Get Git

Exercises from last time:

- Full name greeting:
  - Write a program that asks for a person's first name, then middle and then last. Finally, it should greet the person using their full name.
- Bigger better favorite number:
  - Write a program that asks for a person's favorite number. Have your program add 1 to it and then suggest the result as a bigger and better favorite number.

ARRAYS

Ordered Collections

How To Make an Array?

How To Make an Array:

- rainbow = []

Adding Elements

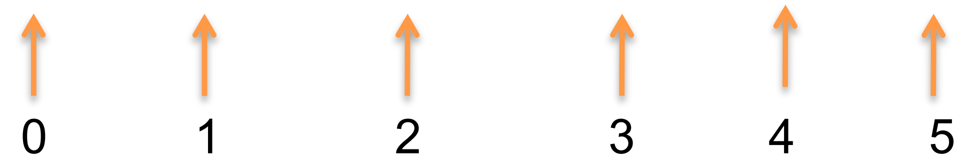
```
>> rainbow = []
=> []
>> rainbow << "red"
=> ["red"]
>> rainbow.push "orange"
=> ["red", "orange"]
>> rainbow + ['yellow']
=> ["red", "orange", "yellow"]
>> rainbow.concat ['green']
=> ["red", "orange", "green"]
```

Doesn’t change rainbow

Changes rainbow

Accessing Array Values

```
a = [ "a", "b", "c", "d", "e" ]
```



```
a[2] #=> "c”

a[6] #=> nil

a[1, 2] #=> ["b", "c”]

a[1..3] #=> ["b", "c", "d”]

a[1…3] #=> ["b", "c"]
```

Accessing an Array

```
>> rainbow[0]
=> "red"
>> rainbow[1]
=> "orange"
>> rainbow[2]
=> "green"
```

Multidimsional Arrays

```
>> multiplication = []
=> []
>> multiplication[0] = [0,0,0,0,0]
=> [0, 0, 0, 0, 0]
>> multiplication[1] = [0,1,2,3,4]
=> [0, 1, 2, 3, 4]
>> multiplication[2] = [0,2,4,6,8]
=> [0, 2, 4, 6, 8]
>> multiplication[3] = [0,3,6,9,12]
=> [0, 3, 6, 9, 12]
>> multiplication[4] = [0,4,8,12,16]
=> [0, 4, 8, 12, 16]
>> multiplication[1][3]
=> 3
>> multiplication[4][4]
=> 16
>> multiplication
=> [[0, 0, 0, 0, 0], [0, 1, 2, 3, 4], [0, 2, 4, 6, 8], [0, 3, 6, 9, 12], [0, 4, 8, 12, 16]]
```

Exercise

- Make a scheduling system.
  - Create an array called schedule
  - Each index of schedule represents a day of the week. 0 is Sunday, 1 is Monday, etc. Indexes should point to an array containing the names of employees scheduled for that day.
  - Employees:
    - “Sally”
    - “Todd”
    - “Kim”
    - “Joe”
  - Joe and Kim work on weekends
  - Sally and Todd work on Tuesday and Thursday
  - Sally and Kim work on Monday, Wednesday and Friday

Overview of Classes, Objects & Methods

Everything is an Object

- Everything in Ruby is an object.
- Objects are instances of Classes

```
>> 5.class
 => Fixnum

>> "hello".class
 => String

>> 4.5.class
 => Float

>> :hello.class
 => Symbol
```

Making a new Class

```
class Daisy
end
```

Instantiating the class:

```
d = Daisy.new
```

Methods

- In Ruby, strings, integers and arrays (objects) are like nouns.
- Methods are like verbs
- Other languages synonyms: ‘function’, ‘procedure’

Making a Method for Our Class

```
class Daisy
  def name
      “Gretta”
  end
end

d = Daisy.new
d.name
```

Exercise

- Make five new methods on the Daisy class:
  - num_petals should return 30 (as an integer)
  - color should return white (as a symbol)
  - smell should return delicious (as a string)
  - age should return 2 days (as a string)
  - height should return 10 inches (as a string)
- Instanciate your daisy class and try calling all your methods on it.

How Methods Work

name = “sally”

name.upcase

‘upcase’ is a method on the string object we put into the name variable. It is an *action* that the string knows about and can do.

Making Our Own Methods

Syntax:

Parameters

```
def add(x, y)
  x + y
end
```

Now we can call the add method:

```
>> add(5, 6)
=> 11
```

Arguments

Arguments/Parameters

- Technical note: parameters appear in method definitions; arguments appear in method calls

A Word About Scope

- Scope means where a variable is available in a program.
- Ruby has different types of variables that are more or less limited in access.
  - Local variables
  - Global variables
  - Instance variables
  - Class variables (we will talk about these later)

Local Variables

```
def add(x,y)
   number = x + y
end

def subtract(x,y)
   number = x - y
end
```

The variable `number` is only accessable within the *scope* of the method. The add method will not know about the `number` variable in the subtract method. If you try to access `number` outside of the method, you will get an error telling you that it is undefined.

Instance Variables

```
def add(x, y)
  @number = x + y
end

def subtract(x, y)
  @number = x – y
end
```

@number is an instance variable. It is accessable outside the methods, so when we call subtract, it assigns a new value to @number in ***both*** methods.

If Statements

```
if x == 54
   puts “x is 54”
elsif x == 63
   puts “x is 63”
else
   puts “x is some other number”
end
```

Exercise: Tic Tac Toe

Make a tic tac toe game. You'll probably want to use methods to accomplish it. I'd suggest using two methods, a `start_game` method and a `play` method. Remember that the two methods can share variables by using instance variables. Since tic tac toe is a two player game, you'll need to switch between players each turn…this brings *if statements* to mind. Oh, and just a hint, tic tac toe is a bit like a multidimensional array…

Introducing Test First Teaching

Exercise: Tic Tac Toe Using TFT

Homework

- Chapters 5, 7, 8
- Chapter 8.3 Building and sorting an array