Programming for Everybody

6. Hashes & Symbols

le wagon

Hashes - a recap

hashes are a type of Ruby collection of key-value pairs, where unique keys are associated with some values

keys must be unique, but values can be repeated

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

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

symbol-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 any given time, so they save memory

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

Symbols as Hash keys (cont.)

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 between 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 => between keys and values) is nicknamed
the *hash rocket* style

However, the Hash syntax changed in Ruby 1.9:
no more *hash rockets* from now on!

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

cat_name = my_hash[:cat]
```

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

cat_name = my_hash[:cat]
```

Reading from hashes:
Setting a default value

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 value for non-existent keys

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

# now if we try to access a non-existent key in
my_hash, we'll get "Bob" as a result

my_hash[:a_key] => 'Bob'
```

Selecting from hashes

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
}

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

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

Printing just 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| puts k }
# puts out: one two three

my_hash.each_value { |v| puts v }
# puts out: 1 2 3
```

Thank you.