    average = sum / count
```