Ruby Data Types

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

Array

Array literals/initialization

```
arr = [1,2,3]
arr2 = [[1,2], [3,4]]
arr3 = [8, "lcd", 0.1]
arr4 = (0..10).to_a # Range -> Array
empty1 = []
empty2 = Array.new
zeroes = Array.new(5, 0)
```

Array

- Arrays are **heterogeneous**
- # of elements
  - `arr.length`, `arr.size`
- Out of bounds access returns `nil`
- Element accessors are similar to strings
- Dynamic resizing
  - Assign past end of array
- `|`, `&` for union, intersection

Compare to Java/C++

More Arrays

- `words = %w(casualties of cool)`
  - `["casualties", "of", "cool"]`
- `<<` to append
- Useful functionality
  - `alphabet = ('A'..'Z').to_a`
  - `alphabet.each { |c| print c }`
- Other methods
  - `clear`, `empty?`, `compact!`, `sort`, `sort!`, `pop`, `push`, `reverse`, etc.

Symbol

- *Immutable*, interned string (only one copy)
- Commonly used as key in hash map
- Symbol table
  - Stores names of classes, methods, variables
  - More efficient than storing as string
  - Can be used with *reflection*
- Symbols preferred over strings as unique identifiers
- Methods available to convert between string and symbol

Get used to symbols, they're very common

Hashes

- AKA maps, associative arrays
- Best to use immutable objects as keys
  - Required sometimes, e.g. Python
- Not covered: hash codes
  - Hashes supported directly in Perl, Python, and Ruby; supported in class libraries of Java, C++, C#

Hashes

```ruby
colors1 = { :John => "blue", :Dave => "red" }
colors2 = { "John" => "blue", "Dave" => "red" }
colors3 = { John: => "blue", Dave: => "red" }

puts colors3[:Cyndi]

colors.each do |key, value|
    puts "#{key}'s favorite color is #{value}"
end
```

Array vs. Hash

- Which is better if need to access items in order?
- Which is useful for direct access?
- Hash: useful for ‘paired’ data
    - Similar to tuples (which Ruby doesn’t have built-in)

Range

- Purpose
  - Determine whether value is within range
  - Iteration
- Any object that implements `<=>` function
  - `<=>` similar to Java's `CompareTo` – why is this needed?
- Bounds
  - `1..10` includes `10`
  - `1...10` excludes `10`

Range

```
coldware = 1945..1989
coldware.include? birthdate.year

(1..3).to_a # parenthesis required, otherwise just
            # applies to 3
```

Subrange introduce in Pascal, also used in Modula-2 and Ada. Others?

Booleans, `nil`

- `TrueClass` (`FalseClass`) singleton – write as `true` (`false`)
    - `true != 1`, `false != 0`
- `nil` means 'no value'
    - Test directly (e.g. `x == nil`) or with `nil?` (e.g. `x.nil?`)

Numeric Types

- `FixNum`
  - `Int` operations that fit in a machine word
- `BigNum`
  - Used for larger integers (`FixNum` converted automatically)
- `Float`
  - Floating points values
- `Complex`
  - Real + imaginary
- `Rational`, e.g. `2/3`