
CSE 341 AA: Section 8

Porter Jones

pbjones@cs.washington.edu

Office Hours: Thursdays 5:30 - 7:30pm





Ruby Arrays

- Very flexible and can be used in many different ways
- Widely used in Ruby programming for a variety of tasks

```
# one way to create an array
```

```
a = [1, 2, 3, 4]
```

```
# create a new array with a given size
```

```
b = Arrays.new(10)
```

```
# initialize it with a block!
```

```
c = Arrays.new(10) { |i| i * i }
```



Ruby Arrays: super dynamic and flexible

```
# dynamic types (of course)
```

```
a = [1, "hello", [2, 3], false]
```

```
# index out of bounds returns nil, negative wraps around
```

```
# the line below assigns last element to nil
```

```
a[-1] = a[10]
```

```
# assigning element outside size is perfectly fine
```

```
# fills in empty spaces with nil
```

```
a[20] = "way off the end"
```



Ruby Arrays: also not arrays

```
# Can be used as a set
```

```
a = [1, 2, 3, 3]
```

```
b = [2, 3, 4]
```

```
# & is set intersection, | is set union, - is subtraction
```

```
a & b # gives [2, 3]
```

```
a | b # gives [1, 2, 3, 4]
```

```
# & and | will automatically remove duplicates
```

```
# can also use .uniq to turn an array into a set
```

```
a.uniq # gives [1, 2, 3]
```



Ruby Arrays: still not arrays

Can be used as stacks and queues!

```
a = []
```

```
a.push 2
```

```
a.push 3
```

```
a.pop # gives 3
```

```
a.pop # gives 2
```

```
a.pop # gives nil
```

shift takes the first element off the array

```
a = [1, 2, 3]
```

```
a.shift # gives 1
```



Ruby Arrays: a few more things

```
# Can alias other arrays
```

```
a = [1, 2, 3]
```

```
b = a # b refers to the same array that a does
```

```
c = a.clone # c actually refers to a shallow copy of a
```

```
# Can splice arrays with arr[start_index, num_elements]
```

```
a = [1, 2, 3, 4, 5, 6, 7, 8, 9]
```

```
a[3, 3] # gives [4, 5, 6]
```

```
# Can also assign splices!!
```

```
a[3, 3] = [1] # a is now [1, 2, 3, 1, 7, 8, 9]
```



Ruby Hashes

```
# Creates empty hash, stores keys and values
```

```
h = {}
```

```
# Add records
```

```
h["best dessert"] = "ice cream"
```

```
h[true] = 32
```

```
# Get the keys and values for a hash
```

```
h.keys
```

```
h.values
```



Ranges

```
# Creates range of values 1 to 100  
(1..100)
```

```
# Ranges can be used in similar ways to arrays (duck typing)  
(1..100).each {|x| puts x }
```

```
# ...but they aren't arrays, no indexing!  
# can't do (1..100)[5]
```

```
# can turn them into arrays if you need to  
(1..100).to_a
```




Enumerables and blocks

- Arrays, Hashes, and Ranges are examples of enumerable objects
- Can use enumerable methods that take a block for performing certain functionalities across all elements in the enumerable



each

```
a = [1, 2, 3, 4]
```

```
sum = 0
```

```
# Note the lexical scope!
```

```
b = a.each {|x| sum += x }
```

```
# sum = 10
```

```
# each returns the enumerable it was called on
```

```
# so b = [1, 2, 3, 4]
```



map, select, inject

```
a = [1, 2, 3, 4]
```

```
# b will become [2, 4, 6, 8]
```

```
b = a.map {|x| x * 2}
```

```
# like filter, c will become [3, 4]
```

```
c = a.select {|x| x > 2}
```

```
# like fold, d will become 10
```

```
d = a.inject(0) {|acc, x| acc + x }
```



calling blocks

```
# use block_given? to know if given a block
# use yield to call the block
def example_block x
  if block_given?
    yield x
  else
    x
  end
end
```