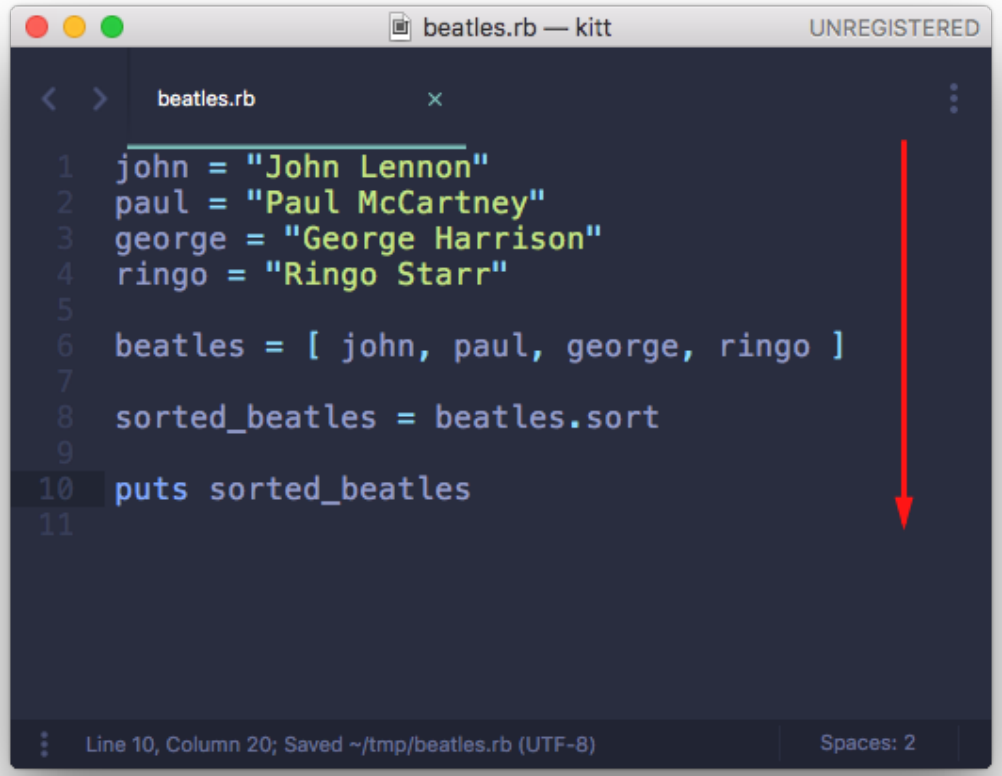
FLOW & ARRAY

BASIC FLOW

Top to bottom / line-by-line



LET'S CHANGE THIS FLOW

IF

```
if condition
  # code executed only when condition is "truthy"
end
```

truthy: anything that is different from `false` or `nil`.

IF !

```
if !condition
  # code executed only when condition is not "truthy"
end
```

UNLESS

```
unless condition
  # code executed only when condition is not "truthy"
end
```

LIVE-CODE - VOTING AGE

Can you vote?

```
puts "How old are you?"
age = gets.chomp.to_i

if age >= 18
  puts "you can vote!"
end
```

IF / ELSE

```
if condition
  # executed when condition is truthy
else
  # executed when condition is not truthy
end
```

LIVE-CODE: VOTING AGE (ENHANCED)

Can you vote? enhanced version

```
puts "what's your age?"

age = gets.chomp.to_i

if age >= 18
  puts "you can vote!"
else
  puts "too young to vote..."
end
```

TERNARY OPERATOR

```
condition ? code_when_truthy : code_when_falsey
```

Live-code: let's flip coins

```
puts "heads or tails?"
choice = gets.chomp
coin = ["heads", "tails"].sample

result = (choice == coin) ? "winner" : "loser"
puts "#{result}, that was #{coin}"
```

IF / ELSIF / ELSE

```
if condition
  # ...
elsif other_condition
  # ...
else
  # ...
end
```

LIVE-CODE: MORNING? AFTERNOON? NIGHT?

What's wrong with this program?

```
hour = Time.now.hour

if hour < 12
  puts "Good morning!"
elsif hour > 12
  puts "Good afternoon!"
elsif hour >= 20
  puts "Good night!"
else
  puts "Lunch time!"
end
```

The order of the conditions may matter!

CASE / WHEN / ELSE

Old school UI

```
puts "What do you want to do?"
action = gets.chomp

case action
when "read"
  puts "You are in READ mode"
when "write"
  puts "You are in WRITE mode"
when "exit"
  puts "Bye Bye"
else
  puts "Wrong action"
end
```

INLINE CONDITIONS

For short single-line statements

```
do_something if condition
do_something unless condition
```

```
number = gets.chomp.to_i
puts "your number #{number} is even!" if number.even?
```

MULTIPLE CONDITIONS - AND

&& is the logical AND

```
true  && true              #=> true
false && false             #=> false
true  && false             #=> false
false && true              #=> false
true  && false && true     #=> ?
```

MULTIPLE CONDITIONS - OR

|| is the logical OR

```
true  || true              #=> true
false || false             #=> false
true  || false             #=> true
false || true              #=> true
true  || false || true     #=> ?
```

LIVE-CODE - OPENING HOURS

When is the shop opened?

```ruby
hour = Time.now.hour

if (hour > 9 && hour < 12) || (hour > 14 && hour < 18)
  puts "The shop is opened!"
else
  puts "Sorry, the shop is closed..."
end
```

LOOPING WITH `WHILE`

```
while condition
  # executed while condition is truthy
end
```

LIVE-CODE: FIND THE RIGHT PRICE

Let's find the right price!

```
price_to_find = rand(1..5)
choice = nil      # variable initialization

while choice != price_to_find
  puts "How much (between $1 and $5)?"
  choice = gets.chomp.to_i
end

puts "You won! Price was #{price_to_find}$"
```

LOOPING WITH UNTIL

```
until condition
  # executed until condition is truthy
end
```

LIVE-CODE: FIND THE RIGHT PRICE (REFACTO)

```
price_to_find = rand(1..5)
choice = nil     # variable initialization

until choice == price_to_find
  puts "How much (between $1 and $5)?"
  choice = gets.chomp.to_i
end

puts "You won! Price was #{price_to_find}$"
```

LOOPING WITH FOR

```
for num in [1, 2, 3]
  puts num
end
```

ARRAYS

DEFINE AN ARRAY

```
empty_array = []                          # an empty array
beatles = ["john", "ringo", "seb"]        # array of 3 strings
```

GET AN ELEMENT USING INDEXES

```
beatles = ["john", "ringo", "seb"]
beatles[0]  #=> "john"
beatles[2]  #=> "seb"
beatles[8]  #=> nil
```

indexes start at 0

```
beatles = ["john", "ringo", "seb"]
# index =>    0        1        2
```

MODIFY AN ELEMENT

```
beatles = ["john", "ringo", "seb"]
beatles[2] = "george"
p beatles    # => ["john", "ringo", "george"]
```

APPEND AN ELEMENT

```
beatles = ["john", "ringo", "george"]
beatles << "paul"
p beatles    # => ["john", "ringo", "george", "paul"]
```

DELETE AN ELEMENT

By element value:

```
beatles.delete("john")
```

By index:

```
beatles.delete_at(2)
```

SIZE / COUNT / LENGTH

```
[1, 2, 3].size    #=> 3
[1, 2, 3].count   #=> 3
[1, 2, 3].length  #=> 3
```

EACH

each is your new best friend

```
beatles.each do |beatle|
 puts "#{beatle} is in the Beatles"
end

beatles.each { |beatle| puts "#{beatle} is in the Beatles" }
```

LOOPING WITH FOR ON INDICES

Let's loop through the **indices**

```
for index in 0...(musicians.size)
  musician = musicians[index]
  puts "#{index + 1} - #{musician}"
end

# => 1 - David Gilmour
#    2 - Roger Waters
#    3 - Richard Wright
#    4 - Nick Mason
```

LOOPING WITH FOR ON ELEMENTS

Or directly on the **elements**

```
for musician in musicians
  puts "Listen to #{musician}"
end

# => Listen to David Gilmour
#    Listen to Roger Waters
#    Listen to Richard Wright
#    Listen to Nick Mason
```

ITERATORS

#EACH

Live-code: Let's greet musicians one by one.

```
musicians.each do |musician|
  puts "Hello #{musician}!"
end

# => Hello David Gilmour!
#    Hello Roger Waters!
#    Hello Richard Wright!
#    Hello Nick Mason!
```

#EACH_WITH_INDEX

Live-code: Let's display an ordered list of musicians.

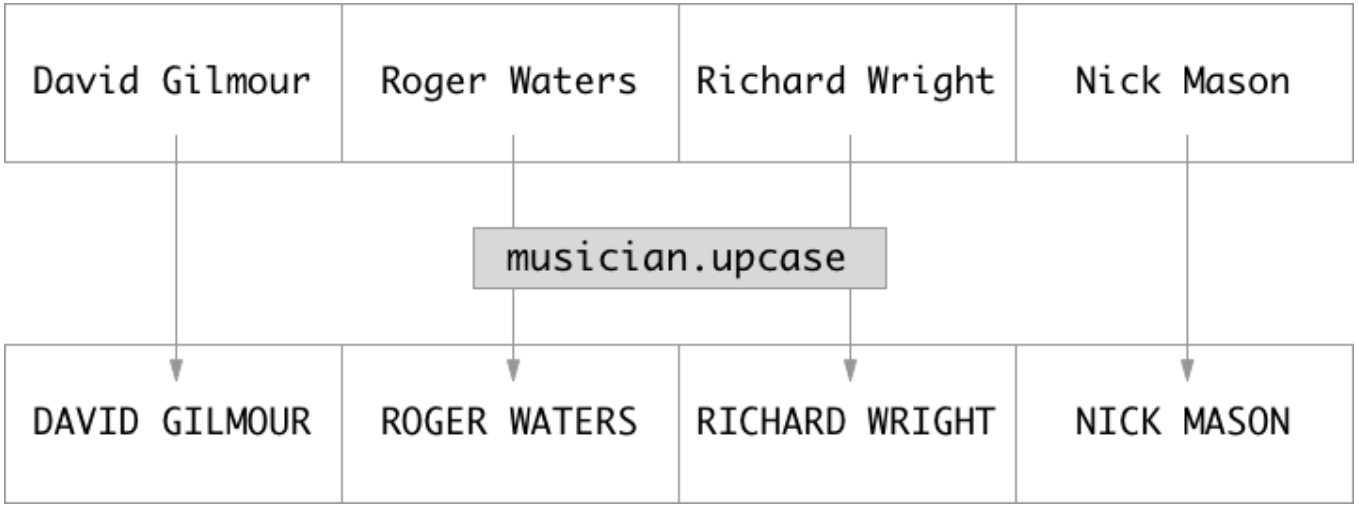
```ruby
musicians.each_with_index do |musician, index|
  puts "#{index + 1} - #{musician}"
end

# => 1 - David Gilmour
#    2 - Roger Waters
#    3 - Richard Wright
#    4 - Nick Mason
```

THERE'S MORE

#MAP

"Transform" one array to another one by applying some code on **each** element.



#map - UPCASED NAMES

Live-code: Let's build an array of upcased musician names.

```
musicians = ['David Gilmour', 'Roger Waters', 'Richard Wright

upcased_musicians = musicians.map do |musician|
  musician.upcase
end

p upcased_musicians
# => ['DAVID GILMOUR', 'ROGER WATERS', 'RICHARD WRIGHT', 'NIC
```

#map - FIRST NAMES

Live-code: Let's build an array of musician first names.

```
musicians = ['David Gilmour', 'Roger Waters', 'Richard Wright

musician_first_names = musicians.map do |musician|
  musician.split.first
end

p musician_first_names
# => ['David', 'Roger', 'Richard', 'Nick']
```

#count

- Count element of an array for which some code is **true**
- **Live-code**: count musicians starting with "R"

```
musicians = ['David Gilmour', 'Roger Waters', 'Richard Wright

r_musicians_count = musicians.count do |musician|
  musician.start_with?("R")
end

p r_musicians_count
# => 2
```

#SELECT

- Filter from an array elements for which some code is **true**.
- **Live-code**: extract musicians starting with "R"

```
musicians = ['David Gilmour', 'Roger Waters', 'Richard Wright

r_musicians = musicians.select do |musician|
  musician.start_with?("R")
end

p r_musicians
# => ['Roger Waters', 'Richard Wright']
```

AND MANY MORE

- http://www.ruby-doc.org/core/Array.html
- http://www.ruby-doc.org/core/Enumerable.html

(**Today's first challenge** is to pick the right methods from this documentation)

TO SUMMARIZE...

#EACH, #MAP, #COUNT...

- Are called **iterators**.
- **Enumerable** is a **module included** in the `Array` class.
- An iterator is just a **method of** Enumerable.

UNDERSTAND WHAT THEY RETURN

```
# .map       [] -> []
# .count     [] -> Integer
# .select    [] -> []
```

UNDERSTAND HOW THEY WORK

The block of code **has a different role** in each case.

```
musicians.each   { |musician| any_code_with(musician) }
musicians.map    { |musician| transform(musician)     }
musicians.select { |musician| condition_on(musician)  }
```

HAPPY HACKING!