IRON
HACK

Problem Solving with *Data Structures*

Data structures

A big part of programming is putting together associated values.

There are different ways to group up or **structure** values in Ruby.

Core Ruby Data structures

We can use Ruby classes, and build more complex data structures with them.

Arrays contain lists of values.

Hashes associate values to keys.

And... our own classes

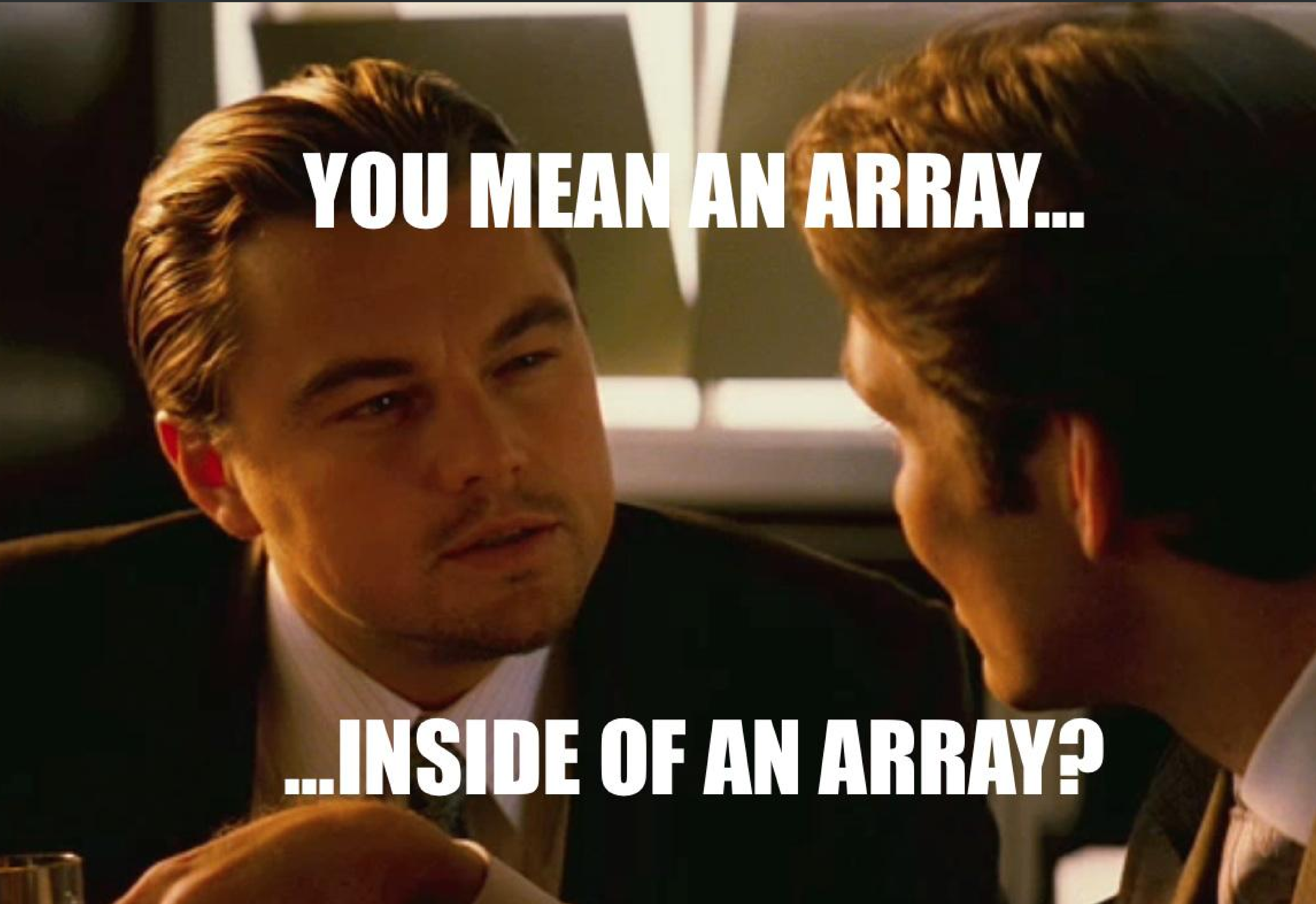
We can use our own **Classes** to group up values and also add methods.

These **Classes** can in turn use **Arrays**, **Hashes** or whatever you want!

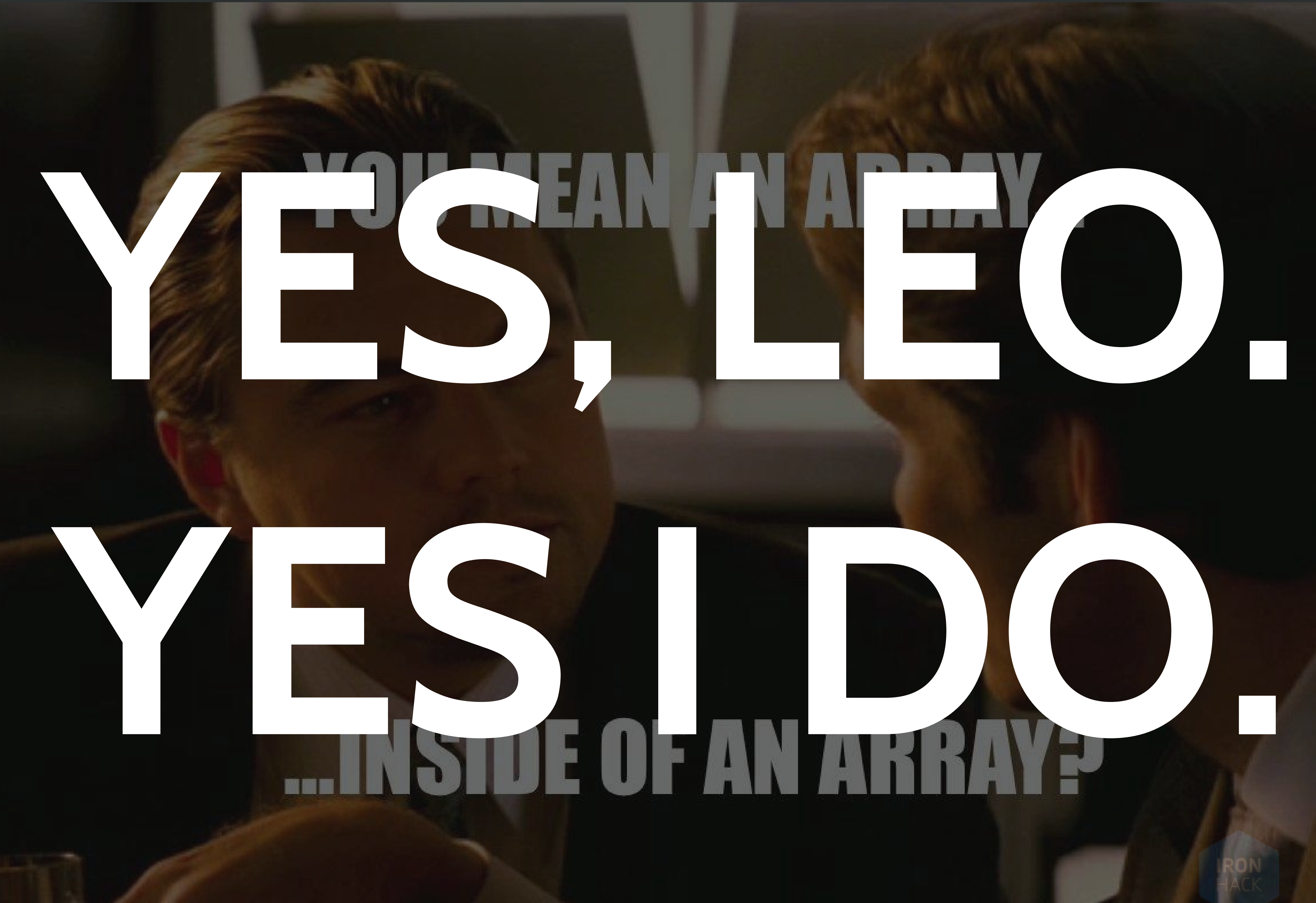
Data structures

But did you know you could put data structures **inside** other data structures?

```
arr = [
  { :arr => [ 1, 2, 3 ] },
  { :arr => [ 4, 5, 6 ] },
  { :arr => [ 7, 8, 9 ] }
]
```


YOU MEAN AN ARRAY...
...INSIDE OF AN ARRAY?

YES, LEO.
YES I DO.



Data structures

And arrays inside hashes.

And hashes inside of arrays.

And instances of objects inside of arrays inside of arrays.

You can't go on forever, but you can essentially go as deep as you need to.

Data structures

So we will go over the basics on dealing with **nested data structures**.

(just a fancy computer science term for structures inside of structures)

Data structures

The important part to remember throughout is that even if an object is inside an array or hash, it still has all the behavior that it would normally have.

What changes is how you *reference* or *access* that object.

Things inside arrays

If you've got an array, you know that you can access its items with the square brackets [ ].

```
arr = [
  "Thing #0",
  "Thing #1",
  "Thing #2",
  "Thing #3"
]

puts arr[2]
#=> "Thing #2"
```

Things inside arrays

Since the items inside this array are strings, `arr[2]` can do anything a string can do.

```ruby
arr = [
  "Thing #0",
  "Thing #1",
  "Thing #2",
  "Thing #3"
]

puts arr[2].size #=> 8
puts arr[2].upcase #=> "THING #2"
```

Arrays inside arrays

But what if there were other arrays inside that array?

```
arr = [
  [    "a",         "b",              "c"     ], # 0
  [     1,          2 ,                3      ], # 1
  [  "pizza",  "asparagus", "chicken wings"], # 2
  [ "coffee",      "tea",            "cola"   ]  # 3
  #     0           1              2
]
```

Arrays inside arrays

`arr[2]` has all the array methods and you can use `[]` to access its items by index.

```
arr = [
  [    "a",         "b",              "c"     ], # 0
  [     1,           2 ,               3      ], # 1
  [  "pizza",   "asparagus", "chicken wings"], # 2
  [ "coffee",      "tea",            "cola"   ]  # 3
  #     0            1               2
]
p arr[2]                #=> [ "pizza", "asparagus", "chicken
wings" ]
puts arr[2][0]          #=> "pizza"
puts arr[2][2]          #=> "chicken wings"
puts arr[2].size        #=> 3
puts arr[2].reverse     #=> [ "chicken wings", "asparagus",
"pizza" ]
```

Arrays inside arrays

If we take a look at `arr[0]`, the same thing applies.

```
arr = [
  [    "a",         "b",              "c"      ], # 0
  [     1,          2 ,               3        ], # 1
  [  "pizza",  "asparagus", "chicken wings"], # 2
  [ "coffee",      "tea",            "cola"    ]  # 3
  #     0           1                2
]
puts arr[0][1]       #=> "b"
puts arr[0][2]       #=> "c"
puts arr[0].size     #=> 3
puts arr[0].reverse  #=> [ "c", "b", "a" ]
```

Hashes inside arrays

Now what if the array was full of hashes? Maybe the the hashes contain coordinates.

```
arr = [
  { :lat => 25, :lng => 80, :name => "Miami"    }, # 0
  { :lat => 48, :lng => 2,  :name => "Paris"    }, # 1
  { :lat => 40, :lng => 3,  :name => "Madrid"   }, # 2
  { :lat => 18, :lng => 66, :name => "San Juan" }  # 3
]
```

Hashes inside arrays

`arr[2]` is now a hash. You can use it just like a hash variable.

```
arr = [
  { :lat => 25, :lng => 80, :name => "Miami"    }, # 0
  { :lat => 48, :lng => 2,  :name => "Paris"    }, # 1
  { :lat => 40, :lng => 3,  :name => "Madrid"   }, # 2
  { :lat => 18, :lng => 66, :name => "San Juan" }  # 3
]
puts arr[2]
#=> { :lat => 40, :lng => 3,  :name => "Madrid" }
```

Hashes inside arrays

`arr[2]` has all the hash methods and you can use `[]` to access its items by key.

```
arr = [
  { :lat => 25, :lng => 80, :name => "Miami"    }, # 0
  { :lat => 48, :lng => 2,  :name => "Paris"    }, # 1
  { :lat => 40, :lng => 3,  :name => "Madrid"   }, # 2
  { :lat => 18, :lng => 66, :name => "San Juan" }  # 3
]

puts arr[2][:name] #=> "Madrid"
puts arr[2][:lat] #=> 40
puts arr[2].empty? #=> false
puts arr[2].invert #=> { 40 => :lat, 3 => :lng, "Madrid"
=> :name }
```

Things inside hashes

If you've got a hash you know you access its items with the square brackets and its keys.

```
lemonade_revenue = {
  :monday    => 20,
  :tuesday   => 15,
  :wednesday => 5,
  :thursday  => 9,
  :friday    => 17
}

puts lemonade_revenue[:monday] #=> 20
```

Things inside hashes

Since the items inside this hash are numbers, `lemonade_revenue[:monday]` can do anything a numbers can do.

```ruby
lemonade_revenue = {
  :monday    => 20,
  :tuesday   => 15,
  :wednesday => 5,
  :thursday  => 9,
  :friday    => 17
}

puts lemonade_revenue[:monday] - 5 #=> 15
puts lemonade_revenue[:monday].zero? #=> false
```

Hashes inside hashes

But if there were hashes inside that hash...

```
lemonade_revenue = {
  :monday    => { :revenue => 20, :costs => 5 },
  :tuesday   => { :revenue => 15, :costs => 4 },
  :wednesday => { :revenue => 5 , :costs => 3 },
  :thursday  => { :revenue => 9 , :costs => 3 },
  :friday    => { :revenue => 17, :costs => 4 }
}
```

Hashes inside hashes

Same deal, just treat `lemonade_revenue[:monday]` as if it was a hash variable.

```ruby
lemonade_revenue = {
  :monday    => { :revenue => 20, :costs => 5 },
  :tuesday   => { :revenue => 15, :costs => 4 },
  :wednesday => { :revenue => 5 , :costs => 3 },
  :thursday  => { :revenue => 9 , :costs => 3 },
  :friday    => { :revenue => 17, :costs => 4 }
}
puts lemonade_revenue[:monday][:revenue] #=> 20
puts lemonade_revenue[:monday][:costs] #=> 5
```

Arrays inside hashes

You can even toss an array in there, just for fun.

```
lemonade_revenue = {
  :monday    => { :revenue => 20, :costs => 5, :feedback => [ "Good", "Too sweet" ]},
  :tuesday   => { :revenue => 15, :costs => 4, :feedback => [ "Meh"] },
  :wednesday => { :revenue => 5 , :costs => 3, :feedback => [ "Wow"] },
  :thursday  => { :revenue => 9 , :costs => 3, :feedback => [ "Best", "Not sweet"] },
  :friday    => { :revenue => 17, :costs => 4, :feedback => [] }
}
```

Arrays inside hashes

Just treat `lemonade_revenue[:monday][:feedback]` as an array variable.

```
lemonade_revenue = {
  :monday    => { :revenue => 20, :costs => 5, :feedback => [ "Good", "Too sweet" ]},
  :tuesday   => { :revenue => 15, :costs => 4, :feedback => [ "Meh"] },
  :wednesday => { :revenue => 5 , :costs => 3, :feedback => [ "Wow"] },
  :thursday  => { :revenue => 9 , :costs => 3, :feedback => [ "Best", "Not sweet"] },
  :friday    => { :revenue => 17, :costs => 4, :feedback => [] }
}

puts lemonade_revenue[:monday][:feedback][1] #=> "Too sweet"
puts lemonade_revenue[:tuesday][:feedback][0] #=> "Meh"
```

Anything inside anything

The point is that there can be a mix of things inside things.

You have to **identify what you are dealing with** on each level so you know what you need to do to access the next level.

Anything inside anything

Go slowly. Print things step by step.

```ruby
puts lemonade_finances
puts lemonade_finances[:monday]
puts lemonade_finances[:monday][:feedback]
puts lemonade_finances[:monday][:feedback][1]
```

Anything inside anything

If you aren't sure what you are dealing with, use the `class` method so know what type the value is.

```
puts lemonade_finances.class #=> Hash
puts lemonade_finances[:monday].class #=> Hash
puts lemonade_finances[:monday][:feedback].class #=> Array
puts lemonade_finances[:monday][:feedback][1].class #=> String
lemonade_finances[:monday][:costs].class #=> Fixnum
```

Exercise

Write two data structures that fulfill
the lines of code below .

Can they go that deep? There's only one way to find out!

```
puts hash[:wat][2][:wut][1][0][9][:bbq]
puts arr[0][5][:secret][:unlock][1]
```

Exercise

Create a class CarDealer which has an inventory of cars. We want to be able to quickly locate all Cars of a given brand to show them to customers.

We should be able to get all the cars of a brand with

```
car_dealer.cars("Ford")
```

We also need a method to print the inventory in a form like

Ford: Fiesta, Mustang

Seat: Ibiza, Leon, Toledo

Start with...

```
class CarDealer
  def initialize(inventory)
    @inventory = inventory
    # What should inventory look like? Is it
a hash? Is it an array?
  end
end
```