CSE 341: Section 8

Yuma & Taylor
University of Washington

Overview

- Homework 5 check-in, early questions
- Enumerable types in Ruby
    - Arrays
    - Hashes
    - Ranges
- Blocks

Arrays

Arrays: definition, indexing

```
irb(main):001:0> a = [ 1, 2, 3 ]
=> [1, 2, 3]
irb(main):002:0> a[0]
=> 1
```

Arrays: reverse indexing

```
irb(main):003:0> a[-1]
=> 3
irb(main):004:0> a[-2]
=> 2
```

Arrays: “out-of-bounds” indexing

```
irb(main):005:0> a[10]
=> nil
irb(main):006:0> a[10] = 5
=> 5
irb(main):007:0> a
=> [1, 2, 3, nil, nil, nil, nil, nil, nil, nil, 5]
```

Arrays: dynamic assignment

```
irb(main):008:0> a[6] = "Hello"

=> "Hello"

irb(main):009:0> a

=> [1, 2, 3, nil, nil, nil, “Hello”, nil, nil, nil, 5]
```

Arrays: range slicing

```
irb(main):010:0> a[8..2] = ["CSE 341", "is great!"]
=> ["CSE 341", "is great!"]
irb(main):011:0> a
=> [1, 2, 3, nil, nil, nil, "Hello", nil, "CSE 341", "is
great!", nil, nil, 5]
```

Arrays: block initialization

```
irb(main):001:0> a = Array.new(10) { 0 }
=> [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

irb(main):002:0> a = Array.new(10) { |i| i ** 2 }
=> [0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

Arrays: additional uses

Arrays are an incredibly flexible and fundamental unit of idiomatic Ruby programming.

- Arrays can be used as sets: `|`, `&`, `-`, `#uniq`
- Arrays can be used as queues: `#push`, `#pop`
- Arrays can be used as stacks: `#shift`, `#unshift`

Lots more: https://ruby-doc.org/core-2.2.0/Array.html

Arrays: aliasing

Array assignment is *aliasing*, which means you have to be careful about mutable interactions, e.g.:

```
irb(main):001:0> a = [1, 2, 3]
irb(main):002:0> b = a
irb(main):003:0> a[2] = 4
```

```
irb(main):004:0> a
=> [1, 2, 4]
irb(main):005:0> b
=> [1, 2, 4]
```

Arrays: aliasing

Array assignment is *aliasing*, which means you have to be careful about mutable interactions, e.g.:

```
irb(main):001:0> a = [1, 2, 3]
irb(main):002:0> b = a.clone
irb(main):003:0> a[2] = 4
```

```
irb(main):004:0> a
=> [1, 2, 4]
irb(main):005:0> b
=> [1, 2, 3]
```

Blocks

Blocks

- Many methods take in *blocks*
  - Kind of like anonymous functions that can be passed as arguments to functions
- Similar to closures but not quite
  - Have lexical scope (uses environment where block was defined)
  - Cannot be assigned to variables (they are “second-class,” not “first-class”)

Blocks (example)

```
3.times { puts “hi” }
```

- `times` is a method of the `Fixnum` class that takes a block and executes it the number of times represented by the Fixnum (in this case, 3)
- `{ puts “hi” }` is a block that prints “hi”

Output:

```
    hi
    hi
    hi
    => 3
```

Blocks (implicit call argument)

```
irb(main):001:0> def my_method (x)
irb(main):001:1>    yield x + 1
irb(main):001:2> end
irb(main):002:0> my_method(1)                    # <- crash!
irb(main):002:0> my_method(1) { |x| puts x }
2
irb(main):002:0> my_method(1) { puts "Block called!" }
Block called!
```

Blocks (explicit call argument)

```
irb(main):001:0> def my_method (x, &block)
irb(main):001:1>    block.call(x + 1)
irb(main):001:2> end
irb(main):002:0> my_method(1)                 # <- crash!
irb(main):002:0> my_method(1) { |x| puts x }
2
irb(main):002:0> my_method(1) { puts “Block called!” }
Block called!
```

Each

- Similar to for-each loops in Java

```
irb(main):001:0> [1, 2, 5, 12].each {|i| puts (i*i)}
    1
    4
    25
    144
    => [1, 2, 5, 12]
```

Map

- Similar to map in SML

```
irb(main):001:0> a = Array.new(10) {|i| i }
=> [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

irb(main):002:0> a.map {|x| x * 2}
=> [0, 2, 4, 6, 8, 10, 12, 14, 16, 18]
```

Inject

- Similar to `fold` in SML

```
irb(main):001:0> a = Array.new(10) {|i| i }
=> [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

irb(main):002:0> a.inject(0) {|acc,elt| acc+elt }
=> 45
```

Select

- Similar to `filter` in SML

```
irb(main):001:0> a = Array.new(10) {|i| i }
=> [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

irb(main):002:0> a.select {|x| x > 7 }
=> [8, 9]
```

Conditionals

```
irb(main):001:0> a = Array.new(10) {|i| i }
=> [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

irb(main):002:0> a.any? {|x| x > 7 }
=> true

irb(main):003:0> a.all? {|x| x > 7 }
=> false

irb(main):004:0> a.all?   # if no block provided…
=> true                   # true iff every element is not false or nil
```

Hashes

Hashes

Arrays identify their elements by index, whereas Hashes identify their elements by name:

```
irb(main):001:0> h = { foo: "bar", baz: "quux" }

=> {:foo=>"bar", :baz=>"quux"}
```

Hashes: #[ ], #[ ]=

```
irb(main):001:0> h = {}
irb(main):002:0> h["foo"] = "bar"
irb(main):003:0> h["foo"]
=> "bar"
irb(main):004:0> h
=> {"foo"=>"bar"}
```

Hashes: #delete

- Delete keys with #delete

```
irb(main):001:0> h = { foo: "bar", baz: "quux" }
irb(main):002:0> h.delete(:foo)
irb(main):003:0> h.delete(:foo)
=> {:baz=>"quux"}
```

Hashes: #each

- Iterate keys with #each

```
irb(main):001:0> h = { foo: "bar", baz: "quux" }
irb(main):002:0> h.each { |k, v| puts "#{k} => #{v}" }
foo => bar
baz => quux
```