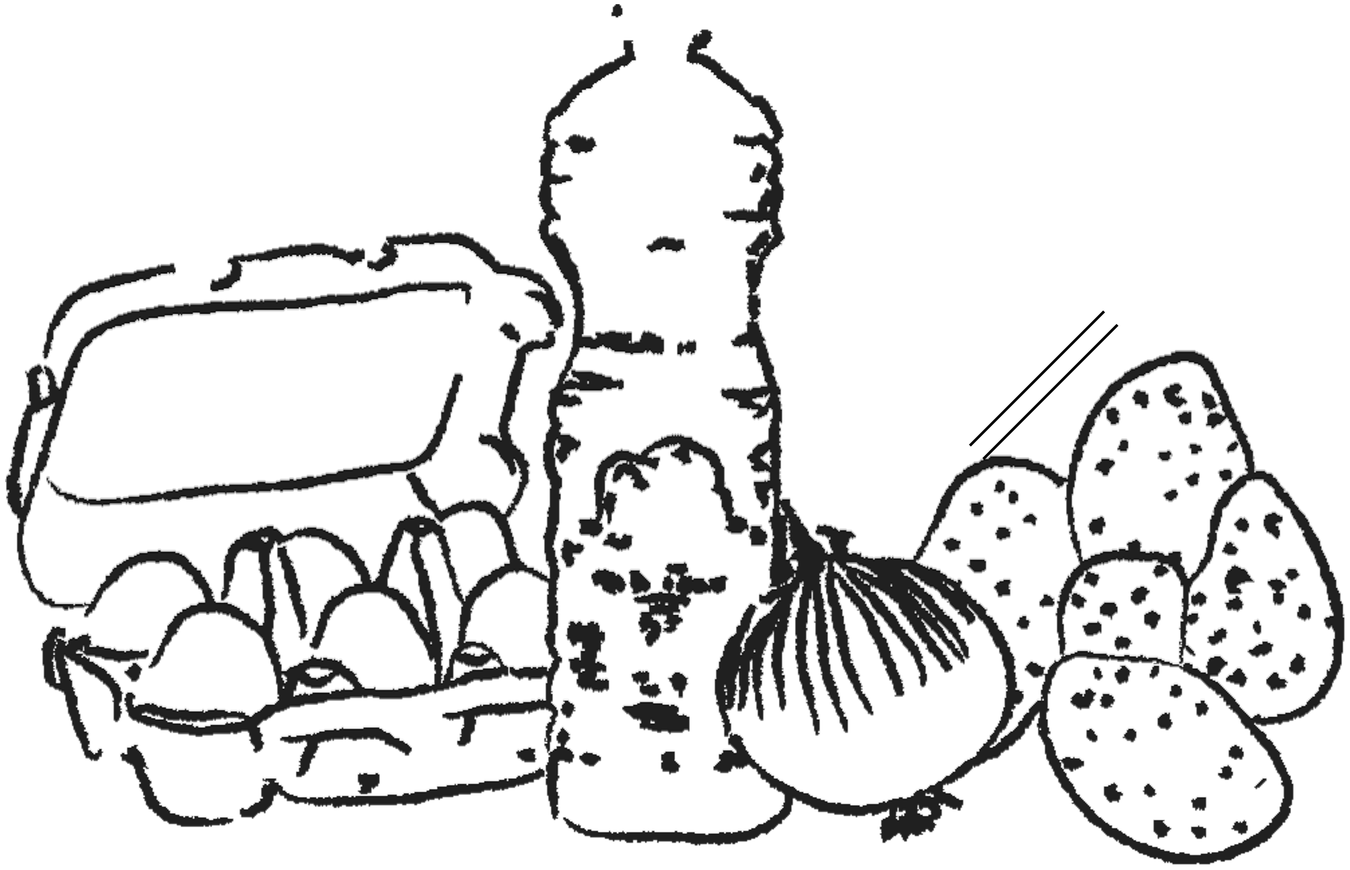
Loops & Iterators

Coding a Spanish Omelette

:{) Codaisseur

@miriamtocino

What Do We Need



Ingredients:

5 potatoes

6 eggs

1 onion

Olive oil

Salt

What They Do

A loop is the **repetitive execution**
of a piece of code.

for a given amount of repetitions
or
until a certain condition is met

Simple Loop

Looping is so simple

The simplest way to create a loop in Ruby.

Simple Loop

A **loop** will execute any code within the block until you manually intervene with **Ctrl+c**.

```
$ ruby recipe.rb

Cooking a Spanish Omelette
Cooking a Spanish Omelette
Cooking a Spanish Omelette
Cooking a Spanish Omelette
Cooking a Spanish Omelette
Cooking a Spanish Omelette
Cooking a Spanish Omelette
Cooking a Spanish Omelette
Cooking a Spanish Omelette
. . .   Interrupt with Ctrl+C
```

recipe.rb

```ruby
loop do
  puts "Cooking a Spanish Omelette"
end

loop { puts "Cooking a Spanish Omelette" }
```

Controlling Loop Execution - Break

break allows us to exit a loop at any point,
so any code after a **break** will not be executed.

```
$ ruby recipe.rb

Scrape potato 1.
```

recipe.rb

```ruby
counter = 0
loop do
  counter = counter + 1
  puts "Scrape potato #{counter}."
  break        # this will exit the loop
end
```

Controlling Loop Execution - Conditions

We can also include **conditions** within a loop,
to make it accomplish what we wish.

```
$ ruby recipe.rb

Scrape potato 1.
Scrape potato 2.
Scrape potato 3.
Scrape potato 4.
Scrape potato 5.
```

recipe.rb

```ruby
counter = 0
loop do
  counter = counter + 1
  puts "Scrape potato #{counter}.”
  if counter == 5
    break
  end
end
```

Controlling Loop Execution - Next

next skips the rest of the current iteration and start executing the next iteration.

```
$ ruby recipe.rb

Scrape potato 1.
Scrape potato 2.
Scrape potato 4.
Scrape potato 5.
```

recipe.rb

```ruby
counter = 0
loop do
  counter = counter + 1
  if counter == 3
    next    # skip rest of iteration
  end
  puts "Scrape potato #{counter}.”
  if counter == 5
    break
  end
end
```

While Loop

Execute, add, repeat

```
Repeat action while certain condition is true.
```

While Loop

As soon as **condition stops being true**,
the while loop stops.

```
$ ruby recipe.rb

Cut potato 1 in slices.
Cut potato 2 in slices.
Cut potato 3 in slices.
Cut potato 4 in slices.
Cut potato 5 in slices.
```

recipe.rb

```ruby
counter = 0
while counter < 5
  counter = counter + 1
  puts "Cut potato #{counter} in slices."
end
```

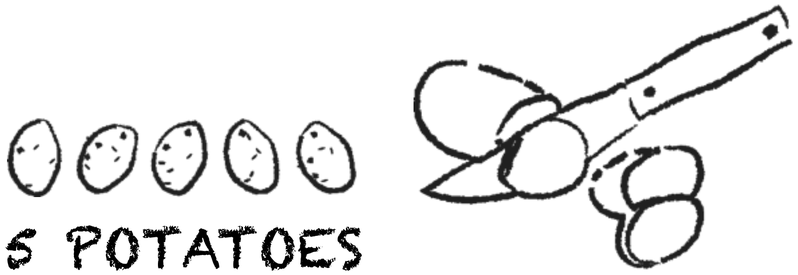
Refactoring Assignment Operators

Nice way to say the same thing
but with less typing.

```
counter = counter + 1
counter += 1

counter = counter - 1
counter -= 1

counter = counter * 1
counter *= 1

counter = counter / 1
counter /= 1
```

recipe.rb

```
counter = 0
while counter < 5
  counter += 1
  puts "Cut potato #{counter} in slices."
end
```



For Loop

Fixed number of times

Looping over a **collection of elements**.

For Loop with Range

`for` loops have a definite end,
since loops over a **finite number** of elements.

```
$ ruby recipe.rb

Add potato 1 to the pan.
Add potato 2 to the pan.
Add potato 3 to the pan.
Add potato 4 to the pan.
Add potato 5 to the pan.
```

recipe.rb

```
for counter in 1...6
  puts "Add potato #{counter} to the pan."
end
```

For Loop with Array

`for` loops also work with `arrays`.

```
$ ruby recipe.rb

Add potato 1 to the pan.
Add potato 2 to the pan.
Add potato 3 to the pan.
Add potato 4 to the pan.
Add potato 5 to the pan.
```

recipe.rb

```ruby
potatoes = [1, 2, 3, 4, 5]

for potato in potatoes
  puts "Add potato #{potato} to the pan."
end
```

Arrays

Ordered lists of items

An array is a collection of items
ordered from first to last.

Arrays

An **array** is an ordered list of elements of any type placed between **brackets []**.

arrays.rb

```
empty_array   = [ ]
string_array  = [ "potatoes", "eggs", "onion", "oil", "salt" ]
number_array  = [ 88, 54, 13, 6, 8 ]
mixed_array   = [ 88, "potatoes", [ 4, 5 ], 0.16 ]
```

Finding Elements within Arrays

To find the first and the last elements of an array, we can just use the **first** and **last** methods.

arrays.rb

```
ingredients = [ "potatoes", "eggs", "onion", "oil", "salt" ]

ingredients.first
=> "potatoes"

ingredients.last
=> "salt"
```

Finding Elements within Arrays

Arrays are what we call **indexed lists**.
Reference any element by its **index** (location) number.

arrays.rb

```
ingredients = [ "potatoes", "eggs", "onion", "oil", "salt" ]
                      0        1       2      3      4
ingredients[2]
=> "onion"

ingredients[3]
=> "oil"
```

Programmers start counting by **0**.

While Loop & Arrays

We can access **array indexes** within a **while loop**.

```
$ ruby recipe.rb

0: potatoes
1: eggs
2: onion
3: oil
4: salt
```

recipe.rb

```ruby
ingredients = [ "potatoes", "eggs",
                "onion", "oil", "salt" ]
index = 0

while index < ingredients.length
  puts "#{index}: #{ingredients[index]}"
  index += 1
end
```

Modifying Arrays

Use the pop method
to remove the last item of an array.

arrays.rb

```
ingredients = [ "potatoes", "eggs", "onion", "oil", "salt" ]

ingredients.pop
=> "salt"

ingredients
=> [ "potatoes", "eggs", "onion", "oil" ]
```

Modifying Arrays

Use the **push** method
to add any item to an array.

arrays.rb

```
ingredients = [ "potatoes", "eggs", "onion", "oil", "salt" ]

ingredients.push("salt")
=> [ "potatoes", "eggs", "onion", “oil”, "salt" ]

# Or using the shovel operator

ingredients << "salt"
=> [ "potatoes", "eggs", "onion", “oil”, "salt" ]
```

Each Method

Powerful iterator

The Ruby Way to iterate over a collection of items.

What Is An Iterator

Iterators are methods that
naturally loop over a given set of data.

They allows you to operate on
each element in the collection.

Each Iterator Over Array

recipe.rb

```ruby
ingredients = [ "potatoes", "eggs", "onion", "oil", "salt" ]

ingredients.each { |ingredient| puts "You need #{ingredient}."}
```

You can use the **each** method to iterate over an **array**.

```
$ ruby recipe.rb

You need potatoes.
You need eggs.
You need onion.
You need oil.
You need salt.
```

Each Iterator Over Array

recipe.rb

```ruby
ingredients = [ "potatoes", "eggs", "onion", "oil", "salt" ]

ingredients.each { |ingredient| puts "You need #{ingredient}."}
```

```
$ ruby recipe.rb

1) You need potatoes.
2) You need eggs.
3) You need onion.
4) You need oil.
5) You need salt.
```

recipe.rb

```ruby
ingredients = [ "potatoes", "eggs",
                "onion", "oil", "salt" ]
counter = 1

ingredients.each do |ingredient|
  puts "#{counter}) You need #{ingredient}."
  counter += 1
end
```

Hashes

Associative collections

A hash is a collection of key/value pairs
storing the association between each key and value.

Hashes

Entries in a hash are often referred to as **key-value pairs**, which creates an **associative representation** of data.

hashes.rb

```ruby
old_syntax_hash = { :name => 'potato' }

# OR

new_syntax_hash = { name: 'potato' }
```

hashes.rb

```ruby
ingredient = { name: 'potato', quantity: 5, unit: 'piece' }
```

Hashes

Add a new key-value pair to existing hash.

hashes.rb

```
ingredient = { name: 'potato', quantity: 5, unit: 'piece' }

ingredient[:origin] = 'Spain'

ingredient
=> { name: 'potato', quantity: 5, unit: 'piece', origin: 'Spain' }
```

Hashes

Remove existing key-value pair to existing hash.

hashes.rb

```
ingredient = { name: 'potato', quantity: 5, unit: 'piece' }

ingredient.delete(:unit)

ingredient
=> { name: 'potato', quantity: 5 }
```

Hashes

Retrieve a piece of information from a hash.

hashes.rb

```
ingredient = { name: 'potato', quantity: 5, unit: 'piece' }

ingredient[:name]
=> 'potato'

ingredient[:quantity]
=> 5

ingredient[:unit]
=> 'piece'
```

Iterating Over Hashes

We can use the **each** method to iterate over hashes (like with arrays) but with small differences.

```
$ ruby recipe.rb

Ingredient's name is potato.
Ingredient's quantity is 5.
Ingredient's unit is piece.
```

recipe.rb

```ruby
ingredient = { name: 'potato',
               quantity: 5,
               unit: 'piece' }

ingredient.each do |key, value|
  puts "Ingredient's #{key} is #{value}."
end
```

Hashes vs. Arrays

Does this data need to be
associated with a specific label?

Hash

Does order matter?

Array

Combining Hashes & Arrays

In some cases it will make sense to combine arrays and hashes.

```
$ ruby recipe.rb
Ingredient's name is potatoes.
Ingredient's quantity is 5.
***
Ingredient's name is eggs.
Ingredient's quantity is 6.
***
Ingredient's name is onion.
Ingredient's quantity is 1.
***
Ingredient's name is oil.
Ingredient's quantity is some.
***
Ingredient's name is salt.
Ingredient's quantity is some.
```

recipe.rb

```ruby
ingredients = [
  { name: 'potatoes', quantity: 5 },
  { name: 'eggs', quantity: 6 },
  { name: 'onion', quantity: 1 },
  { name: 'oil', quantity: 'some' },
  { name: 'salt', quantity: 'some' },
]

ingredients.each do |ingredient|
  ingredient.each do |key, value|
    puts "Ingredient's #{key} is #{value}."
  end
  puts "***"
end
```

Combining Hashes & Arrays

Remember how to access each of the values in a hash?

```
$ ruby recipe.rb
We need 5 potatoes.
We need 6 eggs.
We need 1 onion.
We need some oil.
We need some salt.
```

recipe.rb

```ruby
ingredients = [
  { name: 'potatoes', quantity: 5 },
  { name: 'eggs', quantity: 6 },
  { name: 'onion', quantity: 1 },
  { name: 'oil', quantity: 'some' },
  { name: 'salt', quantity: 'some' },
]

ingredients.each do |ingredient|
   puts "We need #{ingredient[:quantity]}
         #{ingredient[:name]}."
end
```

The End