Don't abuse `#reduce`

A love letter to `Enumerable`

Alexander Momchilov, RubyConf 2022 lightning talk, Nov 30

Don't abuse `#reduce`

A love letter to `Enumerable`

Alexander Momchilov, RubyConf 2022 lightning talk, Nov 30

shop

Alexander Momchilov, RubyConf 2022 lightning talk, Nov 30

When it comes to `for` loops, we get it.

```ruby
honour_roll_students = []

grade_cohorts.each do |grade_cohort|
  grade_cohort.each do |student|
    if student.honour_roll?
      honour_roll_students << student
    end
  end
end
```

```ruby
honour_roll_students = grade_cohorts
    .flat_map { |students| students.filter(&:honour_roll?) }
```

Sample problem

```
names = ["Joe", "Jane", "Bob", "Alice", "Alison", "Betty"]

names_by_initial = # ???


result: {
  "J" => ["Joe", "Jane"],
  "B" => ["Bob", "Betty"],
  "A" => ["Alice", "Alison"],
}
```

Sample problem

```
names = ["Joe", "Jane", "Bob", "Alice", "Alison", "Betty"]

names_by_initial = # ???
```

```
result: {
  "J" => ["Joe", "Jane"],
  "B" => ["Bob", "Betty"],
  "A" => ["Alice", "Alison"],
}
```

...but why do we do this stuff?

```ruby
names = ["Joe", "Jane", "Bob", "Alice", "Alison", "Betty"]

names_by_initial = names.reduce({}) do |accumulator, name|
  if accumulator.key?(name[0])
    accumulator.fetch(name[0]) << name
  else
    accumulator[name[0]] = [name]
  end

  accumulator
end
```

```
result: {
  "J" => ["Joe", "Jane"],
  "B" => ["Bob", "Betty"],
  "A" => ["Alice", "Alison"],
}
```

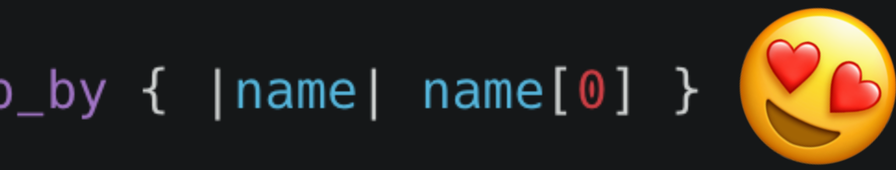
```ruby
names = ["Joe", "Jane", "Bob", "Alice", "Alison", "Betty"]

names_by_initial = names.each_with_object({}) do |name, accumulator|
  initial = name[0]
  if (group = accumulator[initial])
    group << name
  else
    accumulator[initial] = [name]
  end

  # no need to return the accumulator
end
```

#each_with_object is really just #each ... with an object.

```ruby
names = ["Joe", "Jane", "Bob", "Alice", "Alison", "Betty"]

names_by_initial = {}

names.each do |name|
  initial = name[0]
  if (group = names_by_initial[initial])
    group << name
  else
    names_by_initial[initial] = [name]
  end
end
```

Enumerable#group_by to the rescue!

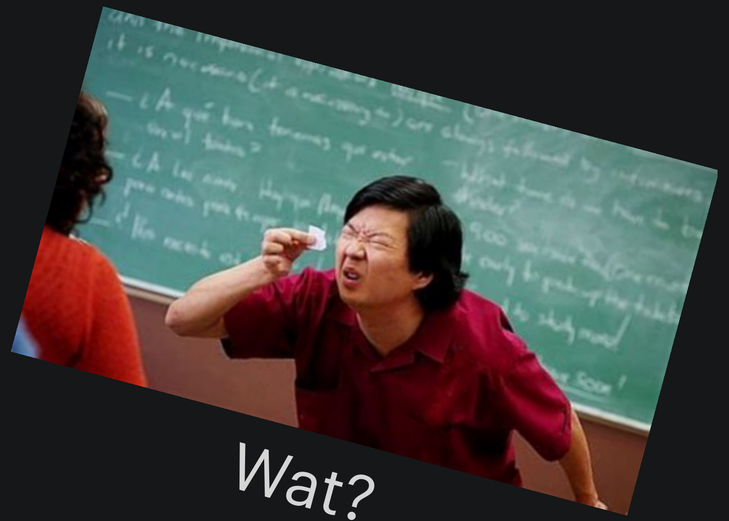
```ruby
names = ["Joe", "Jane", "Bob", "Alice", "Alison", "Betty"]

names_by_initial = names.group_by { |name| name[0] }
```



```ruby
names = ["Joe", "Jane", "Bob", "Alice", "Alison", "Betty"]

names_by_initial = names.reduce({}) do |accumulator, name|
  initial = name[0]
  if (group = accumulator[initial])
    group << name
  else
    accumulator[initial] = [name]
  end

  accumulator
end
```

```ruby
names = ["Joe", "Jane", "Bob", "Alice", "Alison", "Betty"]

names_by_initial = names.reduce({}) do |accumulator, name|
  initial = name[0]
  if (group = accumulator[initial])
    group << name
  else
    accumulator[initial] = [name]
  end

  accumulator
end
```

```ruby
names = ["Joe", "Jane", "Bob", "Alice", "Alison", "Betty"]

names_by_initial = names.reduce({}) do |accumulator, name|
  initial = name[0]
  if (group = accumulator[initial])
    group << name
  else
    accumulator[initial] = [name]
  end

  accumulator
end
```



```ruby
module MyEnumerable
  def include?(needle)
    reduce(false) { |acc, x| acc || x == needle }
  end
end
```

```ruby
module MyEnumerable
  def none?(&predicate)
    reduce(true) { |acc, b| acc && !predicate.call(b) }
  end

  def any?(&predicate)
    reduce(false) { |acc, b| acc || predicate.call(b) }
  end

  def all?(&predicate)
    reduce(true) { |acc, b| acc && predicate.call(b) }
  end
end
```

You can *build* up values

```ruby
module MyEnumerable
  def to_a
    reduce([]) { |acc, e| acc + [e] }
  end

  def to_h
    reduce({}) { |acc, e| acc.merge({ e.first => e.last }) }
  end
end
```

Type Signatures Preview

Show type signatures generated automatically by RBS.

Enable Type Signatures

Instance Methods

all?
any?
chain
chunk
chunk_while
collect
collect_concat
compact
count
cycle
detect
drop
drop_while
each_cons
each_entry
each_slice

Enumerable

Module

What's Here

Module Enumerable provides methods that are useful to a collection class for:

- Querying
- Fetching
- Searching
- Sorting
- Converting
- And more....

Methods for Querying

These methods return information about the Enumerable other than the elements themselves:

include?, member?
Returns `true` if self == object, `false` otherwise.
all?
Returns `true` if all elements meet a specified criterion; `false` otherwise.

Enumerable