

Hashes

CS105 : Saelee

Recap: Arrays

- Arrays allow us to store an indefinite quantity of data and refer to stored elements by “index”
 - Index N refers to the (N+1)th element of the array
 - e.g., [“man”, “canal”, “panama”][2] ⇒ “panama”
- Very useful for write programs where:
 - We are interested in positional access to multiple data
 - We need to store an arbitrary quantity of data and iterate over it later (and/or update it)

Arbitrary # of Inputs

```
arr = []  
loop do  
  input = gets.to_i  
  if input != -1  
    arr << input  
  else  
    break  
  end  
end  
  
puts arr.sort.inspect
```

```
> ruby test.rb  
10  
20  
9  
1  
2  
38  
57  
-1  
[1, 2, 9, 10, 20, 38, 57]
```

More Array Processing

```
arr = []
loop do
  input = gets.to_i
  if input != -1
    arr << input
  else
    break
  end
end

puts "Max: #{arr.max}"
puts "Min: #{arr.min}"
puts "Sum: #{arr.inject { |sum, val| val+sum } }"
```

```
> ruby test.rb
8
5
11
-1
Max: 11
Min: 5
Sum: 24
```

Positional "Buckets"

```
arr = []
1000.times do
  pos = rand(10)
  if arr[pos] == nil
    arr[pos] = 1
  else
    arr[pos] += 1
  end
end

puts arr.inspect
```

```
[117, 106, 100, 115, 76, 99, 106, 86, 94, 101]
```

Write a method to determine
the number of times each value
appears in an array

Enter “Hashes”

- Similar to arrays, but more general
 - Instead of “looking up” elements by index position, look up elements by **keys**
- Hashes allow for **key** \Rightarrow **value** mappings

Hash Syntax

- Empty hash: `{}`
- Hash literal:

```
{ 'name'      => 'Michael',  
  'course'   => 'CS 105',  
  'students' => 38 }
```
- Hash keys and values can be any type of object
- Hash access: `hash_var['name']`
- Hash assignment: `hash_var['key'] = value`
- Default value for unknown key = `nil`

Counting Values

```
def count_times(*vals)
  counts = {}
  vals.each do |val|
    if counts[val] == nil
      counts[val] = 0
    end
    counts[val] += 1
  end
  counts
end
```

```
>> count_times 5,2,2,1,3,4,8,1,5
=> {5=>2, 1=>2, 2=>2, 8=>1, 3=>1, 4=>1}
```

Some Hash Methods

- `[]`, `[]=`
- `default=`
- `length`
- `has_key?`
- `has_value?`

Using a Hash

```
>> hash = {"first" => "Michael"}
=> {"first"=>"Michael"}
>> hash["first"]
=> "Michael"
>> hash["last"]
=> nil
>> hash["last"] = "Lee"
=> "Lee"
>> hash.length
=> 2
>> hash.has_key? "last"
=> true
>> hash.has_key? "FIRST"
=> false
>> hash.has_value? "Michael"
=> true
>> hash.default = "Dunno"
=> "Dunno"
>> hash["middle"]
=> "Dunno"
>> hash
=> {"last"=>"Lee", "first"=>"Michael"}
```

Using Hash Defaults

```
def count_times(*vals)
  counts = {}
  counts.default = 0
  vals.each do |val|
    counts[val] += 1
  end
  counts
end
```

- If a key does not exist, looking it up returns the “default” value

Hash Iterators

- `each_key`
- `each_value`
- `each`, `each_pair`

Drawing a Histogram

```
def count_times(*vals)
  counts = {}
  vals.each do |val|
    if counts[val] == nil
      counts[val] = 0
    end
    counts[val] += 1
  end
  counts
end

def draw_hist(count_hash)
  count_hash.each_pair do |val, num_times|
    label = "#{val}:".rjust(10)
    puts label + ('#' * num_times)
  end
end
```

```
>> draw_hist(count_times(12,1,22,12,1,1,1,345,12,1))
```

```
    22: #
     1: #####
    12: ###
   345: #
```