Collections and Loops

Week 2 / Lesson 1

Agenda

- Review!
- Iteration - Loops
- Collections
  - Arrays
  - Hashes

Sync your repo

- You should do this regularly whenever working in your repository
- ...but at least once at the start of each class
- Run `git pull upstream master`

Reviewing the command line

- How do you change directories?
- How would you list the contents of a directory?
- How do you create a new folder?
- How do you create a new file?

Reviewing Git

- What is a Git repository
- What is a Git remote?
- What is a GitHub “fork”?
- What does a pull request do and how does it work?
- What are the steps to send changes from a local repository to your remote on GitHub?

General Ruby (Review)

- What does the `puts` method do?
- What are two built in methods that the String type has?
- What are the Ruby types for these values:
  - “Hello”
  - true
  - 235
  - 3.14

Ruby Quiz

- Define a method called ‘multiplier’ that takes two Numbers as arguments

- Within your method multiply the two numbers together

- If the result is greater than 50, return the string “Over 50”

- Otherwise return the string “Under 50”

- When finished call the method with different arguments like this: `multiplier 10, 5`

Iteration

Repetition

Repetition

Repetition

Iteration (Ruby-esque Loops)

Times Iterator

```
3.times do
    puts "going…"
end
puts "gone"

# going...
# going...
# going...
# gone
```

Iteration (Ruby-esque Loops)

.upto

```
1.upto(3) do |num|
    puts "#{num}.going"
end

# 1. going
# 2. going
# 3. going
```

Iteration (Ruby-esque Loops)

.downto

```
3.downto(1) do |guess|
    puts "You have #{guess} guesses left"
end

# You have 3 guesses left
# You have 2 guesses left
# You have 1 guesses left
```

Iteration (Ruby-esque Loops)

Less common in Ruby

- These loops are less common in Ruby, but good to know as a programmer.
    - X.times
    - upto
    - downto
- For additional help with syntax, see the Resources at the end of the slides.

Conditional Loops

```
count = 10
while count > 0
    puts "Looping"
    count -=1
end
```

```
count = 10
until count < 1
    puts "Looping"
    count -= 1
end
```

```
count = 10
loop do
    break if count < 1
    puts "Looping"
    count -= 1
end
```

Loops Exercise

- Open the file at Week2/Lesson1/Examples/loops.rb
- Follow the instructions in that file to create a small program that prints the song 99 Bottles of Beer on the Wall
- Before we start, which iterator that we discussed so far do you think would be the best choice for this job and why?

- a) while
- b) loop
- c) until
- d) .times
- e) .downto
- f) .upto

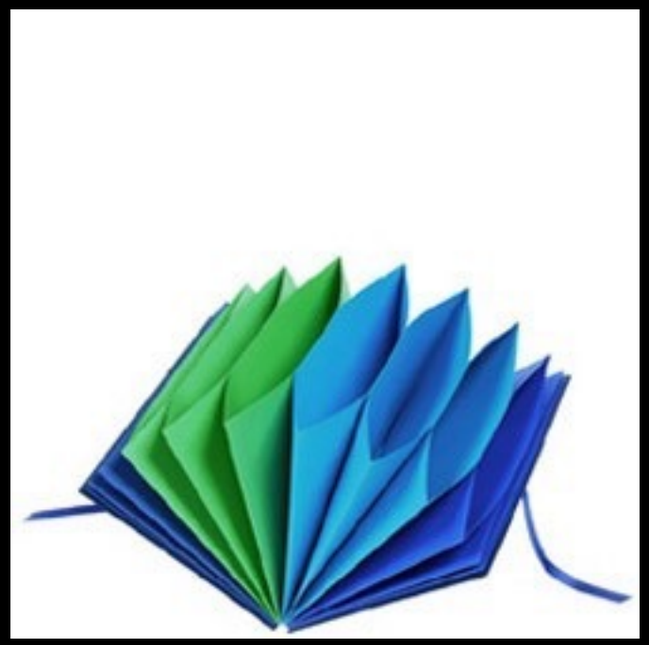
Iteration Recap

- Iteration in programming allows us to keep our code DRY
- Loops are used to repeat ~~lines of code~~ functionality
- ~~Common or~~ Ruby-esque loops are
  - .times
  - .upto
  - .downto
  - .each (we will see in a moment)

Collections

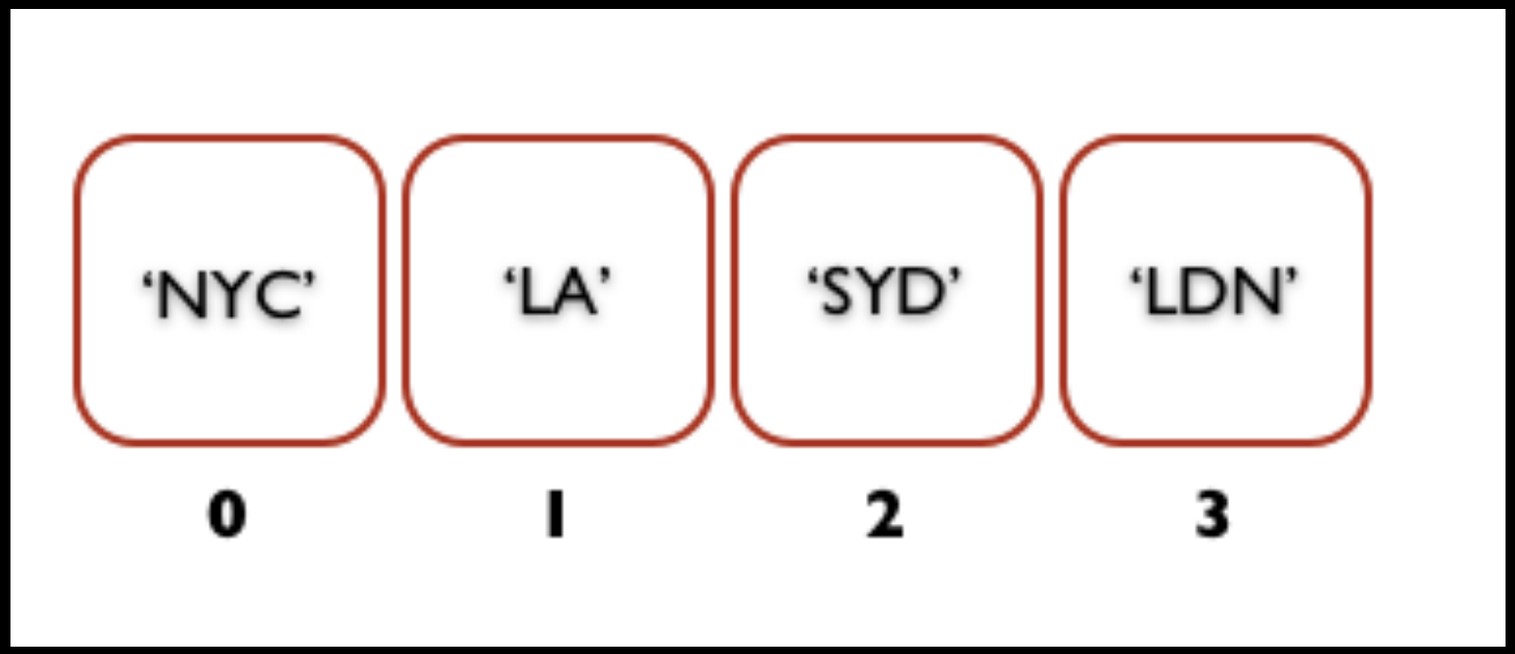
Arrays

They’re just like lists...



Arrays

Find by Index



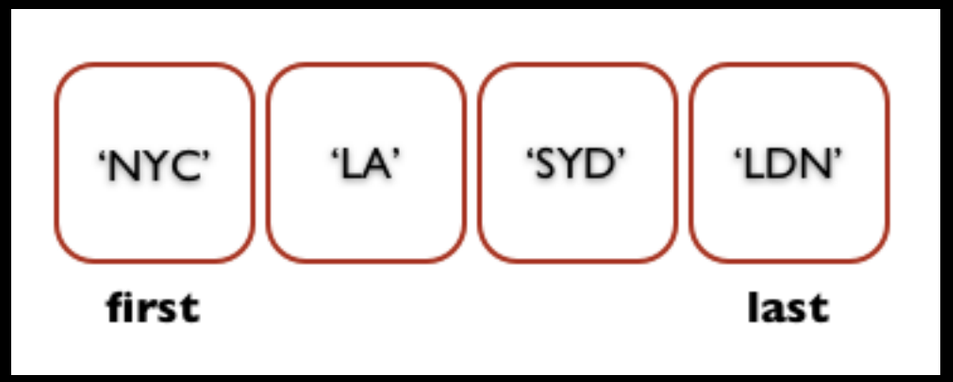
Arrays

Find by Index

```
my_array = ["NYC", "LA", "SYD", "LDN"]
my_array[0] #"NYC"
my_array[1] #"LA"
my_array[-1] #"LDN"
```

Arrays

Find by Position



Arrays

Find by Position

```
my_array = ["NYC", "LA", "SYD", "LDN"]
my_array.first #"NYC"
my_array.last #"LDN"
```

```
# In rails...
# Will not work in IRB
my_array = ["NYC", "LA", "SYD", "LDN"]
my_array.second
my_array.third
my_array.forth
my_array.fifth
my_array.forty_two # known as the reddit
```

Arrays

Array Methods

```
name = "Salman"
name.upcase
```

```
my_array = ["NYC", "LA", "SYD", "LDN"]
my_array.reverse
```

Arrays

- Open up IRB

Arrays - your turn

- Open up Week2/Lesson1/Examples/arrays.rb
- Copy the code into a new file inside of this lesson's homework folder within a folder named after yourself
- Complete the exercise with the person next to you

Arrays Recap

- A collection of data
- Can search an array by index or position
- Arrays are objects and therefore have methods.

BrEaK!

Collections

Hashes

- Often referred to as dictionaries
- Each entry in a hash needs a key and a value
- If you access a hash at a specific key, it will return the value at that key

Hashes

Find by key

```
ga_markets = {"NYC" => "New York
City", "LA" => "Los Angeles", "SYD"
=> "Sydney", "LDN" => "London"}
```

```
ga_markets["NYC"]
ga_markets["LA"]
ga_markets[“SYD”]
```

Hashes

Setting Values

```
user_hash = {}
user_hash["name"] = "Salman"
user_hash["favorite_color"] = "Green"
user_hash
user_hash
```

```
>> {"name"=>"Salman",
"favorite_color"=>"Green"}
```

Symbols

New Ruby type

- A symbol is a special type of object in ruby, used extensively
- It is always preceded by a colon
- Cannot contain **spaces** ~~or numbers~~
- Symbols are used because:
  - they are immutable and **take less memory**
  - they are easier to compare to other objects
  - they are cleaner in syntax
- Examples:
  - `:hello`
  - `:this_is_a_symbol`

Symbols

Primarily used as keys for hashes

```
ga_markets = {}
ga_markets = {:NYC => "New York City"}
ga_markets[:LA] = "Los Angeles"
ga_markets

>> {:NYC => "New York City", :LA => "Los Angeles"}
```

Hash

Methods

```
user = {:user_name => "SalmanAnsari", :email =>
"salman.ansari@gmail.com"}
user.has_key? :email #true
user.key? :email #true
user.include? :email #true
user.has_value? "SalmanAnsari" #true (note: extremely
inefficient!)
```

Hash

Ruby 1.9+ Alternate Syntax

```
user = {:user_name => "SalmanAnsari", :email =>
"salman.ansari@gmail.com"}
# becomes
user = {user: "SalmanAnsari", email: "salman.ansari@gmail.com"}
# a little bit more concise
# more closely matches JSON format
# considered an 'alternate' syntax, not a replacement
```

Collections

Array of Hashes

```
users = [
    {:user => "Salman Ansari", :role => "Instructor"},
    {:user => "Brooks Swinnerton", :role=> "TA"}
    {:user => "Brian Fountain", :role => "TA"]
]

# Alternate syntax for Ruby 1.9+

users = [
    {user: "Salman Ansari", role: "Instructor"},
    {user: "Brooks Swinnerton", role: "TA"},
    {user: "Brian Fountain", role: "TA"}
]
```

Iterating over Collections

.each

```
ga_markets = ["NYC", "LA", "SYD", "LDN"]

ga_markets.each {|market| puts market}
```

Lab Time

Collections

See Week2/Lesson1/Examples/hashes.rb

Recap

Iterating Over Collections

Homework

Continue work on Secret Number. Due next class (lesson 4)

Resources: Collections, Loops & APIs

Arrays

Creating Arrays

```
my_array = ["Apples", "Oranges", "Pears"]
["Apples", "Oranges", "Pears"]
my_array = Array.new
[]
Array.new(3)
[nil, nil, nil]
Array.new(3, "BEWD")
["BEWD", "BEWD", "BEWD"]
```

Assessing Elements

```
arr = ["NYC", "LDN", "LA", "SF", "BOS", "BER"]
arr[0]
arr[100]
arr[-3]
NYC
nil
SF
arr[2, 3] #=> [3, 4, 5]
["LA", "SF", "BOS"]
arr[1..4]
[LDN, LA, SF, BOS]
```

Resources: Collections, Loops & APIs

Hashes

```
    GA_Markets = { "New York City"=>"NYC",
"London"=>"LDN", "Los    Angeles"=>"LA", "San
Francisco"=>"SF", "Boston"=>"BOS",
"Berlin"=>"BER" }
```

```
    GA_Markets["London"]
```

"LDN"

```
    super_heros = { batman: "Bruce Wayne",
superman: "Clark Kent",  spiderman: "Peter
Parker"}
```

```
    super_heros[:superman]
```

"Clark Kent"

Loops

Iterator Loops

```
4.times do
  puts "This will be printed 4 times"
end
```

```
This will be printed 4 times
This will be printed 4 times
This will be printed 4 times
This will be printed 4 times
```

Each Loop

```
        # A list of GA Courses
        courses = [ "FEWD", "BEWD", "CSF" ]
```

```
    names.each do|n|
        puts "GA has a course on #{n}"
    end
```

GA has a course on FEWD
GA has a course on BEWD
GA has a course on CSF