Loops and Collections

Coding a Spanish Omelet

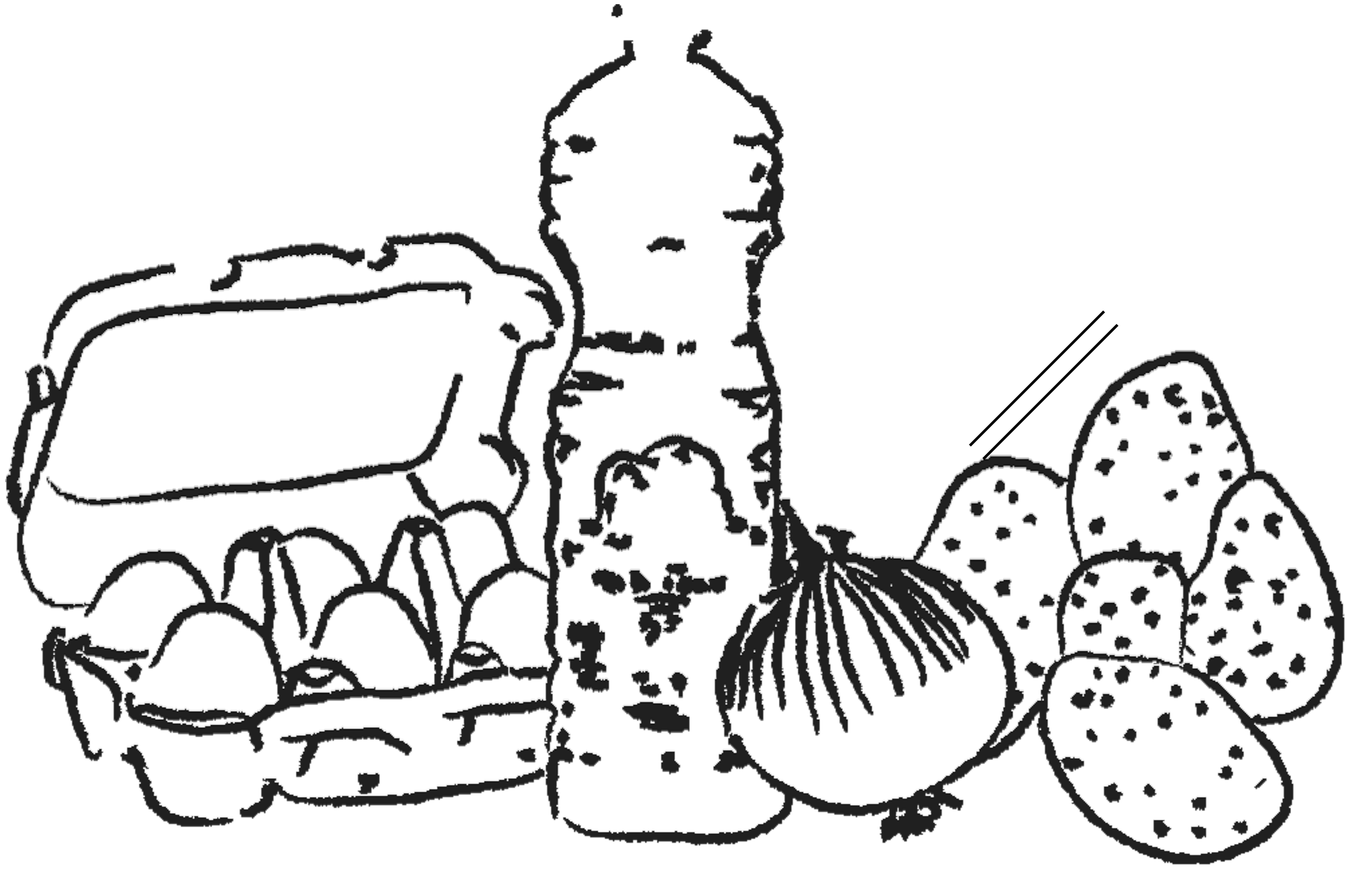
Miriam Tocino

:{) Codaisseur

What They Do

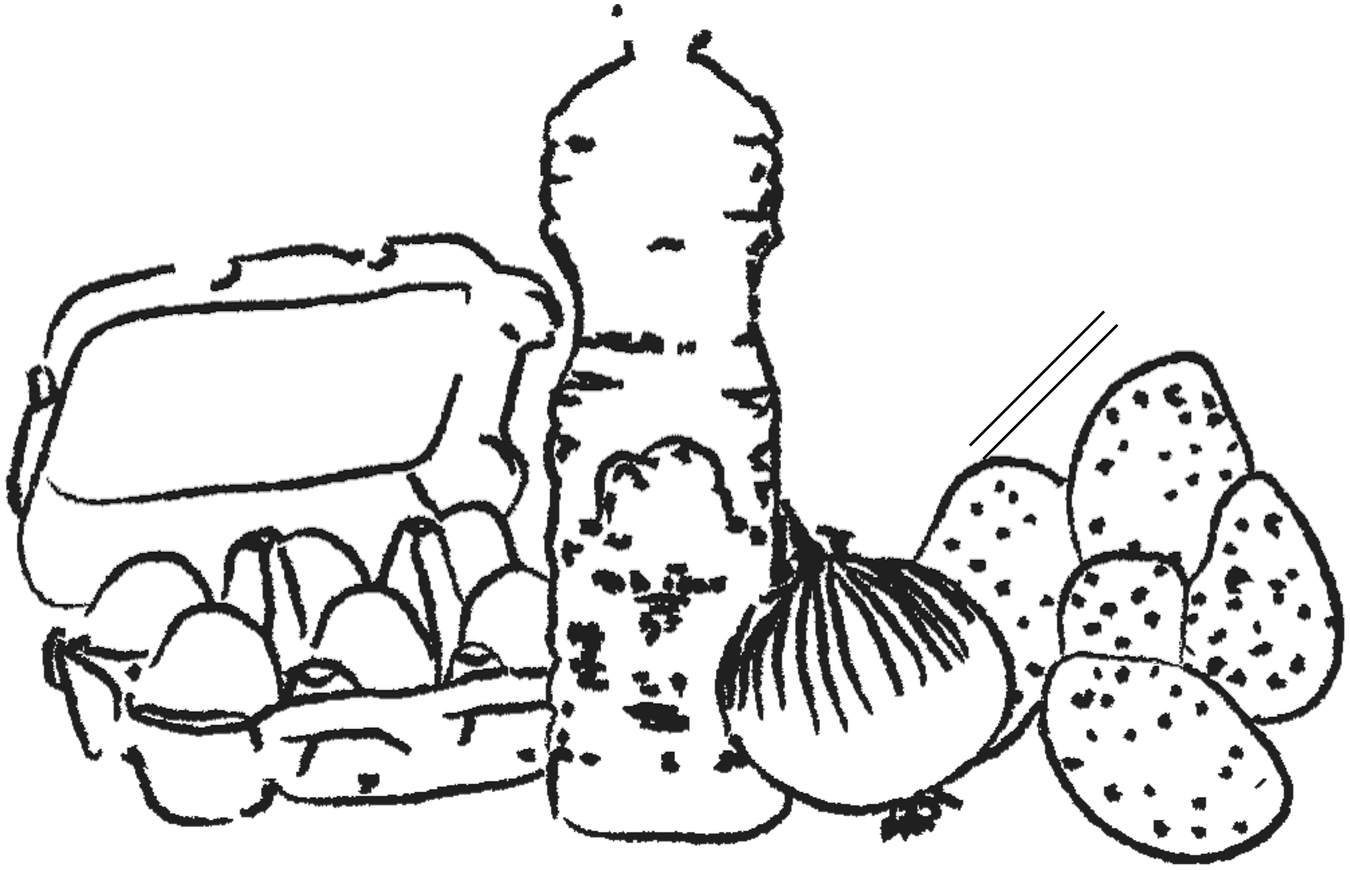
Automate repetitive tasks
quickly and easily.

What Do We Need



Ingredients:

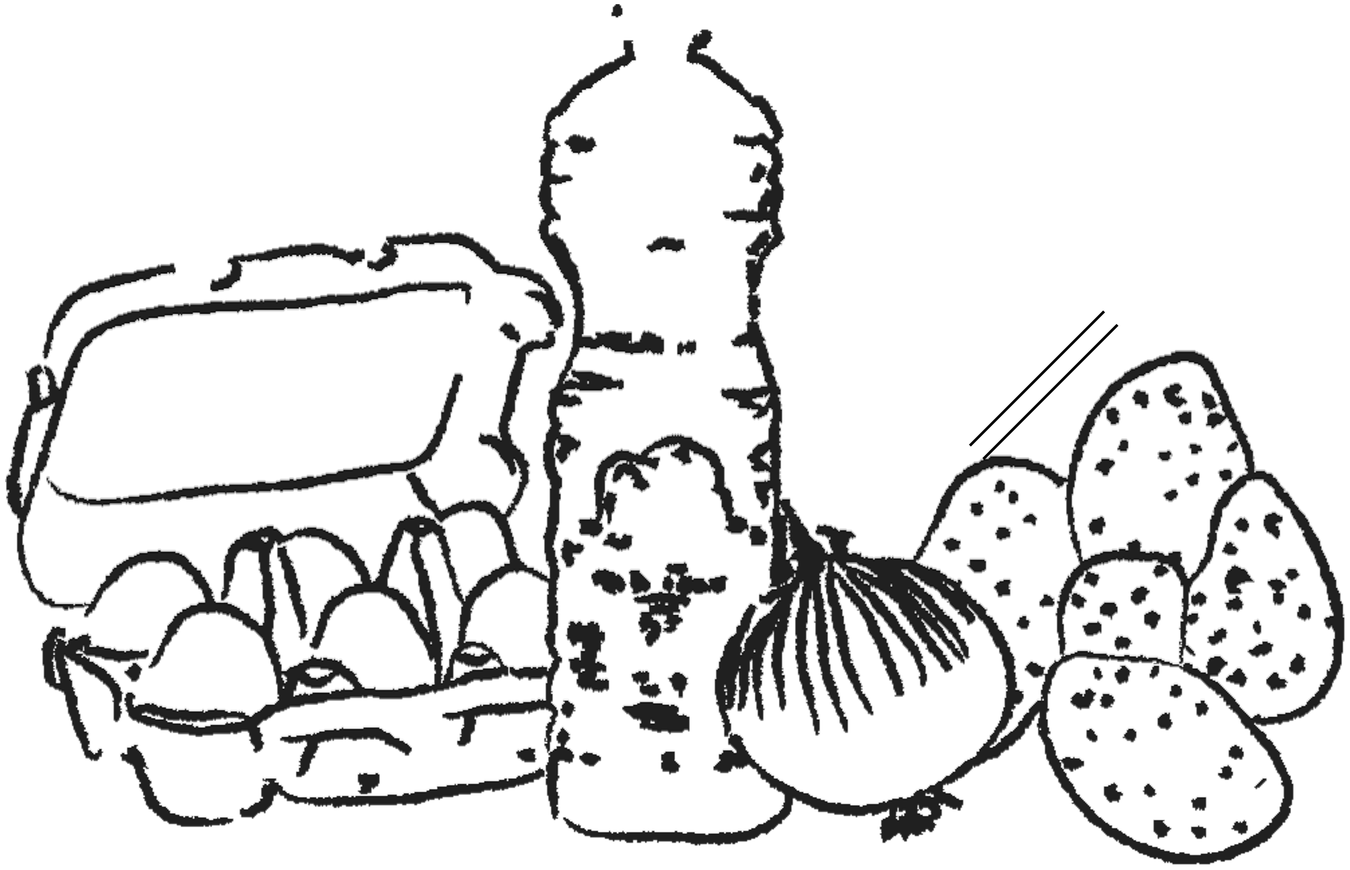
What Do We Need



Ingredients:

5 potatoes

What Do We Need

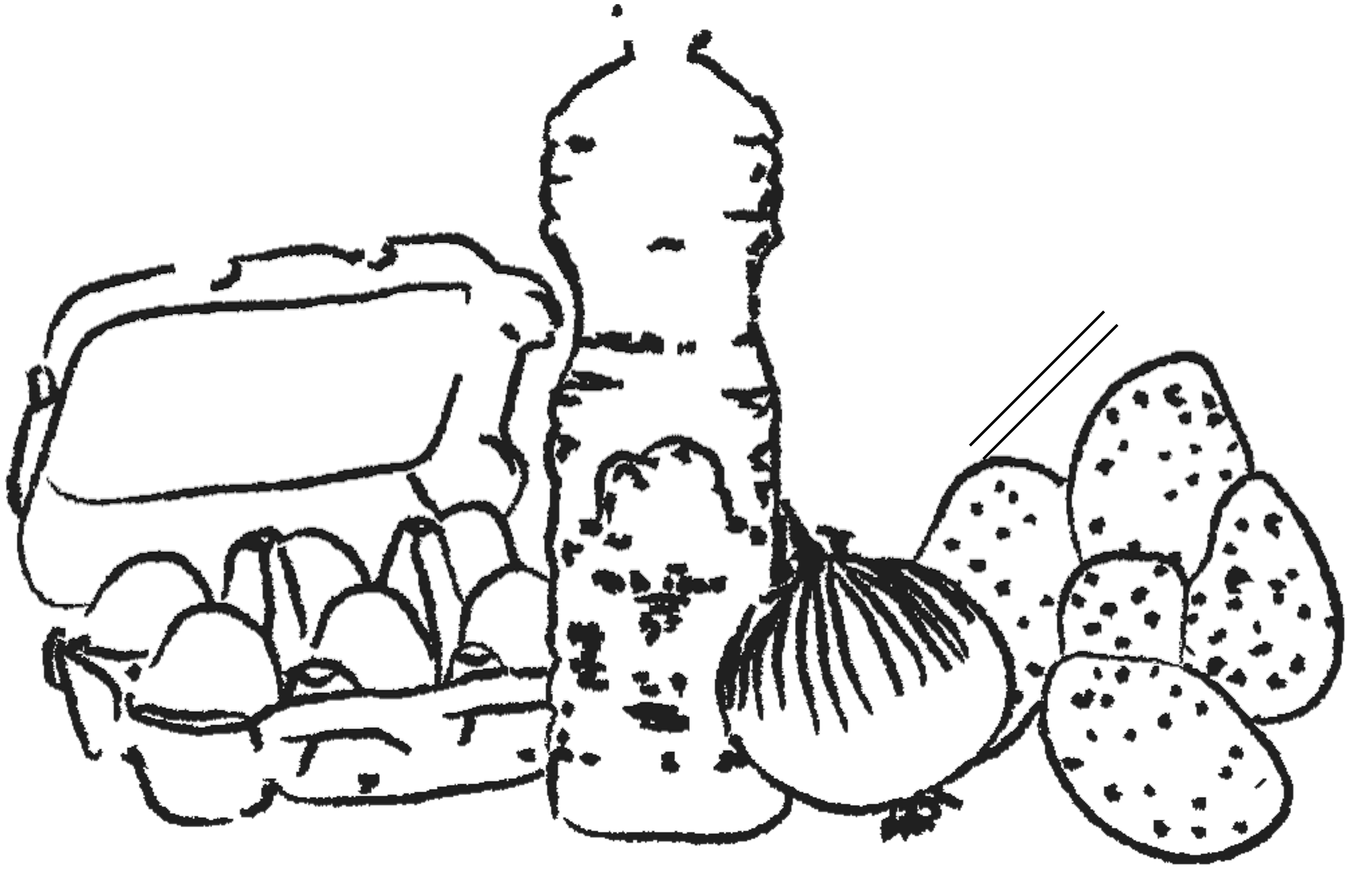


Ingredients:

5 potatoes

6 eggs

What Do We Need



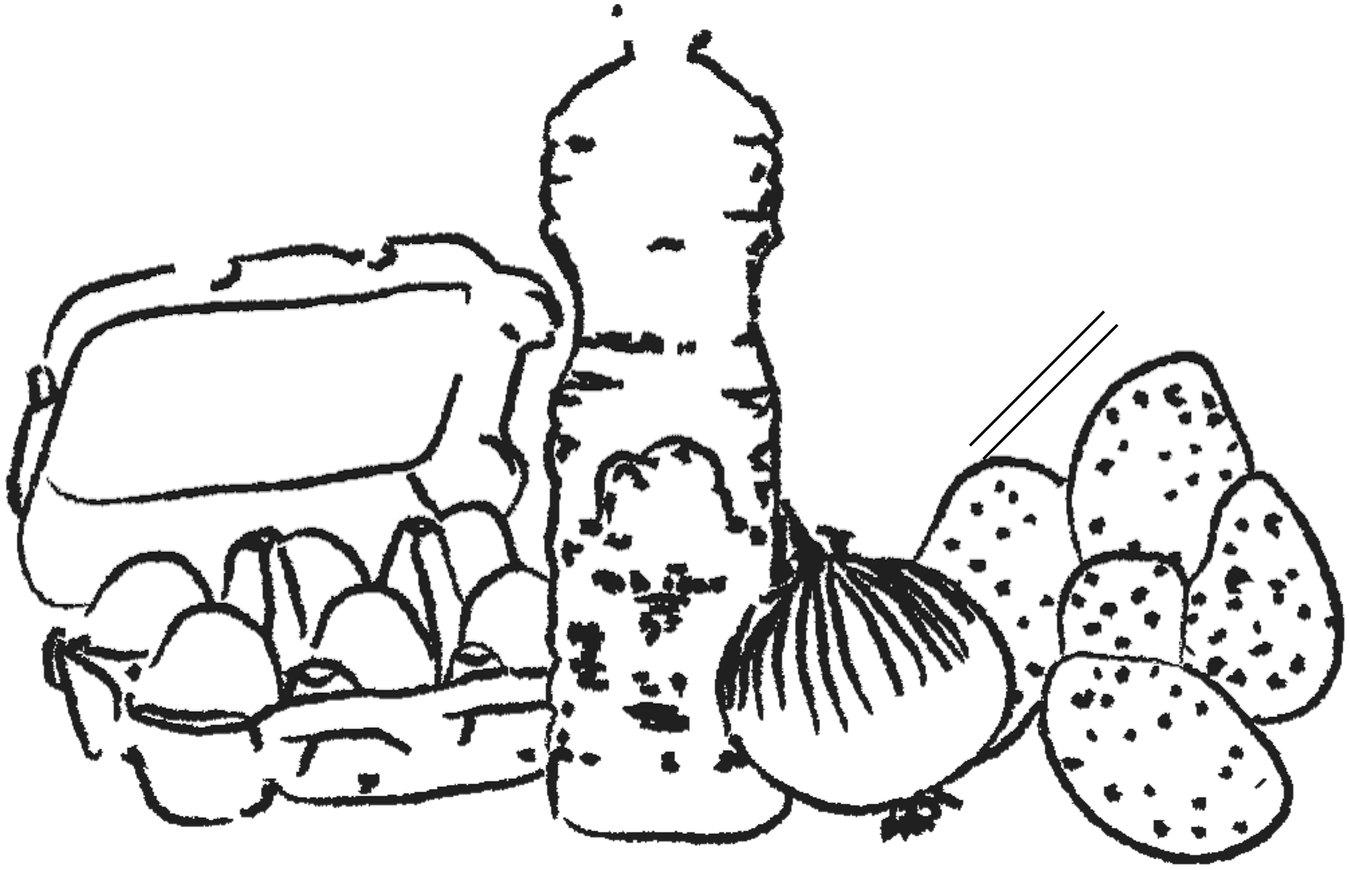
Ingredients:

5 potatoes

6 eggs

1 onion

What Do We Need



Ingredients:

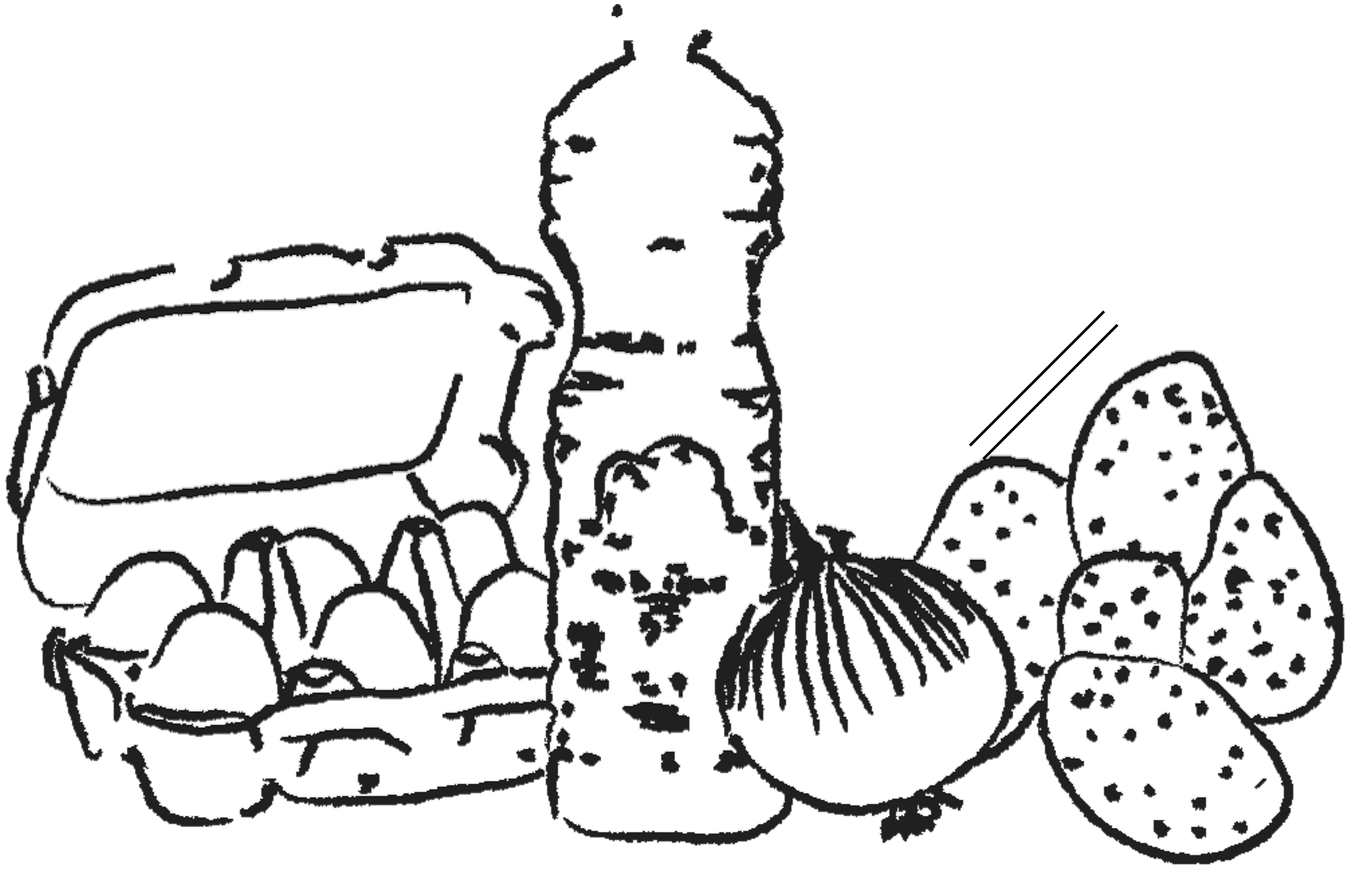
5 potatoes

6 eggs

1 onion

Olive oil

What Do We Need



Ingredients:

5 potatoes

6 eggs

1 onion

Olive oil

Salt

While Loop

Execute, add, repeat

Repeat action while certain condition is true.
As soon as the condition stops being true, the loop stops.

Syntax

```
while <conditional>
  # do something
end
```

While loop

```
$ ruby recipe.rb
```

```
# scrape the potatoes
```

While loop

```
$ ruby recipe.rb
```

```
# scrape the potatoes
counter = 1
```

While loop

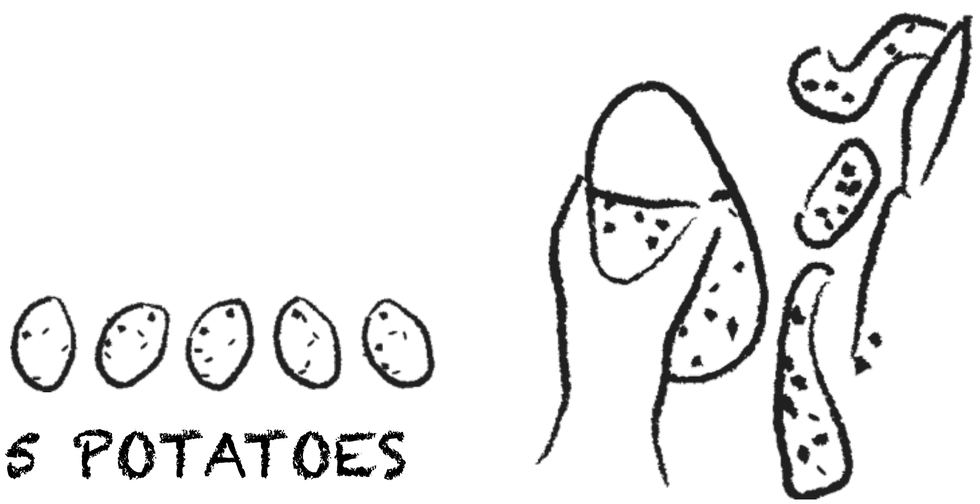
```
$ ruby recipe.rb
```

```ruby
# scrape the potatoes
counter = 1
while counter < 6


end
```

While loop

```
$ ruby recipe.rb
```

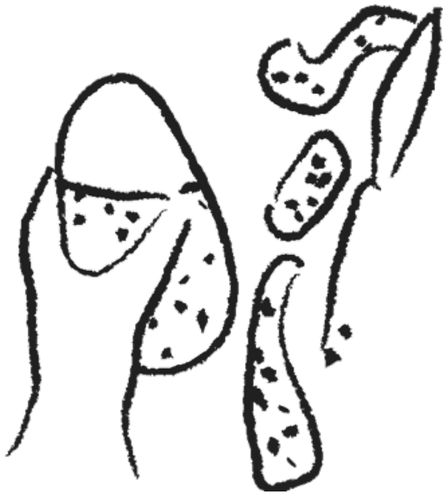
```ruby
# scrape the potatoes
counter = 1
while counter < 6
  puts "Scrape potato #{counter}."

end
```



While loop

```
$ ruby recipe.rb
```

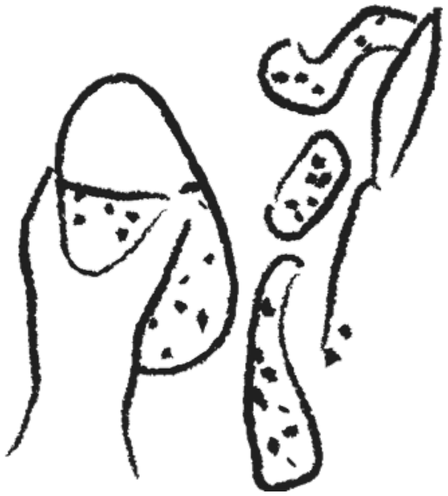
```ruby
# scrape the potatoes
counter = 1
while counter < 6
  puts "Scrape potato #{counter}."
  counter = counter + 1
end
```



While loop

```
$ ruby recipe.rb
```

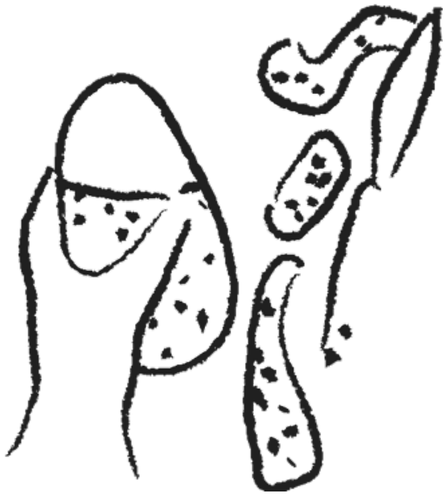
```ruby
# scrape the potatoes
counter = 1
while counter < 6
  puts "Scrape potato #{counter}."
  counter = counter + 1
end
```



While loop

```
$ ruby recipe.rb

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

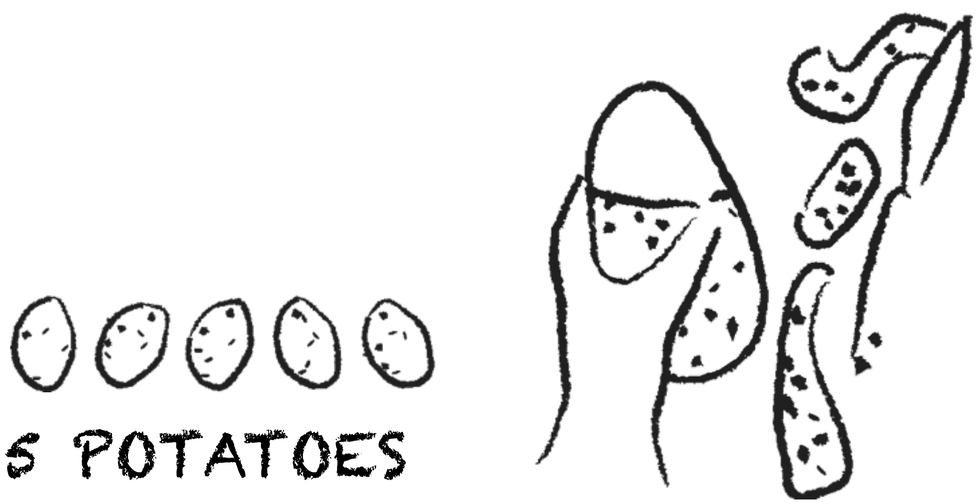
```ruby
# scrape the potatoes
counter = 1
while counter < 6
  puts "Scrape potato #{counter}."
  counter = counter + 1
end
```



Infinite loop

```
$ ruby recipe.rb
```

```ruby
# scrape the potatoes
counter = 1
while counter < 6
  puts "Scrape potato #{counter}."
  counter = counter + 1
end
```



Infinite loop

```
$ ruby recipe.rb
```

```ruby
# scrape the potatoes
counter = 1
while counter < 6
  puts "Scrape potato #{counter}."
  counter = counter + 1
end
```

Infinite loop

```
$ ruby recipe.rb
```

```ruby
# scrape the potatoes
counter = 1
while counter < 6
  puts "Scrape potato #{counter}."

end
```

Infinite loop

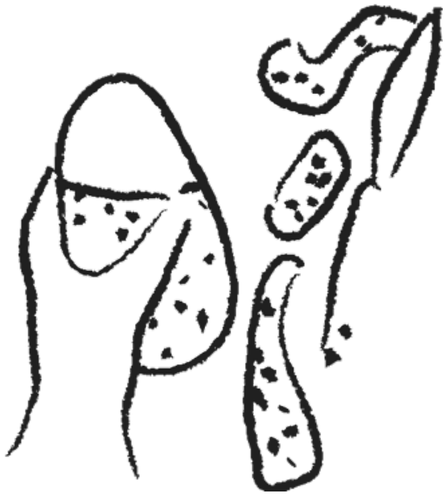
```
$ ruby recipe.rb
```

```ruby
# scrape the potatoes
counter = 1
while counter < 6
  puts "Scrape potato #{counter}."

end
```

Infinite loop

```
$ ruby recipe.rb

Scrape potato 1.
Scrape potato 1.
Scrape potato 1.
Scrape potato 1.
Scrape potato 1.
Scrape potato 1.
Scrape potato 1.
Scrape potato 1.
Scrape potato 1.
Scrape potato 1.
Scrape potato 1.
Scrape potato 1.
Scrape potato 1.
Scrape potato 1.
…
```

```ruby
# scrape the potatoes
counter = 1
while counter < 6
  puts "Scrape potato #{counter}."

end
```



Infinite loop

```
$ ruby recipe.rb

Scrape potato 1.
Scrape potato 1.
Scrape potato 1.
Scrape potato 1.
Scrape potato 1.
Scrape potato 1.
Scrape potato 1.
Scrape potato 1.
Scrape potato 1.
Scrape potato 1.
```

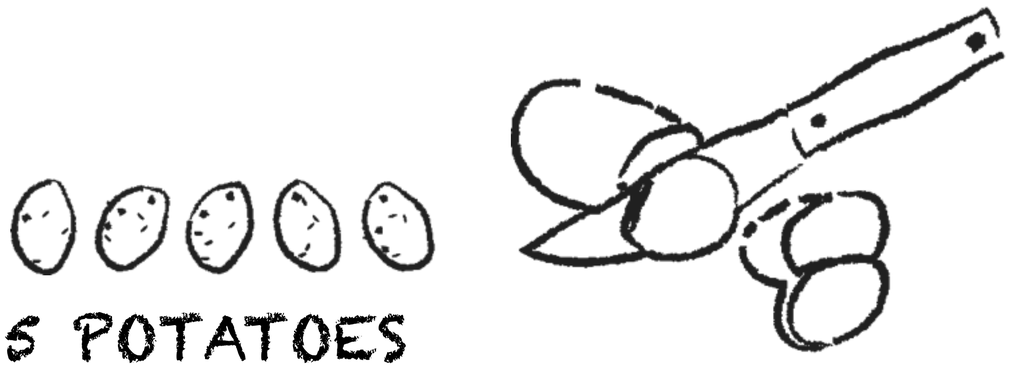
An infinite loop is
a **loop that never exits**.
(stop it with ctrl + c)

```ruby
# scrape the potatoes
counter = 1
while counter < 6
  puts "Scrape potato #{counter}."

end
```

While loop

```
$ ruby recipe.rb
```

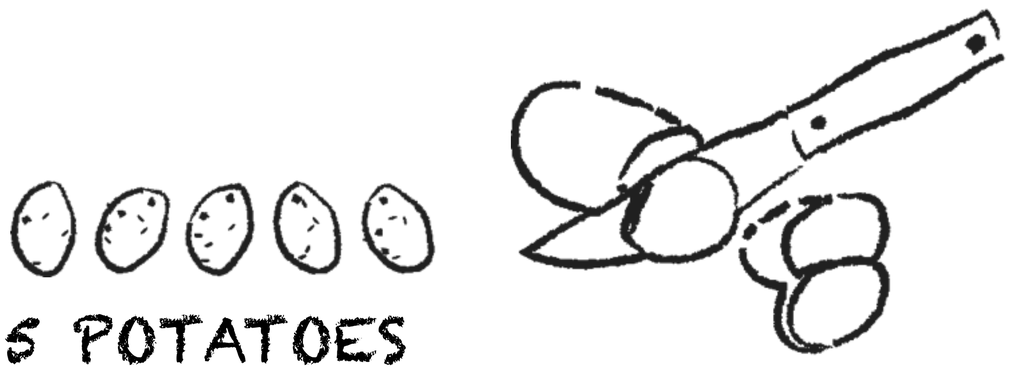
```
# cut potatoes into thick slices
```



While loop

```
$ ruby recipe.rb
```

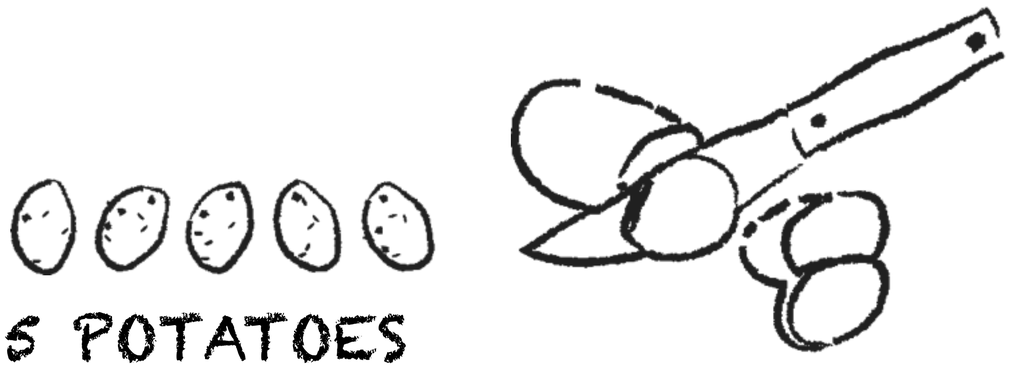
```ruby
# cut potatoes into thick slices
counter = 1
```



While loop

```
$ ruby recipe.rb
```

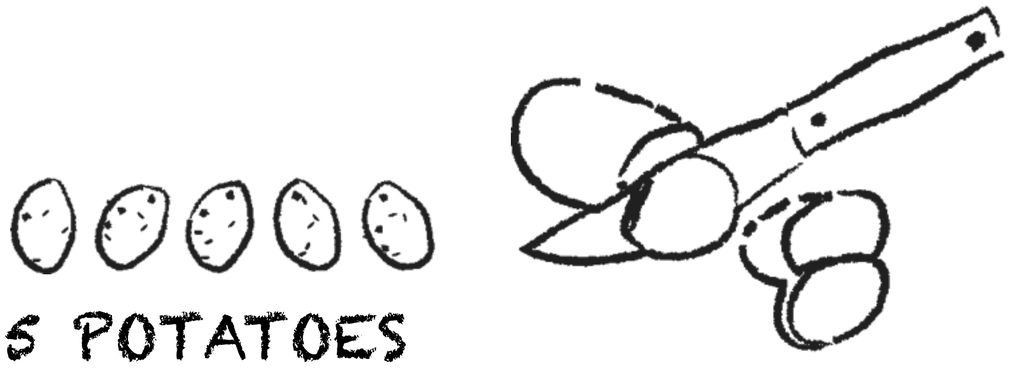
```ruby
# cut potatoes into thick slices
counter = 1
while counter <= 5


end
```



While loop

```
$ ruby recipe.rb
```

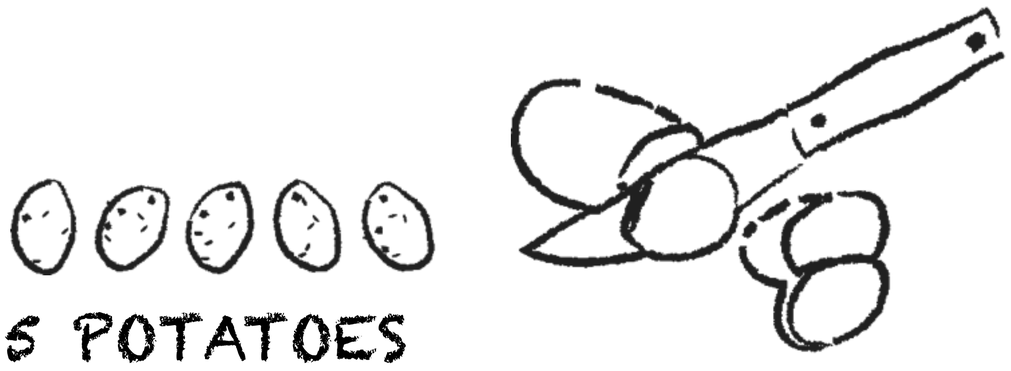
```ruby
# cut potatoes into thick slices
counter = 1
while counter <= 5
  puts "Cut potato #{counter}."

end
```



While loop

```
$ ruby recipe.rb
```

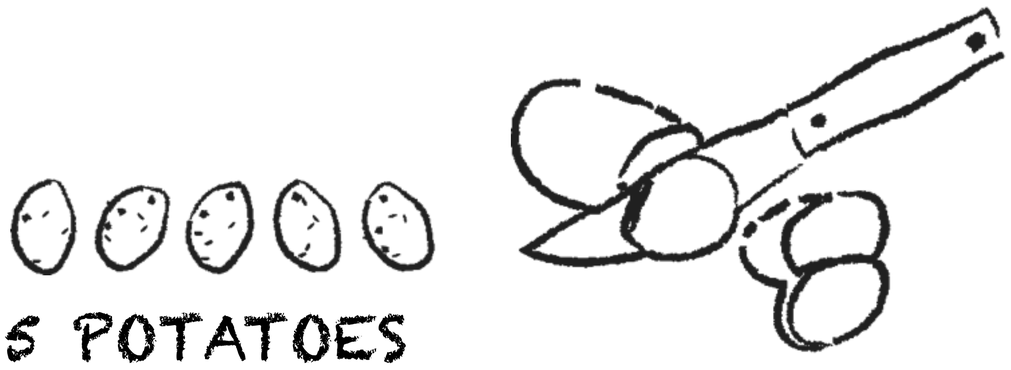
```ruby
# cut potatoes into thick slices
counter = 1
while counter <= 5
  puts "Cut potato #{counter}."
  counter = counter + 1
end
```



While loop

```
$ ruby recipe.rb
```

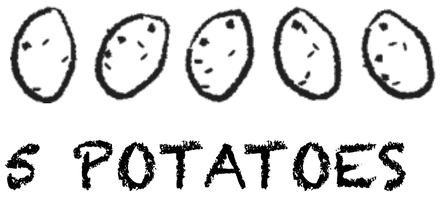
```ruby
# cut potatoes into thick slices
counter = 1
while counter <= 5
  puts "Cut potato #{counter}."
  counter = counter + 1
end
```

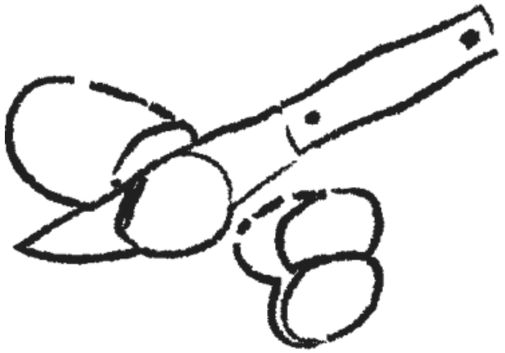


While loop

```
$ ruby recipe.rb

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

```ruby
# cut potatoes into thick slices
counter = 1
while counter <= 5
  puts "Cut potato #{counter}."
  counter = counter + 1
end
```





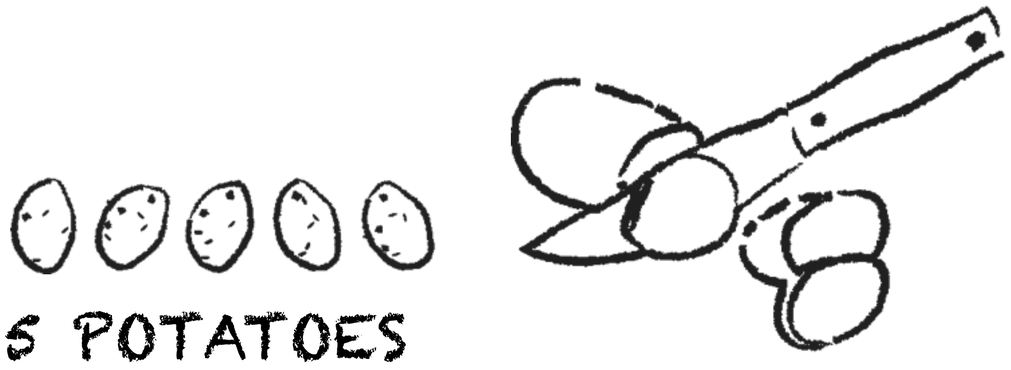
Assignment Operators

There’s always
another way with Ruby.

Assignment Operators

```
# more assignment operators
```

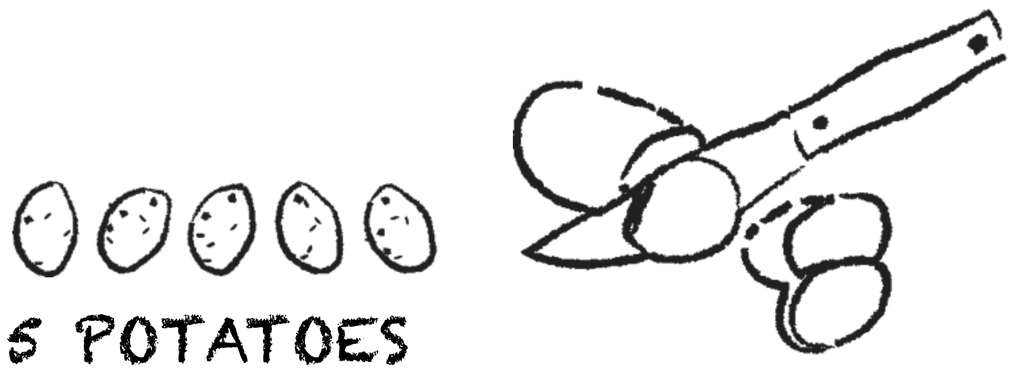
```
# cut potatoes into thick slices
counter = 1
while counter < 6
  puts "Cut potato #{counter}."
  counter = counter + 1
end
```



Assignment Operators

```ruby
# more assignment operators
counter = counter + 1
```

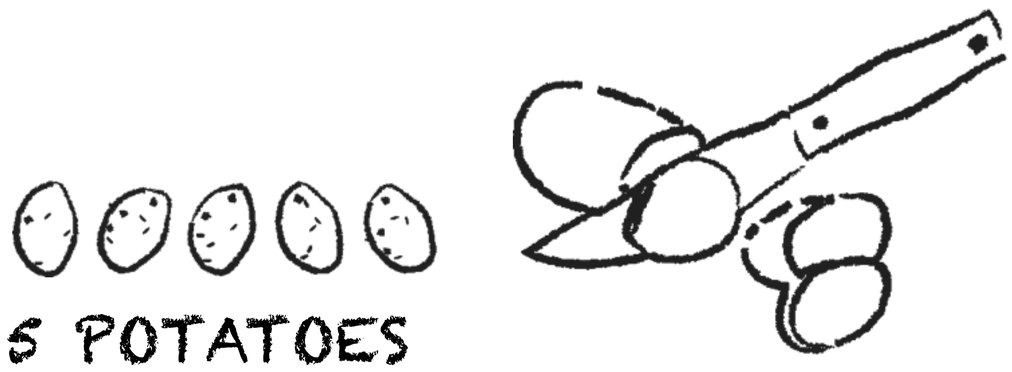
```ruby
# cut potatoes into thick slices
counter = 1
while counter < 6
  puts "Cut potato #{counter}."
  counter = counter + 1
end
```



Assignment Operators

```ruby
# more assignment operators
counter = counter + 1
```

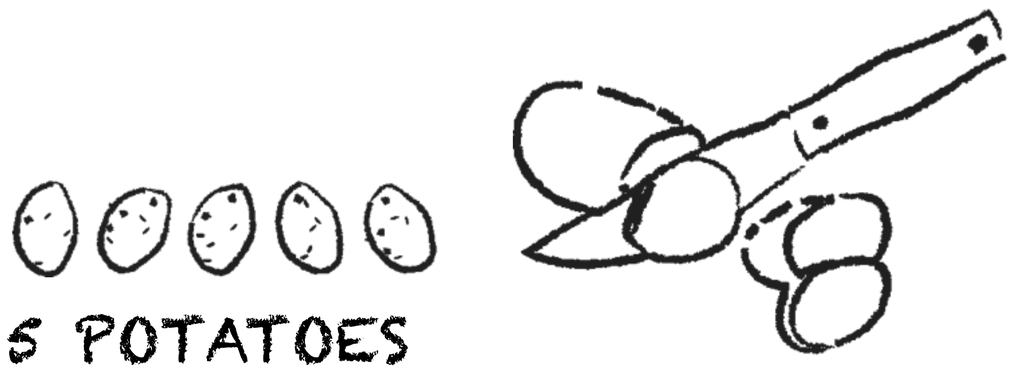
```ruby
# cut potatoes into thick slices
counter = 1
while counter < 6
  puts "Cut potato #{counter}."
  counter = counter + 1
end
```



Assignment Operators

```ruby
# more assignment operators
counter = counter + 1
counter += 1
```

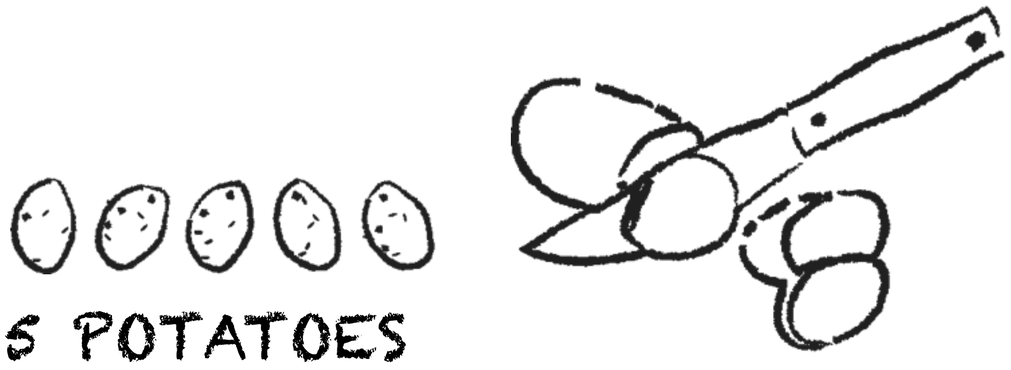
```ruby
# cut potatoes into thick slices
counter = 1
while counter < 6
  puts "Cut potato #{counter}."
  counter = counter + 1
end
```



Assignment Operators

```ruby
# more assignment operators
counter = counter + 1
counter += 1
```

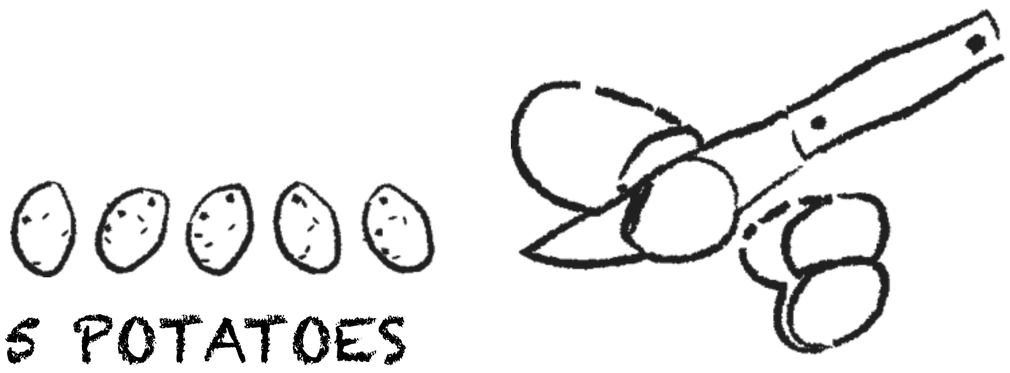
```ruby
# cut potatoes into thick slices
counter = 1
while counter < 6
  puts "Cut potato #{counter}."
  counter = counter + 1
end
```



Assignment Operators

```ruby
# more assignment operators
counter = counter + 1
counter += 1

counter = counter - 1
counter -= 1
```

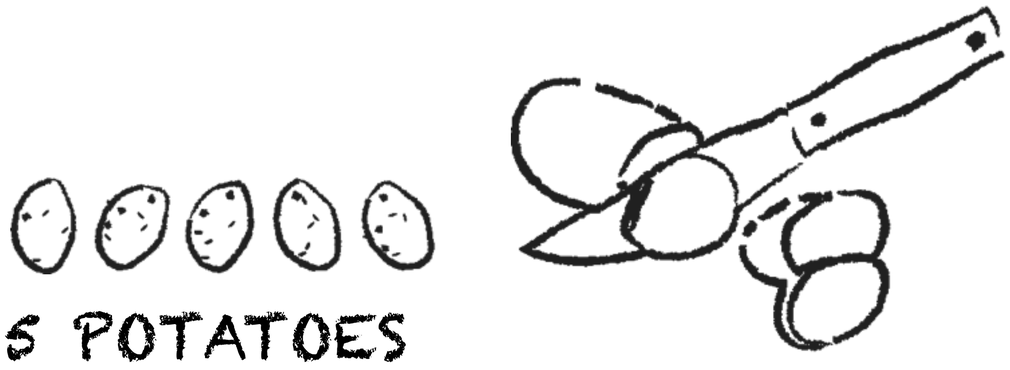
```ruby
# cut potatoes into thick slices
counter = 1
while counter < 6
  puts "Cut potato #{counter}."
  counter = counter + 1
end
```



Assignment Operators

```ruby
# more assignment operators
counter = counter + 1
counter += 1

counter = counter - 1
counter -= 1
```

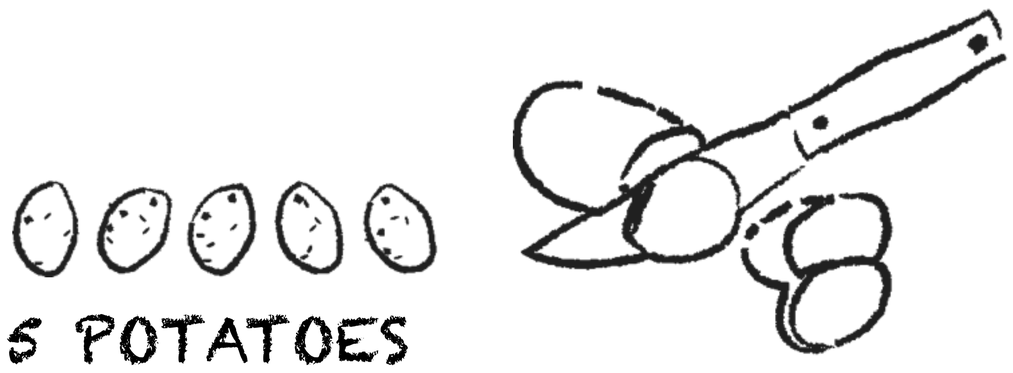
```ruby
# cut potatoes into thick slices
counter = 1
while counter < 6
  puts "Cut potato #{counter}."
  counter = counter + 1
end
```



Assignment Operators

```ruby
# more assignment operators
counter = counter + 1
counter += 1

counter = counter - 1
counter -= 1

counter = counter * 1
counter *= 1
```

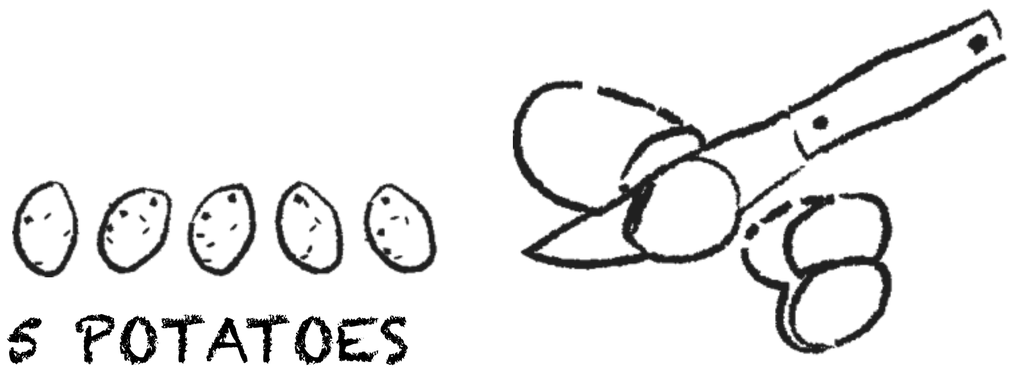
```ruby
# cut potatoes into thick slices
counter = 1
while counter < 6
  puts "Cut potato #{counter}."
  counter = counter + 1
end
```



Assignment Operators

```ruby
# more assignment operators
counter = counter + 1
counter += 1

counter = counter - 1
counter -= 1

counter = counter * 1
counter *= 1
```

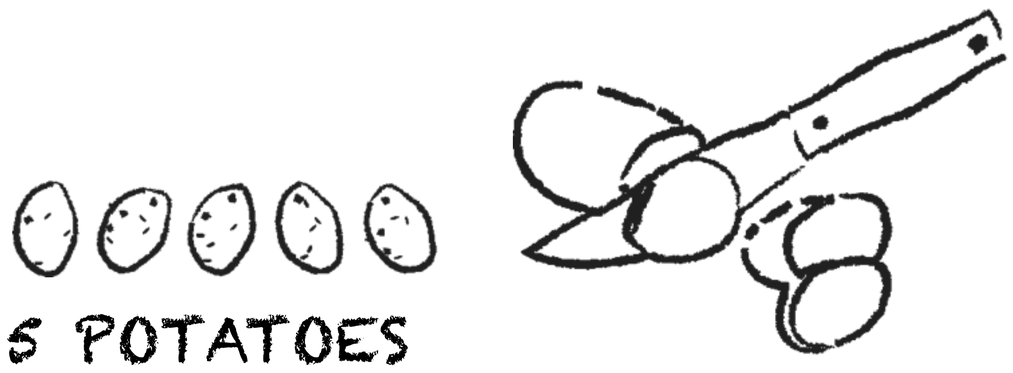
```ruby
# cut potatoes into thick slices
counter = 1
while counter < 6
  puts "Cut potato #{counter}."
  counter = counter + 1
end
```



Assignment Operators

```ruby
# more assignment operators
counter = counter + 1
counter += 1

counter = counter - 1
counter -= 1

counter = counter * 1
counter *= 1

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

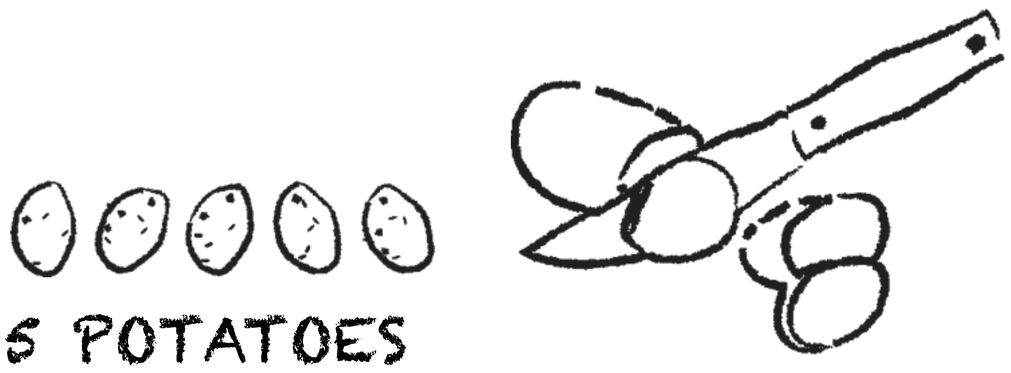
```ruby
# cut potatoes into thick slices
counter = 1
while counter < 6
  puts "Cut potato #{counter}."
  counter = counter + 1
end
```



Assignment Operators

```ruby
# more assignment operators
counter = counter + 1
counter += 1

counter = counter - 1
counter -= 1

counter = counter * 1
counter *= 1

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

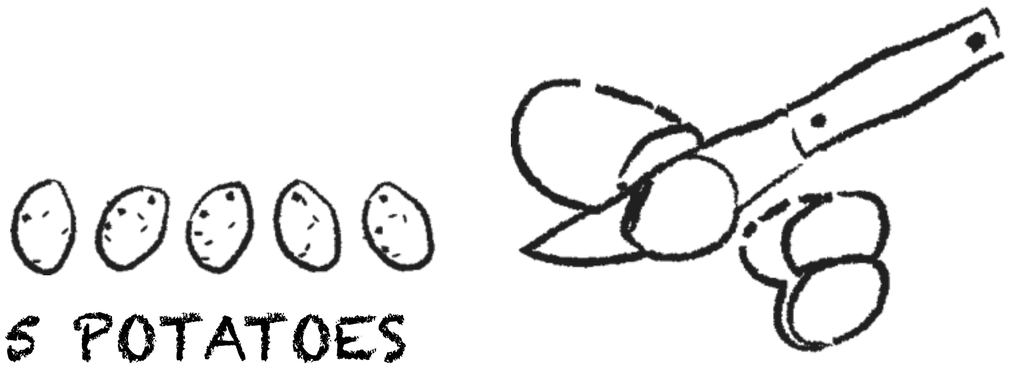
```ruby
# cut potatoes into thick slices
counter = 1
while counter < 6
  puts "Cut potato #{counter}."
  counter = counter + 1
end
```



Assignment Operators

```ruby
# more assignment operators
counter = counter + 1
counter += 1

counter = counter - 1
counter -= 1

counter = counter * 1
counter *= 1

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

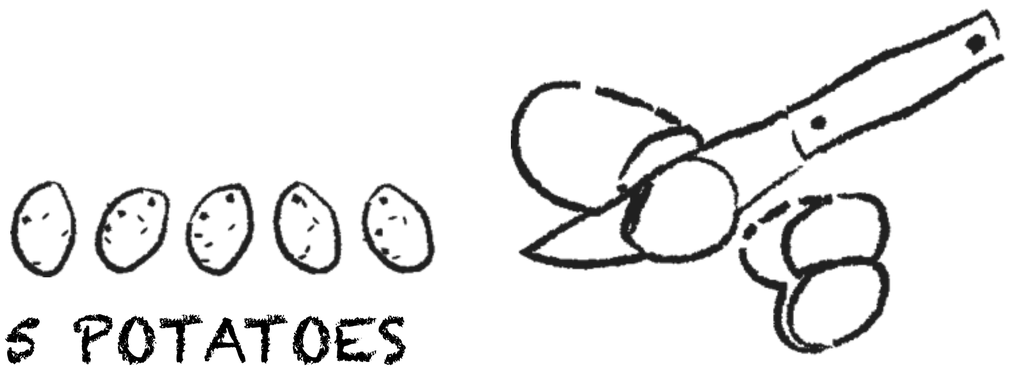
```ruby
# cut potatoes into thick slices
counter = 1
while counter < 6
  puts "Cut potato #{counter}."
  counter = counter + 1
end
```



Assignment Operators

```ruby
# more assignment operators
counter = counter + 1
counter += 1

counter = counter - 1
counter -= 1

counter = counter * 1
counter *= 1

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

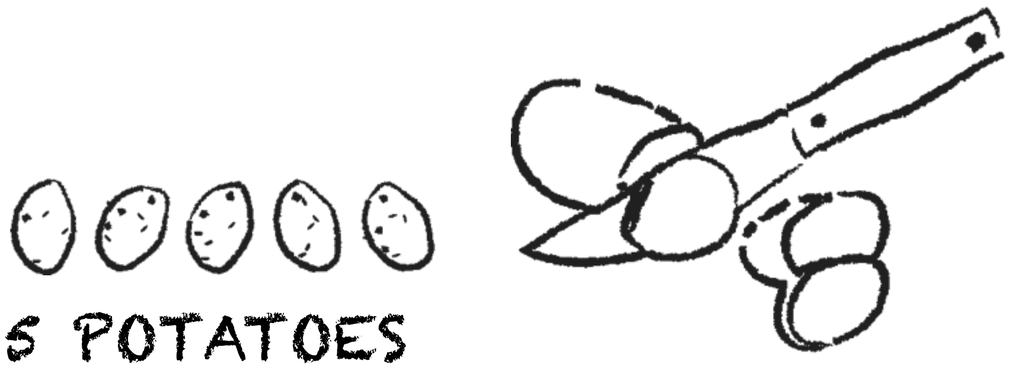
```ruby
# cut potatoes into thick slices
counter = 1
while counter < 6
  puts "Cut potato #{counter}."

end
```



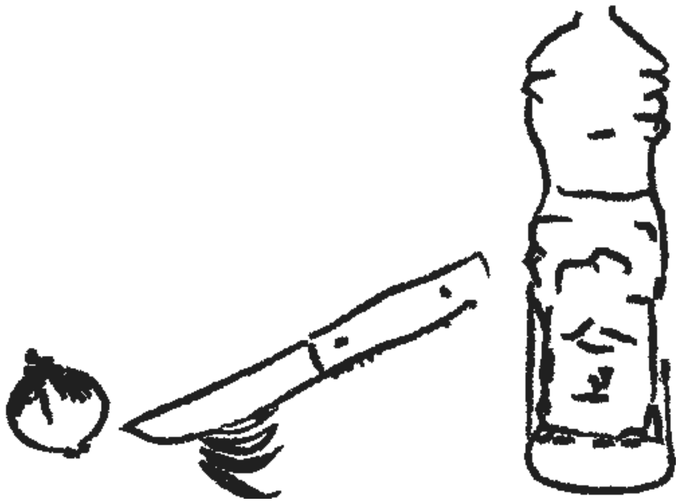
Assignment Operators

```ruby
# more assignment operators
counter = counter + 1
counter += 1

counter = counter - 1
counter -= 1

counter = counter * 1
counter *= 1

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

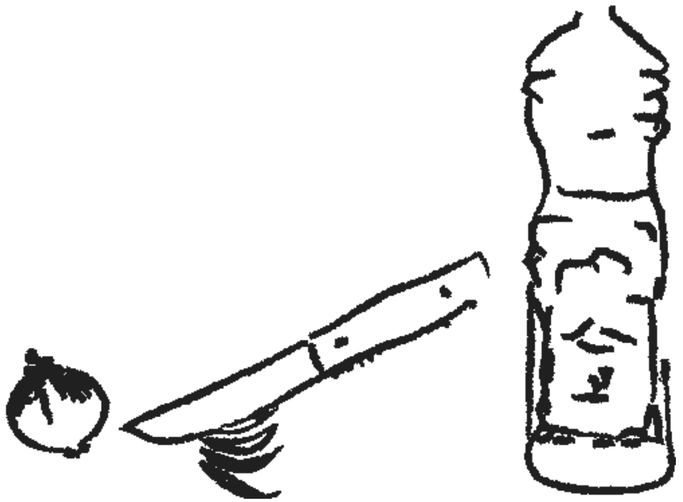
```ruby
# cut potatoes into thick slices
counter = 1
while counter < 6
  puts "Cut potato #{counter}."
  counter += 1
end
```



Keep Cooking



Keep Cooking



```
puts "Chop the onion."
puts "Heat the oil in a large frying pan."
```

For Loop

Fixed number of times

Whenever you know
how many times you will be looping.

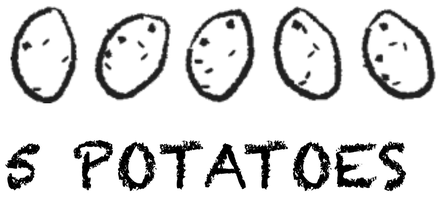
Syntax

```
for <variable> in <range>
   # Do something
end
```

For Loop

```
$ ruby recipe.rb
```

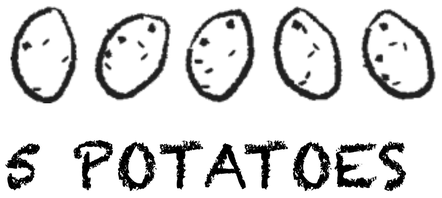
```
# Add potatoes and onion to the pan
```



For Loop

```
$ ruby recipe.rb
```

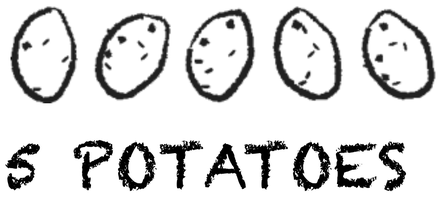
```ruby
# Add potatoes and onion to the pan
for counter in 1...6
```



For Loop

```
$ ruby recipe.rb
```

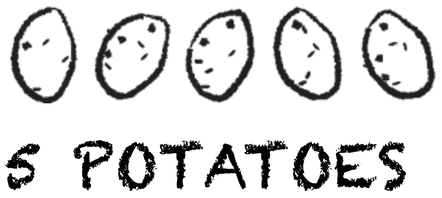
```ruby
# Add potatoes and onion to the pan
for counter in 1...6


end
```



For Loop

```
$ ruby recipe.rb
```

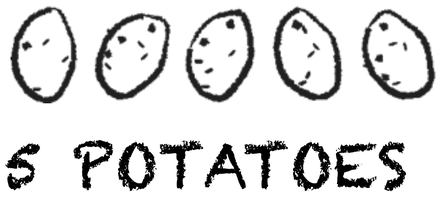
```ruby
# Add potatoes and onion to the pan
for counter in 1...6
  puts "Add potato #{counter}
       to the pan.”
end

puts "Add onion to the pan."
```



For Loop

```
$ ruby recipe.rb
```

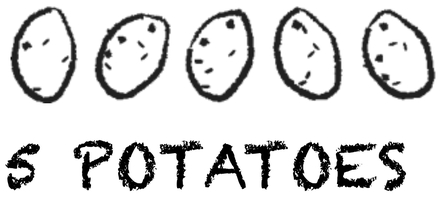
```ruby
# Add potatoes and onion to the pan
for counter in 1...6
  puts "Add potato #{counter}
       to the pan.”
end

puts "Add onion to the pan."
```



For Loop

```
$ 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.

Add onion to the pan.
```

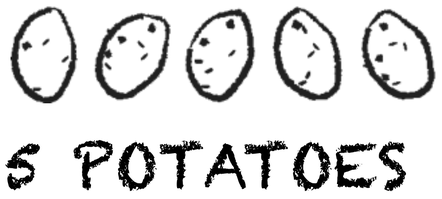
```ruby
# Add potatoes and onion to the pan
for counter in 1...6
  puts "Add potato #{counter}
       to the pan.”
end

puts "Add onion to the pan."
```



For Loop

```
$ 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.

Add onion to the pan.
```

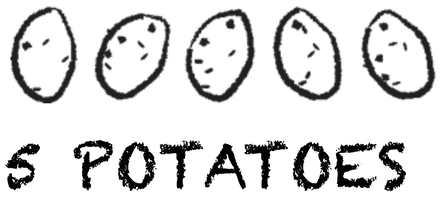
```ruby
# Add potatoes and onion to the pan
for counter in 1...6
  puts "Add potato #{counter}
       to the pan.”
end

puts "Add onion to the pan."
```



For Loop

```
$ 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.

Add onion to the pan.
```

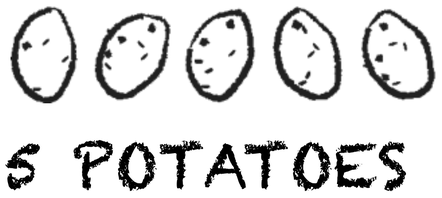
```
# Add potatoes and onion to the pan

   puts "Add potato #{counter}
        to the pan.”
end

puts "Add onion to the pan."
```



For Loop

```
$ 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.

Add onion to the pan.
```

```ruby
# Add potatoes and onion to the pan
for counter in 1..5
  puts "Add potato #{counter}
       to the pan.”
end

puts "Add onion to the pan."
```



Keep Cooking

Keep Cooking

```
puts "Strain potatoes and onions through a colander into a large bowl."
```

Arrays

Ordered lists of items

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

Syntax

Syntax

```
collection = [item0, item1, item2, …]
```

Syntax

collection = [item0, item1, item2, …]

empty_collection = [ ]
string_collection = ['jan', 'jaap', 'karel', 'kees']
number_collection = [88, 54, 13, 6, 8]
mixed_collection = [88, 'hallo', [4, 5], 0.16]

Adding items

Adding items

```
collection.push( 'my new item' )

OR

collection << 'my new item'
```

Retrieving items

Retrieving items

Each item in an array has an index.
You can get at that item with square brackets.

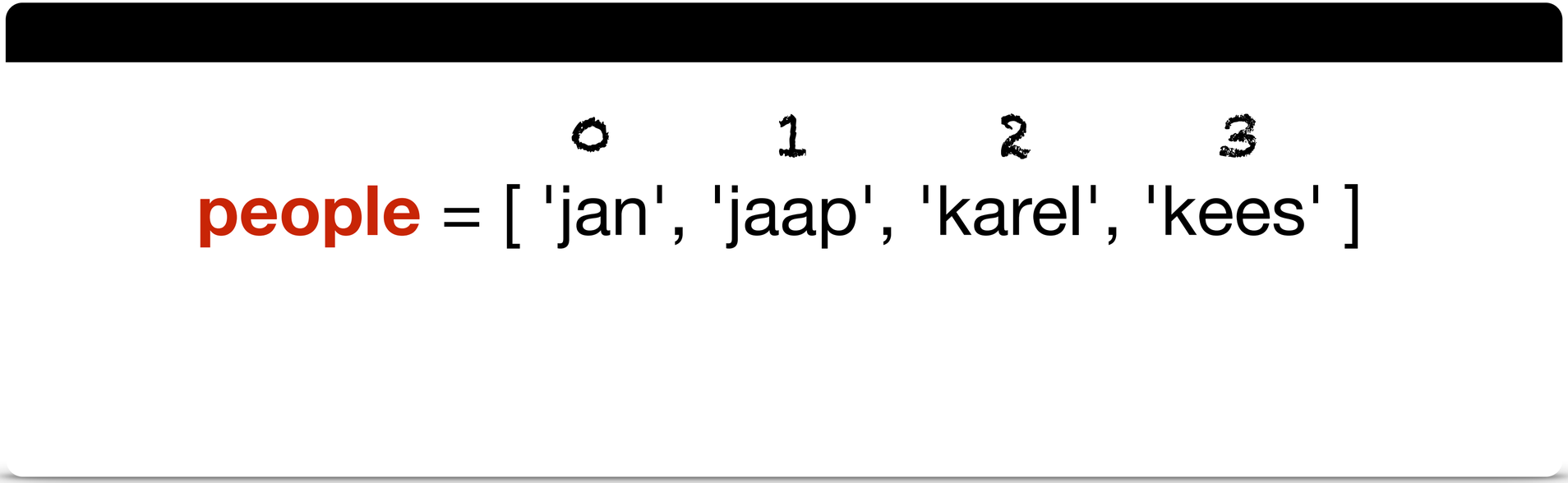
Retrieving items

Each item in an array has an index.
You can get at that item with square brackets.

people = ['jan', 'jaap', 'karel', 'kees']

Retrieving items

Each item in an array has an index.
You can get at that item with square brackets.



Retrieving items

Each item in an array has an index.
You can get at that item with square brackets.

```
                0      1       2       3
people = [ 'jan', 'jaap', 'karel', 'kees' ]
```

Here, **people[0]** is **'jan'** and **people[2]** is **'karel'**.

Retrieving items

Each item in an array has an index.
You can get at that item with square brackets.

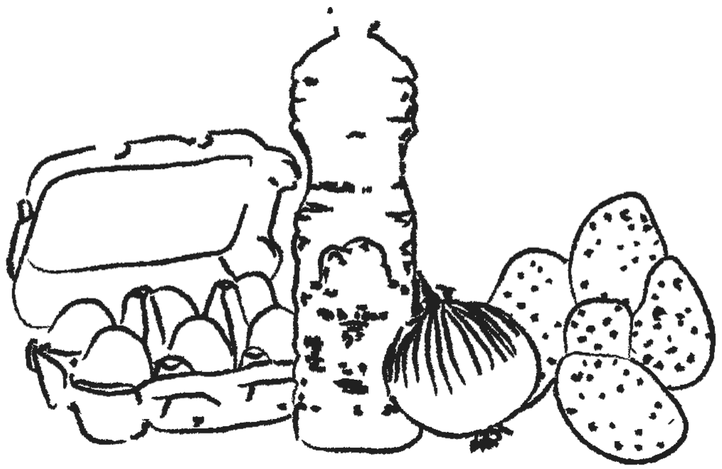
```
                0       1       2       3
people = [ 'jan', 'jaap', 'karel', 'kees' ]
```

Here, **people[0]** is **'jan'** and **people[2]** is **'karel'**.

Programmers start counting at zero!

Accessing index

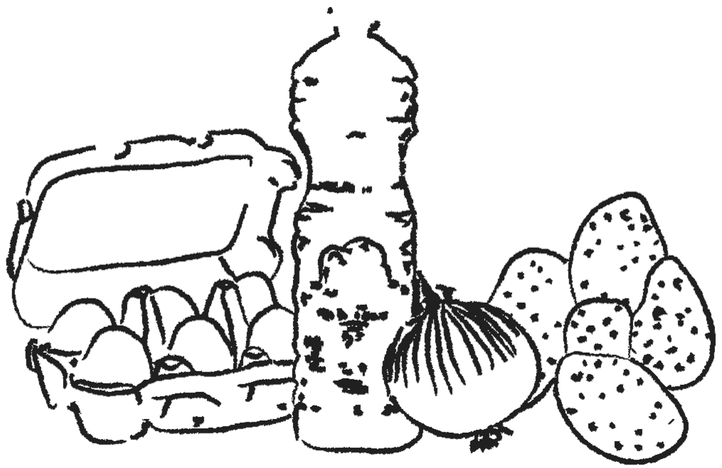
```
$ ruby recipe.rb
```



Accessing index

```
$ ruby recipe.rb
```

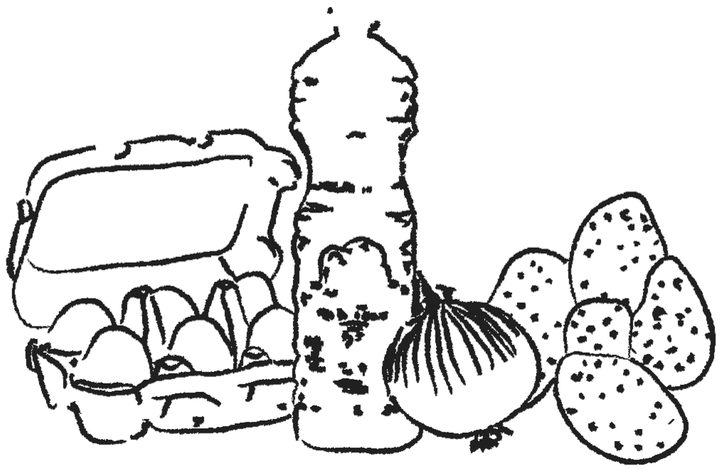
```
ingredients = ['potatoes','eggs','salt']
```



Accessing index

```
$ ruby recipe.rb
```

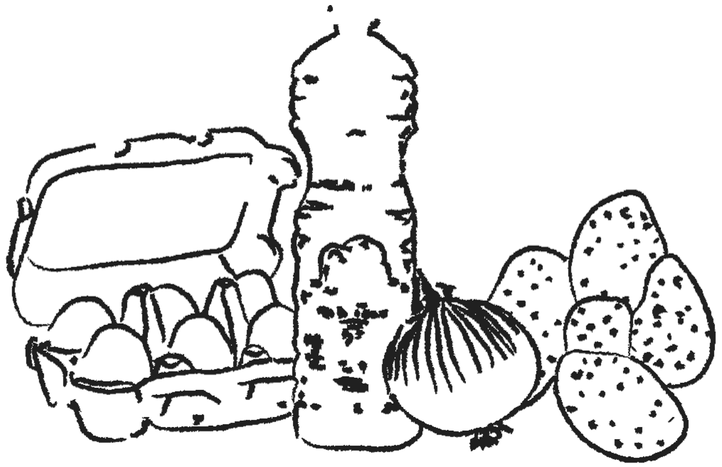
```
ingredients = ['potatoes','eggs','salt']
index = 0
```



Accessing index

```
$ ruby recipe.rb
```

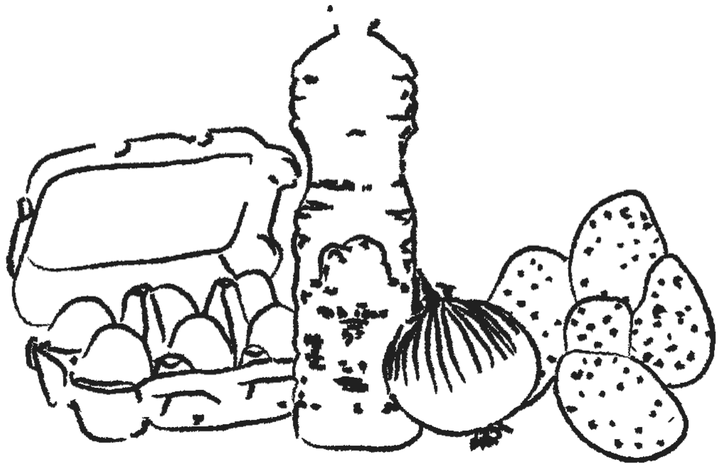
```
ingredients = ['potatoes','eggs','salt']
index = 0

while index < ingredients.length


end
```



Accessing index

```
$ ruby recipe.rb
```

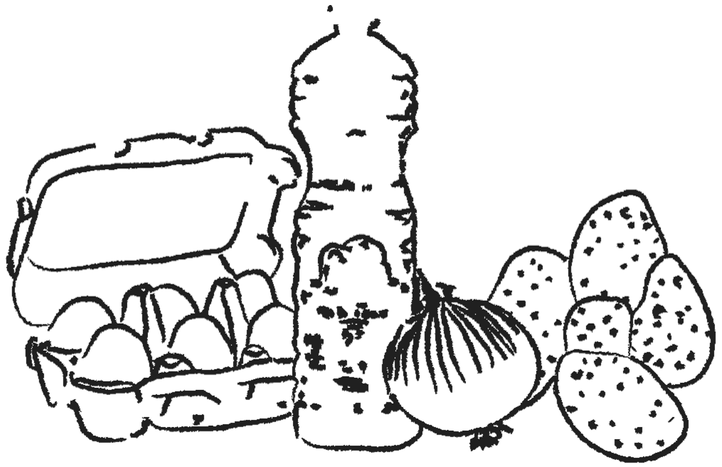
```ruby
ingredients = ['potatoes','eggs','salt']
index = 0

while index < ingredients.length
  puts "#{index}: #{ingredients[index]}"

end
```



Accessing index

```
$ ruby recipe.rb
```

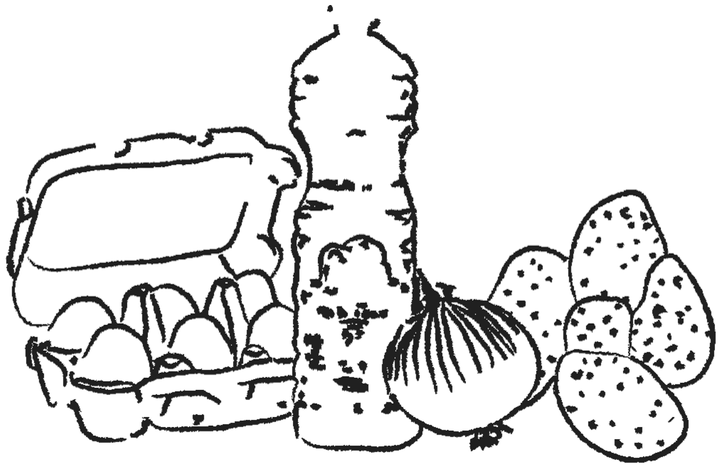
```ruby
ingredients = ['potatoes','eggs','salt']
index = 0

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



Accessing index

```
$ ruby recipe.rb
```

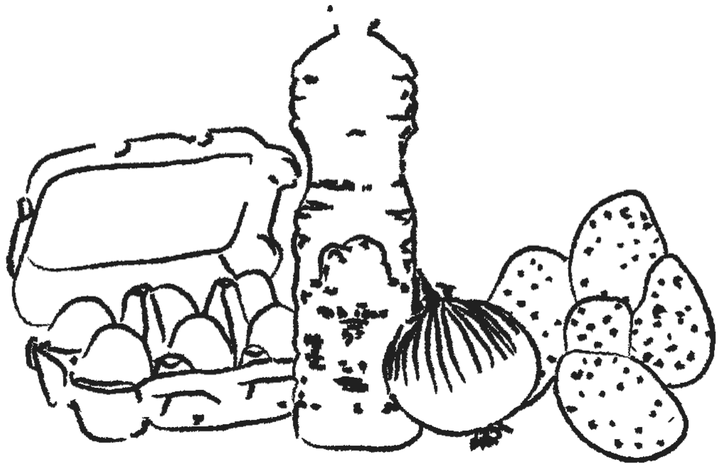
```ruby
ingredients = ['potatoes','eggs','salt']
index = 0

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



Accessing index

```
$ ruby recipe.rb

0: potatoes
1: eggs
2: salt
```

```ruby
ingredients = ['potatoes','eggs','salt']
index = 0

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



Each Method

Powerful iterator

The Ruby Way to iterate over a collection of items,
like an array, one at a time.

Syntax

Syntax

```
collection = [item0, item1, item2, …]
```

Syntax

```
collection = [item0, item1, item2, ...]

collection.each do | item |
  # do something
end
```

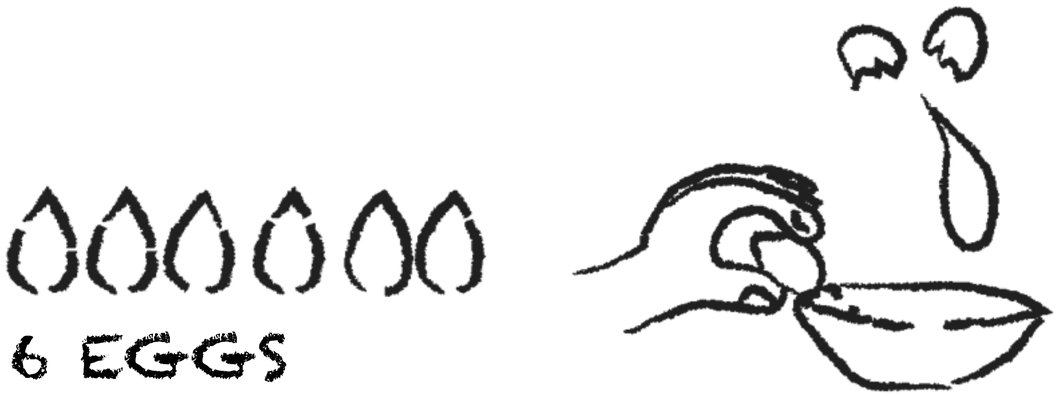
Syntax

```
collection = [item0, item1, item2, ...]

collection.each do | item |
  # do something
end

mycollection.each { | item | # do something }
```

Each Operator

```
$ ruby recipe.rb
```

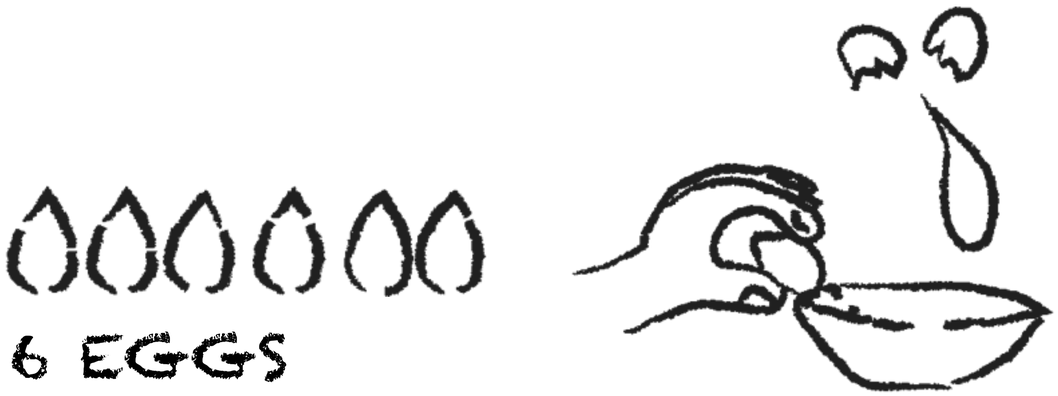
```
# Break the eggs
and beat them separately.
```



Each Operator

```
$ ruby recipe.rb
```

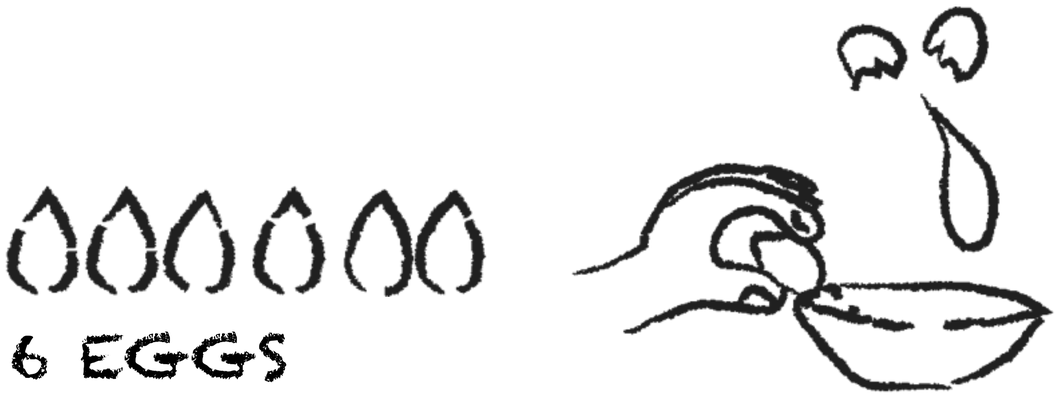
```
# Break the eggs
and beat them separately.
eggs = [1, 2, 3, 4, 5, 6]
```



Each Operator

```
$ ruby recipe.rb
```

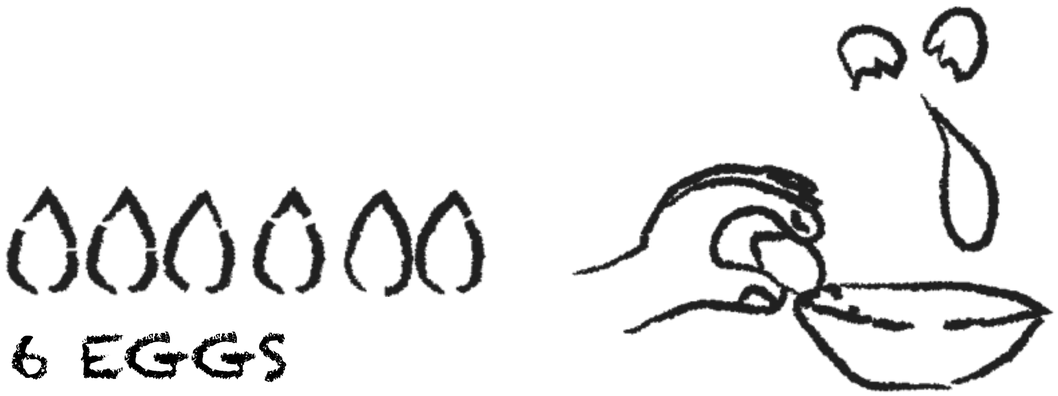
```ruby
# Break the eggs
and beat them separately.
eggs = [1, 2, 3, 4, 5, 6]
eggs.each do |egg|

end
```



Each Operator

```
$ ruby recipe.rb
```

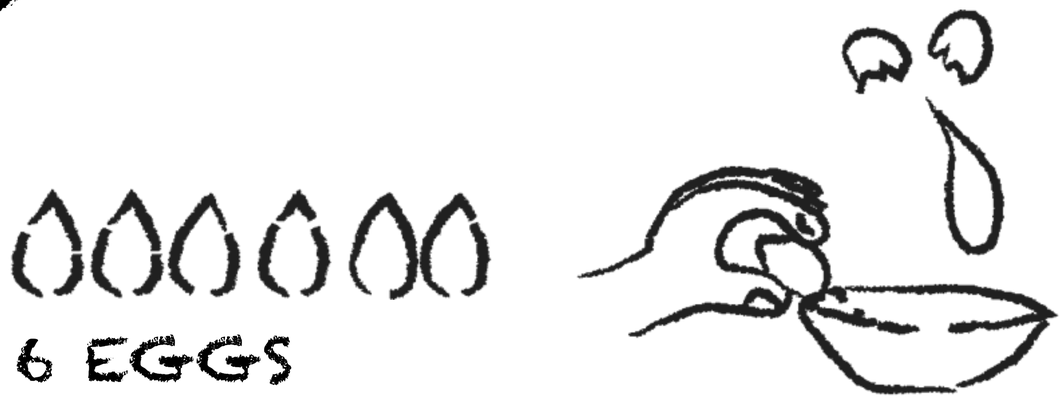
```ruby
# Break the eggs
and beat them separately.
eggs = [1, 2, 3, 4, 5, 6]
eggs.each do |egg|
  puts "Break egg #{egg}.”
end

puts "Beat the eggs separately."
```



Each Operator

```
$ ruby recipe.rb
```

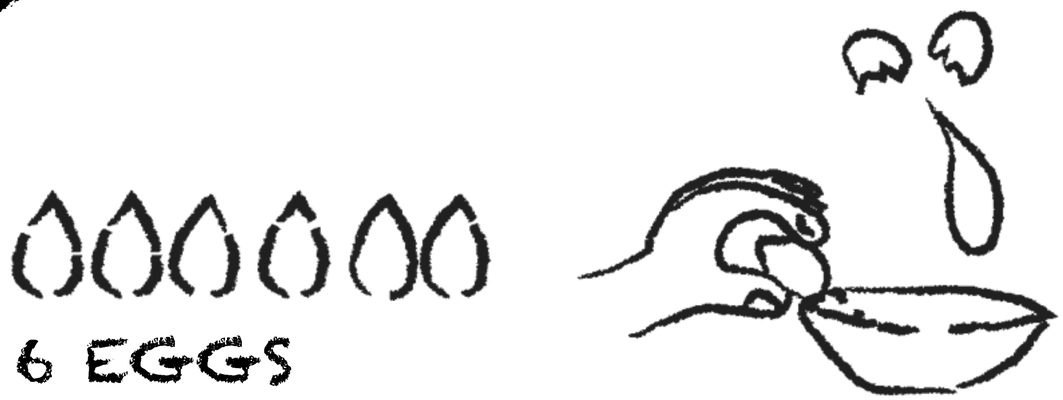
```ruby
# Break the eggs
and beat them separately.
eggs = [1, 2, 3, 4, 5, 6]
eggs.each do |egg|
  puts "Break egg #{egg}.”
end

puts "Beat the eggs separately."
```



Each Operator

```
$ ruby recipe.rb

Break egg 1.
Break egg 2.
Break egg 3.
Break egg 4.
Break egg 5.
Break egg 6.

Beat the eggs separately.
```

```ruby
# Break the eggs
and beat them separately.
eggs = [1, 2, 3, 4, 5, 6]
eggs.each do |egg|
  puts "Break egg #{egg}.”
end

puts "Beat the eggs separately."
```



Hashes

Associative collections

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

Syntax

Hashes associate keys with values.

Syntax

Hashes associate keys with values.

```
myhash = { 'jan' => 34, 'jaap' => 26, 'karel' => 40 }
```

Syntax

Hashes associate keys with values.

```
myhash = { 'jan' => 34, 'jaap' => 26, 'karel' => 40 }

myhash['anna'] = 28
```

Each Operator

```
$ ruby recipe.rb
```

Each Operator

```
$ ruby recipe.rb
```

```ruby
ingredients = {
  "onions" => 2,
  "potatoes" => 5,
  "eggs" => 6 }

ingredients.each do | name, number |
    puts "Take #{number} #{name}."
end
```

Each Operator

```
$ ruby recipe.rb
```

```ruby
ingredients = {
  "onions" => 2,
  "potatoes" => 5,
  "eggs" => 6 }

ingredients.each do | name, number |
    puts "Take #{number} #{name}."
end
```

Each Operator

```
$ ruby recipe.rb

Take 2 onions.
Take 5 potatoes.
Take 6 eggs.
```

```ruby
ingredients = {
  "onions" => 2,
  "potatoes" => 5,
  "eggs" => 6 }

ingredients.each do | name, number |
    puts "Take #{number} #{name}."
end
```

Keep Cooking

LISTO!

Keep Cooking


```
puts "Then stir into the bowl the potatoes with plenty of salt."
puts "Heat a little of the strained oil in a smaller pan."
puts "Tip everything into the pan and cook on a moderate heat."
puts "When almost set, invert on a plate and slide back into the pan."
puts "Cook a few more minutes."
puts "Slide on to a plate and cool for 10 minutes before serving."
```

Final review

```
while <conditional>
    # do something
end
```

Questions?

```
for <variable> in <range>
    # do something
end
```

```
collection.each do |item|
    # do something
end
```

Time to code

Go and shine looping!

Choose one of your favourite recipes
and give it a try!

Go to Your Shop
Use Loops and Collections

Go to Your Shop
Use Loops and Collections

Use Arrays/Hashes where needed

Go to Your Shop
Use Loops and Collections

Use Arrays/Hashes where needed
Add a Shopping Cart (Array or Hash)

Go to Your Shop
Use Loops and Collections

Use Arrays/Hashes where needed
Add a Shopping Cart (Array or Hash)
Let User Order Items in a Loop

Go to Your Shop
Use Loops and Collections

Use Arrays/Hashes where needed
Add a Shopping Cart (Array or Hash)
Let User Order Items in a Loop
Use Loops to Print List of Items & Cart

Go to Your Shop
Use Loops and Collections

Use Arrays/Hashes where needed
Add a Shopping Cart (Array or Hash)
Let User Order Items in a Loop
Use Loops to Print List of Items & Cart
Show Total Price of Cart