Programming for Everybody

6. Hashes and symbols

le wagon

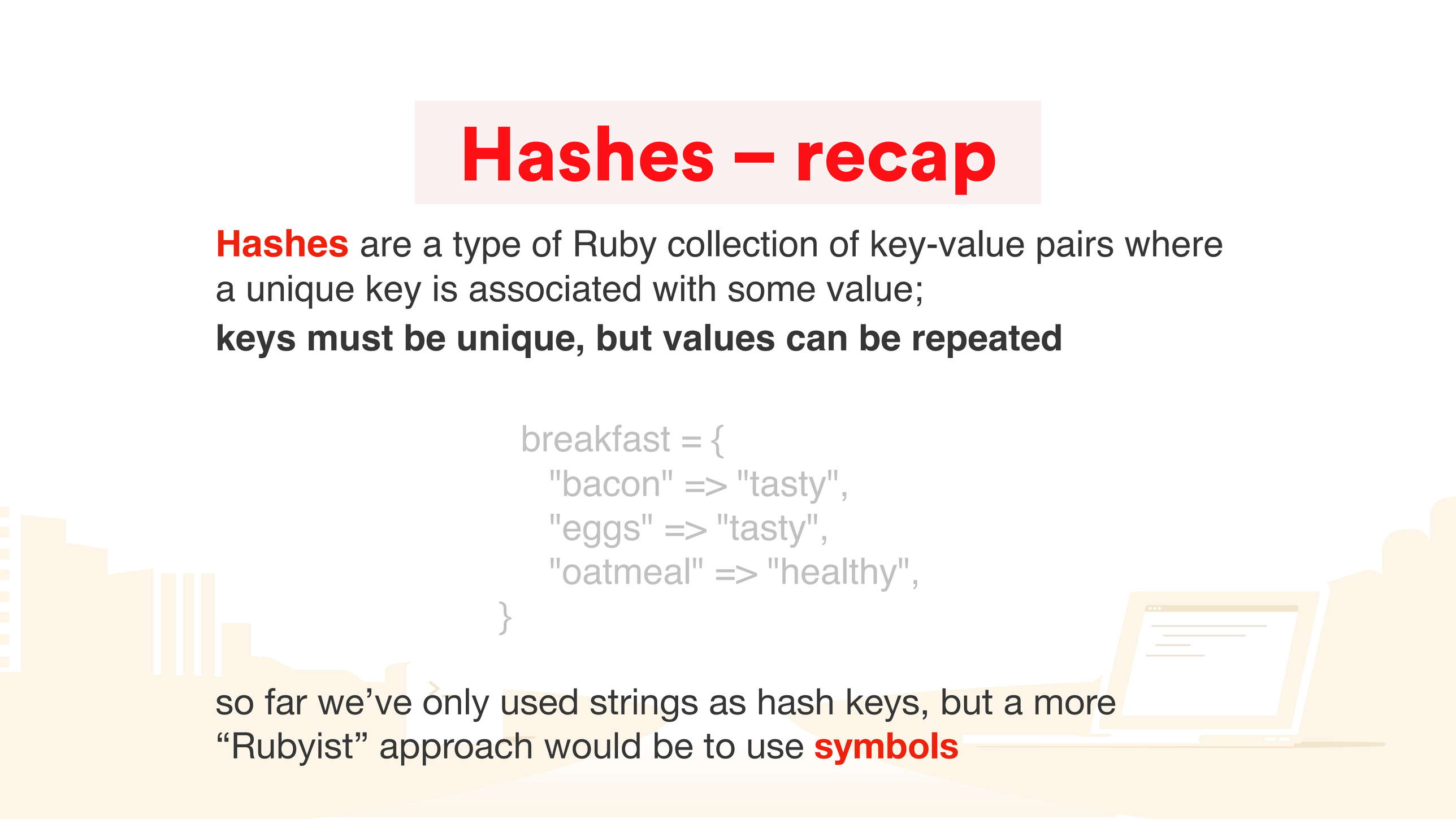
Hashes – recap

Hashes are a type of Ruby collection of key-value pairs where a unique key is associated with some value;

keys must be unique, but values can be repeated

```
breakfast = {
  "bacon" => "tasty",
  "eggs" => "tasty",
  "oatmeal" => "healthy",
}
```

so far we've only used strings as hash keys, but a more "Rubyist" approach would be to use **symbols**



Symbols as hash keys

symbols are mainly used in Ruby either as **hash keys** or for referencing **method names**

symbols-as-keys are faster than strings-as-keys because:

- they can’t be changed once they’re created
- only one copy of any symbol exists at a given time, so they save memory

symbols always start with a colon (:), the first character after the colon has to be a letter or underscore (_)
(ex :my_symbol)

Symbols as hash keys

no more strings as keys from now on!

```
my_hash = {
  “cat” => “Garfield”,
  “dog” => “Snoopy”,
  “bird” => “Tweety”
}

cat_name = my_hash[“cat”]
```

```
my_hash = {
  :cat => “Garfield”,
  :dog => “Snoopy”,
:bird => “Tweety”
}

cat_name = my_hash[:cat]
```

Converting symbols and strings

1. Converting symbols to strings

```
:test.to_s
result -> “test"
```

2. Converting strings to symbols

```
“hello".to_sym
result -> :hello
```

or

```
"hello".intern
result -> :hello
```

New symbol syntax

the hash syntax we've seen so far (with the => symbol between keys and values) is nicknamed *hash rocket* style

however, the hash syntax changed in Ruby 1.9 - no more hash rockets from now on!

```
my_hash = {
  :cat => "Garfield",
  :dog => "Snoopy",
:bird => "Tweety"
}

cat_name = my_hash[:cat]
```

```
my_hash = {
   cat: "Garfield",
   dog: "Snoopy",
 bird: "Tweety"
}

cat_name = my_hash[:cat]
```

Selecting from a hash

to filter a hash for values that meet certain criteria we can use the **.select** method

```
grades = {
  alice: 100,
  bob: 92,
  chris: 95,
  dave: 97
}

puts grades.select { |name, grade| grade < 97 }

prints out { :bob => 92, :chris => 95 }
```

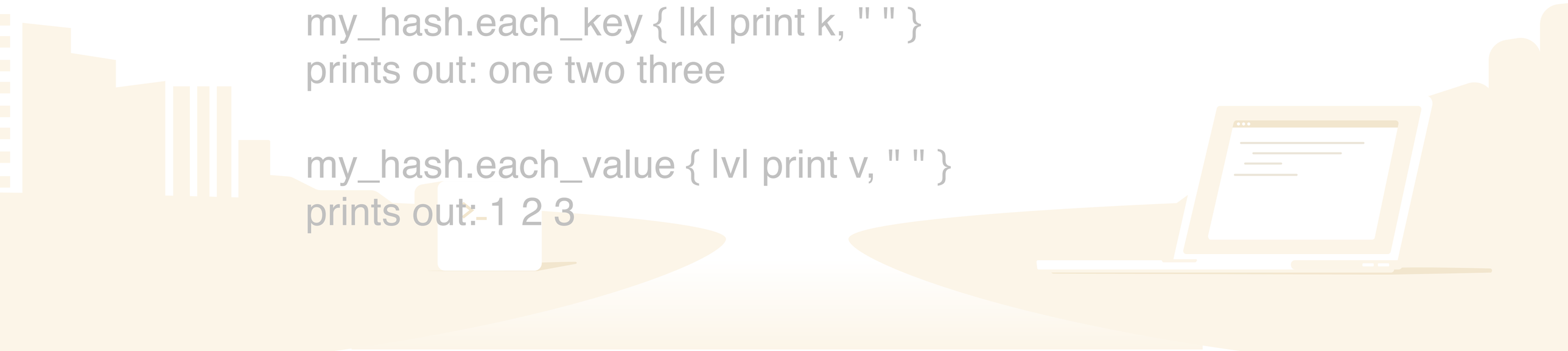
Printing keys / values

we can also iterate over **just keys** or **just values** using the **.each_key** and the **.each_value** methods

```
my_hash = { one: 1, two: 2, three: 3 }

my_hash.each_key { |k| print k, " " }
prints out: one two three

my_hash.each_value { |v| print v, " " }
prints out: 1 2 3
```



Selecting from a hash

if we try to access a key that doesn't exist we'll get *nil* as a result

but if we create our hash using the Hash.new syntax we can specify a default

```
my_hash = Hash.new("Bob")
```

=> now if we try to access a **nonexistent key** in my_hash we'll get "Bob" as a result

Thank you!