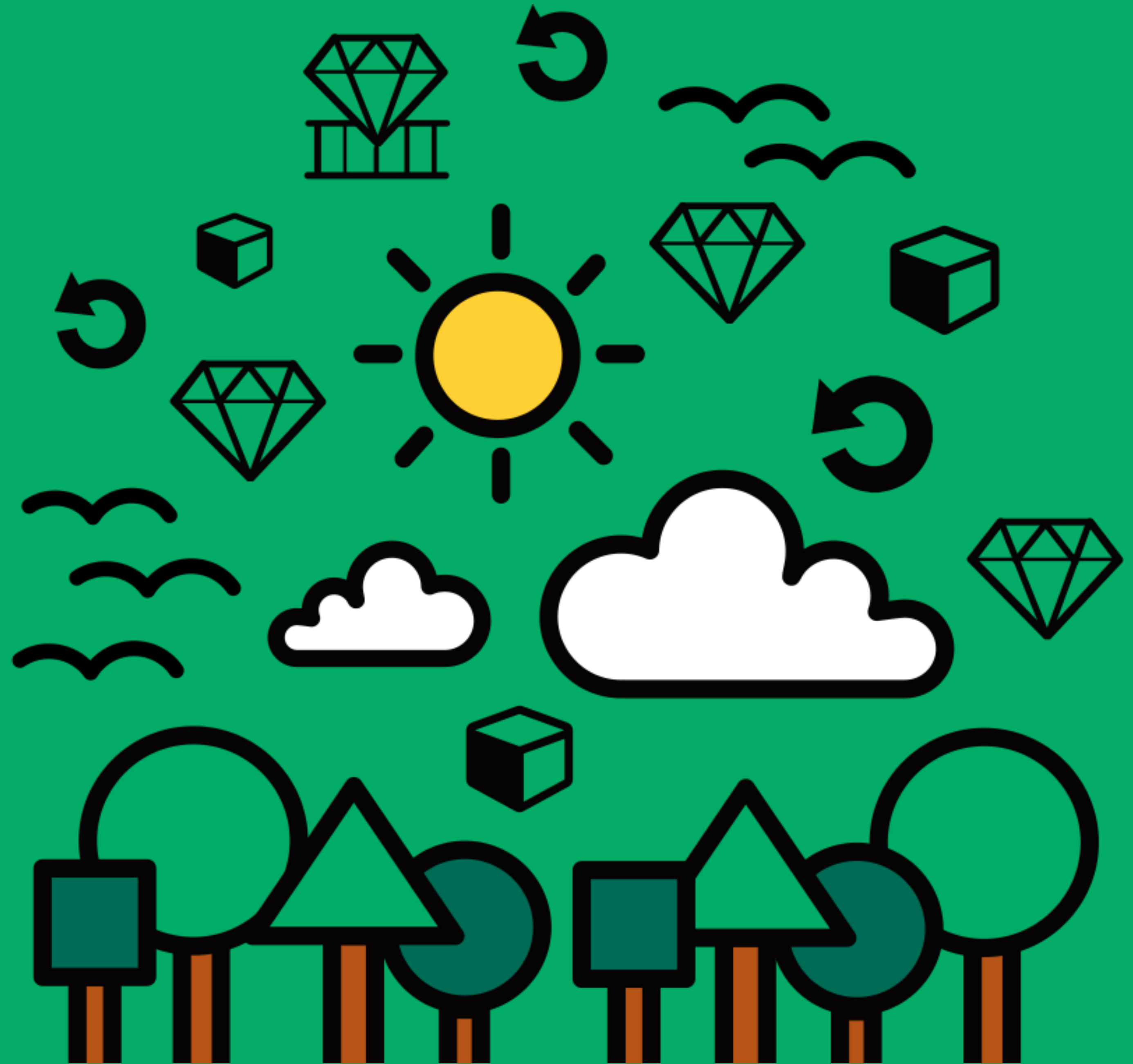


Loops and Lists

by Miriam Tocino

Coding a
Spanish Omelette



What They Do

Automate **repetitive tasks**
quickly and easily.



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

← true

```
while conditional  
  # Do something  
end
```


While Loop

```
$ ruby recipe.rb
```

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

```
# Scrape the Potatoes
```

```
counter = 1
```

```
while counter < 6
```

```
  puts "Scrape potato #{counter}."
```

```
  counter = counter + 1
```

```
end
```



5 POTATOES



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.
```

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

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

5 POTATOES



While Loop

```
$ ruby recipe.rb
```

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

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



5 POTATOES



Assignment Operators

There's always
another way with Ruby.

Assignment Operators

```
# More Assignment Operators
```

```
counter = counter + 1
```

```
counter += 1
```

```
counter = counter - 1
```

```
counter -= 1
```

```
counter = counter * 1
```

```
counter *= 1
```

```
counter = counter / 1
```

```
counter /= 1
```

```
# Cut potatoes into thick slices
```

```
counter = 1
```

```
while counter < 6
```

```
  puts "Cut potato #{counter}."
```

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

```
end
```





```
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 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
```

```
for counter in 1..56
```

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

```
end
```

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



5 POTATOES





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

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

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

OR

```
collection << 'my new item'
```


Retrieving Items

Each item in an array has an **index**.

You can get at that item with square brackets.

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

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

Programmers start counting at zero!

Iterate w/ while

```
$ ruby recipe.rb
```

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

```
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

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

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

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


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.
```

```
# 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."
```

6 EGGS





Hashes

Associative collections

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

Syntax

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

```
myhash['anna'] = 28
```

Hashes associate keys with values!


Each Operator

```
$ ruby recipe.rb
```

```
Take 5 potatoes.  
Take 6 eggs.  
Take 1 onions.
```

Note that a Hash is
unordered!

```
ingredients = {  
  "onions" => 1,  
  "potatoes" => 5,  
  "eggs" => 6 }  
  
ingredients.each do | name, number |  
  puts "Take #{number} #{name}."  
end
```



LISTO!



```
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

Questions?



```
while conditional (true)  
  # Do something  
end
```

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

```
mycollection.each do |item|  
  # Do something  
end
```


Time to Code Your Own Recipe

**Choose your favourite recipe
and give it a try.**



Workshop

Now it's your turn to shine!

You can choose any of the **following exercises**, related to your e-store project.

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